



# Locan Documentation

*Release 0.20.0.dev41+g755b969*

**Locan Developers**

**Mar 14, 2024**

# GETTING STARTED

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Installation</b>  | <b>2</b> |
| 1.1      | Dependencies . . . . .   | 2        |
| 1.2      | Install from pypi . . . . .                                      | 2        |
| 1.3      | Install from conda-forge . . . . .                               | 2        |
| 1.4      | Install from distribution or sources . . . . .                   | 2        |
| 1.5      | Run tests . . . . .  | 3        |
| 1.6      | Using conda to set up a dedicated environment: . . . . .         | 3        |
| 1.7      | Jupyter . . . . .  | 3        |
| 1.8      | Various installation issues . . . . .                            | 4        |
| <b>2</b> | <b>Tutorials</b>   | <b>5</b> |
| 2.1      | Tutorial about the LocData class . . . . .                       | 5        |
| 2.1.1    | Sample data . . . . .  | 6        |
| 2.1.2    | Instantiate LocData from a dataframe . . . . .                   | 6        |
| 2.1.3    | LocData attributes . . . . .                                     | 7        |
| 2.1.4    | Metadata . . . . .   | 8        |
| 2.1.5    | Instantiate locdata from selection . . . . .                     | 10       |
| 2.1.6    | Instantiate locdata from collection . . . . .                    | 11       |
| 2.1.7    | Concatenating LocData objects . . . . .                          | 12       |
| 2.1.8    | Modifying data in place . . . . .                                | 13       |
| 2.1.9    | Copy LocData . . . . .   | 15       |
| 2.1.10   | Adding a property . . . . .                                      | 17       |
| 2.1.11   | Adding a property to each localization in LocData.data . . . . . | 17       |
| 2.2      | Tutorial about loading localization data from file . . . . .     | 20       |
| 2.2.1    | Load rapidSTORM data file . . . . .                              | 22       |
| 2.2.2    | Load Zeiss Elyra data file . . . . .                             | 23       |
| 2.2.3    | Localization data from a custom text file . . . . .              | 25       |
| 2.2.4    | Load localization data file . . . . .                            | 26       |
| 2.2.5    | Adjust data types . . . . .                                      | 27       |
| 2.3      | Tutorial about computing hulls for localization data . . . . .   | 29       |
| 2.3.1    | Synthetic data . . . . .   | 30       |
| 2.3.2    | Minimal bounding box for spatial coordinates . . . . .           | 31       |
| 2.3.3    | Oriented minimal bounding box for spatial coordinates . . . . .  | 32       |
| 2.3.4    | Convex hull . . . . .  | 33       |
| 2.3.5    | Alpha shape for spatial coordinates . . . . .                    | 36       |
| 2.4      | Tutorial about metadata . . . . .                                | 45       |
| 2.4.1    | Metadata definition . . . . .                                    | 46       |
| 2.4.2    | Set metadata fields . . . . .                                    | 48       |
| 2.4.3    | Metadata scheme . . . . .  | 50       |

|        |   |     |
|--------|---|-----|
| 2.4.4  | Metadata for LocData . . . . .  | 54  |
| 2.5    | Tutorial about regions . . . . .                                      | 56  |
| 2.5.1  | Region definitions . . . . .  | 57  |
| 2.5.2  | Use Region classes . . . . .  | 58  |
| 2.5.3  | Plot regions . . . . .  | 60  |
| 2.5.4  | Intersection, union, difference of regions . . . . .                  | 61  |
| 2.5.5  | Check if point is in region . . . . .                                 | 63  |
| 2.5.6  | LocData and regions . . . . .   | 64  |
| 2.5.7  | Select localizations within regions . . . . .                         | 65  |
| 2.5.8  | Regions of interest . . . . .   | 66  |
| 2.6    | Tutorial about filtering LocData objects . . . . .                    | 69  |
| 2.6.1  | Synthetic data . . . . .  | 69  |
| 2.6.2  | Select localizations according to property conditions . . . . .       | 70  |
| 2.6.3  | Select localizations in regions . . . . .                             | 71  |
| 2.6.4  | Select localizations from a region of interest (ROI) . . . . .        | 72  |
| 2.6.5  | Select a random subset of localizations . . . . .                     | 74  |
| 2.7    | Tutorial about clustering localizations data . . . . .                | 75  |
| 2.7.1  | Synthetic data . . . . .  | 76  |
| 2.7.2  | Cluster localizations by dbscan . . . . .                             | 77  |
| 2.7.3  | Cluster localizations in the presence of noise . . . . .              | 79  |
| 2.8    | Tutorial about rendering LocData . . . . .                            | 80  |
| 2.8.1  | Synthetic data . . . . .  | 81  |
| 2.8.2  | Render by simple 2D binning . . . . .                                 | 82  |
| 2.8.3  | Use different libraries for rendering . . . . .                       | 90  |
| 2.8.4  | Choose colormaps . . . . .  | 91  |
| 2.9    | Tutorial about simulating localization data . . . . .                 | 93  |
| 2.9.1  | Use random number generator . . . . .                                 | 94  |
| 2.9.2  | Simulate localization data . . . . .                                  | 95  |
| 2.9.3  | Resample data . . . . .   | 104 |
| 2.10   | Tutorial about tracking LocData objects . . . . .                     | 105 |
| 2.10.1 | Synthetic data . . . . .  | 106 |
| 2.10.2 | Track locdata . . . . .   | 107 |
| 2.10.3 | Track using trackpy with locdata.data as input . . . . .              | 112 |
| 2.10.4 | filter . . . . .  | 113 |
| 2.10.5 | drift . . . . .   | 114 |
| 2.10.6 | Mean square displacement (msd) . . . . .                              | 115 |
| 2.11   | Tutorial about transforming LocData . . . . .                         | 118 |
| 2.11.1 | Spatially randomize a structured set of localizations . . . . .       | 119 |
| 2.11.2 | Apply an affine transformation to localization coordinates . . . . .  | 121 |
| 2.11.3 | Apply a BunwarpJ transformation to localization coordinates . . . . . | 122 |
| 2.12   | Tutorial about localizations per frame . . . . .                      | 124 |
| 2.12.1 | Load rapidSTORM data file . . . . .                                   | 125 |
| 2.12.2 | Visualization . . . . .   | 126 |
| 2.12.3 | Plot normalized values . . . . .                                      | 128 |
| 2.13   | Tutorial about localization precision . . . . .                       | 130 |
| 2.13.1 | Load SMLM data file . . . . .   | 130 |
| 2.13.2 | Visualization . . . . .   | 131 |
| 2.13.3 | Analyze localization precision . . . . .                              | 132 |
| 2.13.4 | Fit distributions and show parameters . . . . .                       | 136 |
| 2.14   | Tutorial about analyzing localization properties . . . . .            | 137 |
| 2.14.1 | Load rapidSTORM data file . . . . .                                   | 137 |

|        |   |     |
|--------|---|-----|
| 2.14.2 | Visualization . . . . .   | 138 |
| 2.14.3 | Fitting different distribution models . . . . .                               | 142 |
| 2.14.4 | Showing correlations between two properties . . . . .                         | 143 |
| 2.14.5 | 2-dimensional distribution of localization properties . . . . .               | 145 |
| 2.15   | Tutorial about analyzing blink statistics . . . . .                           | 148 |
| 2.15.1 | Synthetic data . . . . .  | 149 |
| 2.15.2 | Blinking statistics . . . . .   | 151 |
| 2.15.3 | Geometric distribution . . . . .  | 153 |
| 2.16   | Tutorial about coordinate-based colocalization . . . . .                      | 154 |
| 2.16.1 | Synthetic data . . . . .  | 155 |
| 2.16.2 | CBC computation . . . . .   | 157 |
| 2.16.3 | Results . . . . .   | 157 |
| 2.16.4 | CBC for various shifts . . . . .  | 158 |
| 2.16.5 | CBC on various length scales (for small cluster) . . . . .                    | 160 |
| 2.16.6 | CBC on various length scales (for larger cluster) . . . . .                   | 161 |
| 2.17   | Tutorial about drift analysis and correction . . . . .                        | 163 |
| 2.17.1 | Synthetic data . . . . .  | 164 |
| 2.17.2 | Add linear drift . . . . .  | 166 |
| 2.17.3 | Estimate RMS errors . . . . .   | 166 |
| 2.17.4 | Estimate drift . . . . .  | 166 |
| 2.17.5 | Model drift . . . . .   | 169 |
| 2.17.6 | Drift correction . . . . .  | 172 |
| 2.17.7 | Drift analysis by a cross-correlation algorithm . . . . .                     | 174 |
| 2.18   | Tutorial about nearest neighbor distances . . . . .                           | 178 |
| 2.18.1 | Load rapidSTORM data file . . . . .   | 178 |
| 2.18.2 | Visualization . . . . .   | 179 |
| 2.18.3 | Analyze nearest neighbor distances . . . . .                                  | 180 |
| 2.18.4 | Meta data for the analysis procedure . . . . .                                | 181 |
| 2.19   | Tutorial about Ripley's k function . . . . .                                  | 182 |
| 2.19.1 | Synthetic data . . . . .  | 182 |
| 2.19.2 | Analyze Ripley's h function . . . . .   | 185 |
| 2.19.3 | Estimate Ripley's h function . . . . .  | 186 |
| 2.19.4 | Compute Ripley's k, l and h function . . . . .                                | 188 |
| 2.19.5 | Estimate Ripley's h function for 3D data . . . . .                            | 189 |
| 2.20   | Tutorial about analyzing grouped cluster properties . . . . .                 | 190 |
| 2.20.1 | Synthetic data . . . . .  | 191 |
| 2.20.2 | True clusters . . . . .   | 192 |
| 2.20.3 | Investigate the convex hull areas . . . . .                                   | 194 |
| 2.20.4 | Investigate the position variances . . . . .                                  | 195 |
| 2.20.5 | Investigate any grouped property . . . . .                                    | 196 |
| 2.20.6 | Cluster localizations by dbscan . . . . .                                     | 197 |
| 2.20.7 | Investigate the position variances . . . . .                                  | 198 |
| 2.20.8 | Investigate the convex hull areas . . . . .                                   | 200 |
| 2.20.9 | Investigate the uncertainties for cluster centroids . . . . .                 | 200 |
| 2.21   | Tutorial about how to use a standard Analysis class . . . . .                 | 201 |
| 2.21.1 | Load SMLM data file . . . . .   | 202 |
| 2.21.2 | Visualization . . . . .   | 203 |
| 2.21.3 | A simple analysis procedure: localization precision . . . . .                 | 204 |
| 2.21.4 | Metadata can be used to rerun the analysis with the same parameter. . . . .   | 209 |
| 2.22   | Tutorial about setting up an analysis pipeline and batch processing . . . . . | 210 |
| 2.22.1 | Apply a pipeline of different analysis routines . . . . .                     | 211 |

|          |   |            |
|----------|---|------------|
| 2.22.2   | Apply the pipeline on multiple datasets - a batch process . . . . . | 214        |
| 2.23     | Tutorial about managing files in batch processing . . . . .         | 217        |
| 2.23.1   | Some file structure to be analysed . . . . .                        | 217        |
| 2.23.2   | The Files class . . . . .   | 218        |
| 2.23.3   | Identify files . . . . .  | 218        |
| 2.23.4   | Exclude files . . . . .   | 219        |
| 2.23.5   | Match corresponding files . . . . .                                 | 220        |
| 2.23.6   | Match metadata files . . . . .                                      | 221        |
| 2.23.7   | Group files . . . . .   | 221        |
| 2.23.8   | Indexing and iterating over files . . . . .                         | 222        |
| 2.24     | Tutorial about multiprocessing using ray . . . . .                  | 223        |
| 2.24.1   | Activate logging . . . . .  | 224        |
| 2.24.2   | Synthetic data . . . . .  | 224        |
| 2.24.3   | Analysis pipeline . . . . .   | 225        |
| 2.24.4   | Run analysis in parallel . . . . .                                  | 225        |
| 2.24.5   | Visualize the combined results . . . . .                            | 226        |
| 2.25     | Tutorial about example datasets . . . . .                           | 227        |
| 2.25.1   | Load SMLM data from ShareLoc.XYZ . . . . .                          | 228        |
| 2.25.2   | Load SMLM data from LocanDatasets . . . . .                         | 231        |
| 2.26     | Tutorial about logging . . . . .                                    | 237        |
| 2.26.1   | Activate logging . . . . .  | 238        |
| 2.26.2   | Logging in locan . . . . .  | 239        |
| 2.26.3   | Logging in locan - third party libraries . . . . .                  | 242        |
| 2.27     | Notebook execution table . . . . .                                  | 245        |
| <b>3</b> | <b>Docker</b>   | <b>246</b> |
| 3.1      | Prepare for using Docker . . . . .                                  | 246        |
| 3.2      | Dockerfiles . . . . .   | 246        |
| 3.3      | Build a docker image . . . . .                                      | 246        |
| 3.4      | Start a container from the image . . . . .                          | 247        |
| 3.4.1    | Run project tests: . . . . .  | 247        |
| 3.4.2    | Run project in an interactive environment: . . . . .                | 247        |
| 3.4.3    | Use Jupyter notebooks: . . . . .                                    | 247        |
| 3.5      | Clean up . . . . .  | 247        |
| <b>4</b> | <b>Package Design</b>   | <b>248</b> |
| 4.1      | Data structures . . . . .   | 248        |
| 4.1.1    | Comments on LocData: . . . . .                                      | 248        |
| 4.2      | Metadata . . . . .  | 249        |
| 4.2.1    | Structure of metadata for LocData . . . . .                         | 249        |
| 4.2.2    | Structure of metadata for analysis classes . . . . .                | 251        |
| 4.3      | Properties . . . . .  | 251        |
| 4.3.1    | Localization properties: . . . . .                                  | 252        |
| 4.3.2    | LocData properties: . . . . .                                       | 253        |
| 4.4      | Methods for data analysis . . . . .                                 | 254        |
| 4.5      | Aim . . . . .   | 255        |
| 4.6      | Outline . . . . .   | 255        |
| <b>5</b> | <b>Command-line interface</b>                                       | <b>256</b> |
| <b>6</b> | <b>Colormaps</b>  | <b>257</b> |

|          |   |            |
|----------|---|------------|
| <b>7</b> | <b>User Interface</b>                           | <b>258</b> |
| <b>8</b> | <b>API Reference</b>                            | <b>259</b> |
| 8.1      | locan.analysis . . . . .                        | 259        |
| 8.1.1    | Submodules: . . . . .                           | 260        |
| 8.2      | locan.constants . . . . .                       | 339        |
| 8.2.1    | locan.constants.ROOT_DIR . . . . .              | 339        |
| 8.2.2    | locan.constants.PROPERTY_KEYS . . . . .         | 339        |
| 8.2.3    | locan.constants.DECODE_KEYS . . . . .           | 340        |
| 8.2.4    | locan.constants.ELYRA_KEYS . . . . .            | 340        |
| 8.2.5    | locan.constants.NANOIMAGER_KEYS . . . . .       | 340        |
| 8.2.6    | locan.constants.RAPIDSTORM_KEYS . . . . .       | 341        |
| 8.2.7    | locan.constants.SMAP_KEYS . . . . .             | 341        |
| 8.2.8    | locan.constants.SMLM_KEYS . . . . .             | 341        |
| 8.2.9    | locan.constants.THUNDERSTORM_KEYS . . . . .     | 341        |
| 8.2.10   | locan.constants.FileType . . . . .              | 342        |
| 8.2.11   | locan.constants.HullType . . . . .              | 344        |
| 8.2.12   | locan.constants.PropertyDescription . . . . .   | 344        |
| 8.2.13   | locan.constants.PropertyKey . . . . .           | 345        |
| 8.2.14   | locan.constants.RenderEngine . . . . .          | 351        |
| 8.3      | locan.configuration . . . . .                   | 352        |
| 8.3.1    | locan.configuration.DATASETS_DIR . . . . .      | 352        |
| 8.3.2    | locan.configuration.RENDER_ENGINE . . . . .     | 352        |
| 8.3.3    | locan.configuration.N_JOBS . . . . .            | 352        |
| 8.3.4    | locan.configuration.COLORMAP_DEFAULTS . . . . . | 352        |
| 8.3.5    | locan.configuration.TQDM_LEAVE . . . . .        | 353        |
| 8.3.6    | locan.configuration.TQDM_DISABLE . . . . .      | 353        |
| 8.4      | locan.data . . . . .                            | 353        |
| 8.4.1    | Submodules: . . . . .                           | 353        |
| 8.5      | locan.datasets . . . . .                        | 453        |
| 8.5.1    | locan.datasets.load_npc . . . . .               | 454        |
| 8.5.2    | locan.datasets.load_tubulin . . . . .           | 454        |
| 8.6      | locan.dependencies . . . . .                    | 455        |
| 8.6.1    | locan.dependencies.INSTALL_REQUIRES . . . . .   | 455        |
| 8.6.2    | locan.dependencies.EXTRAS_REQUIRE . . . . .     | 455        |
| 8.6.3    | locan.dependencies.IMPORT_NAMES . . . . .       | 456        |
| 8.6.4    | locan.dependencies.HAS_DEPENDENCY . . . . .     | 456        |
| 8.6.5    | locan.dependencies.QtBindings . . . . .         | 456        |
| 8.6.6    | locan.dependencies.needs_package . . . . .      | 457        |
| 8.7      | locan.gui . . . . .                             | 458        |
| 8.7.1    | Submodules: . . . . .                           | 458        |
| 8.8      | locan.locan_io . . . . .                        | 459        |
| 8.8.1    | Submodules: . . . . .                           | 459        |
| 8.9      | locan.rois . . . . .                            | 481        |
| 8.9.1    | Submodules: . . . . .                           | 481        |
| 8.10     | locan.scripts . . . . .                         | 485        |
| 8.10.1   | Submodules: . . . . .                           | 486        |
| 8.11     | locan.simulation . . . . .                      | 493        |
| 8.11.1   | Submodules: . . . . .                           | 493        |
| 8.12     | locan.tests . . . . .                           | 508        |
| 8.12.1   | locan.tests.test . . . . .                      | 509        |

|          |                                       |            |
|----------|---------------------------------------|------------|
| 8.13     | locan.utils . . . . .                 | 509        |
| 8.13.1   | Submodules: . . . . .                 | 509        |
| 8.14     | locan.visualize . . . . .             | 514        |
| 8.14.1   | Submodules: . . . . .                 | 514        |
| <b>9</b> | <b>Changelog</b>                      | <b>543</b> |
| 9.1      | 0.20 - dev . . . . .                  | 543        |
| 9.1.1    | New Features . . . . .                | 543        |
| 9.1.2    | API Changes . . . . .                 | 543        |
| 9.1.3    | Bug Fixes . . . . .                   | 543        |
| 9.1.4    | Other Changes and Additions . . . . . | 543        |
| 9.2      | 0.19 - 2023-12-12 . . . . .           | 544        |
| 9.2.1    | Other Changes and Additions . . . . . | 544        |
| 9.3      | 0.18 - 2023-12-06 . . . . .           | 544        |
| 9.3.1    | API Changes . . . . .                 | 544        |
| 9.3.2    | Bug Fixes . . . . .                   | 544        |
| 9.3.3    | Other Changes and Additions . . . . . | 544        |
| 9.4      | 0.17 - 2023-10-26 . . . . .           | 544        |
| 9.4.1    | New Features . . . . .                | 544        |
| 9.4.2    | API Changes . . . . .                 | 544        |
| 9.4.3    | Bug Fixes . . . . .                   | 545        |
| 9.4.4    | Other Changes and Additions . . . . . | 545        |
| 9.5      | 0.16 - 2023-10-05 . . . . .           | 545        |
| 9.5.1    | API Changes . . . . .                 | 545        |
| 9.5.2    | Bug Fixes . . . . .                   | 545        |
| 9.5.3    | Other Changes and Additions . . . . . | 545        |
| 9.6      | 0.15 - 2023-09-28 . . . . .           | 545        |
| 9.6.1    | New Features . . . . .                | 545        |
| 9.6.2    | API Changes . . . . .                 | 545        |
| 9.6.3    | Bug Fixes . . . . .                   | 546        |
| 9.6.4    | Other Changes and Additions . . . . . | 546        |
| 9.7      | 0.14 - 2023-06-30 . . . . .           | 546        |
| 9.7.1    | New Features . . . . .                | 546        |
| 9.7.2    | API Changes . . . . .                 | 546        |
| 9.7.3    | Bug Fixes . . . . .                   | 547        |
| 9.7.4    | Other Changes and Additions . . . . . | 547        |
| 9.8      | 0.13 - 2023-02-15 . . . . .           | 547        |
| 9.8.1    | New Features . . . . .                | 547        |
| 9.8.2    | API Changes . . . . .                 | 547        |
| 9.8.3    | Bug Fixes . . . . .                   | 547        |
| 9.8.4    | Other Changes and Additions . . . . . | 548        |
| 9.9      | 0.12 - 2022-06-02 . . . . .           | 548        |
| 9.9.1    | API Changes . . . . .                 | 548        |
| 9.9.2    | Bug Fixes . . . . .                   | 549        |
| 9.9.3    | Other Changes and Additions . . . . . | 549        |
| 9.10     | 0.11.1 - 2022-04-08 . . . . .         | 549        |
| 9.10.1   | Bug Fixes . . . . .                   | 549        |
| 9.11     | 0.11 - 2022-03-22 . . . . .           | 549        |
| 9.11.1   | New Features . . . . .                | 549        |
| 9.11.2   | API Changes . . . . .                 | 550        |
| 9.11.3   | Bug Fixes . . . . .                   | 550        |

|           |                                       |            |
|-----------|---------------------------------------|------------|
| 9.11.4    | Other Changes and Additions . . . . . | 550        |
| 9.12      | 0.10 - 2021-11-23 . . . . .           | 550        |
| 9.12.1    | New Features . . . . .                | 550        |
| 9.12.2    | Other Changes and Additions . . . . . | 551        |
| 9.13      | 0.9 - 2021-11-11 . . . . .            | 551        |
| 9.13.1    | API Changes . . . . .                 | 551        |
| 9.13.2    | Other Changes and Additions . . . . . | 551        |
| 9.14      | 0.8 - 2021-05-06 . . . . .            | 552        |
| 9.14.1    | API Changes . . . . .                 | 552        |
| 9.14.2    | Bug Fixes . . . . .                   | 552        |
| 9.14.3    | Other Changes and Additions . . . . . | 552        |
| 9.15      | 0.7 - 2021-03-26 . . . . .            | 552        |
| 9.15.1    | API Changes . . . . .                 | 552        |
| 9.15.2    | Other Changes and Additions . . . . . | 552        |
| 9.16      | 0.6 - 2021-03-04 . . . . .            | 553        |
| 9.16.1    | New Features . . . . .                | 553        |
| 9.16.2    | API Changes . . . . .                 | 553        |
| 9.16.3    | Bug Fixes . . . . .                   | 554        |
| 9.16.4    | Other Changes and Additions . . . . . | 555        |
| 9.17      | 0.5.1 - 2020-03-25 . . . . .          | 555        |
| 9.18      | 0.5 - 2020-03-22 . . . . .            | 555        |
| 9.18.1    | New Features . . . . .                | 555        |
| 9.18.2    | API Changes . . . . .                 | 556        |
| 9.18.3    | Bug Fixes . . . . .                   | 556        |
| 9.19      | 0.4.1 - 2020-02-16 . . . . .          | 556        |
| 9.19.1    | Bug Fixes . . . . .                   | 556        |
| 9.19.2    | Other Changes and Additions . . . . . | 556        |
| 9.20      | 0.4 - 2020-02-13 . . . . .            | 556        |
| 9.20.1    | New Features . . . . .                | 556        |
| 9.20.2    | API Changes . . . . .                 | 557        |
| 9.20.3    | Bug Fixes . . . . .                   | 558        |
| 9.20.4    | Other Changes and Additions . . . . . | 558        |
| 9.21      | 0.3 - 2019-07-09 . . . . .            | 558        |
| 9.21.1    | New Features . . . . .                | 558        |
| 9.21.2    | API Changes . . . . .                 | 558        |
| 9.21.3    | Other Changes and Additions . . . . . | 559        |
| 9.22      | 0.2 - 2019-03-22 . . . . .            | 559        |
| 9.22.1    | New Features . . . . .                | 559        |
| 9.22.2    | API Changes . . . . .                 | 559        |
| 9.22.3    | Bug Fixes . . . . .                   | 560        |
| 9.23      | 0.1 - 2018-12-09 . . . . .            | 560        |
| 9.23.1    | New Features . . . . .                | 560        |
| 9.23.2    | Other Changes and Additions . . . . . | 561        |
| <b>10</b> | <b>Contributions</b>                  | <b>562</b> |
| 10.1      | Idea and Lead . . . . .               | 562        |
| 10.2      | Various Contributions . . . . .       | 562        |
| <b>11</b> | <b>License</b>                        | <b>563</b> |
| <b>12</b> | <b>Other Licenses</b>                 | <b>564</b> |



|                                     |            |
|-------------------------------------|------------|
| <b>13 Documentation</b>             | <b>565</b> |
| 13.1 Update documentation . . . . . | 565        |
| 13.2 Type hints . . . . .           | 565        |
| 13.3 Example docstring . . . . .    | 565        |
| <b>14 Development</b>               | <b>568</b> |
| 14.1 Install . . . . .              | 568        |
| 14.2 Import Conventions . . . . .   | 568        |
| 14.3 Unit tests . . . . .           | 569        |
| 14.4 Coverage . . . . .             | 569        |
| 14.5 Versioning . . . . .           | 569        |
| 14.6 To remember: . . . . .         | 569        |
| <b>15 Indices and tables</b>        | <b>570</b> |
| <b>Python Module Index</b>          | <b>571</b> |
| <b>Index</b>                        | <b>573</b> |



Locan is a python-based library with code for analyzing single-molecule localization data. Such data is typically generated in fluorescence-based super-resolution microscopy methods. Single-molecule localization microscopy (SMLM) techniques rely on finding the position of single-molecule emitters in time and space and reconstructing a super-resolved image or movie. The generated localizations are analyzed point-by-point for statistical and structural insight.

## INSTALLATION

### 1.1 Dependencies

- python 3
- standard scipy and other open source libraries

A list with all hard and optional dependencies is given in *pyproject.toml*, *environment.yml* and *requirements.txt*.

### 1.2 Install from pypi

Install locan directly from the Python Package Index:

```
pip install locan
```

Extra dependencies can be included:

```
pip install locan[all]
```

### 1.3 Install from conda-forge

Install locan with the conda package manager:

```
conda install -c conda-forge locan
```

### 1.4 Install from distribution or sources

In order to get the latest changes install from the GitHub repository main branch:

```
pip install git+https://github.com/super-resolution/Locan.git@main
```

or download distribution or wheel archive and install with pip:

```
pip install <distribution_file>
```

Install from local sources:

```
pip install <locan_directory>
```

## 1.5 Run tests

Use pytest to run the tests from the source directory:

```
pytest
```

Or run a locan script from any directory:

```
locan test
```

## 1.6 Using conda to set up a dedicated environment:

- 1) Install miniconda or anaconda (platform-independent)
- 2) Setup a new environment from the environment.yml file:

```
conda env create --file "./environment.yml"
```

or with specific python version:

```
conda create --name locan python=3.10  
conda env update --name locan --file "./environment.yml"
```

- 3) Activate the environment and install locan.

We recommend using [mamba](#) to speed up dependency resolution:

```
mamba create --name locan python=3.10  
mamba env update --name locan --file "./environment.yml"  
mamba install --name locan -c conda-forge locan  
conda activate locan
```

## 1.7 Jupyter

To work with jupyter notebooks install jupyter lab:

```
pip install jupyterlab
```

or inside a conda environment:

```
conda install -c conda-forge jupyterlab
```

Make sure to add the appropriate lab extensions:

```
jupyter labextension install @jupyter-widgets/jupyterlab-manager \
                             @pyviz/jupyterlab_pyviz \
                             jupyter-matplotlib
```

You may need to install node.js for rebuilding jupyter lab:

```
conda install -c conda-forge nodejs
```

## 1.8 Various installation issues

### 1) Numba requires specific numpy version:

Numba might not be compatible with the latest numpy version.

Solution: Install numba first.

### 2) Building a wheel for hdbscan raises error:

Building a wheel for hdbscan during installation might cause the following error:

“ValueError: numpy.ndarray size changed, may indicate binary incompatibility. “

This error arises from version incompatibility between the numpy version installed in the current environment and the one used for building the wheel.

Solution: Install wheel, cython, and numpy (or numba) with oldest-supported-numpy first, then build hdbscan using the installed versions (and not in isolation as done by default), and finally install locan:

```
pip install wheel cython numba oldest-supported-numpy
pip install hdbscan --no-build-isolation
pip install locan[all]
```

### 3) Running “locan napari” in conda environment raises error:

Starting napari in a conda environment with python>=3.8 and pyside2 causes the following error:

“RuntimeError: PySide2 rcc binary not found in...”

Seems like napari>0.4.5 does not work with pyside2<5.14 due to the replacement of pyside2-uic/pyside2-rcc.

Solution: Set up an environment with python 3.7. Or set up an environment with only pyqt5 instead of pyside2. Or, if both pyqt5 and pyside2 are installed, set the environment variable “QT\_API”:

```
import os
os.environ["QT_API"] = "pyqt5"
```

## TUTORIALS

Tutorials give selected application examples to get familiar with typical use cases.

Tutorials are provided as Jupyter notebooks.

### 2.1 Tutorial about the LocData class

```
import numpy as np
import pandas as pd

import locan as lc
```

```
/tmp/ipykernel_1475/3754365477.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
↳of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳54466

import pandas as pd
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.1.1 Sample data

A localization has certain properties such as 'position\_x'. A list of localizations can be assembled into a dataframe:

```
df = pd.DataFrame(
    {
        'position_x': np.arange(0,10),
        'position_y': np.random.random(10),
        'frame': np.arange(0,10),
    })
```

### 2.1.2 Instantiate LocData from a dataframe

A LocData object carries localization data together with metadata and aggregated properties for the whole set of localizations.

We first instantiate a LocData object from the dataframe:

```
locdata = lc.LocData.from_dataframe(dataframe=df)
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

```
attributes = [x for x in dir(locdata) if not x.startswith('_')]
attributes
```

```
['alpha_shape',
'bounding_box',
'centroid',
'concat',
'convex_hull',
'coordinate_keys',
'coordinates',
'count',
'data',
'dataframe',
'dimension',
'from_chunks',
'from_collection',
'from_coordinates',
'from_dataframe',
'from_selection',
'indices',
'inertia_moments',
'meta',
'oriented_bounding_box',
'print_meta',
'print_summary',
```

(continues on next page)

(continued from previous page)

```
'projection',
'properties',
'reduce',
'references',
'region',
'reset',
'uncertainty_keys',
'update',
'update_alpha_shape',
'update_alpha_shape_in_references',
'update_convex_hulls_in_references',
'update_inertia_moments_in_references',
'update_oriented_bounding_box_in_references',
'update_properties_in_references']
```

### 2.1.3 LocData attributes

The class attribute `Locdata.count` represents the number of all current `LocData` instantiations.

```
print('LocData count: ', lc.LocData.count)
```

```
LocData count: 1
```

The localization dataset is provided by the data attribute:

```
print(locdata.data.head())
```

|   | position_x | position_y | frame |
|---|------------|------------|-------|
| 0 | 0          | 0.721990   | 0     |
| 1 | 1          | 0.408696   | 1     |
| 2 | 2          | 0.684785   | 2     |
| 3 | 3          | 0.313493   | 3     |
| 4 | 4          | 0.740382   | 4     |

Aggregated properties are provided by the attribute properties. E.g. the property `position_x` represents the mean of the `position_x` for all localizations. We keep the name, since the aggregated dataset can be treated as just a single locdata event with `position_x`. This is used when dealing with data clusters.

```
locdata.properties
```

```
{'localization_count': 10,
'position_x': 4.5,
'uncertainty_x': 0.9574271077563381,
'position_y': 0.551084446025565,
'uncertainty_y': 0.08032460018240109,
'frame': 0,
'region_measure_bb': 6.405344503496165,
'localization_density_bb': 1.561196278286328,
'subregion_measure_bb': 19.423409889665816}
```



Since spatial coordinates are quite important one can check on *coordinate\_keys* and *dimension*:

```
locdata.coordinate_keys
```

```
['position_x', 'position_y']
```

```
locdata.dimension
```

```
2
```

A numpy array of spatial coordinates is returned by:

```
locdata.coordinates
```

```
array([[0.      , 0.72198962],
       [1.      , 0.40869622],
       [2.      , 0.68478475],
       [3.      , 0.31349324],
       [4.      , 0.7403815 ],
       [5.      , 0.94204647],
       [6.      , 0.33209912],
       [7.      , 0.32689362],
       [8.      , 0.23034153],
       [9.      , 0.81011839]])
```

## 2.1.4 Metadata

For detailed information see the Tutorial about metadata.

Metadata is provided by the attribute *meta* and can be printed as

```
locdata.print_meta()
```

```
identifier: "1"
source: DESIGN
state: RAW
history {
  name: "LocData.from_dataframe"
}
element_count: 10
frame_count: 10
creation_time {
  2024-03-14T11:08:55.025282Z
}
```

A summary of the most important metadata fields is printed as:

```
locdata.print_summary()
```

```

identifier: "1"
comment: ""
source: DESIGN
state: RAW
element_count: 10
frame_count: 10
creation_time {
  2024-03-14T11:08:55.025282Z
}

```

Metadata fields can be printed and changed individually:

```

print(locdata.meta.comment)
locdata.meta.comment = 'user comment'
print(locdata.meta.comment)

```

```

user comment

```

LocData.meta.map represents a dictionary structure that can be filled by the user. Both key and value have to be strings, if not a TypeError is thrown.

```

print(locdata.meta.map)
locdata.meta.map['user field'] = 'more information'
print(locdata.meta.map)

```

```

{}
{'user field': 'more information'}

```

Metadata can also be added at Instantiation:

```

locdata_2 = lc.LocData.from_dataframe(dataframe=df, meta={'identifier': 'myID_
↪1',
                                                         'comment': 'my own user_
↪comment'})
locdata_2.print_summary()

```

```

identifier: "myID_1"
comment: "my own user comment"
source: DESIGN
state: RAW
element_count: 10
frame_count: 10
creation_time {
  2024-03-14T11:08:55.107563Z
}

```

### 2.1.5 Instantiate locdata from selection

A LocData object can also be instantiated from a selection of localizations. In this case the LocData object keeps a reference to the original locdata together with a list of indices (or a slice object). The new dataset is assembled on request of the data attribute.

*Typically a selection is derived using a selection method such that using `LocData.from_selection()` is not often necessary.*

```
locdata_2 = lc.LocData.from_selection(locdata, indices=[1,2,3,4])
locdata_3 = lc.LocData.from_selection(locdata, indices=[5,6,7,8])

print('count: ', lc.LocData.count)
print('')
print(locdata_2.data)
```

```
count: 3

   position_x  position_y  frame
1           1    0.408696      1
2           2    0.684785      2
3           3    0.313493      3
4           4    0.740382      4
```

```
locdata_2.print_summary()
```

```
identifier: "3"
comment: "user comment"
source: DESIGN
state: MODIFIED
element_count: 4
frame_count: 4
creation_time {
  2024-03-14T11:08:55.025282Z
}
modification_time {
  2024-03-14T11:08:55.114205Z
}
```

The reference is kept in a private attribute as are the indices.

```
print(locdata_2.references)
print(locdata_2.indices)
```

```
<locan.data.locdata.LocData object at 0x7ff80c12add0>
[1, 2, 3, 4]
```

The reference is the same for both selections.

```
print(locdata_2.references is locdata_3.references)
```

True

### 2.1.6 Instantiate locdata from collection

A LocDat object can further be instantiated from a collection of other LocData objects.

```
del(locdata_2, locdata_3)

locdata_1 = lc.LocData.from_selection(locdata, indices=[0,1,2])
locdata_2 = lc.LocData.from_selection(locdata, indices=[3,4,5])
locdata_3 = lc.LocData.from_selection(locdata, indices=[6,7,8])
locdata_c = lc.LocData.from_collection(locdatas=[locdata_1, locdata_2,
↪locdata_3], meta={'identifier': 'my_collection'})

print('count: ', lc.LocData.count, '\n')
print(locdata_c.data, '\n')
print(locdata_c.properties, '\n')
locdata_c.print_summary()
```

```
count:  5
```

|   | localization_count | position_x | uncertainty_x | position_y | uncertainty_y \ |
|---|--------------------|------------|---------------|------------|-----------------|
| 0 | 3                  | 1.0        | 0.57735       | 0.605157   | 0.098816        |
| 1 | 3                  | 4.0        | 0.57735       | 0.665307   | 0.185290        |
| 2 | 3                  | 7.0        | 0.57735       | 0.296445   | 0.033086        |

|   | frame | region_measure_bb | localization_density_bb | subregion_measure_bb |
|---|-------|-------------------|-------------------------|----------------------|
| 0 | 0     | 0.626587          | 4.787844                | 4.626587             |
| 1 | 3     | 1.257106          | 2.386433                | 5.257106             |
| 2 | 6     | 0.203515          | 14.740915               | 4.203515             |

```
{'localization_count': 3, 'position_x': 4.0, 'uncertainty_x': 1.
↪7320508075688772, 'position_y': 0.3369779066935461, 'uncertainty_y': 0.
↪055178550208717245, 'frame': 0, 'region_measure_bb': 2.213173912125862,
↪'localization_density_bb': 1.3555193216236465, 'subregion_measure_bb': 12.
↪737724637375287}
```

```
identifier: "my_collection"
comment: ""
source: DESIGN
state: RAW
element_count: 3
frame_count: 3
creation_time {
  2024-03-14T11:08:55.264923Z
}
```

In this case the reference are also kept in case the original localizations from the collected LocData object are requested.

```
print(locdata_c.references)
```

```
[<locan.data.locdata.LocData object at 0x7ff7fe8295d0>, <locan.data.locdata.
↳LocData object at 0x7ff80c10ac90>, <locan.data.locdata.LocData object at_
↳0x7ff7fe85d490>]
```

In case the collected LocData objects are not needed anymore and should be free for garbage collection the references can be deleted by a dedicated Locdata method

```
locdata_c.reduce()
print(locdata_c.references)
```

```
None
```

## 2.1.7 Concatenating LocData objects

Lets have a second dataset with localization data:

```
del(locdata_2)

df_2 = pd.DataFrame(
    {
        'position_x': np.arange(0,10),
        'position_y': np.random.random(10),
        'frame': np.arange(0,10),
    })

locdata_2 = lc.LocData.from_dataframe(dataframe=df_2)

print('First locdata:')
print(locdata.data.head())
print('')
print('Second locdata:')
print(locdata_2.data.head())
```

```
First locdata:
   position_x  position_y  frame
0           0    0.721990      0
1           1    0.408696      1
2           2    0.684785      2
3           3    0.313493      3
4           4    0.740382      4
```

```
Second locdata:
   position_x  position_y  frame
0           0    0.920945      0
1           1    0.278350      1
2           2    0.306932      2
```

(continues on next page)

(continued from previous page)

|   |   |          |   |
|---|---|----------|---|
| 3 | 3 | 0.843144 | 3 |
| 4 | 4 | 0.623920 | 4 |

In order to combine two sets of localization data from two `LocData` objects into a single `LocData` object use the class method `LocData.concat`:

```
locdata_new = lc.LocData.concat([locdata, locdata_2])
print('Number of localizations in locdata_new: ', len(locdata_new))
locdata_new.data.head()
```

```
Number of localizations in locdata_new: 20
```

|   | position_x | position_y | frame |
|---|------------|------------|-------|
| 0 | 0          | 0.721990   | 0     |
| 1 | 1          | 0.408696   | 1     |
| 2 | 2          | 0.684785   | 2     |
| 3 | 3          | 0.313493   | 3     |
| 4 | 4          | 0.740382   | 4     |

## 2.1.8 Modifying data in place

In case localization data has been modified in place, i.e. the dataset attribute is changed, all properties and hulls must be recomputed. This is best done by re-instantiating the `LocData` object using `LocData.from_dataframe()`; but it can also be done using the `LocData.reset()` function.

```
del(df, locdata)

df = pd.DataFrame(
    {
        'position_x': np.arange(0,10),
        'position_y': np.random.random(10),
        'frame': np.arange(0,10),
    })

locdata = lc.LocData.from_dataframe(dataframe=df)

print(locdata.data.head())
```

|   | position_x | position_y | frame |
|---|------------|------------|-------|
| 0 | 0          | 0.335945   | 0     |
| 1 | 1          | 0.437390   | 1     |
| 2 | 2          | 0.181114   | 2     |
| 3 | 3          | 0.530118   | 3     |
| 4 | 4          | 0.680089   | 4     |

```
locdata.centroid
```

```
array([4.5      , 0.51240132])
```

Now if localization data is changed in place (which you should not do unless you have a good reason), properties and bounding box are not automatically adjusted.

```
locdata.dataframe = pd.DataFrame(
    {
        'position_x': np.arange(0,8),
        'position_y': np.random.random(8),
        'frame': np.arange(0,8),
    })

print(locdata.data.head())
```

|   | position_x | position_y | frame |
|---|------------|------------|-------|
| 0 | 0          | 0.138824   | 0     |
| 1 | 1          | 0.606688   | 1     |
| 2 | 2          | 0.559792   | 2     |
| 3 | 3          | 0.130943   | 3     |
| 4 | 4          | 0.775142   | 4     |

```
locdata.centroid # so this returns incorrect values here
```

```
array([4.5      , 0.51240132])
```

Update them by re-instantiating a new LocData object:

```
locdata_new = lc.LocData.from_dataframe(dataframe=locdata.data)
```

```
locdata_new.centroid
```

```
array([3.5      , 0.37336717])
```

```
locdata_new.meta
```

```
identifier: "12"
source: DESIGN
state: RAW
history {
  name: "LocData.from_dataframe"
}
element_count: 8
frame_count: 8
creation_time {
  seconds: 1710414535
  nanos: 343495000
}
```

Alternatively you can use `reset()`. In this case, however, metadata is not updated and will provide wrong information.

```
locdata.reset()
```

```
<locan.data.locdata.LocData at 0x7ff7fe670e10>
```

```
locdata.centroid
```

```
array([3.5      , 0.37336717])
```

```
locdata.meta
```

```
identifier: "11"  
source: DESIGN  
state: RAW  
history {  
  name: "LocData.from_dataframe"  
}  
element_count: 10  
frame_count: 10  
creation_time {  
  seconds: 1710414535  
  nanos: 313441000  
}
```

### 2.1.9 Copy LocData

Shallow and deep copies can be made from LocData instances. In either case the class variable count and the metadata is not just copied but adjusted accordingly.

```
print('count: ', lc.LocData.count)  
print('')  
print(locdata_2.meta)
```

```
count: 7  
  
identifier: "9"  
source: DESIGN  
state: RAW  
history {  
  name: "LocData.from_dataframe"  
}  
element_count: 10  
frame_count: 10  
creation_time {  
  seconds: 1710414535  
  nanos: 289072000  
}
```



```
from copy import copy, deepcopy

print('count before: ', lc.LocData.count)
locdata_copy = copy(locdata_2)
locdata_deepcopy = deepcopy(locdata_2)
print('count after: ', lc.LocData.count)
```

```
count before:  7
count after:   9
```

```
print(locdata_copy.meta)
```

```
identifier: "13"
source: DESIGN
state: MODIFIED
history {
  name: "LocData.from_dataframe"
}
history {
  name: "LocData.copy"
  parameter: "None"
}
ancestor_identifiers: "9"
element_count: 10
frame_count: 10
creation_time {
  seconds: 1710414535
  nanos: 289072000
}
modification_time {
  seconds: 1710414535
  nanos: 390941000
}
```

```
print(locdata_deepcopy.meta)
```

```
identifier: "14"
source: DESIGN
state: MODIFIED
history {
  name: "LocData.from_dataframe"
}
history {
  name: "LocData.deepcopy"
  parameter: "None"
}
ancestor_identifiers: "9"
element_count: 10
frame_count: 10
```

(continues on next page)

(continued from previous page)

```
creation_time {
  seconds: 1710414535
  nanos: 289072000
}
modification_time {
  seconds: 1710414535
  nanos: 392597000
}
```

### 2.1.10 Adding a property

Any property that is created for a set of localizations (and represented as a python dictionary) can be added to the Locdata object. As an example, we compute the maximum distance between any two localizations and add that `max_distance` as new property to `locdata`.

```
max_distance = lc.max_distance(locdata)
max_distance
```

```
{'max_distance': 7.00140004632628}
```

```
locdata.properties.update(max_distance)
locdata.properties
```

```
{'localization_count': 8,
 'position_x': 3.5,
 'uncertainty_x': 0.8660254037844386,
 'position_y': 0.37336717472931963,
 'uncertainty_y': 0.08596222573944916,
 'frame': 0,
 'region_measure_bb': 4.509393984363259,
 'localization_density_bb': 1.7740743052704513,
 'subregion_measure_bb': 15.288398281246645,
 'region_measure_ch': 2.8511486808868756,
 'localization_density_ch': 2.8058866426816884,
 'subregion_measure_ch': 14.152580785545538,
 'max_distance': 7.00140004632628}
```

### 2.1.11 Adding a property to each localization in LocData.data

In case you have processed your data and come up with a new property for each localization in the LocData object, this property can be added to data. As an example, we compute the nearest neighbor distance for each localization and add `nn_distance` as new property.

```
locdata.data
```

|   | position_x | position_y | frame |
|---|------------|------------|-------|
| 0 | 0          | 0.138824   | 0     |
| 1 | 1          | 0.606688   | 1     |
| 2 | 2          | 0.559792   | 2     |
| 3 | 3          | 0.130943   | 3     |
| 4 | 4          | 0.775142   | 4     |
| 5 | 5          | 0.180445   | 5     |
| 6 | 6          | 0.316270   | 6     |
| 7 | 7          | 0.278833   | 7     |

```
nn = lc.NearestNeighborDistances().compute(locdata)
nn.results
```

|   | nn_distance | nn_index |
|---|-------------|----------|
| 0 | 1.104037    | 1        |
| 1 | 1.001099    | 2        |
| 2 | 1.001099    | 1        |
| 3 | 1.088077    | 2        |
| 4 | 1.163471    | 5        |
| 5 | 1.009182    | 6        |
| 6 | 1.000701    | 7        |
| 7 | 1.000701    | 6        |

To add `nn_distance` as new property to each localization in `LocData` object, use the `pandas.assign` function on the `locdata.dataframe`.

```
locdata.dataframe = locdata.dataframe.assign(nn_distance=nn.results['nn_
↪distance'])
locdata.data
```

|   | position_x | position_y | frame | nn_distance |
|---|------------|------------|-------|-------------|
| 0 | 0          | 0.138824   | 0     | 1.104037    |
| 1 | 1          | 0.606688   | 1     | 1.001099    |
| 2 | 2          | 0.559792   | 2     | 1.001099    |
| 3 | 3          | 0.130943   | 3     | 1.088077    |
| 4 | 4          | 0.775142   | 4     | 1.163471    |
| 5 | 5          | 0.180445   | 5     | 1.009182    |
| 6 | 6          | 0.316270   | 6     | 1.000701    |
| 7 | 7          | 0.278833   | 7     | 1.000701    |

## Adding nn\_distance as new property to each localization in LocData object with dataframe=None

In case the LocData object was created with LocData.from\_selection() the LocData.dataframe attribute is None and LocData.data is generated from the referenced locdata and the index list.

In this case LocData.dataframe can still be filled with additional data that is merged upon returning LocData.data.

```
locdata_selection = lc.LocData.from_selection(locdata, indices=[1, 3, 4, 5])
locdata_selection.data
```

|   | position_x | position_y | frame | nn_distance |
|---|------------|------------|-------|-------------|
| 1 | 1          | 0.606688   | 1     | 1.001099    |
| 3 | 3          | 0.130943   | 3     | 1.088077    |
| 4 | 4          | 0.775142   | 4     | 1.163471    |
| 5 | 5          | 0.180445   | 5     | 1.009182    |

```
locdata_selection.dataframe
```

```
Empty DataFrame
Columns: []
Index: []
```

```
nn_selection = lc.NearestNeighborDistances().compute(locdata_selection)
nn_selection.results
```

|   | nn_distance | nn_index |
|---|-------------|----------|
| 0 | 2.055805    | 1        |
| 1 | 1.189535    | 2        |
| 2 | 1.163471    | 3        |
| 3 | 1.163471    | 2        |

Make sure the indices in nn.results match those in dat\_selection.data:

```
locdata_selection.data.index
```

```
Index([1, 3, 4, 5], dtype='int64')
```

```
nn_selection.results.index = locdata_selection.data.index
nn_selection.results
```

|   | nn_distance | nn_index |
|---|-------------|----------|
| 1 | 2.055805    | 1        |
| 3 | 1.189535    | 2        |
| 4 | 1.163471    | 3        |
| 5 | 1.163471    | 2        |

Then assign the corresponding result to dataframe:

```
locdata_selection.dataframe = locdata_selection.dataframe.assign(nn_distance=
↳nn_selection.results['nn_distance'])
locdata_selection.dataframe
```

```
nn_distance
1      2.055805
3      1.189535
4      1.163471
5      1.163471
```

Calling data will return the complete dataset.

```
locdata_selection.data
```

|   | position_x | position_y | frame | nn_distance_x | nn_distance_y |
|---|------------|------------|-------|---------------|---------------|
| 1 | 1          | 0.606688   | 1     | 1.001099      | 2.055805      |
| 3 | 3          | 0.130943   | 3     | 1.088077      | 1.189535      |
| 4 | 4          | 0.775142   | 4     | 1.163471      | 1.163471      |
| 5 | 5          | 0.180445   | 5     | 1.009182      | 1.163471      |

## 2.2 Tutorial about loading localization data from file

```
from pathlib import Path

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

Localization data is typically provided as text or binary file with different formats depending on the fitting software. Locan provides functions for loading various localization files.

All available functions can be looked up in the [API documentation](#).

In locan there are functions available to deal with file types according to the constant enum `FileType`:

```
list(lc.FileType._member_names_)
```

```
['UNKNOWN_FILE_TYPE',
 'CUSTOM',
 'RAPIDSTORM',
 'ELYRA',
 'THUNDERSTORM',
```

(continues on next page)

(continued from previous page)

```
'ASDF',
'NANOIMAGER',
'RAPIDSTORMTRACK',
'SMLM',
'DECODE',
'SMAP']
```

Currently the following io functions are available:

```
[name for name in dir(lc.locan_io) if not name.startswith("__")]
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

```
['Files',
'annotations',
'convert_property_names',
'convert_property_types',
'files',
'find_file_upstream',
'load_Elyra_file',
'load_Elyra_header',
'load_Nanoimager_file',
'load_Nanoimager_header',
'load_SMAP_file',
'load_SMAP_header',
'load_SMLM_file',
'load_SMLM_header',
'load_SMLM_manifest',
'load_asdf_file',
'load_decode_file',
'load_decode_header',
'load_locdata',
'load_rapidSTORM_file',
'load_rapidSTORM_header',
'load_rapidSTORM_track_file',
'load_rapidSTORM_track_header',
'load_thunderstorm_file',
'load_thunderstorm_header',
'load_txt_file',
'locdata',
'manifest_file_info_from_locdata',
'manifest_format_from_locdata',
'manifest_from_locdata',
'save_SMAP_csv',
'save_SMLM',
'save_asdf',
'save_thunderstorm_csv',
```

(continues on next page)

(continued from previous page)

```
'utilities']
```

Throughout this manual it might be helpful to use pathlib to provide path information. In all cases a string path is also usable.

## 2.2.1 Load rapidSTORM data file

Here we identify some data in the test\_data directory and provide a path using pathlib (a pathlib object is returned by `lc.ROOT_DIR`):

```
path = lc.ROOT_DIR / 'tests/test_data/rapidSTORM_dstorm_data.txt'
print(path, '\n')
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.txt
```

The data is then loaded from a rapidSTORM localization file. The file header is read to provide correct property names. The number of localisations to be read can be limited by *nrows*

```
dat = lc.load_rapidSTORM_file(path=path, nrows=10)
```

Print information about the data:

```
print('Data head:')
print(dat.data.head(), '\n')
print('Summary:')
dat.print_summary()
print('Properties:')
print(dat.properties)
```

```
Data head:
   position_x  position_y  frame  intensity  chi_square  local_background
0    9657.40    24533.5      0    33290.10    1192250.0      767.732971
1    16754.90    18770.0      0    21275.40    2106810.0      875.460999
2    14457.60    18582.6      0    20748.70     526031.0      703.369995
3     6820.58    16662.8      0     8531.77    3179190.0      852.789001
4    19183.20    22907.2      0    14139.60     448631.0      662.770020

Summary:
identifier: "1"
comment: ""
source: EXPERIMENT
state: RAW
element_count: 10
frame_count: 1
file {
  type: RAPIDSTORM
  path: "/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/
python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.
```

(continues on next page)

(continued from previous page)

```

↪txt"
}
creation_time {
    2024-03-14T11:08:51.023763Z
}

Properties:
{'localization_count': 10, 'position_x': 16878.898, 'uncertainty_x': 2252.
↪823069869743, 'position_y': 18209.502, 'uncertainty_y': 1569.5233876858572,
↪'intensity': 147591.47, 'local_background': 707.37335, 'frame': 0, 'region_
↪measure_bb': 378484578.7175999, 'localization_density_bb': 2.
↪6421155741358055e-08, 'subregion_measure_bb': 80399.79999999999}

```

Column names are exchanged with standard locan property names according to the following mapping. If no mapping is defined a warning is issued and the original column name is kept.

```
lc.RAPIDSTORM_KEYS
```

```

{'Position-0-0': 'position_x',
 'Position-1-0': 'position_y',
 'Position-2-0': 'position_z',
 'ImageNumber-0-0': 'frame',
 'Amplitude-0-0': 'intensity',
 'FitResidues-0-0': 'chi_square',
 'LocalBackground-0-0': 'local_background',
 'TwoKernelImprovement-0-0': 'two_kernel_improvement',
 'Position-0-0-uncertainty': 'uncertainty_x',
 'Position-1-0-uncertainty': 'uncertainty_y',
 'Position-2-0-uncertainty': 'uncertainty_z'}

```

## 2.2.2 Load Zeiss Elyra data file

The Elyra super-resolution microscopy system from Zeiss uses as slightly different file format. Elyra column names are exchanged with locan property names upon loading the data.

```

path_Elyra = lc.ROOT_DIR / 'tests/test_data/Elyra_dstorm_data.txt'
print(path_Elyra, '\n')

```

```

/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↪python3.11/site-packages/locan/tests/test_data/Elyra_dstorm_data.txt

```

```
dat_Elyra = lc.load_Elyra_file(path=path_Elyra, nrows=10)
```

```

print('Data head:')
print(dat_Elyra.data.head(), '\n')
print('Summary:')
dat_Elyra.print_summary()

```

(continues on next page)



(continued from previous page)

```
print('Properties:')
print(dat_Elyra.properties)
```

Data head:

|   | original_index | frame | frames_number | frames_missing | position_x | \ |
|---|----------------|-------|---------------|----------------|------------|---|
| 0 | 1              | 1     | 1             | 0              | 15850.6    |   |
| 1 | 2              | 1     | 1             | 0              | 25617.3    |   |
| 2 | 3              | 1     | 1             | 0              | 20155.8    |   |
| 3 | 4              | 1     | 1             | 0              | 10776.9    |   |
| 4 | 5              | 1     | 1             | 0              | 28966.9    |   |

|   | position_y | uncertainty | intensity | local_background_sigma | chi_square | \ |
|---|------------|-------------|-----------|------------------------|------------|---|
| 0 | 23502.1    | 8.6         | 472.0     | 5.33                   | 0.28       |   |
| 1 | 24310.2    | 9.5         | 529.0     | 4.38                   | 0.31       |   |
| 2 | 24039.1    | 13.0        | 306.0     | 3.06                   | 0.23       |   |
| 3 | 10047.4    | 13.4        | 369.0     | 3.98                   | 0.25       |   |
| 4 | 8731.6     | 18.1        | 428.0     | 14.73                  | 0.41       |   |

|   | psf_half_width | channel | slice_z |
|---|----------------|---------|---------|
| 0 | 110.000000     | 1       | 1.0     |
| 1 | 129.800003     | 1       | 1.0     |
| 2 | 131.100006     | 1       | 1.0     |
| 3 | 143.000000     | 1       | 1.0     |
| 4 | 150.100006     | 1       | 1.0     |

Summary:

identifier: "2"

comment: ""

source: EXPERIMENT

state: RAW

element\_count: 10

frame\_count: 1

file {

type: ELYRA

path: "/home/docs/checkouts/readthedocs.org/user\_builds/locan/envs/latest/  
→lib/python3.11/site-packages/locan/tests/test\_data/Elyra\_dstorm\_data.txt"

}

creation\_time {

2024-03-14T11:08:51.060046Z

}

Properties:

```
{'localization_count': 10, 'position_x': 19610.811087722697, 'uncertainty_x': 2109.405021031108, 'position_y': 18319.131814537763, 'uncertainty_y': 2608.8543142720196, 'intensity': 3145.0, 'frame': 1, 'region_measure_bb': 351167887.24, 'localization_density_bb': 2.847640790447807e-08, 'subregion_measure_bb': 75072.8}
```

### 2.2.3 Localization data from a custom text file

Other custom text files can be read with a function that wraps the `pandas.read_table()` method.

```
path_csv = lc.ROOT_DIR / 'tests/test_data/five_blobs.txt'
print(path_csv, '\n')
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
→python3.11/site-packages/locan/tests/test_data/five_blobs.txt
```

Here data is loaded from a comma-separated-value file. Column names are read from the first line and a warning is given if the naming does not comply with locan conventions. Column names can also be provided as *column*. The separator, e.g. a tab 't' can be provided as *sep*.

```
dat_csv = lc.load_txt_file(path=path_csv, sep=',', columns=None, nrows=10)
```

```
print('Data head:')
print(dat_csv.data.head(), '\n')
print('Summary:')
dat_csv.print_summary()
print('Properties:')
print(dat_csv.properties)
```

```
Data head:
   index  position_x  position_y  cluster_label  frame
0      0      624.0      919.0             3      0
1      1      611.0      873.0             3      0
2      2      388.0     1015.0             0      0
3      3      209.0      465.0             2      0
4      4     1001.0      851.0             4      0

Summary:
identifier: "3"
comment: ""
source: EXPERIMENT
state: RAW
element_count: 10
frame_count: 1
file {
  type: CUSTOM
  path: "/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/
→lib/python3.11/site-packages/locan/tests/test_data/five_blobs.txt"
}
creation_time {
  2024-03-14T11:08:51.086592Z
}

Properties:
{'localization_count': 10, 'position_x': 517.5, 'uncertainty_x': 87.
→40648971583543, 'position_y': 815.3, 'uncertainty_y': 65.24586832385123,
```

(continues on next page)

(continued from previous page)

```
↪ 'frame': 0, 'region_measure_bb': 488950.0, 'localization_density_bb': 2.
↪ 0451988955925965e-05, 'subregion_measure_bb': 2878.0}
```

## 2.2.4 Load localization data file

A general function for loading localization data is provided. Targeting specific localization file formats is done through the `file_format` parameter.

```
path = lc.ROOT_DIR / 'tests/test_data/rapidSTORM_dstorm_data.txt'
print(path, '\n')
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↪ python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.txt
```

```
dat = lc.load_locdata(path=path, file_type=lc.FileType.RAPIDSTORM, nrows=10)
```

```
dat.print_summary()
```

```
identifier: "4"
comment: ""
source: EXPERIMENT
state: RAW
element_count: 10
frame_count: 1
file {
  type: RAPIDSTORM
  path: "/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/
↪ lib/python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.
↪ txt"
}
creation_time {
  2024-03-14T11:08:51.111785Z
}
```

The file type can be specified by using the enum class `FileType` and use tab control to make a choice.

```
list(lc.FileType._member_names_)
```

```
['UNKNOWN_FILE_TYPE',
 'CUSTOM',
 'RAPIDSTORM',
 'ELYRA',
 'THUNDERSTORM',
 'ASDF',
 'NANOIMAGER',
 'RAPIDSTORMTRACK',
 'SMLM',
```

(continues on next page)

(continued from previous page)

```
'DECODE',
'SMAP']
```

```
lc.FileType.RAPIDSTORM
```

```
<FileType.RAPIDSTORM: 2>
```

## 2.2.5 Adjust data types

The data types of localization properties are adjusted in all load functions by default to the following standard types:

```
lc.PROPERTY_KEYS
```

```
{'index': 'integer',
 'original_index': 'integer',
 'position_x': 'float',
 'position_y': 'float',
 'position_z': 'float',
 'frame': 'integer',
 'frames_number': 'integer',
 'frames_missing': 'integer',
 'time': 'float',
 'intensity': 'float',
 'local_background': 'float',
 'local_background_sigma': 'float',
 'signal_noise_ratio': 'float',
 'signal_background_ratio': 'float',
 'chi_square': 'float',
 'two_kernel_improvement': 'float',
 'psf_amplitude': 'float',
 'psf_width': 'float',
 'psf_width_x': 'float',
 'psf_width_y': 'float',
 'psf_width_z': 'float',
 'psf_half_width': 'float',
 'psf_half_width_x': 'float',
 'psf_half_width_y': 'float',
 'psf_half_width_z': 'float',
 'psf_sigma': 'float',
 'psf_sigma_x': 'float',
 'psf_sigma_y': 'float',
 'psf_sigma_z': 'float',
 'uncertainty': 'float',
 'uncertainty_x': 'float',
 'uncertainty_y': 'float',
 'uncertainty_z': 'float',
 'channel': 'integer',
```

(continues on next page)

(continued from previous page)

```
'slice_z': 'float',
'plane': 'integer',
'cluster_label': 'integer'}
```

If this is not what you want, add `convert = False`.

```
path = lc.ROOT_DIR / 'tests/test_data/rapidSTORM_dstorm_data.txt'
print(path, '\n')
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.txt
```

```
locdata = lc.load_locdata(path=path, file_type=lc.FileType.RAPIDSTORM,
↳nrows=10, convert=False)
locdata.data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
#   Column                Non-Null Count  Dtype
---  -
0   position_x            10 non-null    float64
1   position_y            10 non-null    float64
2   frame                 10 non-null    int64
3   intensity              10 non-null    float64
4   chi_square            10 non-null    float64
5   local_background      10 non-null    float64
dtypes: float64(5), int64(1)
memory usage: 612.0 bytes
```

Maybe adjust types for selected localization properties.

```
other_types = {"frame": float}
df = lc.convert_property_types(locdata.data, types=other_types)
locdata.update(dataframe=df)
locdata.data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 6 columns):
#   Column                Non-Null Count  Dtype
---  -
0   position_x            10 non-null    float64
1   position_y            10 non-null    float64
2   frame                 10 non-null    float64
3   intensity              10 non-null    float64
4   chi_square            10 non-null    float64
5   local_background      10 non-null    float64
dtypes: float64(6)
```

(continues on next page)

(continued from previous page)

memory usage: 612.0 bytes

## 2.3 Tutorial about computing hulls for localization data

For each set of localizations with 2D or 3D spatial coordinates various hull can be computed. A hull can be the minimal bounding box, the oriented minimal bounding box, the convex hull, or an alpha shape.

You can trigger computation of specific hull objects using a specific hull class or from the corresponding LocData attribute.

```
from pathlib import Path

%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import locan as lc
from locan.data.hulls import BoundingBox, ConvexHull, OrientedBoundingBox
```

```
/tmp/ipykernel_1512/1272412987.py:6: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳ 54466

import pandas as pd
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.3.1 Synthetic data

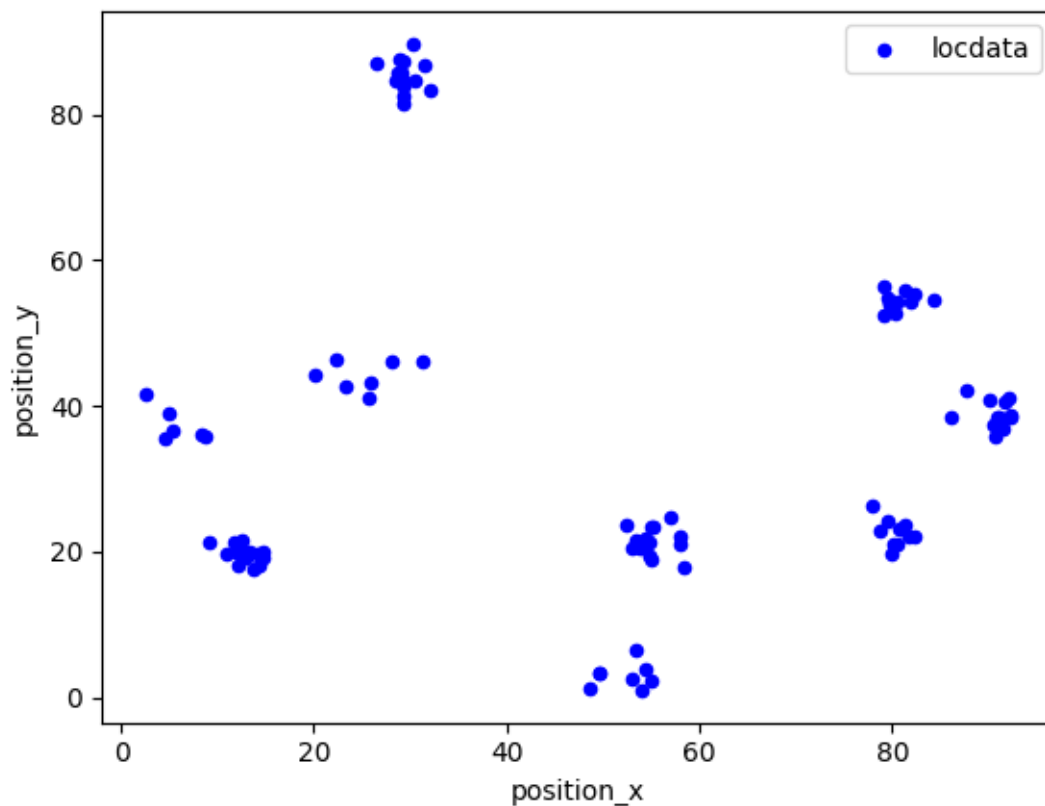
```
rng = np.random.default_rng(seed=1)
```

```
locdata = lc.simulate_Thomas(parent_intensity=1e-3, region=((0, 100), (0, 100)), cluster_mu=10, cluster_std=2, seed=rng)
```

```
locdata.print_summary()
```

```
identifier: "1"
comment: ""
source: SIMULATION
state: RAW
element_count: 100
frame_count: 0
creation_time {
  2024-03-14T11:09:00.343556Z
}
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    label='locdata')
plt.show()
```



### 2.3.2 Minimal bounding box for spatial coordinates

```
# H = BoundingBox(locdata.coordinates)
```

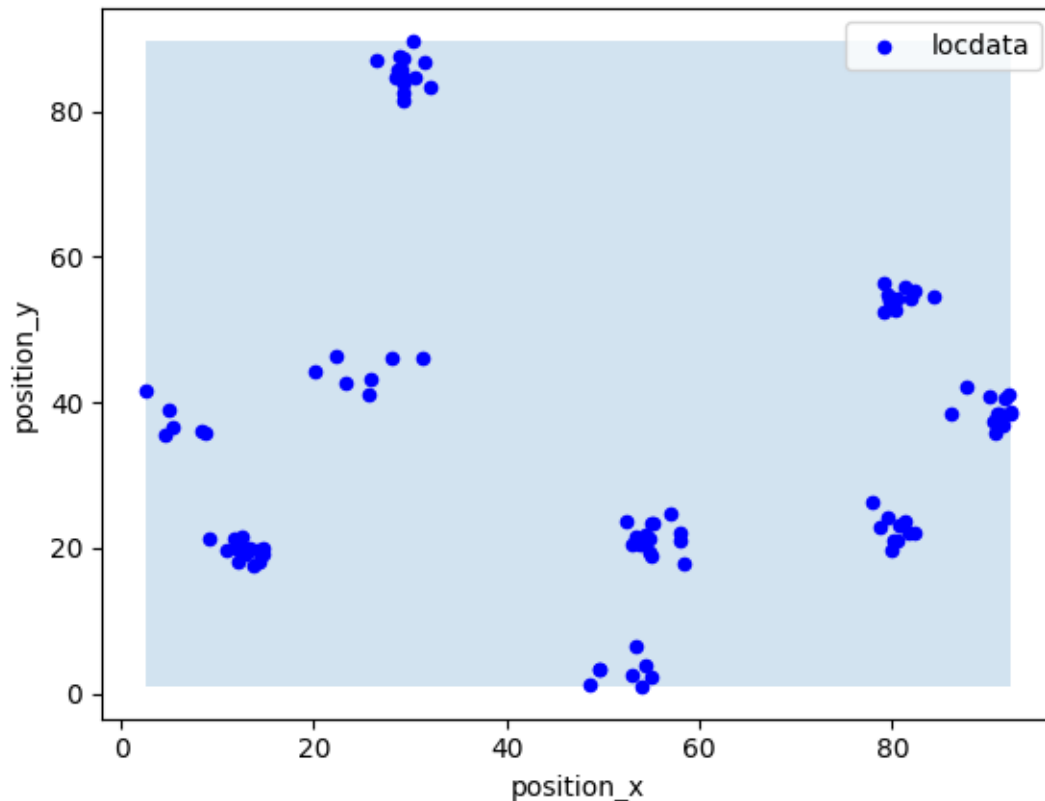
```
H = locdata.bounding_box
```

```
print('dimension: ', H.dimension)
print('hull: ', H.hull)
print('width: ', H.width)
print('vertices: ', H.vertices)
print('region_measure: ', H.region_measure)
print('subregion_measure: ', H.subregion_measure)
print('region: ', H.region)
```

```
dimension: 2
hull: [[ 2.490921    1.02997497]
 [92.36001123  89.71010693]]
width: [89.86909023  88.68013196]
vertices: [[ 2.490921    92.36001123]
 [ 1.02997497  89.71010693]]
region_measure: 7969.602780428899
subregion_measure: 357.0984443733635
region: Rectangle((2.490921001041831, 1.0299749716244107), 89.86909022532446,
↪ 88.6801319613573, 0)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.add_patch(locdata.bounding_box.region.as_artist(alpha=0.2))
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
↪ label='locdata')
plt.show()
```





### 2.3.3 Oriented minimal bounding box for spatial coordinates

```
# H = OrientedBoundingBox(locdata.coordinates)

H = locdata.oriented_bounding_box

print('dimension: ', H.dimension)
print('hull: ', H.hull)
print('vertices: ', H.vertices)
print('width: ', H.width)
print('region_measure: ', H.region_measure)
print('subregion_measure: ', H.subregion_measure)
print('region: ', H.region)
```

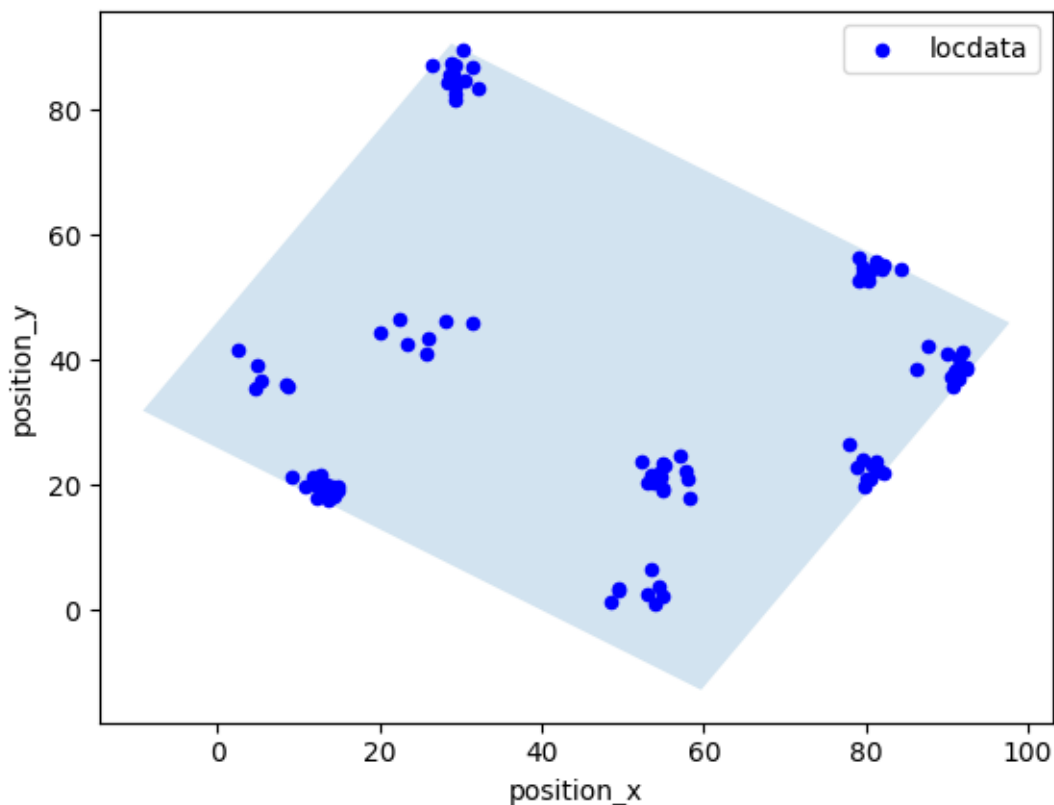
```
dimension: 2
hull: POLYGON ((-9.220392369565374 31.865082145844063, 59.69001056824682 -12.
↪797641894321849, 97.75586149761332 45.934403275030405, 28.845458559801138,
↪90.59712731519632, -9.220392369565374 31.865082145844063))
vertices: [[ -9.22039237 31.86508215]
[ 59.69001057 -12.79764189]
[ 97.7558615 45.93440328]
[ 28.84545856 90.59712732]
[ -9.22039237 31.86508215]]
width: [82.11822302 69.9890144 ]
region_measure: 5747.373493394239
```

(continues on next page)

(continued from previous page)

```
subregion_measure: 304.21447483826086
region: Rectangle((-9.220392369565374, 31.865082145844063), 82.
↪ 11822301864336, 69.98901440048706, -32.948380191618504)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.add_patch(locdata.oriented_bounding_box.region.as_artist(alpha=0.2))
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
↪ label='locdata')
plt.show()
```



### 2.3.4 Convex hull

#### Convex hull for spatial coordinates (scipy)

```
# H = ConvexHull(locdata.coordinates, method='scipy')

H = locdata.convex_hull

print('dimension: ', H.dimension)
print('hull: ', H.hull)
# print('vertex_indices: ', H.vertex_indices)
# print('vertices: ', H.vertices)
print('region_measure: ', H.region_measure)
print('subregion_measure: ', H.subregion_measure)
```

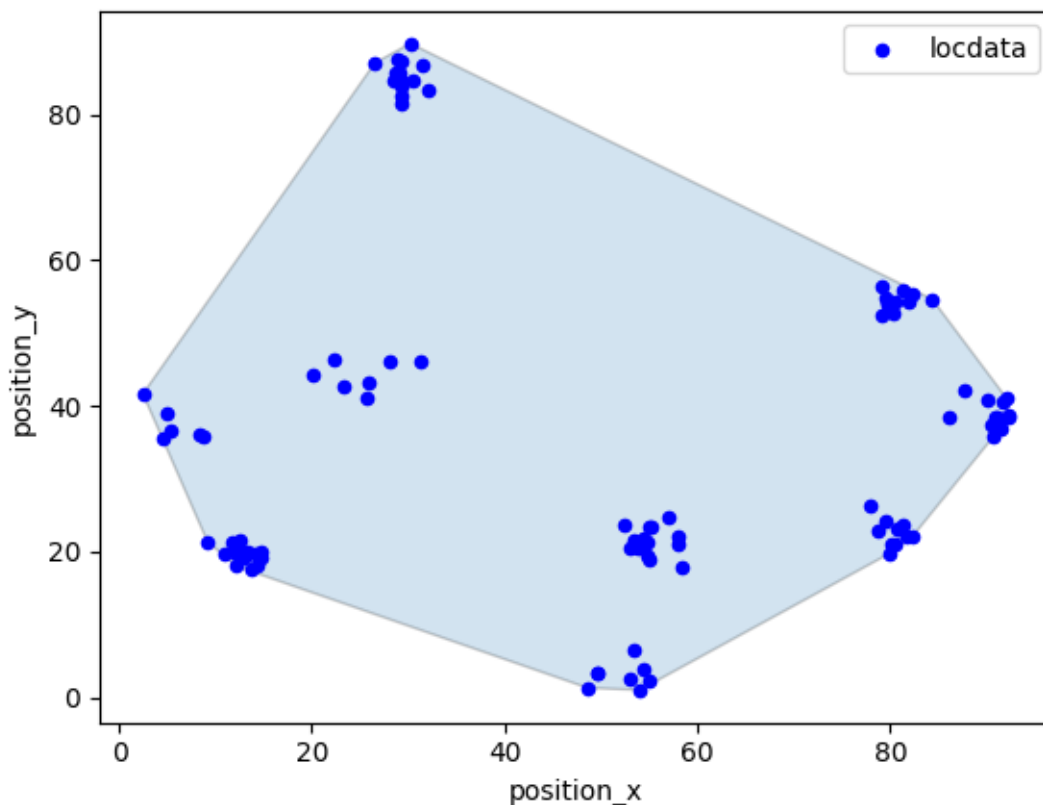
(continues on next page)

(continued from previous page)

```
print('points on boundary: ', H.points_on_boundary)
print('points on boundary relative to all points: ', H.points_on_boundary_rel)
print('region: ', H.region)
```

```
dimension: 2
hull: <scipy.spatial._qhull.ConvexHull object at 0x7fa3a7266ad0>
region_measure: 4908.170783871255
subregion_measure: 264.4870519254037
points on boundary: 13
points on boundary relative to all points: 0.13
region: Polygon(<self.points>, <self.holes>)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.add_patch(locdata.convex_hull.region.as_artist(alpha=0.2))
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↪ label='locdata')
plt.show()
```



## Convex hull for spatial coordinates (shapely)

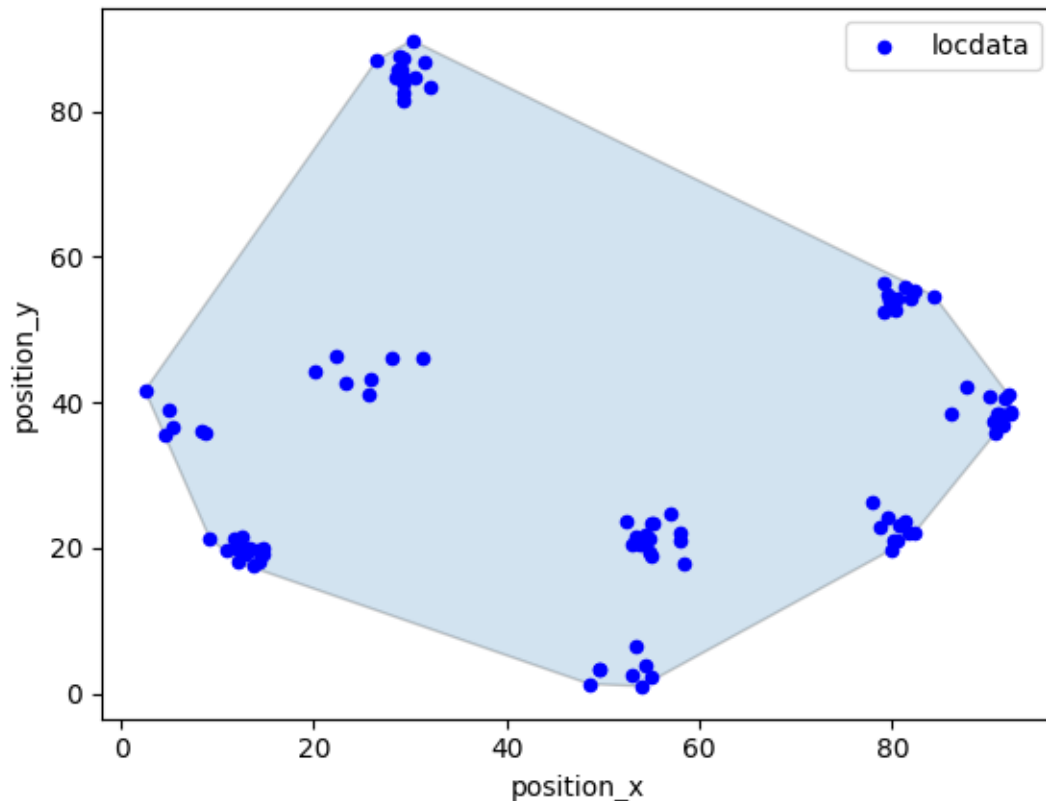
Some hulls can be computed from different algorithms. If implemented, use the `methods` parameter to specify the algorithm.

```
H = ConvexHull(locdata.coordinates, method='shapely')

print('dimension: ', H.dimension)
print('hull: ', H.hull)
# print('vertices: ', H.vertices)
print('region_measure: ', H.region_measure)
print('subregion_measure: ', H.subregion_measure)
print('points on boundary: ', H.points_on_boundary)
print('points on boundary relative to all points: ', H.points_on_boundary_rel)
print('region: ', H.region)
```

```
dimension: 2
hull: POLYGON ((54.029321929694525 1.0299749716244107, 48.634206414428576 1.
↪ 294631346507537, 12.175420163612081 17.997868633149913, 9.18991822286495 21.
↪ 36788078938049, 2.490921001041831 41.64287079543503, 26.54041545299338 87.
↪ 040661544405, 30.214047987922243 89.71010693298172, 84.34954910628653 54.
↪ 62340239857981, 91.9734281153137 41.17838012011845, 92.33940616216016 38.
↪ 808549274650204, 92.36001122636628 38.48950874170735, 82.28290378326082 22.
↪ 061077730066597, 79.83990217703906 19.69136737327613, 54.029321929694525 1.
↪ 0299749716244107))
region_measure: 4908.170783871255
subregion_measure: 264.4870519254038
points on boundary: 13
points on boundary relative to all points: 0.13
region: Polygon(<self.points>, <self.holes>)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.add_patch(H.region.as_artist(alpha=0.2))
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
↪ label='locdata')
plt.show()
```



### 2.3.5 Alpha shape for spatial coordinates

The alpha shape depends on a single parameter alpha (not to confuse with the alpha to specify opacity in figures). The alpha complex is an alpha-independent representation of all alpha shapes.

You can get all apha values for which the corresponding alpha shape changes.

```
lc.AlphaComplex(locdata.coordinates).alphas()
```

```
array([4.79010715e-02, 6.26886019e-02, 9.00848038e-02, 1.26327560e-01,
       1.27719280e-01, 1.35082505e-01, 1.48065929e-01, 1.59852612e-01,
       1.80345970e-01, 2.29967352e-01, 2.38528055e-01, 2.40927392e-01,
       2.41691338e-01, 2.66695006e-01, 2.77822554e-01, 2.94376165e-01,
       3.06743479e-01, 3.25282310e-01, 3.38425410e-01, 3.43041012e-01,
       3.43184730e-01, 3.52204265e-01, 3.57124009e-01, 3.58975685e-01,
       3.62059409e-01, 3.74195019e-01, 3.78291044e-01, 3.97084183e-01,
       4.00186670e-01, 4.02716245e-01, 4.02739384e-01, 4.02815627e-01,
       4.09114331e-01, 4.18957692e-01, 4.19645900e-01, 4.33534796e-01,
       4.47358936e-01, 4.47427686e-01, 4.50668384e-01, 4.56424309e-01,
       4.67180830e-01, 4.71727787e-01, 4.75216918e-01, 4.81397069e-01,
       4.81997429e-01, 4.82183979e-01, 4.97024836e-01, 4.99990289e-01,
       5.06241764e-01, 5.08056086e-01, 5.13489111e-01, 5.25320369e-01,
       5.31023418e-01, 5.34700206e-01, 5.39200680e-01, 5.49237405e-01,
       5.53496939e-01, 5.58987615e-01, 5.59448067e-01, 5.59500547e-01,
       5.61663190e-01, 5.67027377e-01, 5.67749948e-01, 5.71279300e-01,
       5.79365256e-01, 5.81500820e-01, 5.92288917e-01, 5.97516286e-01,
```

(continues on next page)

(continued from previous page)

```

6.00674458e-01, 6.01810488e-01, 6.03314803e-01, 6.06574504e-01,
6.06739584e-01, 6.09413711e-01, 6.10483544e-01, 6.12523184e-01,
6.18571992e-01, 6.19312766e-01, 6.20713067e-01, 6.24167482e-01,
6.40082879e-01, 6.43777056e-01, 6.51568398e-01, 6.62458725e-01,
6.64871359e-01, 6.82998920e-01, 6.89013581e-01, 6.89811876e-01,
6.95662528e-01, 6.97159840e-01, 6.97171260e-01, 7.00419524e-01,
7.00672239e-01, 7.09851240e-01, 7.11029883e-01, 7.12728093e-01,
7.15296258e-01, 7.19308431e-01, 7.21811422e-01, 7.28223700e-01,
7.34206960e-01, 7.37019513e-01, 7.38740770e-01, 7.39835123e-01,
7.40429317e-01, 7.41294381e-01, 7.46229214e-01, 7.49077113e-01,
7.50780227e-01, 7.51443768e-01, 7.58053002e-01, 7.59431180e-01,
7.60230149e-01, 7.61763351e-01, 7.62155416e-01, 7.63885515e-01,
7.69292534e-01, 7.70264128e-01, 7.71759040e-01, 7.73355937e-01,
7.84442099e-01, 7.90069055e-01, 7.91274054e-01, 7.93463861e-01,
7.94356991e-01, 7.99868788e-01, 8.03569058e-01, 8.11065191e-01,
8.22856563e-01, 8.25466835e-01, 8.26052181e-01, 8.33933862e-01,
8.39957725e-01, 8.44739640e-01, 8.45350979e-01, 8.46103046e-01,
8.49224572e-01, 8.49880234e-01, 8.51344679e-01, 8.52808232e-01,
8.53931434e-01, 8.56196465e-01, 8.61652127e-01, 8.69376152e-01,
8.70334553e-01, 8.79922419e-01, 8.83510627e-01, 8.87756937e-01,
8.87768808e-01, 8.89103801e-01, 8.90296830e-01, 9.03293099e-01,
9.03545920e-01, 9.04654590e-01, 9.13895586e-01, 9.14988676e-01,
9.19769930e-01, 9.21694582e-01, 9.25606663e-01, 9.42225183e-01,
9.53726892e-01, 9.60716235e-01, 9.66079966e-01, 9.79994236e-01,
9.82754216e-01, 9.85767096e-01, 9.88304664e-01, 9.93925311e-01,
9.98617093e-01, 1.01493196e+00, 1.01705667e+00, 1.01784033e+00,
1.02895805e+00, 1.03002195e+00, 1.03079742e+00, 1.03230173e+00,
1.03437394e+00, 1.03953428e+00, 1.04708194e+00, 1.04779857e+00,
1.05264932e+00, 1.06179970e+00, 1.06613206e+00, 1.06634542e+00,
1.08018457e+00, 1.08501873e+00, 1.09518923e+00, 1.09869778e+00,
1.10539327e+00, 1.11120765e+00, 1.11301014e+00, 1.11444086e+00,
1.11952155e+00, 1.13356306e+00, 1.13477392e+00, 1.13733855e+00,
1.15128479e+00, 1.15750823e+00, 1.16736695e+00, 1.16926401e+00,
1.17387209e+00, 1.18535636e+00, 1.18551348e+00, 1.19292617e+00,
1.19896186e+00, 1.20048927e+00, 1.20720985e+00, 1.23595162e+00,
1.23873969e+00, 1.25573200e+00, 1.25617021e+00, 1.26070725e+00,
1.26125038e+00, 1.26240010e+00, 1.26532192e+00, 1.27488250e+00,
1.27724424e+00, 1.28234004e+00, 1.28497224e+00, 1.29840781e+00,
1.30029954e+00, 1.31068196e+00, 1.31351678e+00, 1.31691837e+00,
1.31734890e+00, 1.33458446e+00, 1.33515579e+00, 1.34370910e+00,
1.34568550e+00, 1.34663685e+00, 1.34907880e+00, 1.35972579e+00,
1.36367900e+00, 1.36709957e+00, 1.36739479e+00, 1.41309146e+00,
1.42226765e+00, 1.42571135e+00, 1.43547786e+00, 1.43628331e+00,
1.45069543e+00, 1.46565390e+00, 1.48152517e+00, 1.48366972e+00,
1.48804400e+00, 1.49016566e+00, 1.52334060e+00, 1.52574832e+00,
1.52870001e+00, 1.53542071e+00, 1.53864429e+00, 1.54408771e+00,
1.54595694e+00, 1.55976987e+00, 1.57562096e+00, 1.58411018e+00,
1.61295349e+00, 1.61741832e+00, 1.62776308e+00, 1.63307779e+00,
1.63940332e+00, 1.64733370e+00, 1.65626521e+00, 1.65788434e+00,

```

(continues on next page)

(continued from previous page)

```

1.66678718e+00, 1.69711226e+00, 1.70174793e+00, 1.71050339e+00,
1.71714404e+00, 1.72508327e+00, 1.73141334e+00, 1.73774244e+00,
1.74668941e+00, 1.74747826e+00, 1.75133241e+00, 1.75463569e+00,
1.77157747e+00, 1.77362769e+00, 1.77744793e+00, 1.78223117e+00,
1.78501180e+00, 1.79207963e+00, 1.79918946e+00, 1.80015981e+00,
1.82414077e+00, 1.82536675e+00, 1.87379584e+00, 1.89357272e+00,
1.91381607e+00, 1.92656279e+00, 1.94884908e+00, 2.00577717e+00,
2.01728998e+00, 2.03034431e+00, 2.04408104e+00, 2.05038818e+00,
2.05349095e+00, 2.07285481e+00, 2.16816869e+00, 2.20910083e+00,
2.22509947e+00, 2.24795084e+00, 2.25112215e+00, 2.27054590e+00,
2.27811322e+00, 2.27983143e+00, 2.31343855e+00, 2.32264410e+00,
2.32880977e+00, 2.34437210e+00, 2.37718224e+00, 2.37723128e+00,
2.38295290e+00, 2.39569231e+00, 2.41090942e+00, 2.43550447e+00,
2.45645949e+00, 2.45891276e+00, 2.52246605e+00, 2.54215315e+00,
2.62136822e+00, 2.65112497e+00, 2.65972828e+00, 2.70080147e+00,
2.79515289e+00, 2.84755996e+00, 2.85697323e+00, 2.90416046e+00,
3.01656649e+00, 3.07917748e+00, 3.13521975e+00, 3.17783655e+00,
3.26467278e+00, 3.38255925e+00, 3.56990862e+00, 3.57574974e+00,
3.59260960e+00, 3.60194779e+00, 3.63036096e+00, 3.69906825e+00,
3.95779301e+00, 4.26390245e+00, 4.29842013e+00, 4.90182079e+00,
4.97166259e+00, 6.07922626e+00, 6.24118004e+00, 6.24890464e+00,
6.35586273e+00, 6.39866293e+00, 6.48277005e+00, 6.65001925e+00,
6.94378885e+00, 7.14344811e+00, 7.21201452e+00, 7.24911348e+00,
7.26452608e+00, 7.32337306e+00, 7.41191471e+00, 7.42595448e+00,
7.43392888e+00, 7.43433203e+00, 7.72806825e+00, 7.74336647e+00,
7.76087118e+00, 7.78944873e+00, 7.80165047e+00, 7.82652969e+00,
7.85021562e+00, 7.87716234e+00, 7.88724780e+00, 7.96646223e+00,
8.04566727e+00, 8.08224562e+00, 8.09483397e+00, 8.26562220e+00,
8.31591730e+00, 8.51342985e+00, 8.71703225e+00, 8.74628027e+00,
8.79373820e+00, 8.94187403e+00, 8.95071004e+00, 9.00003635e+00,
9.01595702e+00, 9.63640804e+00, 9.86227445e+00, 1.01897890e+01,
1.02221135e+01, 1.02751450e+01, 1.03599403e+01, 1.03913311e+01,
1.04465878e+01, 1.04471172e+01, 1.04597259e+01, 1.05636147e+01,
1.05639307e+01, 1.05738881e+01, 1.06432177e+01, 1.06765138e+01,
1.08061930e+01, 1.08267337e+01, 1.10126155e+01, 1.11019300e+01,
1.17944937e+01, 1.17991373e+01, 1.19695055e+01, 1.20037920e+01,
1.21184799e+01, 1.22848303e+01, 1.30919383e+01, 1.50017504e+01,
1.51307502e+01, 1.53817915e+01, 1.58322884e+01, 1.59250873e+01,
1.59542235e+01, 1.59653821e+01, 1.66977766e+01, 1.75114153e+01,
1.76752204e+01, 1.77796252e+01, 1.77821903e+01, 1.78396689e+01,
1.78495038e+01, 1.78527366e+01, 1.80088206e+01, 1.87142131e+01,
1.89454045e+01, 1.89693286e+01, 1.89963288e+01, 1.90443274e+01,
1.91047288e+01, 1.91360626e+01, 1.91361642e+01, 1.91392739e+01,
1.91597195e+01, 1.91653265e+01, 1.91703660e+01, 1.91811000e+01,
1.92761620e+01, 1.94600972e+01, 1.96390949e+01, 2.00514415e+01,
2.02227234e+01, 2.15705094e+01, 2.39869621e+01, 2.41473984e+01,
2.41540455e+01, 2.44485075e+01, 2.46186883e+01, 2.56872419e+01,
2.63949037e+01, 2.70955193e+01, 2.78666861e+01, 2.82256845e+01,
2.95858969e+01, 2.98130814e+01, 3.06827051e+01, 3.22557333e+01,

```

(continues on next page)

(continued from previous page)

```
3.70800737e+01, 5.09825269e+01, 5.34146215e+01, 7.99051776e+01,
9.91221385e+01, 1.42276409e+02, 4.34636400e+02, 4.71113175e+02,
1.32363587e+03,          inf])
```

You can determine an optimal alpha, i.e. the smallest alpha for which all points are still part of the alpha shape.

```
opt_alpha = lc.AlphaComplex(locdata.coordinates).optimal_alpha()
opt_alpha
```

```
32.25573327495765
```

```
# H = lc.AlphaShape(opt_alpha, locdata.coordinates)

locdata.update_alpha_shape(alpha=opt_alpha)
H = locdata.alpha_shape

print('dimension: ', H.dimension)
# print('vertex_indices: ', H.vertex_indices)
# print('vertices: ', H.vertices)
print('region_measure: ', H.region_measure)
print('subregion_measure: ', H.subregion_measure)
print('points in alpha shape: ', H.n_points_alpha_shape)
print('points in alpha shape relative to all points: ', H.n_points_alpha_
    ↪ shape_rel)
print('points on boundary: ', H.n_points_on_boundary)
print('points on boundary relative to all points: ', H.n_points_on_boundary_
    ↪ rel)
print('region: ', H.region)
```

```
dimension: 2
region_measure: 4908.1707838712555
subregion_measure: 264.4870519254038
points in alpha shape: 100
points in alpha shape relative to all points: 1.0
points on boundary: 22
points on boundary relative to all points: 0.22
region: Polygon(<self.points>, <self.holes>)
```

The alpha shape is made of different vertex types that can be differentiated as *exterior*, *interior*, *regular* or *singular*.

```
ac_simplices_all = H.alpha_complex.get_alpha_complex_lines(H.alpha, type='all
    ↪')
ac_simplices_exterior = H.alpha_complex.get_alpha_complex_lines(H.alpha, type=
    ↪'exterior')
ac_simplices_interior = H.alpha_complex.get_alpha_complex_lines(H.alpha, type=
    ↪'interior')
ac_simplices_regular = H.alpha_complex.get_alpha_complex_lines(H.alpha, type=
```

(continues on next page)



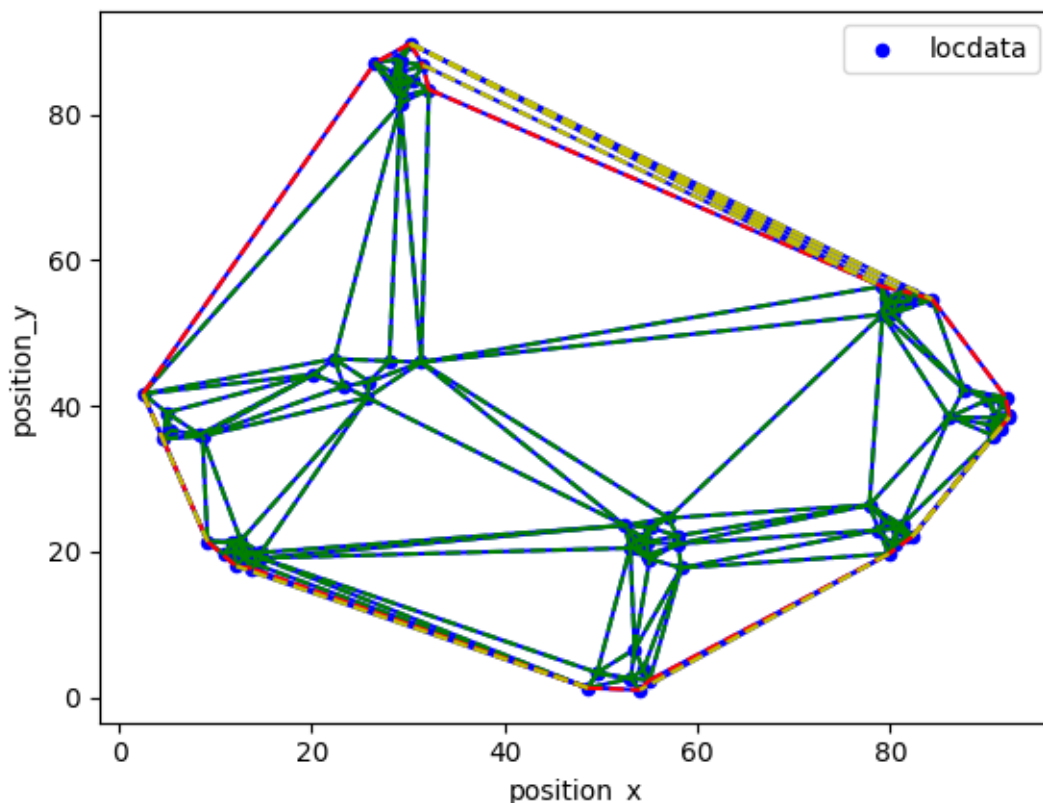
(continued from previous page)

```
↪ 'regular')
ac_simplices_singular = H.alpha_complex.get_alpha_complex_lines(H.alpha, type=
↪ 'singular')
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)

for simp in ac_simplices_all:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '-b')
for simp in ac_simplices_interior:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '--g')
for simp in ac_simplices_regular:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '--r')
for simp in ac_simplices_singular:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '--y')

locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
↪ label='locdata')
plt.show()
```



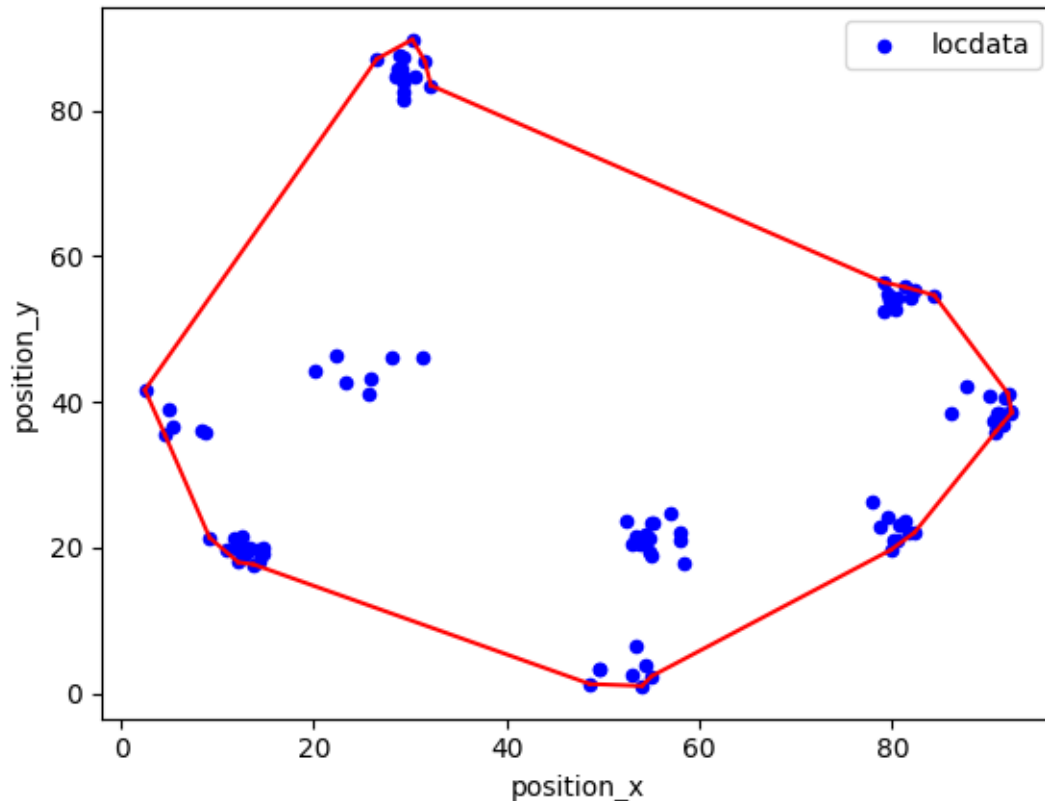
Often the *regular* representation is good enough.

```
fig, ax = plt.subplots(nrows=1, ncols=1)
for simp in ac_simplices_regular:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '-r')
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
↪ label='locdata')
```

(continues on next page)

(continued from previous page)

```
plt.show()
```



You can get the connected components as list of Region.

```
H.connected_components
```

```
[Polygon([[26.54041545299338, 87.040661544405], [30.214047987922243, 89.
↪ 71010693298172], [84.34954910628653, 54.62340239857981], [91.9734281153137,
↪ 41.17838012011845], [92.33940616216016, 38.808549274650204], [92.
↪ 36001122636628, 38.48950874170735], [82.28290378326082, 22.061077730066597],
↪ [79.83990217703906, 19.69136737327613], [54.029321929694525, 1.
↪ 0299749716244107], [48.634206414428576, 1.294631346507537], [12.
↪ 175420163612081, 17.997868633149913], [9.18991822286495, 21.36788078938049],
↪ [2.490921001041831, 41.64287079543503], [26.54041545299338, 87.
↪ 040661544405]], [])]
```

```
connected_component_0 = H.connected_components[0]
```

```
print('dimension: ', connected_component_0.dimension)
print('region_measure: ', connected_component_0.region_measure)
print('subregion_measure: ', connected_component_0.subregion_measure)
```

```
dimension: 2
region_measure: 4908.1707838712555
subregion_measure: 264.4870519254038
```

The alpha shape for a smaller alpha can have multiple connected components.

```
H = lc.AlphaShape(5, locdata.coordinates)

print('dimension: ', H.dimension)
# print('vertex_indices: ', H.vertex_indices)
# print('vertices: ', H.vertices)
print('region_measure: ', H.region_measure)
print('subregion_measure: ', H.subregion_measure)
print('points in alpha shape: ', H.n_points_alpha_shape)
print('points in alpha shape relative to all points: ', H.n_points_alpha_
    ↪ shape_rel)
print('points on boundary: ', H.n_points_on_boundary)
print('points on boundary relative to all points: ', H.n_points_on_boundary_
    ↪ rel)
```

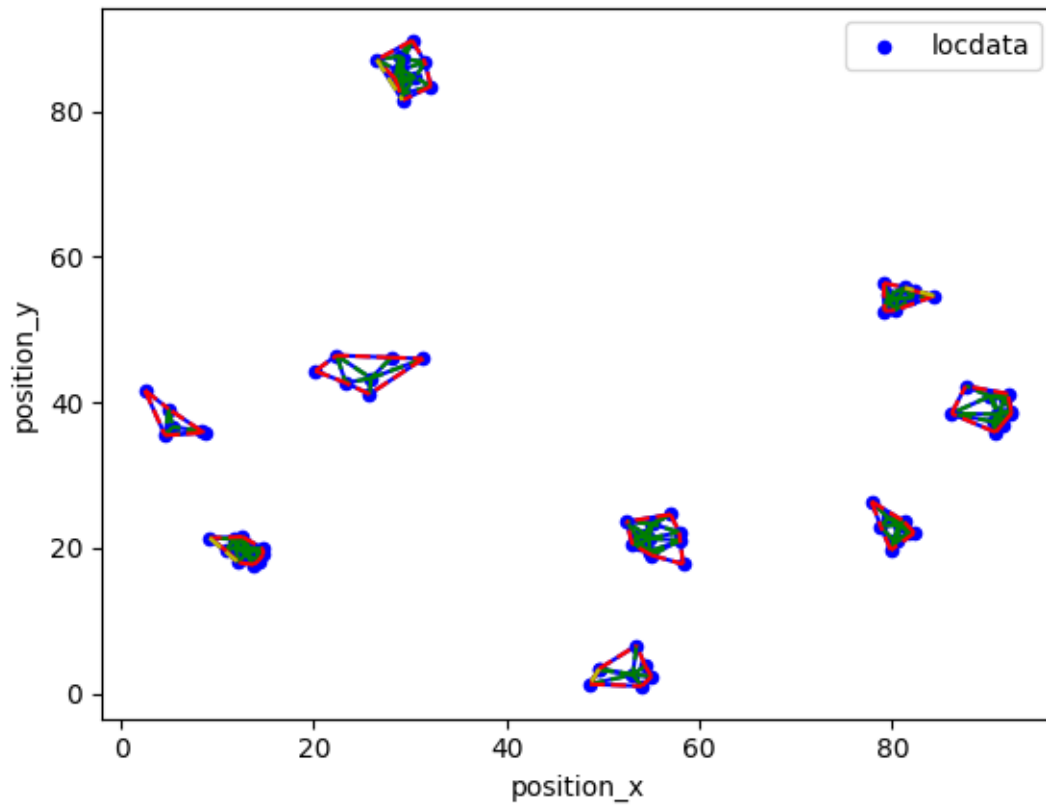
```
dimension: 2
region_measure: 178.41254317639843
subregion_measure: 171.67742405936463
points in alpha shape: 100
points in alpha shape relative to all points: 1.0
points on boundary: 57
points on boundary relative to all points: 0.57
```

```
ac_simplices_all = H.alpha_complex.get_alpha_complex_lines(H.alpha, type='all'
    ↪)
ac_simplices_exterior = H.alpha_complex.get_alpha_complex_lines(H.alpha, type=
    ↪ 'exterior')
ac_simplices_interior = H.alpha_complex.get_alpha_complex_lines(H.alpha, type=
    ↪ 'interior')
ac_simplices_regular = H.alpha_complex.get_alpha_complex_lines(H.alpha, type=
    ↪ 'regular')
ac_simplices_singular = H.alpha_complex.get_alpha_complex_lines(H.alpha, type=
    ↪ 'singular')
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)

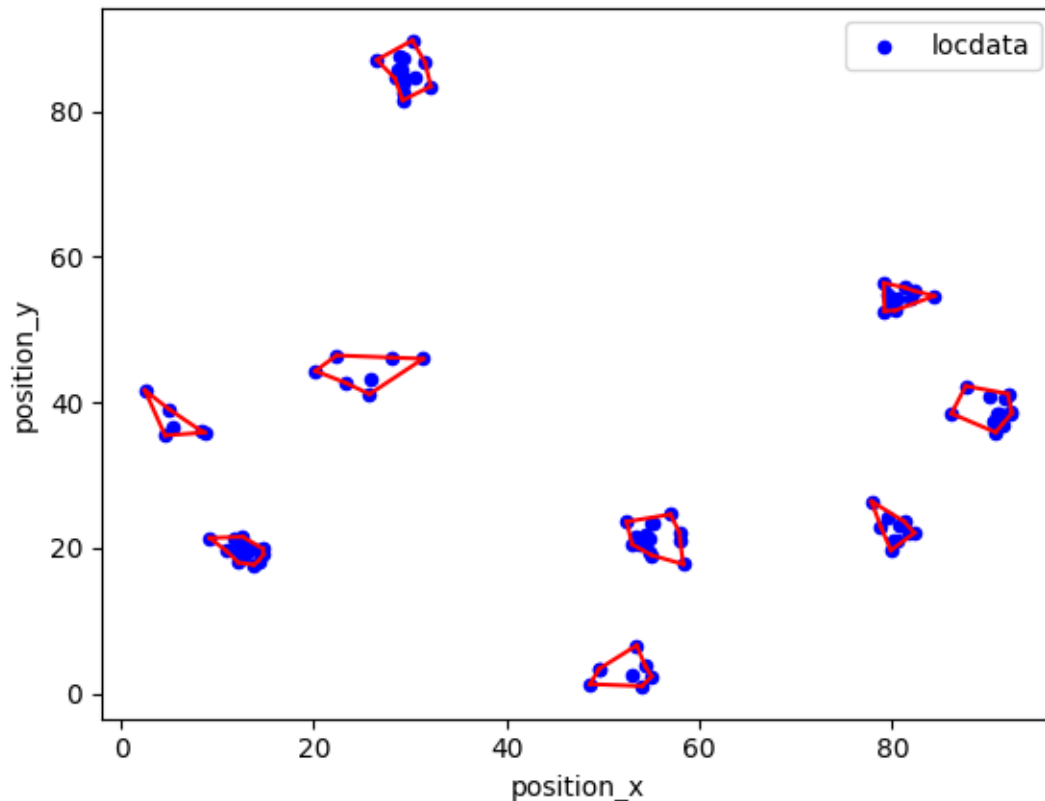
for simp in ac_simplices_all:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '-b')
for simp in ac_simplices_interior:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '--g')
for simp in ac_simplices_regular:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '--r')
for simp in ac_simplices_singular:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '--y')

locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↪ label='locdata')
plt.show()
```



The *regular* representation:

```
fig, ax = plt.subplots(nrows=1, ncols=1)
for simp in ac_simplices_regular:
    ax.plot(locdata.coordinates[simp, 0], locdata.coordinates[simp, 1], '-r')
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↪ label='locdata')
plt.show()
```



The connected components:

H.connected\_components

```
[Polygon([[4.574030584926964, 35.4547363609949], [2.490921001041831, 41.
→ 64287079543503], [4.957382196904078, 39.08282122381802], [8.355461487809666,
→ 36.04765129727869], [8.66131719294932, 35.8564682341916], [4.
→ 574030584926964, 35.4547363609949]], []),
Polygon([[25.770117329307787, 41.124276031608815], [23.31756949147803, 42.
→ 6744624631213], [20.173587957294433, 44.31714346765541], [22.
→ 361223833116604, 46.453014294097834], [28.048700032796415, 46.
→ 15804762285728], [31.339293507790977, 45.9942555369238], [25.
→ 770117329307787, 41.124276031608815]], []),
Polygon([[9.18991822286495, 21.36788078938049], [12.636282213847565, 21.
→ 529820077010385], [14.734872660721315, 19.880462363217678], [14.
→ 769094009431546, 19.124654613922488], [14.266019685598483, 18.
→ 241818179484355], [13.747576761626673, 17.702091783845788], [12.
→ 175420163612081, 17.997868633149913], [9.18991822286495, 21.
→ 36788078938049]], []),
Polygon([[26.54041545299338, 87.040661544405], [30.214047987922243, 89.
→ 71010693298172], [31.521384860659595, 86.75121552412102], [32.
→ 14133335513356, 83.41408741528058], [29.20688463095611, 81.48951050894098],
→ [26.54041545299338, 87.040661544405]], []),
Polygon([[79.18789302054475, 52.543982100600005], [79.11703202181116, 56.
→ 39645603252982], [81.32782142473194, 55.75284367902573], [84.34954910628653,
→ 54.62340239857981], [80.25475558909794, 52.61761293856212], [79.
→ 18789302054475, 52.543982100600005]], []),
```

(continues on next page)

(continued from previous page)

```

Polygon([[87.6064843703255, 42.247461144808746], [91.9734281153137, 41.
→17838012011845], [92.33940616216016, 38.808549274650204], [92.
→36001122636628, 38.48950874170735], [91.40098079916173, 36.989250622930314],
→[90.7011218417029, 35.889994988035625], [86.13341308357771, 38.
→463382259593324], [87.6064843703255, 42.247461144808746]], []),
Polygon([[54.029321929694525, 1.0299749716244107], [48.634206414428576, 1.
→294631346507537], [49.59358344008751, 3.4015003371361145], [53.
→482683026695234, 6.614947738231005], [54.37183424361821, 3.788452129115082],
→[55.03000267597442, 2.3642990669580453], [54.029321929694525, 1.
→0299749716244107]], []),
Polygon([[53.02310815665912, 20.494919113980316], [52.4417533406316, 23.
→609344720491215], [56.99055629523848, 24.610250050117035], [57.
→896406888972834, 22.172515398810596], [57.98560316805971, 21.
→06912087658576], [58.31284235106916, 17.772793788937257], [55.
→04194051962414, 18.999536120747145], [53.02310815665912, 20.
→494919113980316]], []),
Polygon([[78.79284317381992, 22.962147412197464], [77.83372328947765, 26.
→395146222813697], [81.19928424977152, 23.660768547251035], [82.
→28290378326082, 22.061077730066597], [79.83990217703906, 19.69136737327613],
→[78.79284317381992, 22.962147412197464]], [])]
```

```

connected_component_0 = H.connected_components[0]

print('dimension: ', connected_component_0.dimension)
print('region_measure: ', connected_component_0.region_measure)
print('subregion_measure: ', connected_component_0.subregion_measure)
```

```

dimension: 2
region_measure: 12.16313947011497
subregion_measure: 19.10814169604162
```

## 2.4 Tutorial about metadata

```

import numpy as np
import pandas as pd
from google.protobuf import json_format, text_format

import locan as lc
```

```

/tmp/ipykernel_1586/2539966667.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
→of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
→better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
```

(continues on next page)

(continued from previous page)

`↪54466`

```
import pandas as pd
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

Locan:

version: 0.20.0.dev41+g755b969

Python:

version: 3.11.6

## 2.4.1 Metadata definition

We have define a canonical set of metadata to accompany localization data.

Metadata is described by protobuf messages. Googles protobuf format is advantageous to enforce metdata definitions that can be easily attached to various file formats, exchanged with other programmes and implemented in different programming languages.

Metadata is instantiated through messages defined in the `locan.data.metadata_pb2` module.

```
list(lc.data.metadata_pb2.DESRIPTOR.message_types_by_name.keys())
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
```

```
[Open3D INFO] WebRTC GUI backend enabled.
```

```
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

```
['Operation',
 'File',
 'Address',
 'Affiliation',
 'Person',
 'ExperimentalSample',
 'ExperimentalSetup',
 'OpticalUnit',
 'Illumination',
 'Detection',
 'Camera',
 'Acquisition',
 'Lightsheet',
 'Experiment',
 'Localizer',
 'Relation',
 'Property',
 'Metadata']
```

Each class contains a logical set of information that is integrated in the main class `Metadata`.

`Metadata` contains the following keys:

```
metadata = lc.data.metadata_pb2.Metadata()
list(metadata.DESRIPTOR.fields_by_name.keys())
```

```
['identifier',
 'comment',
 'source',
 'state',
 'history',
 'ancestor_identifiers',
 'properties',
 'localization_properties',
 'element_count',
 'frame_count',
 'file',
 'relations',
 'experiment',
 'localizer',
 'map',
 'creation_time',
 'modification_time',
 'production_time']
```

Each field has a predefined type and can be set to appropriate values:

```
metadata.comment = "This is a comment"
```

```
try:
    metadata.comment = 1
except Exception as e:
    print(e)
```

```
bad argument type for built-in operation
```

```
metadata
```

```
comment: "This is a comment"
```

Metadata values including the default values can be shown in JSON format or as dictionary:

```
json_format.MessageToDict(metadata)
```

```
{'comment': 'This is a comment'}
```

```
# except empty fields with repeated message classes
json_format.MessageToDict(metadata, including_default_value_fields=True,
↪preserving_proto_field_name=True)
```

```
{'comment': 'This is a comment',
 'identifier': '',
```

(continues on next page)



(continued from previous page)

```
'source': 'UNKNOWN_SOURCE',
'state': 'UNKNOWN_STATE',
'history': [],
'ancestor_identifiers': [],
'properties': [],
'localization_properties': [],
'element_count': '0',
'frame_count': '0',
'relations': [],
'map': {}}
```

```
json_format.MessageToJson(metadata, including_default_value_fields=True,
↪preserving_proto_field_name=True)
```

```
{\n  "comment": "This is a comment",\n  "identifier": "",\n  "source":
↪"UNKNOWN_SOURCE",\n  "state": "UNKNOWN_STATE",\n  "history": [],\n
↪"ancestor_identifiers": [],\n  "properties": [],\n  "localization_properties
↪": [],\n  "element_count": "0",\n  "frame_count": "0",\n  "relations": [],\n
↪\n  "map": {}\n}
```

To print metadata with timestamp and duration in a well formatted string use:

```
lc.metadata_to_formatted_string(metadata)
```

```
'comment: "This is a comment"\n'
```

## 2.4.2 Set metadata fields

### Repeated fields

To set selected fields instantiate the appropriate messages. For list fields use `message.add()`.

```
metadata = lc.data.metadata_pb2.Metadata()

ou = metadata.experiment.setups.add().optical_units.add()
ou.detection.camera.electrons_per_count = 13.26

metadata
```

```
experiment {
  setups {
    optical_units {
      detection {
        camera {
          electrons_per_count: 13.26
        }
      }
    }
  }
}
```

(continues on next page)

(continued from previous page)

```
}
}
```

## Timestamp fields

Timestamp fields contain information on date and time zone and are of type `google.protobuf.Timestamp`.

```
import time
metadata = lc.data.metadata_pb2.Metadata()
metadata.creation_time.GetCurrentTime()
metadata.creation_time
```

```
seconds: 1710414556
nanos: 851861000
```

```
metadata.creation_time.FromJsonString('2022-05-14T06:58:00.514893Z')
metadata.creation_time.ToJsonString()
```

```
'2022-05-14T06:58:00.514893Z'
```

Time duration fields contain information on time intervals and are of type `google.protobuf.Duration`.

```
metadata.experiment.setups.add().optical_units.add().detection.camera.
↪integration_time.FromMilliseconds(20)
metadata.experiment.setups[0].optical_units[0].detection.camera.integration_
↪time.ToMilliseconds()
# metadata.experiment.setups[0].optical_units[0].detection.camera.integration_
↪time.ToJsonString()
```

```
20
```

To print metadata with timestamp and duration in a well formatted string use:

```
lc.metadata_to_formatted_string(metadata)
```

```
'experiment {\n  setups {\n    optical_units {\n      detection {\n        ↪
↪camera {\n          integration_time {\n            0.020s\n          }\n        ↪
↪    }\n      }\n    }\n  }\n}\ncreation_time {\n  2022-05-14T06:58:00.
↪514893Z\n}\n'
```

### 2.4.3 Metadata scheme

The overall scheme can be instantiated and visualized:

```
metadata = lc.data.metadata_pb2.Metadata()
scheme = lc.message_scheme(metadata)
scheme
```

```
{'identifier': '',
 'comment': '',
 'source': 'UNKNOWN_SOURCE',
 'state': 'UNKNOWN_STATE',
 'history': {'name': '', 'parameter': ''},
 'ancestor_identifiers': [],
 'properties': {'identifier': '',
 'comment': '',
 'name': '',
 'unit': '',
 'type': '',
 'map': {}},
 'localization_properties': {'identifier': '',
 'comment': '',
 'name': '',
 'unit': '',
 'type': '',
 'map': {}},
 'element_count': '0',
 'frame_count': '0',
 'relations': {'identifier': '',
 'comment': '',
 'map': {}},
 'file': {'identifier': '',
 'comment': '',
 'type': 'UNKNOWN_FILE_TYPE',
 'path': '',
 'groups': []}},
 'map': {},
 'file': {'identifier': '',
 'comment': '',
 'type': 'UNKNOWN_FILE_TYPE',
 'path': '',
 'groups': []},
 'experiment': {'identifier': '',
 'comment': '',
 'experimenters': {'identifier': '',
 'comment': '',
 'first_name': '',
 'last_name': '',
 'title': '',
 'affiliations': {'institute': '',
```

(continues on next page)

(continued from previous page)

```

'department': '',
'address': {'address_lines': [],
'city': '',
'city_code': '',
'country': ''}},
'emails': [],
'roles': [],
'address': {'address_lines': [],
'city': '',
'city_code': '',
'country': ''}},
'samples': {'identifier': '',
'comment': '',
'targets': [],
'fluorophores': [],
'buffers': [],
'map': {}},
'setups': {'identifier': '',
'comment': '',
'optical_units': {'identifier': '',
'comment': '',
'illumination': {'identifier': '',
'comment': '',
'lightsource': '',
'power': 0.0,
'area': 0.0,
'power_density': 0.0,
'wavelength': 0.0,
'map': {}},
'detection': {'identifier': '',
'comment': '',
'map': {}},
'camera': {'identifier': '',
'comment': '',
'name': '',
'model': '',
'gain': 0.0,
'electrons_per_count': 0.0,
'pixel_count_x': 0,
'pixel_count_y': 0,
'pixel_size_x': 0.0,
'pixel_size_y': 0.0,
'flipped': False,
'map': {}},
'offset': 0.0,
'serial_number': '',
'integration_time': '0s'}},
'acquisition': {'identifier': '',
'comment': '',

```

(continues on next page)

(continued from previous page)

```

'frame_count': 0,
'frame_of_interest_first': 0,
'frame_of_interest_last': 0,
'stack_count': 0,
'stack_step_count': 0,
'stack_step_size': 0.0,
'map': {},
'time_start': '1970-01-01T00:00:00Z',
'time_end': '1970-01-01T00:00:00Z',
'lightsheet': {'identifier': '',
'comment': '',
'angle_x': 0.0,
'angle_y': 0.0,
'angle_z': 0.0,
'map': {}},
'map': {},
'map': {},
'localizer': {'identifier': '',
'comment': '',
'software': '',
'intensity_threshold': 0.0,
'psf_fixed': False,
'psf_size': 0.0,
'map': {}},
'creation_time': '1970-01-01T00:00:00Z',
'modification_time': '1970-01-01T00:00:00Z',
'production_time': '1970-01-01T00:00:00Z'}

```

## Metadata from toml file

You can provide metadata in a [toml](#) file.

```

metadata_toml = \
"""
# Define the class (message) instances.

[[messages]]
name = "metadata"
module = "locan.data.metadata_pb2"
class_name = "Metadata"

# Fill metadata attributes
# Headings must be a message name or valid attribute.
# Use [[]] to add repeated elements.
# Use string '2022-05-14T06:58:00Z' for Timestamp elements.
# Use int in nanoseconds for Duration elements.

```

(continues on next page)

(continued from previous page)

```
[metadata]
identifier = "123"
comment = "my comment"
ancestor_identifiers = ["1", "2"]
production_time = '2022-05-14T06:58:00Z'

[[metadata.experiment.experimenters]]
first_name = "First name"
last_name = "Last name"

[[metadata.experiment.experimenters.affiliations]]
institute = "Institute"
department = "Department"

[[metadata.experiment.setups]]
identifier = "1"

[[metadata.experiment.setups.optical_units]]
identifier = "1"

[metadata.experiment.setups.optical_units.detection.camera]
identifier = "1"
name = "camera name"
model = "camera model"
electrons_per_count = 3.1
integration_time = 10_000_000

[metadata.localizer]
software = "rapidSTORM"

[[metadata.relations]]
identifier = "1"
"""
```

```
toml_out = lc.metadata_from_toml_string(metadata_toml)
for k, v in toml_out.items():
    print(k, ":\n\n", v)
```

```
metadata :

  identifier: "123"
  comment: "my comment"
  ancestor_identifiers: "1"
  ancestor_identifiers: "2"
  relations {
    identifier: "1"
  }
  experiment {
    experimenters {
```

(continues on next page)

(continued from previous page)

```

first_name: "First name"
last_name: "Last name"
affiliations {
  institute: "Institute"
  department: "Department"
}
}
setups {
  identifier: "1"
  optical_units {
    identifier: "1"
    detection {
      camera {
        identifier: "1"
        name: "camera name"
        model: "camera model"
        electrons_per_count: 3.1
        integration_time {
          nanos: 100000000
        }
      }
    }
  }
}
}
}
}
localizer {
  software: "rapidSTORM"
}
production_time {
  seconds: 1652511480
}

```

To load from file:

#### 2.4.4 Metadata for LocData

Metadata is instantiated for each LocData object and accessible through the LocData.meta attribute.

#### Sample data

```

df = pd.DataFrame(
    {
        'position_x': np.arange(0,10),
        'position_y': np.random.random(10),
        'frame': np.arange(0,10),
    })
locdata = lc.LocData.from_dataframe(dataframe=df)

```

(continues on next page)

(continued from previous page)

```
locdata.meta
```

```
identifier: "1"
source: DESIGN
state: RAW
history {
  name: "LocData.from_dataframe"
}
element_count: 10
frame_count: 10
creation_time {
  seconds: 1710414556
  nanos: 906704000
}
```

Fields can also be printed as well formatted string (using `lc.metadata_to_formatted_string`):

```
locdata.print_meta()
```

```
identifier: "1"
source: DESIGN
state: RAW
history {
  name: "LocData.from_dataframe"
}
element_count: 10
frame_count: 10
creation_time {
  2024-03-14T11:09:16.906704Z
}
```

A summary of the most important metadata is printed as:

```
locdata.print_summary()
```

```
identifier: "1"
comment: ""
source: DESIGN
state: RAW
element_count: 10
frame_count: 10
creation_time {
  2024-03-14T11:09:16.906704Z
}
```

Metadata fields can be printed and changed individually:

```
print(locdata.meta.comment)
```

(continues on next page)



(continued from previous page)

```
locdata.meta.comment = 'user comment'
print(locdata.meta.comment)
```

```
user comment
```

Metadata can also be added at instantiation:

```
locdata_2 = lc.LocData.from_dataframe(dataframe=df, meta={'identifier': 'myID_
↳1',
                                                         'comment': 'my own user_
↳comment'})
locdata_2.print_summary()
```

```
identifier: "myID_1"
comment: "my own user comment"
source: DESIGN
state: RAW
element_count: 10
frame_count: 10
creation_time {
  2024-03-14T11:09:16.933277Z
}
```

## 2.5 Tutorial about regions

Regions define a support for localization data or specify a hull that captures a set of localizations. Locan provides various region classes with a standard set of attributes and methods.

```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import locan as lc
```

```
/tmp/ipykernel_1910/3049569888.py:4: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release_
↳of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and_
↳better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳54466

import pandas as pd
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.5.1 Region definitions

The standard set of attributes and methods is defined by the abstract base class `Region` that all region classes inherit.

```
lc.Region?
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

```
print("Methods:")
[method for method in dir(lc.Region) if not method.startswith('_')]
```

```
Methods:
```

```
['bounding_box',
 'bounds',
 'buffer',
 'centroid',
 'contains',
 'dimension',
 'extent',
 'from_intervals',
 'intersection',
 'max_distance',
 'points',
 'region_measure',
 'subregion_measure',
 'symmetric_difference',
 'union']
```

Further definitions can be found in the abstract classes `Region1D`, `Region2D` and `Region3D` and all specific region classes.

```
import inspect
inspect.getmembers(lc.data.region, inspect.isabstract)
```

```
[('Region', lc.data.region.Region),
 ('Region1D', lc.data.region.Region1D),
```

(continues on next page)

(continued from previous page)

```
('Region2D', locan.data.region.Region2D),
('Region3D', locan.data.region.Region3D),
('RegionND', locan.data.region.RegionND)]
```

## 2.5.2 Use Region classes

Use one of the following classes to define a region in 1, 2 or 3 dimensions:

```
print("Empty Region:\n", [lc.EmptyRegion.__name__], "\n")
print("Regions in 1D:\n", [cls.__name__ for cls in lc.Region1D.__subclasses__
→()], "\n")
print("Regions in 2D:\n", [cls.__name__ for cls in lc.Region2D.__subclasses__
→()], "\n")
print("Regions in 3D:\n", [cls.__name__ for cls in lc.Region3D.__subclasses__
→()], "\n")
print("Regions in nD:\n", [cls.__name__ for cls in lc.RegionND.__subclasses__
→()], "\n")
```

```
Empty Region:
['EmptyRegion']
```

```
Regions in 1D:
['Interval']
```

```
Regions in 2D:
['Rectangle', 'Ellipse', 'Polygon', 'MultiPolygon']
```

```
Regions in 3D:
['AxisOrientedCuboid', 'Cuboid']
```

```
Regions in nD:
['AxisOrientedHyperCuboid']
```

The region constructors take different parameters.

REMEMBER: Angles are taken in degrees.

```
region = lc.Rectangle(corner=(0, 0), width=1, height=2, angle=45)
region
```

```
Rectangle((0, 0), 1, 2, 45)
```

```
points = ((0, 0), (0, 1), (1, 1), (1, 0.5), (0, 0))
holes = (((0.2, 0.2), (0.2, 0.4), (0.4, 0.4), (0.3, 0.25)), ((0.5, 0.5), (0.5,
→ 0.8), (0.8, 0.8), (0.7, 0.45)))
region = lc.Polygon(points, holes)
print(region)
region
```

```
Polygon(<self.points>, <self.holes>)
```

```
Polygon([[0.0, 0.0], [0.0, 1.0], [1.0, 1.0], [1.0, 0.5], [0.0, 0.0]], [[0.2, 0.2], [0.2, 0.4], [0.4, 0.4], [0.3, 0.25]], [[0.5, 0.5], [0.5, 0.8], [0.8, 0.8], [0.7, 0.45]])
```

Several attributes are available, e.g. about the area or circumference.

```
dict(dimension=region.dimension, bounds=region.bounds, extent=region.extent,
    bounding_box=region.bounding_box, centroid=region.centroid, max_
    distance=region.max_distance,
    region_measure= region.region_measure, subregion_measure=region.
    subregion_measure)
```

```
{'dimension': 2,
 'bounds': array([0., 0., 1., 1.]),
 'extent': array([1., 1.]),
 'bounding_box': Rectangle((0.0, 0.0), 1.0, 1.0, 0),
 'centroid': array([0.42723735, 0.61770428]),
 'max_distance': 1.4142135623730951,
 'region_measure': 0.6425,
 'subregion_measure': 5.480275727142994}
```

A list of points defining a polygon that resembles the region is available.

```
print("Points:\n", region.points, "\n")
print("Holes:\n", region.holes)
```

```
Points:
[[0.  0. ]
 [0.  1. ]
 [1.  1. ]
 [1.  0.5]
 [0.  0. ]]

Holes:
[array([[0.2 , 0.2 ],
        [0.2 , 0.4 ],
        [0.4 , 0.4 ],
        [0.3 , 0.25]]), array([[0.5 , 0.5 ],
        [0.5 , 0.8 ],
        [0.8 , 0.8 ],
        [0.7 , 0.45]])]
```

Region can be constructed from interval tuples indicating feature ranges.

```
region_1d = lc.Region.from_intervals((0, 1))
region_2d = lc.Region.from_intervals(((0, 1), (0, 1)))
region_3d = lc.Region.from_intervals([(0, 1)] * 3)
region_4d = lc.Region.from_intervals([(0, 1)] * 4)
```

```
for region in (region_1d, region_2d, region_3d, region_4d):
    print(region)
```

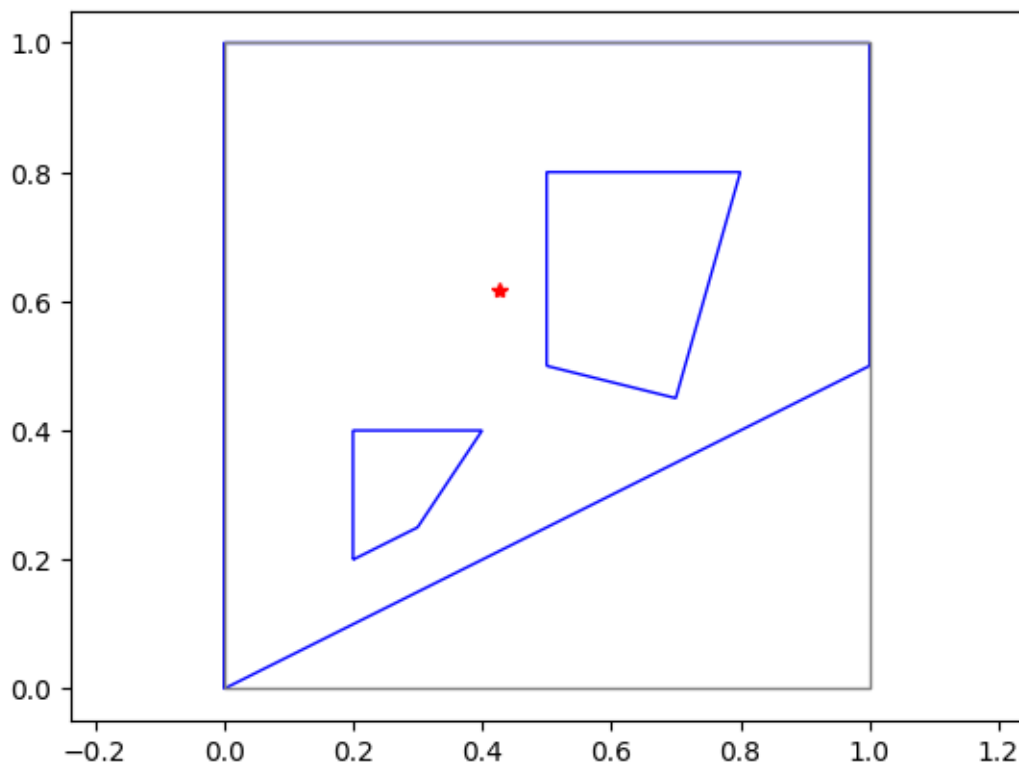
```
Interval(0, 1)
Rectangle((0, 0), 1, 1, 0)
AxisOrientedCuboid((0, 0, 0), 1, 1, 1)
AxisOrientedHyperCuboid((0, 0, 0, 0), (1, 1, 1, 1))
```

### 2.5.3 Plot regions

Regions can be plotted as patch in matplotlib figures.

```
points = ((0, 0), (0, 1), (1, 1), (1, 0.5), (0, 0))
holes = [((0.2, 0.2), (0.2, 0.4), (0.4, 0.4), (0.3, 0.25)), ((0.5, 0.5), (0.5,
→ 0.8), (0.8, 0.8), (0.7, 0.45))]
region = lc.Polygon(points, holes)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.add_patch(region.as_artist(fill=False, color='Blue'))
ax.add_patch(region.bounding_box.as_artist(fill=False, color='Grey'))
ax.plot(*region.centroid, '*', color='Red')
ax.axis('equal')
plt.show()
```



### 2.5.4 Intersection, union, difference of regions

Methods are provided to check for intersection, difference, union and membership.

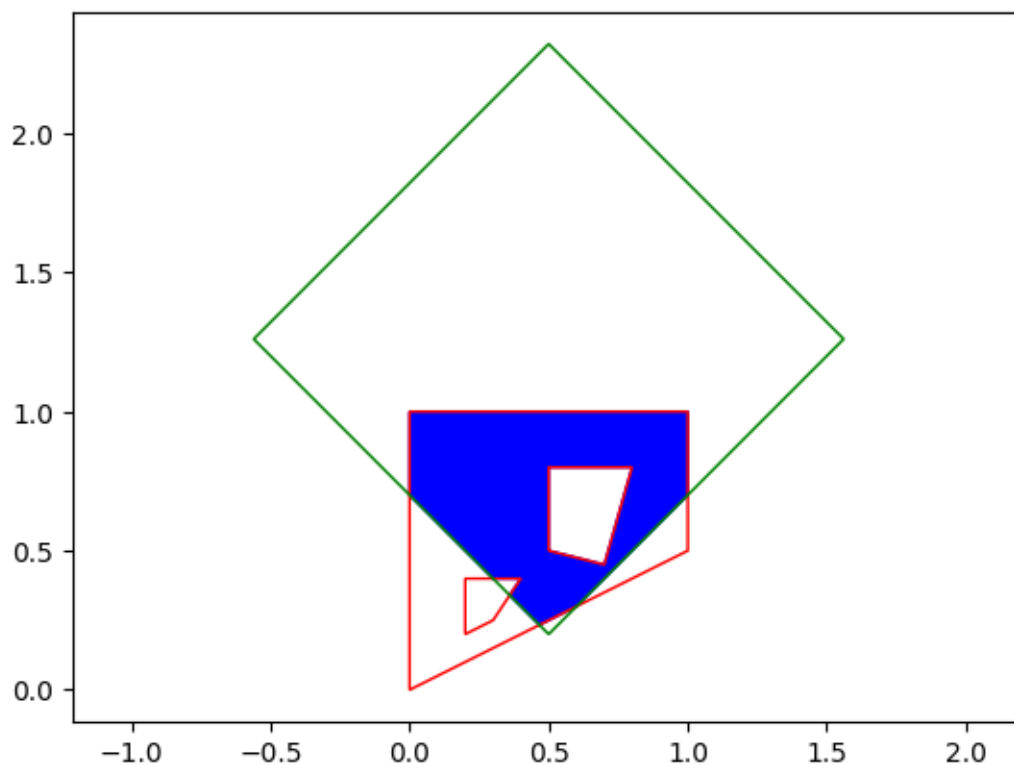
```
other_region = lc.Rectangle(corner=(0.5, 0.2), width=1.5, height=1.5,
↪angle=45)
other_region.shapely_object
```

```
<POLYGON ((0.5 0.2, -0.561 1.261, 0.5 2.321, 1.561 1.261, 0.5 0.2))>
```

```
result = region.intersection(other_region)
print(result)
```

```
Polygon(<self.points>, <self.holes>)
```

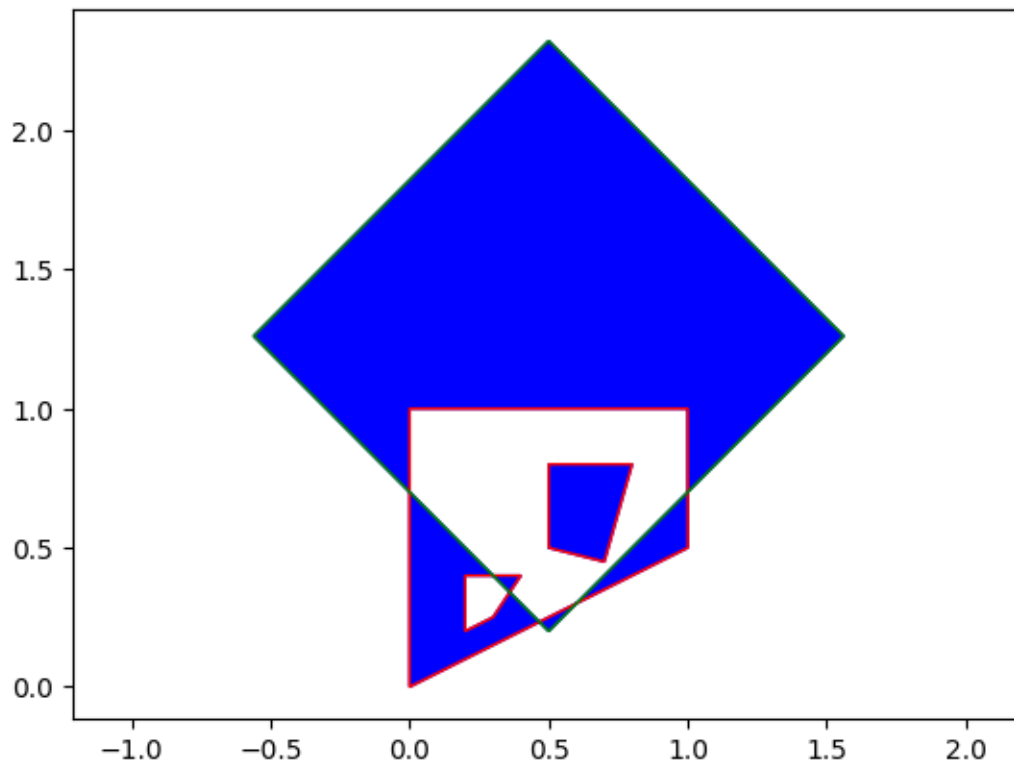
```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.add_patch(result.as_artist(fill=True, color='Blue'))
ax.add_patch(region.as_artist(fill=False, color='Red'))
ax.add_patch(other_region.as_artist(fill=False, color='Green'))
ax.axis('equal')
plt.show()
```



```
result = region.symmetric_difference(other_region)
print(result)
```

```
MultiPolygon(<self.polygons>)
```

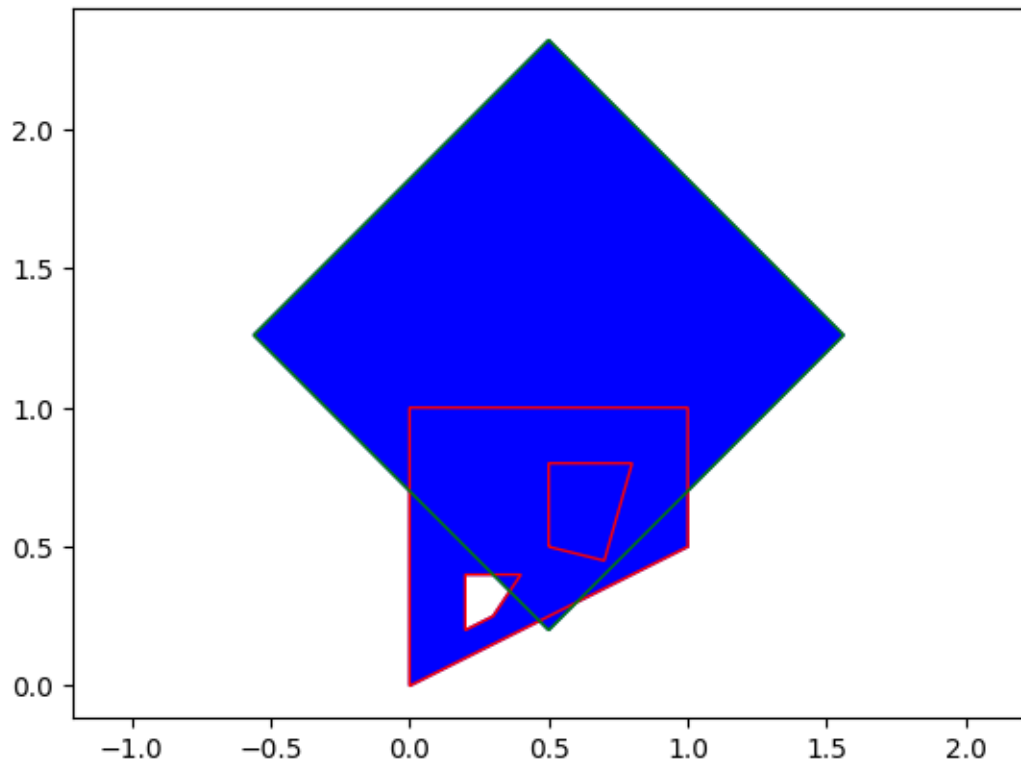
```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.add_patch(result.as_artist(fill=True, color='Blue'))
ax.add_patch(region.as_artist(fill=False, color='Red'))
ax.add_patch(other_region.as_artist(fill=False, color='Green'))
ax.axis('equal')
plt.show()
```



```
result = region.union(other_region)
print(result)
```

```
Polygon(<self.points>, <self.holes>)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.add_patch(result.as_artist(fill=True, color='Blue'))
ax.add_patch(region.as_artist(fill=False, color='Red'))
ax.add_patch(other_region.as_artist(fill=False, color='Green'))
ax.axis('equal')
plt.show()
```



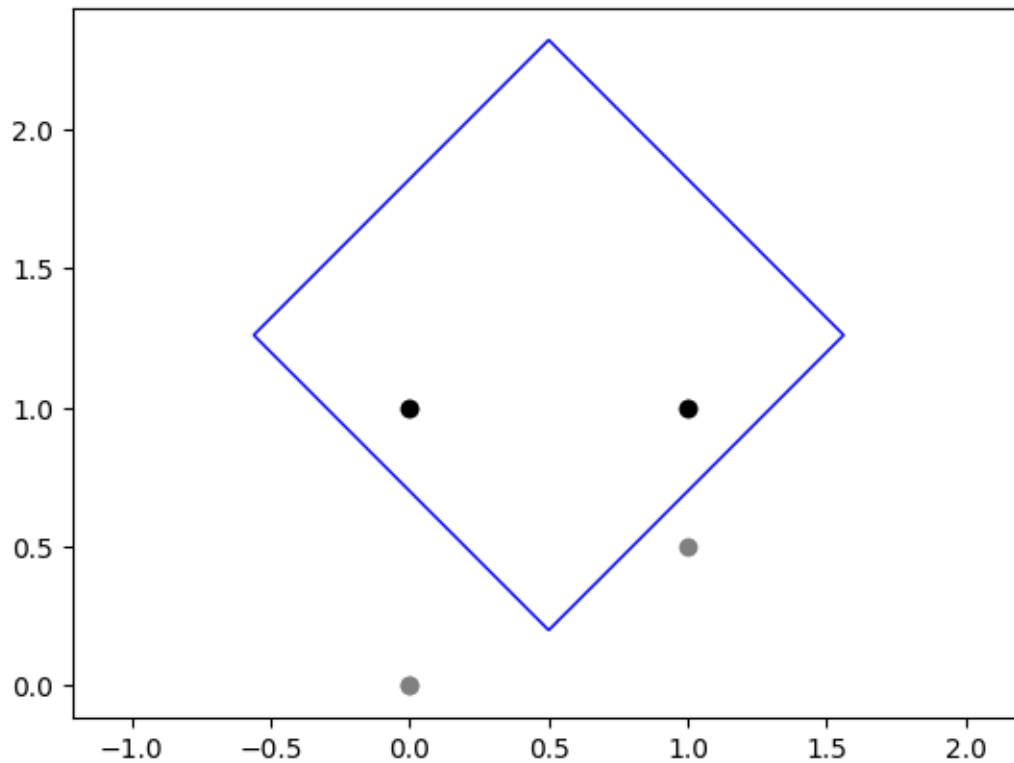
### 2.5.5 Check if point is in region

Region has a `contains` method to select points that are within the region.

```
inside_indices = other_region.contains(region.points)
contained_points = region.points[inside_indices]
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.scatter(*region.points.T, color='Grey')
ax.scatter(*contained_points.T, color='Black')
ax.add_patch(other_region.as_artist(fill=False, color='Blue'))
ax.axis('equal')
plt.show()
```





### 2.5.6 LocData and regions

LocData bring various hulls that define regions. Also LocData typically has a unique region defined as support. This can e.g. result from the definition of a region of interest using the ROI function or as specified in a corresponding yaml file.

#### Create data in region:

A random dataset is created within a specified region (for other methods see simulation tutorial).

```
region = lc.Rectangle(corner=(0, 0), width=1, height=1, angle=45)
locdata = lc.simulate_uniform(n_samples=1000, region=region, seed=1)
locdata.print_summary()
```

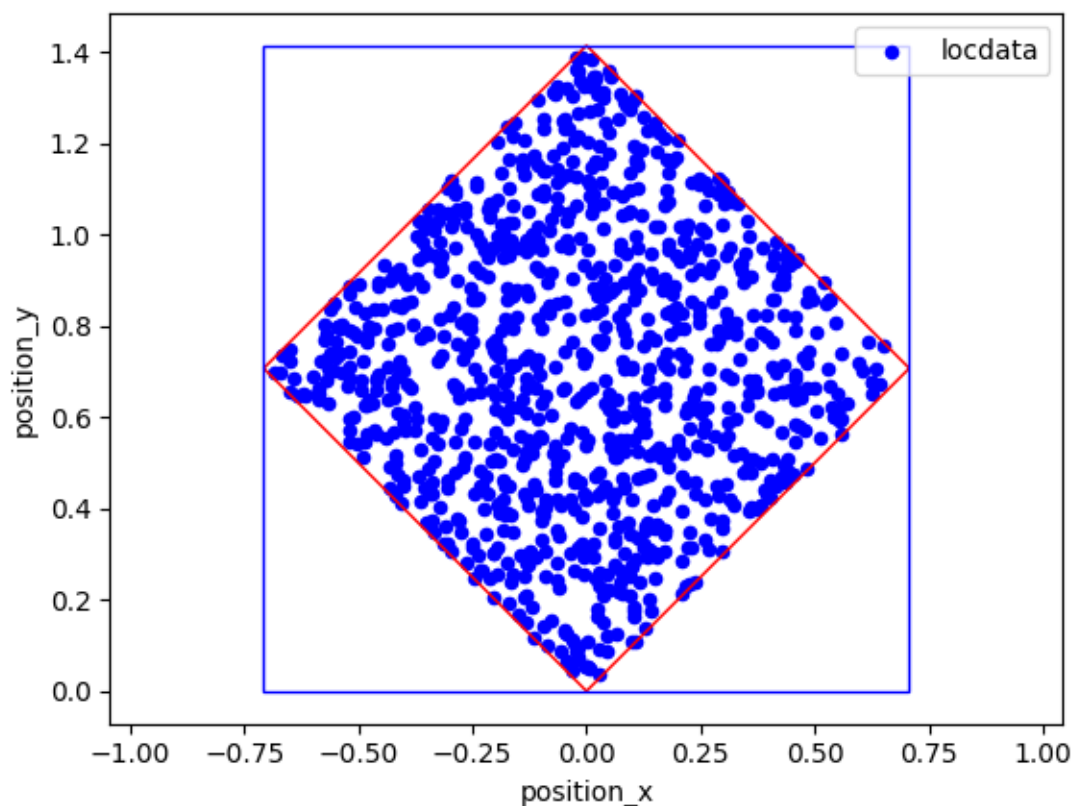
```
identifier: "1"
comment: ""
source: SIMULATION
state: RAW
element_count: 1000
frame_count: 0
creation_time {
  2024-03-14T11:09:42.437524Z
}
```

```
region
```

```
Rectangle((0, 0), 1, 1, 45)
```

### Show scatter plots together with regions

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    → label='locdata')
ax.add_patch(locdata.region.as_artist(fill=False, color='Red'))
ax.add_patch(locdata.region.bounding_box.as_artist(fill=False, color='Blue'))
ax.axis('equal')
plt.show()
```



### 2.5.7 Select localizations within regions

LocData can be selected for localizations being inside the region.

```
region = lc.Ellipse(center=(0, 0.5), width=1, height=0.5, angle=45)
locdata_in_region = lc.select_by_region(locdata, region)
locdata_in_region.region
```

```
Ellipse((0.0, 0.5), 1, 0.5, 45)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
```

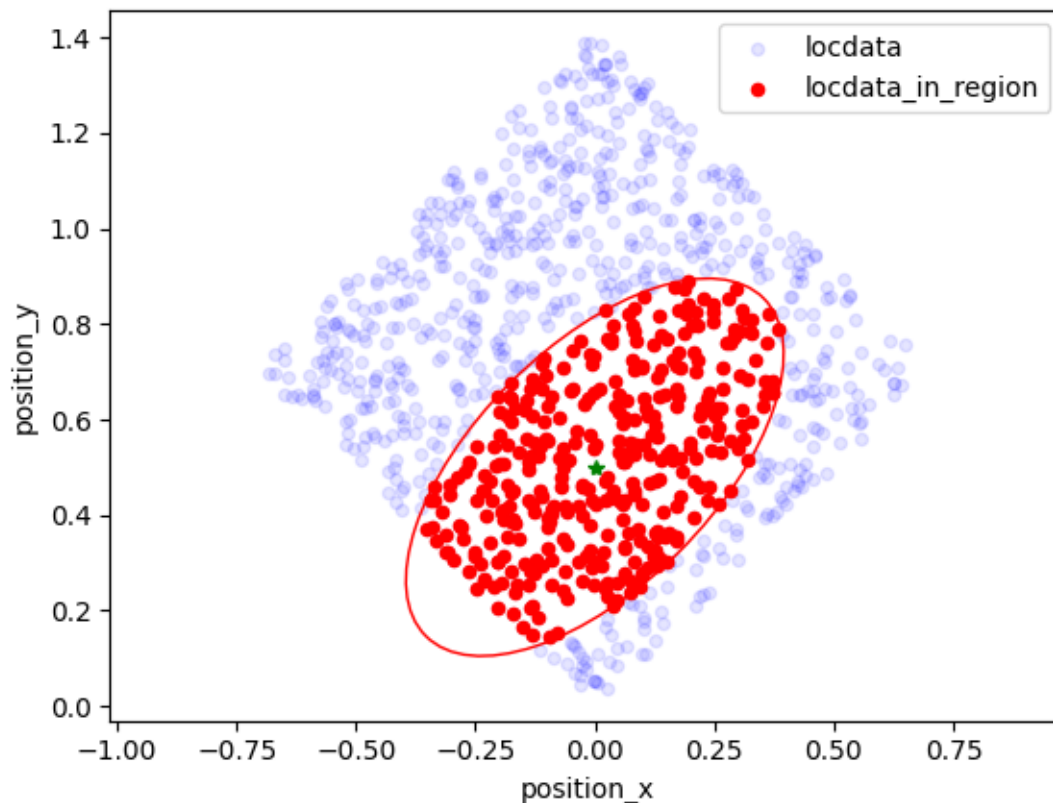
(continues on next page)

(continued from previous page)

```

↪ label='locdata', alpha=0.1)
ax.add_patch(region.as_artist(fill=False, color='Red'))
locdata_in_region.data.plot.scatter(x='position_x', y='position_y', ax=ax,
↪ color='Red', label='locdata_in_region')
ax.plot(*region.centroid, '*', color='Green')
ax.axis('equal')
plt.show()

```



## 2.5.8 Regions of interest

The Roi class is an object that defines a region of interest for a specific localization dataset. It is mostly used to save and reload regions of interest after having selected them interactively, e.g. in napari. It is therefore related to region specifications and a unique LocData object.

Define a region of interest (roi):

```

roi = lc.Roi(reference=locdata, region=lc.Ellipse(center=(0, 0.5), width=1,
↪ height=0.5, angle=80))
roi

```

```

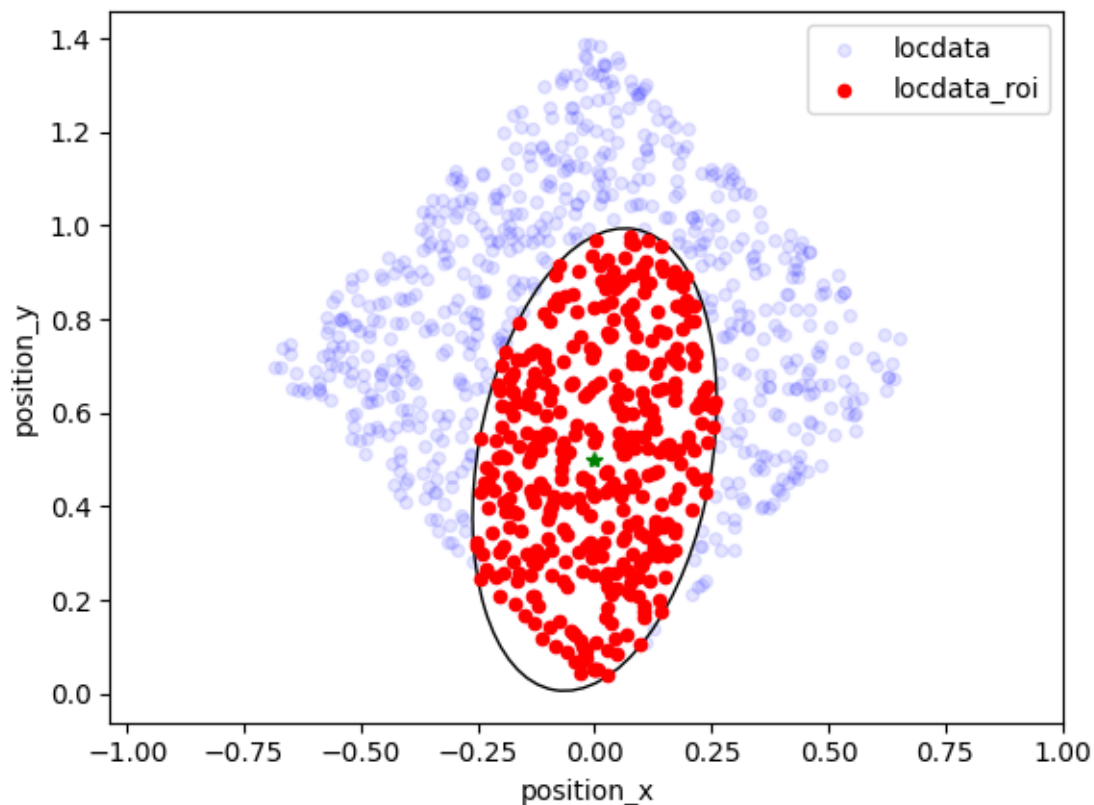
Roi(reference=<locan.data.locdata.LocData object at 0x7f9c41de7550>,
↪ region=Ellipse((0.0, 0.5), 1, 0.5, 80), loc_properties=())

```

Create new LocData instance by selecting localizations within a roi.

```
locdata_roi = roi.locdata()
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↳ label='locdata', alpha=0.1)
ax.add_patch(roi.region.as_artist(fill=False))
locdata_roi.data.plot.scatter(x='position_x', y='position_y', ax=ax, color=
    ↳ 'Red', label='locdata_roi')
ax.plot(*roi.region.centroid, '*', color='Green')
ax.axis('equal')
plt.show()
```



## ROI input/output

If you have prepared rois and saved them as roi.yaml file you can read that data back in:

```
import tempfile
from pathlib import Path

with tempfile.TemporaryDirectory() as tmp_directory:
    file_path = Path(tmp_directory) / 'roi.yaml'

    roi.to_yaml(path=file_path)

    roi_new = lc.Roi.from_yaml(path = file_path)
```

(continues on next page)

(continued from previous page)

```

roi_new.reference = roi.reference

new_locdata = roi_new.locdata()
new_locdata.meta

```

```

/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/rois/roi.py:300: UserWarning: The
↳localization data has to be saved and the file path provided, or the
↳reference is lost.
warnings.warn(

```

```

identifier: "4"
source: SIMULATION
state: MODIFIED
history {
  name: "make_uniform"
  parameter: "{\n_samples\: 1000, \region\: Rectangle((0, 0), 1, 1, 45), \
↳seed\: 1}"
}
history {
  name: "locdata"
  parameter: "{\self\: Roi(reference=<locan.data.locdata.LocData object at
↳0x7f9c41de7550>, region=Ellipse((0.0, 0.5), 1, 0.5, 80), loc_
↳properties=[]), \reduce\: True}"
}
ancestor_identifiers: "1"
element_count: 388
frame_count: 0
creation_time {
  seconds: 1710414582
  nanos: 437524000
}
modification_time {
  seconds: 1710414582
  nanos: 437524000
}

```

```
roi_new
```

```

Roi(reference=<locan.data.locdata.LocData object at 0x7f9c41de7550>,
↳region=Ellipse((0.0, 0.5), 1, 0.5, 80), loc_properties=[])

```

## 2.6 Tutorial about filtering LocData objects

```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import locan as lc
```

```
/tmp/ipykernel_1407/3049569888.py:4: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳ 54466

import pandas as pd
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.6.1 Synthetic data

A random dataset is created.

```
rng = np.random.default_rng(seed=1)
```

```
locdata = lc.simulate_Thomas(parent_intensity=1e-5, region=((0, 1000), (0,
↳ 1000)), cluster_mu=100, cluster_std=10, seed=rng)

locdata.print_summary()
```

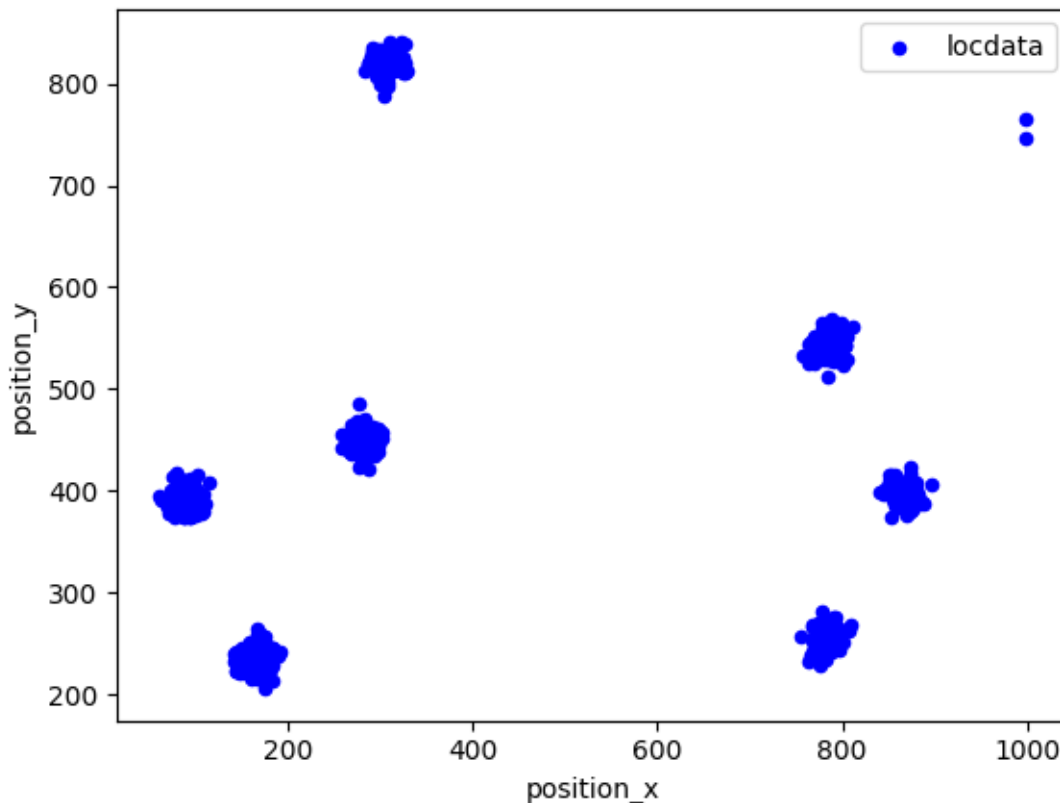
```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
identifier: "1"
comment: ""
source: SIMULATION
state: RAW
```

(continues on next page)

(continued from previous page)

```
element_count: 705
frame_count: 0
creation_time {
  2024-03-14T11:08:45.309935Z
}
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↪ label='locdata')
plt.show()
```



## 2.6.2 Select localizations according to property conditions

A LocData object carries localization data with certain properties for each localization.

We can select localisations according to property conditions.

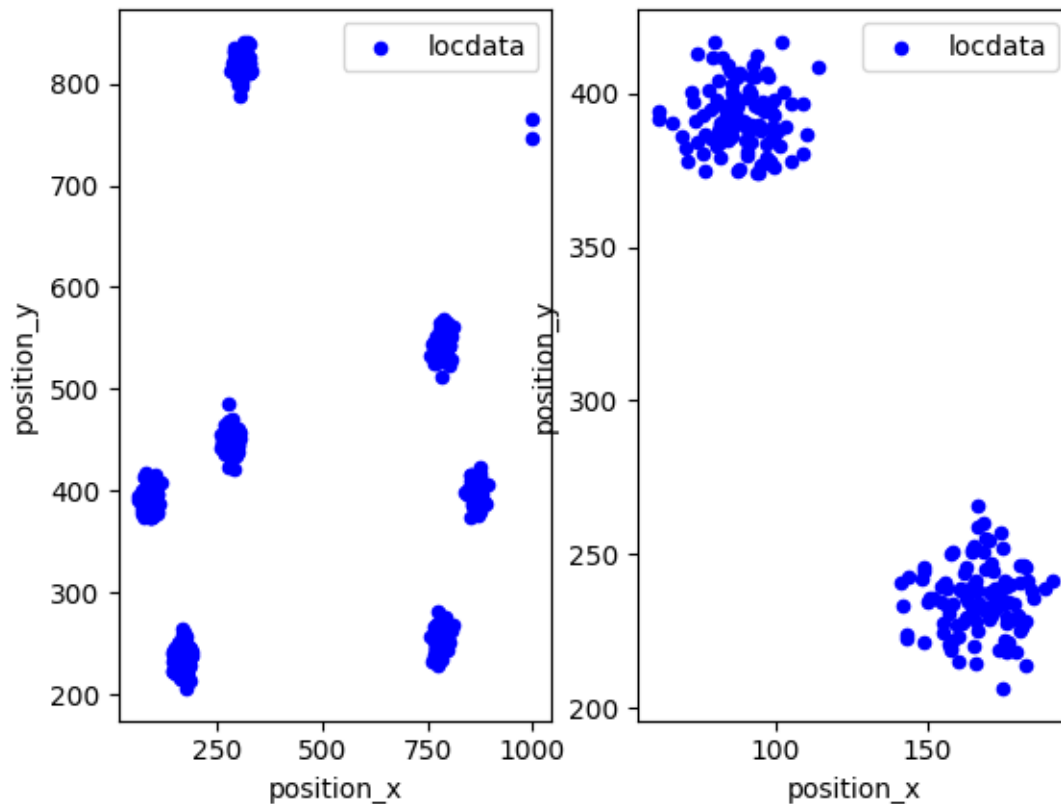
```
locdata_select = lc.select_by_condition(locdata, condition='position_x<200')
```

```
fig, ax = plt.subplots(nrows=1, ncols=2)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax[0], color=
    ↪ 'Blue', label='locdata')
locdata_select.data.plot.scatter(x='position_x', y='position_y', ax=ax[1],
    ↪ color='Blue', label='locdata')
```

(continues on next page)

(continued from previous page)

```
plt.tight_layout
plt.show()
```



### 2.6.3 Select localizations in regions

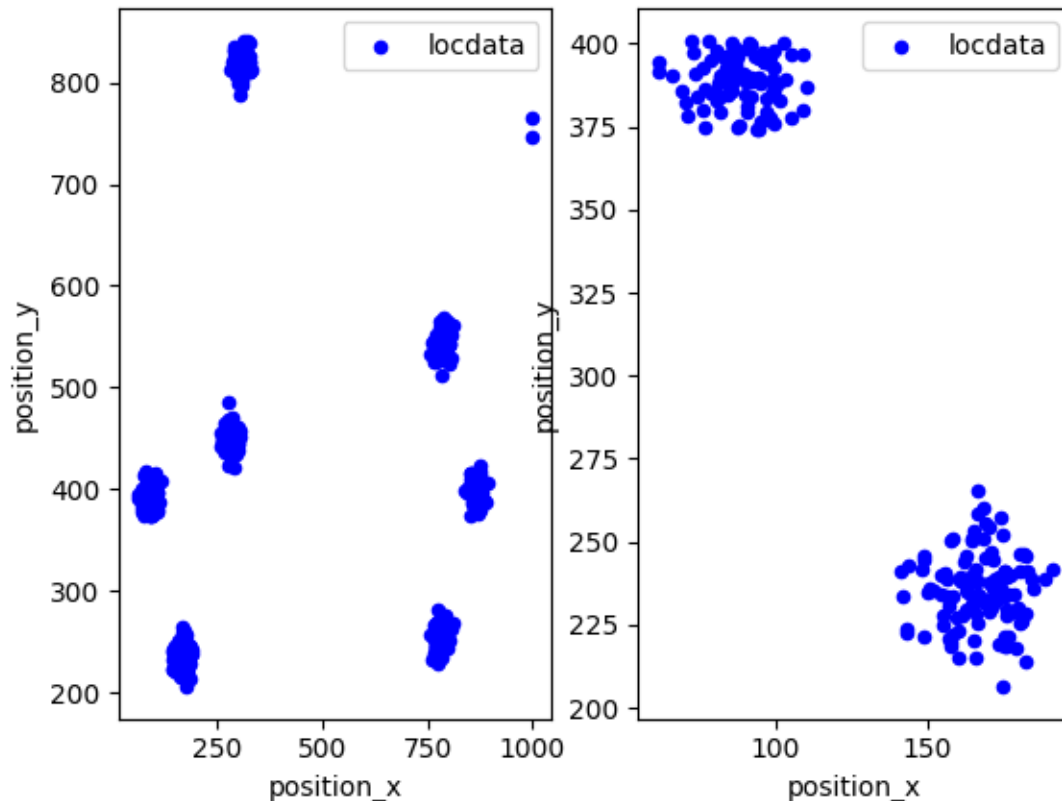
Regions can be defined using the classes defined in the `locan.data.region` module. Please see the tutorial on Regions.

We can select localizations that within a given region.

```
region = region=lc.Rectangle((1, 1), 400, 400, 0)
locdata_roi = lc.select_by_region(locdata, region=region)
```

```
fig, ax = plt.subplots(nrows=1, ncols=2)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax[0], color=
    ↪ 'Blue', label='locdata')
locdata_roi.data.plot.scatter(x='position_x', y='position_y', ax=ax[1], color=
    ↪ 'Blue', label='locdata')
plt.tight_layout
plt.show()
```





### 2.6.4 Select localizations from a region of interest (ROI)

Typically a region of interest is defined for a single dataset. The `Roi` class combines a region definition with a specific data reference and provides convenience methods like input/output.

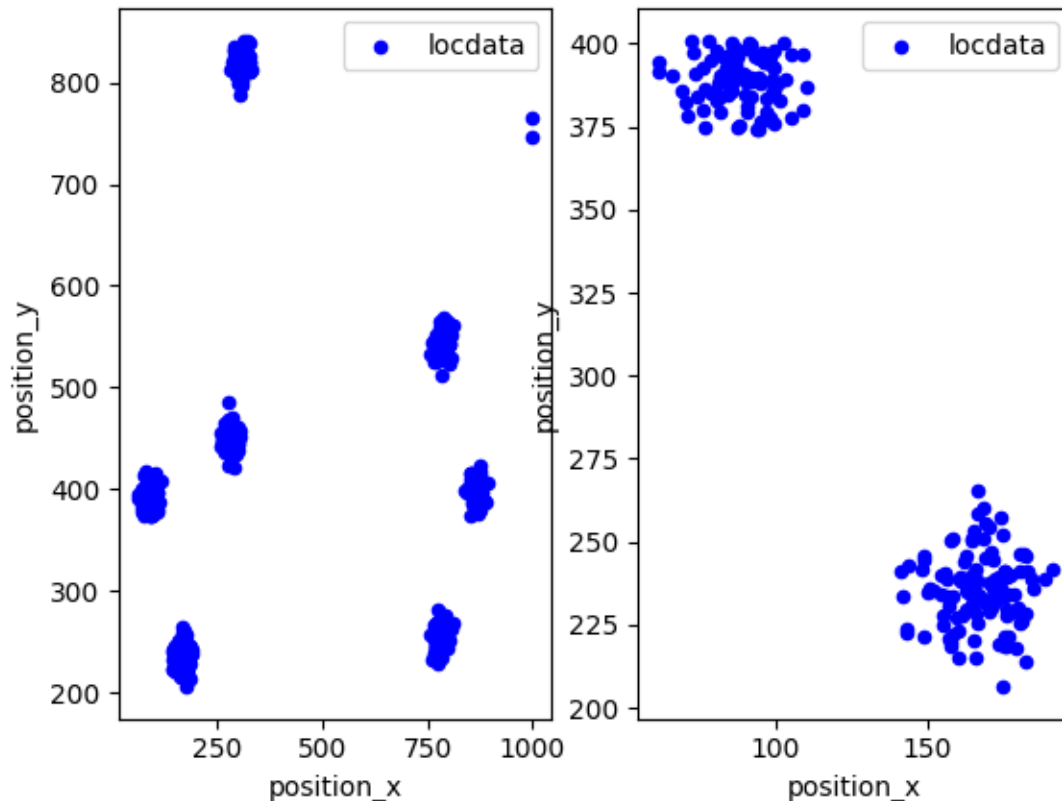
Define a region of interest (roi):

```
roi = lc.Roi(region=lc.Rectangle((1, 1), 400, 400, 0), reference=locdata)
```

Create new `LocData` instance by selecting localizations within a roi.

```
locdata_roi = roi.locdata()
```

```
fig, ax = plt.subplots(nrows=1, ncols=2)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax[0], color=
    ↪ 'Blue', label='locdata')
locdata_roi.data.plot.scatter(x='position_x', y='position_y', ax=ax[1], color=
    ↪ 'Blue', label='locdata')
plt.tight_layout
plt.show()
```



Save roi definitions (including region and reference) as yaml file:

```
roi
```

```
Roi(reference=<locan.data.locdata.LocData object at 0x7f0671269510>,
     region=Rectangle((1, 1), 400, 400, 0), loc_properties=())
```

```
import tempfile
from pathlib import Path

with tempfile.TemporaryDirectory() as tmp_directory:
    file_path = Path(tmp_directory) / 'roi.yaml'

    roi.to_yaml(path=file_path)

    roi_new = lc.Roi.from_yaml(path = file_path)
    roi_new.reference = roi.reference

locdata_new = roi_new.locdata()
locdata_new.meta
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/locan/rois/roi.py:300: UserWarning: The
localization data has to be saved and the file path provided, or the
reference is lost.
warnings.warn(
```

```

identifier: "5"
source: SIMULATION
state: MODIFIED
history {
  name: "make_Thomas"
  parameter: "{\'parent_intensity\': 1e-05, \'region\': ((0, 1000), (0,
↪1000)), \'expansion_factor\': 6, \'cluster_mu\': 100, \'cluster_std\': 10, \
↪\'clip\': True, \'shuffle\': True, \'seed\': Generator(PCG64) at_
↪0x7F068CA67680}"
}
history {
  name: "locdata"
  parameter: "{\'self\': Roi(reference=<locan.data.locdata.LocData object at_
↪0x7f0671269510>, region=Rectangle((1, 1), 400, 400, 0), loc_properties=[]),
↪ \'reduce\': True}"
}
ancestor_identifiers: "1"
element_count: 206
frame_count: 0
creation_time {
  seconds: 1710414525
  nanos: 309935000
}
modification_time {
  seconds: 1710414525
  nanos: 309935000
}

```

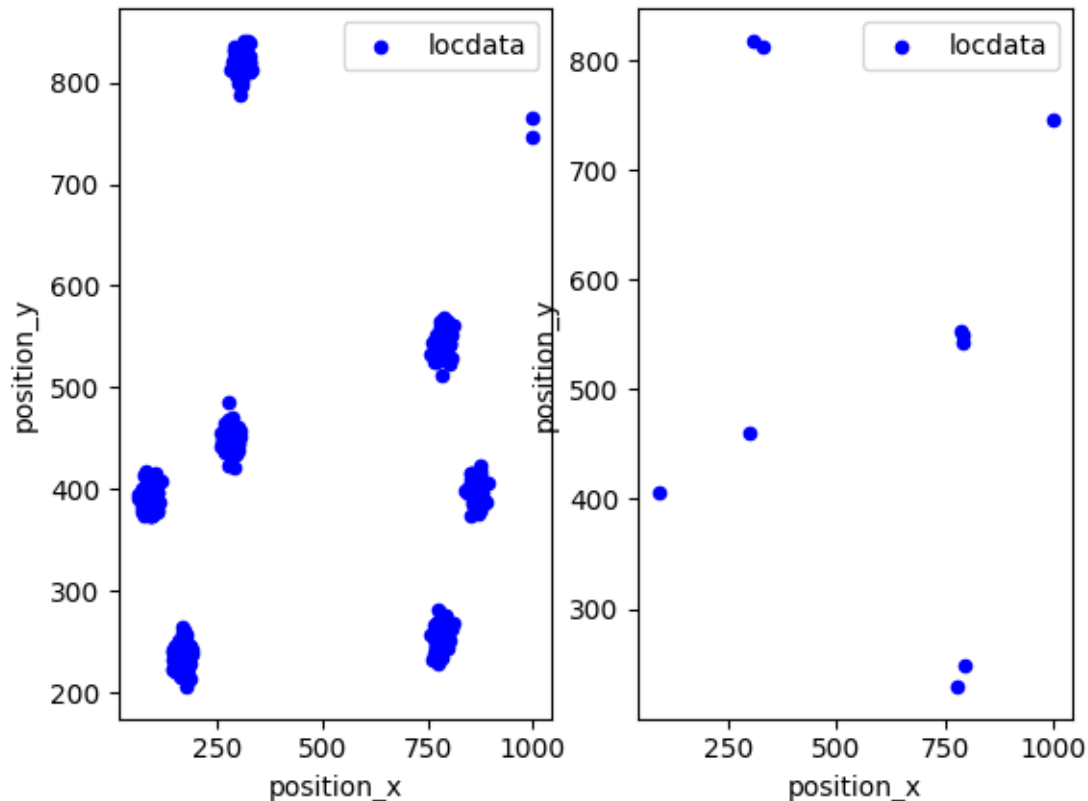
## 2.6.5 Select a random subset of localizations

```
locdata_random = lc.random_subset(locdata, n_points=10, seed=rng)
```

```

fig, ax = plt.subplots(nrows=1, ncols=2)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax[0], color=
↪'Blue', label='locdata')
locdata_random.data.plot.scatter(x='position_x', y='position_y', ax=ax[1],
↪color='Blue', label='locdata')
plt.tight_layout
plt.show()

```



## 2.7 Tutorial about clustering localizations data

Locan provides methods for clustering localizations in LocData objects. The methods all return a new LocDat object that represents the collected selections for each cluster.

```
from pathlib import Path

%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import locan as lc
```

```
/tmp/ipykernel_1299/1692805095.py:6: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳ 54466
```

(continues on next page)

(continued from previous page)

```
import pandas as pd
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.7.1 Synthetic data

We simulate localization data that follows a Neyman-Scott distribution in 2D:

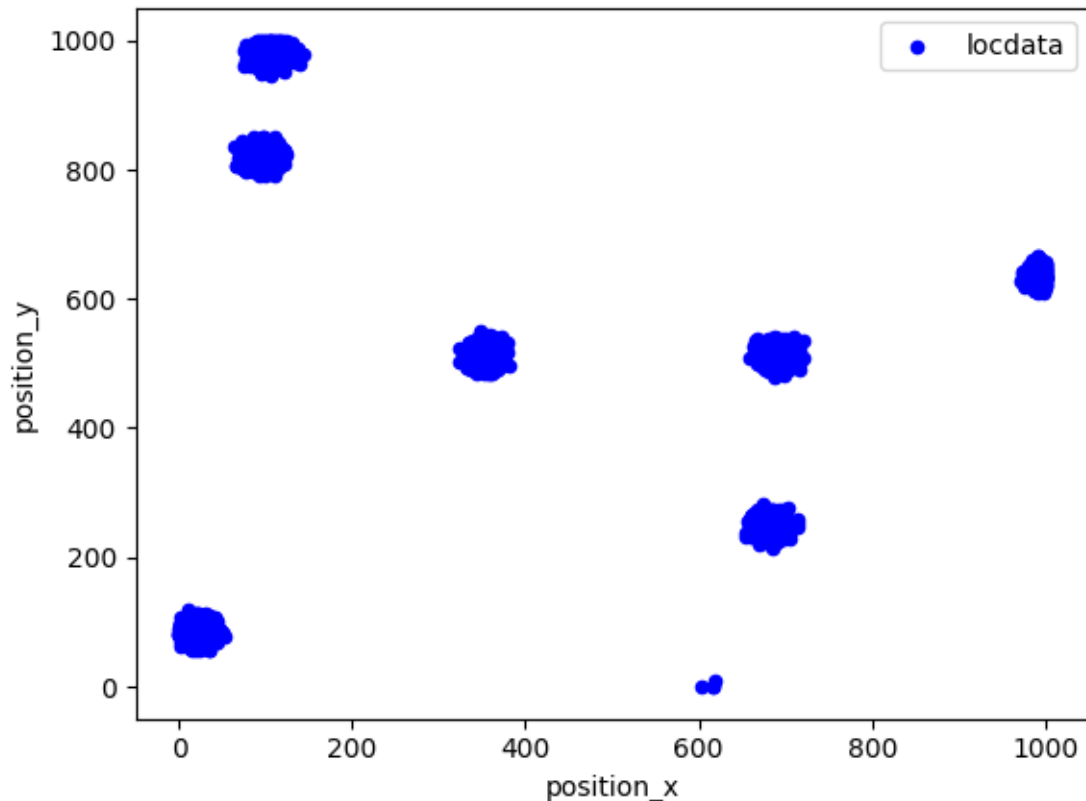
```
rng = np.random.default_rng(seed=11)
```

```
locdata = lc.simulate_Thomas(parent_intensity=1e-5, region=((0, 1000), (0, 1000)), cluster_mu=1000, cluster_std=10, seed=rng)
```

```
locdata.print_summary()
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
identifier: "1"
comment: ""
source: SIMULATION
state: RAW
element_count: 6409
frame_count: 0
creation_time {
  2024-03-14T11:08:06.174673Z
}
```

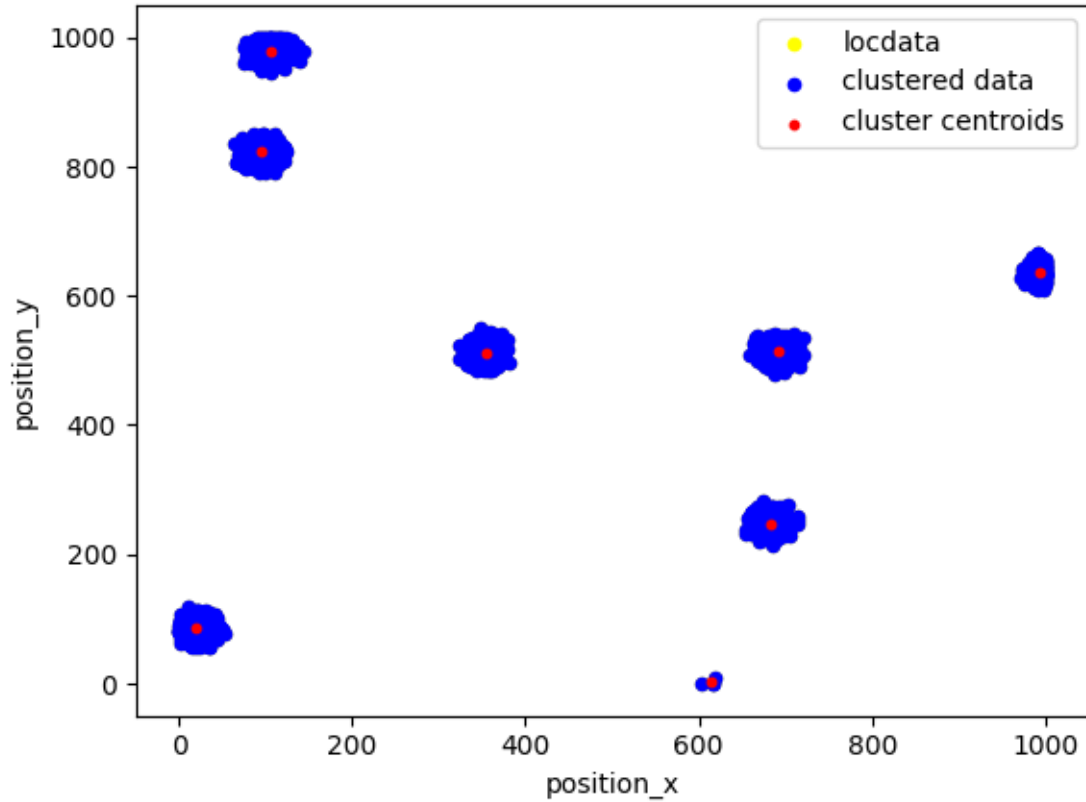
```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
  label='locdata')
plt.show()
```



### 2.7.2 Cluster localizations by dbscan

```
noise, clust = lc.cluster_dbscan(locdata, eps=20, min_samples=3)
assert noise.data.empty
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Yellow',
    ↳ label='locdata')
lc.LocData.concat(clust.references).data.plot.scatter(x='position_x', y=
    ↳ 'position_y', ax=ax, color='Blue', label='clustered data')
clust.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Red',
    ↳ s=10, label='cluster centroids')
plt.show()
```



```
clust.data.head()
```

|   | localization_count | position_x              | uncertainty_x        | position_y | uncertainty_y | \ |
|---|--------------------|-------------------------|----------------------|------------|---------------|---|
| 0 | 1016               | 20.379759               | 0.300643             | 85.374381  | 0.314089      |   |
| 1 | 1050               | 681.909709              | 0.305784             | 248.276785 | 0.310572      |   |
| 2 | 986                | 353.614004              | 0.302981             | 512.621728 | 0.320747      |   |
| 3 | 996                | 690.951712              | 0.315008             | 513.619337 | 0.313548      |   |
| 4 | 368                | 992.523811              | 0.288485             | 636.239028 | 0.548494      |   |
|   | region_measure_bb  | localization_density_bb | subregion_measure_bb | \          |               |   |
| 0 | 3262.794976        | 0.311389                | 229.091796           |            |               |   |
| 1 | 4122.205751        | 0.254718                | 257.346576           |            |               |   |
| 2 | 3814.439353        | 0.258491                | 248.125971           |            |               |   |
| 3 | 3935.485208        | 0.253082                | 250.933843           |            |               |   |
| 4 | 1650.773324        | 0.222926                | 170.375014           |            |               |   |
|   | region_measure     | localization_density    | subregion_measure    |            |               |   |
| 0 | 1000000            | 0.001016                | 4000                 |            |               |   |
| 1 | 1000000            | 0.001050                | 4000                 |            |               |   |
| 2 | 1000000            | 0.000986                | 4000                 |            |               |   |
| 3 | 1000000            | 0.000996                | 4000                 |            |               |   |
| 4 | 1000000            | 0.000368                | 4000                 |            |               |   |

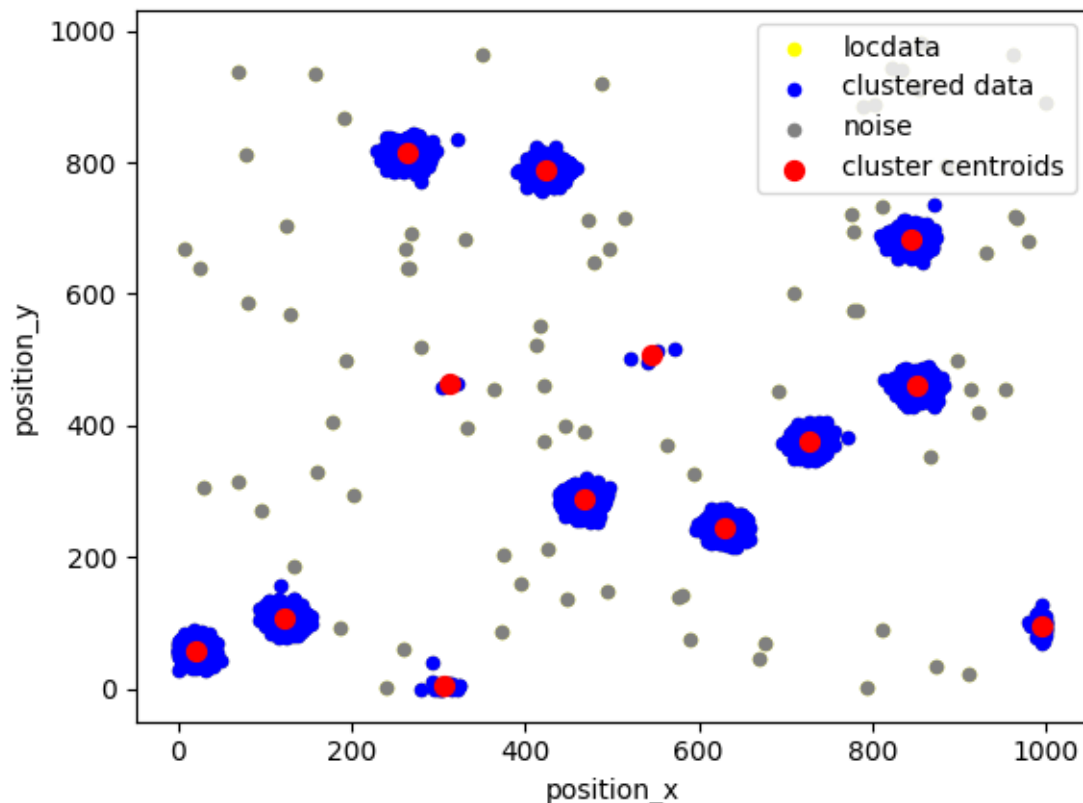
### 2.7.3 Cluster localizations in the presence of noise

Often homogeneously distributed localizations are present that cannot be clustered (so-called noise). In this case *noise* should be set True such that two LocData objects are returned that hold noise and the cluster collection. If *noise* is False it will be part of the returned cluster collection.

```
locdata_cluster = lc.simulate_Thomas(parent_intensity=1e-5, region=((0, 1000),
→ (0, 1000)), cluster_mu=1000, cluster_std=10, seed=rng)
locdata_noise = lc.simulate_Poisson(intensity=1e-4, region=((0, 1000), (0, 1000)), seed=rng)
locdata = lc.LocData.concat([locdata_cluster, locdata_noise])
```

```
noise, clust = lc.cluster_dbscan(locdata, eps=30, min_samples=3)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Yellow',
→ label='locdata')
lc.LocData.concat(clust.references).data.plot.scatter(x='position_x', y=
→ 'position_y', ax=ax, color='Blue', label='clustered data')
noise.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Gray',
→ label='noise')
clust.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Red',
→ s=50, label='cluster centroids')
plt.show()
```

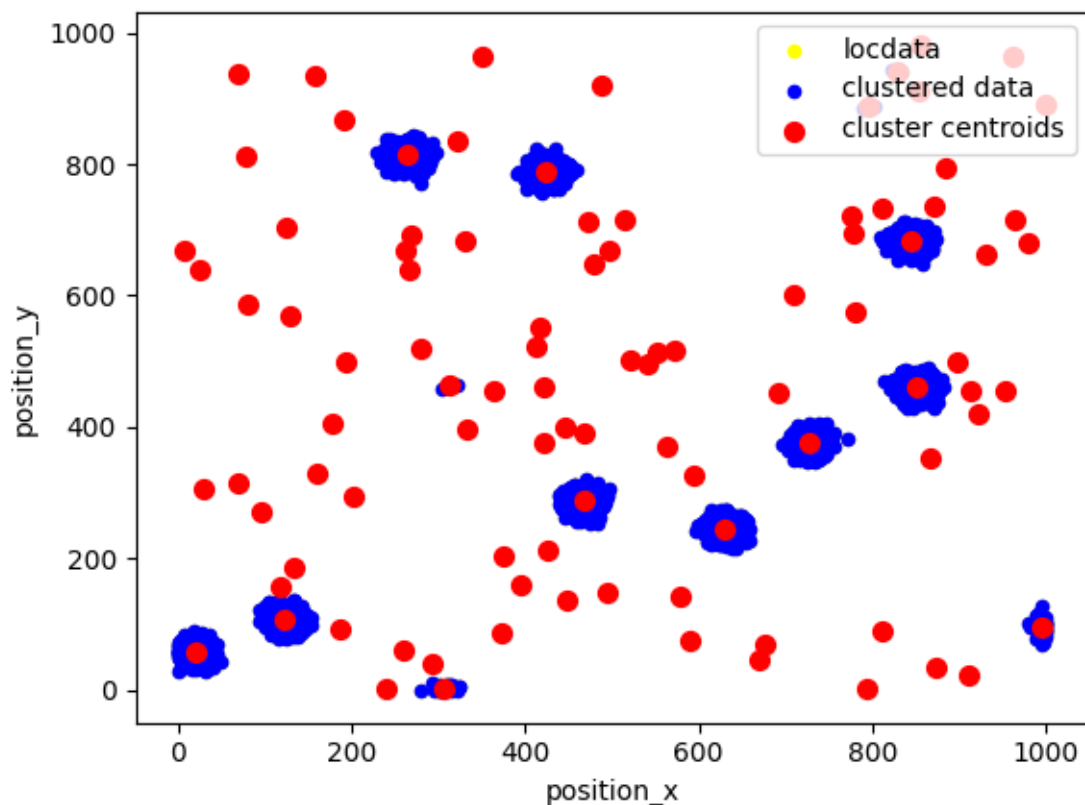


If single localizations should be included as individual clusters, we need to reduce *min\_samples* to 1. In that case *noise* will always be None.



```
noise, clust = lc.cluster_dbscan(locdata, eps=20, min_samples=1)
assert noise.data.empty
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Yellow',
    label='locdata')
lc.LocData.concat(clust.references).data.plot.scatter(x='position_x', y='position_y',
    ax=ax, color='Blue', label='clustered data')
# noise.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Gray',
    label='noise')
clust.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Red',
    s=50, label='cluster centroids')
plt.show()
```



Other cluster functions are available in the `locan.data.clustermodule`.

## 2.8 Tutorial about rendering LocData

```
from pathlib import Path

%matplotlib inline
# %matplotlib widget

import numpy as np
import pandas as pd
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import locan as lc
```

```
/tmp/ipykernel_1947/3690890720.py:7: DeprecationWarning:
Pyyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳ 54466

import pandas as pd
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

## 2.8.1 Synthetic data

Localizations are simulated that are distributed according to a Neyman-Scott distribution (blobs).

```
rng = np.random.default_rng(seed=1)
```

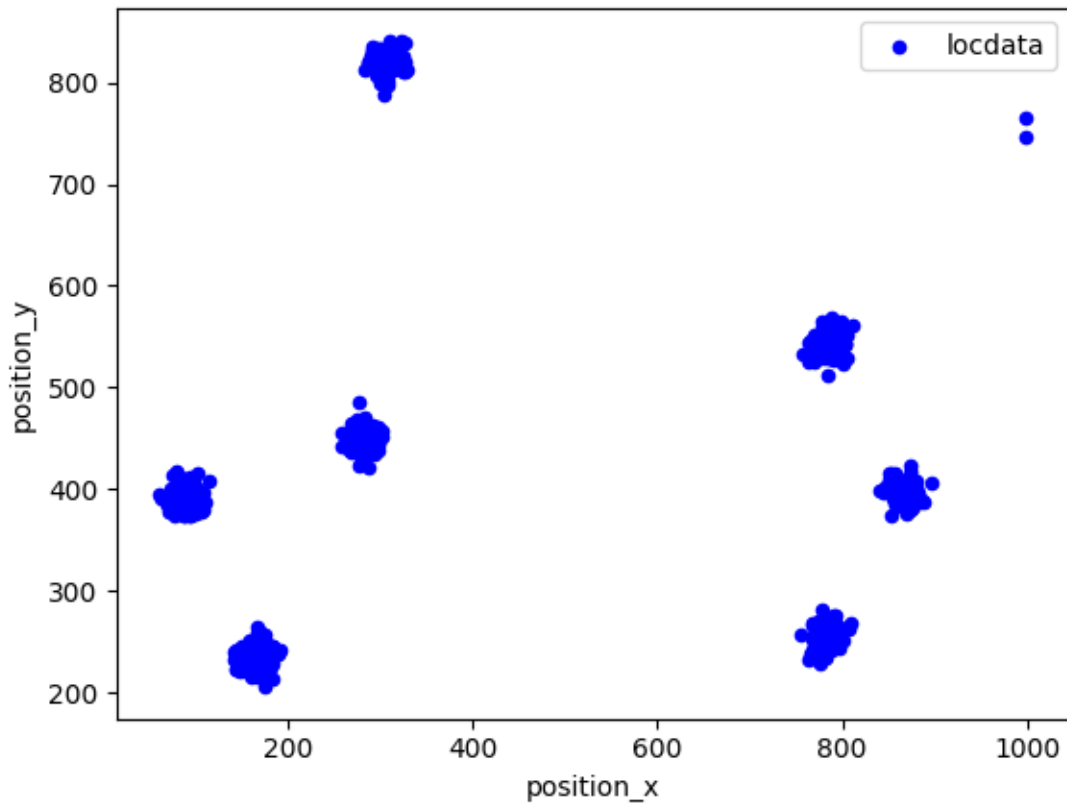
```
locdata = lc.simulate_Thomas(parent_intensity=1e-5, region=((0, 1000), (0,
↳ 1000)), cluster_mu=100, cluster_std=10, seed=rng)

locdata.print_summary()
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
identifier: "1"
comment: ""
source: SIMULATION
state: RAW
element_count: 705
frame_count: 0
creation_time {
  2024-03-14T11:09:47.243568Z
}
```

Since localization data is kept as a pandas dataframe standard plotting routines from pandas or matplotlib can be used.

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
→ label='locdata')
plt.show()
```



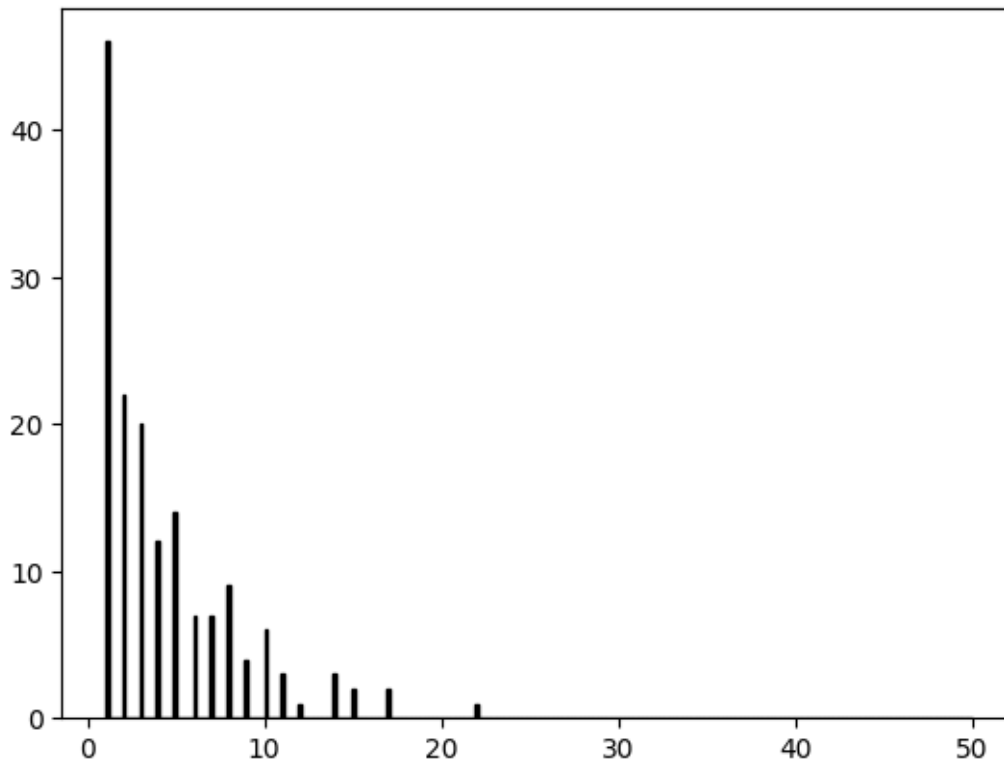
## 2.8.2 Render by simple 2D binning

A method for simply binning localization data in 2D pixels is provided.

```
img, bins, label = lc.histogram(locdata, bin_size=10)
```

The intensity values of the binned locdata are distributed as:

```
plt.hist(img.ravel(), bins=256, range=(1, 50), fc='k', ec='k');
```

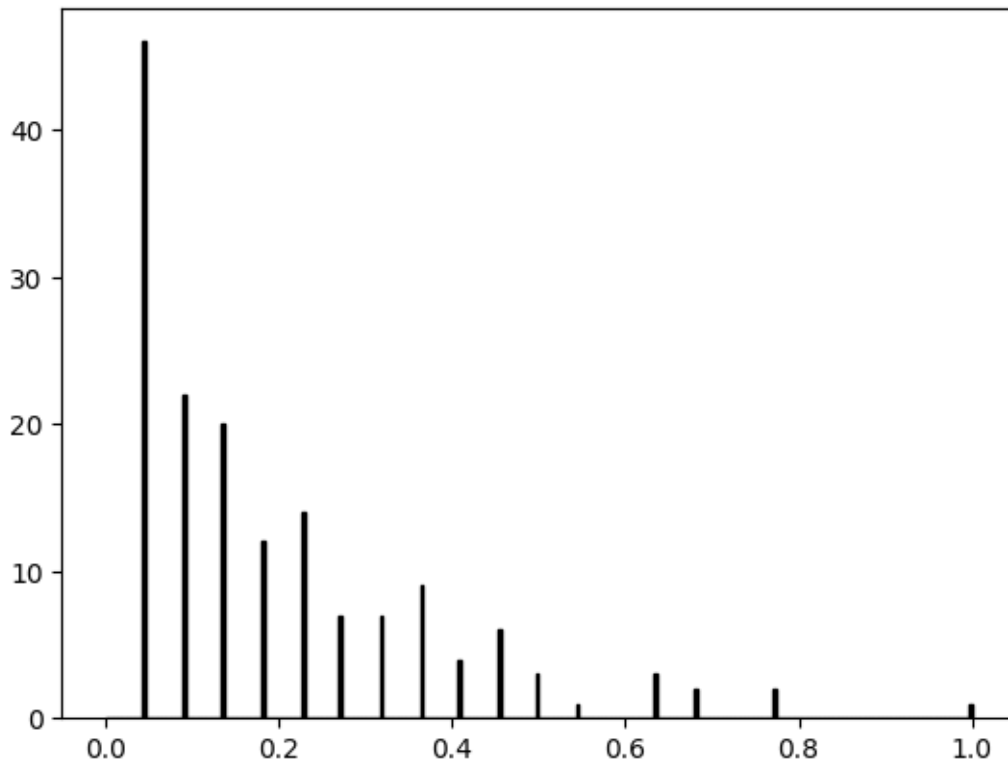


The intensity values can be rescaled in many ways. There are normalization classes and a convenience function in the `locan.render.transformation` module with predefined transformations as listed in `locan.Trafo`:

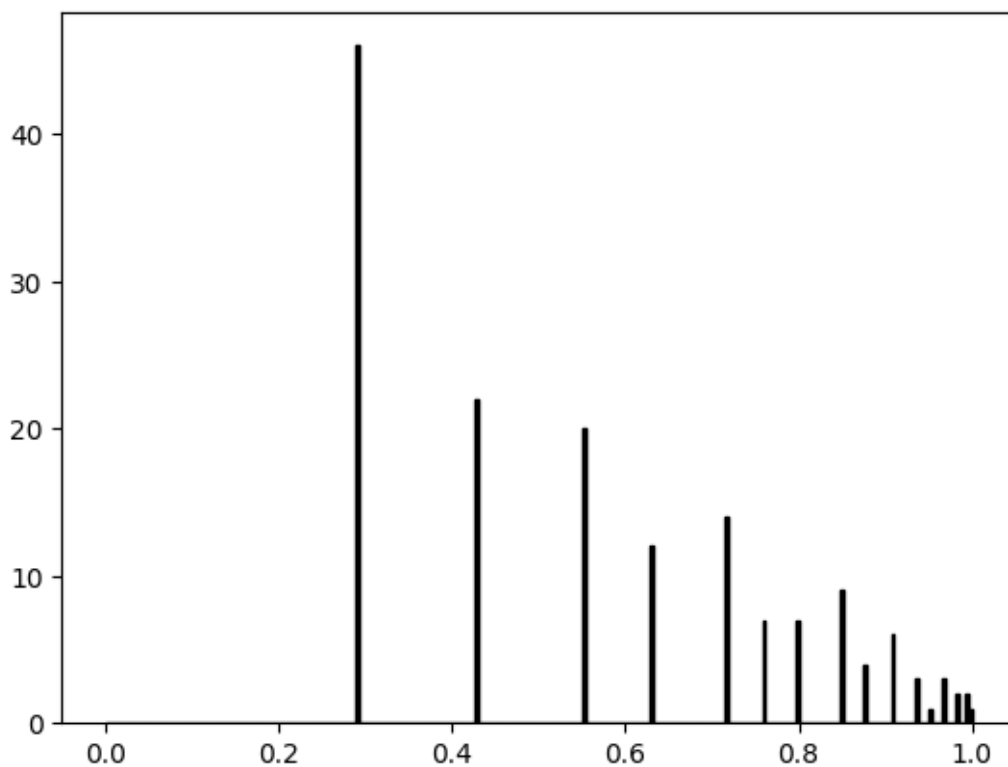
```
list(lc.Trafo)
```

```
[<Trafo.NONE: 0>,
 <Trafo.STANDARDIZE: 1>,
 <Trafo.STANDARDIZE_UINT8: 2>,
 <Trafo.ZERO: 3>,
 <Trafo.ZERO_UINT8: 4>,
 <Trafo.EQUALIZE: 5>,
 <Trafo.EQUALIZE_UINT8: 6>,
 <Trafo.EQUALIZE_ALL: 7>,
 <Trafo.EQUALIZE_ALL_UINT8: 8>,
 <Trafo.EQUALIZE_OP3: 9>,
 <Trafo.EQUALIZE_OP3_UINT8: 10>,
 <Trafo.EQUALIZE_OP3_ALL: 11>,
 <Trafo.EQUALIZE_OP3_ALL_UINT8: 12>]
```

```
img_new = lc.adjust_contrast(img, rescale=lc.Trafo.STANDARDIZE)
epsilon = np.finfo(float).resolution
plt.hist(img_new.ravel(), bins=256, range=(epsilon, 1), fc='k', ec='k');
```

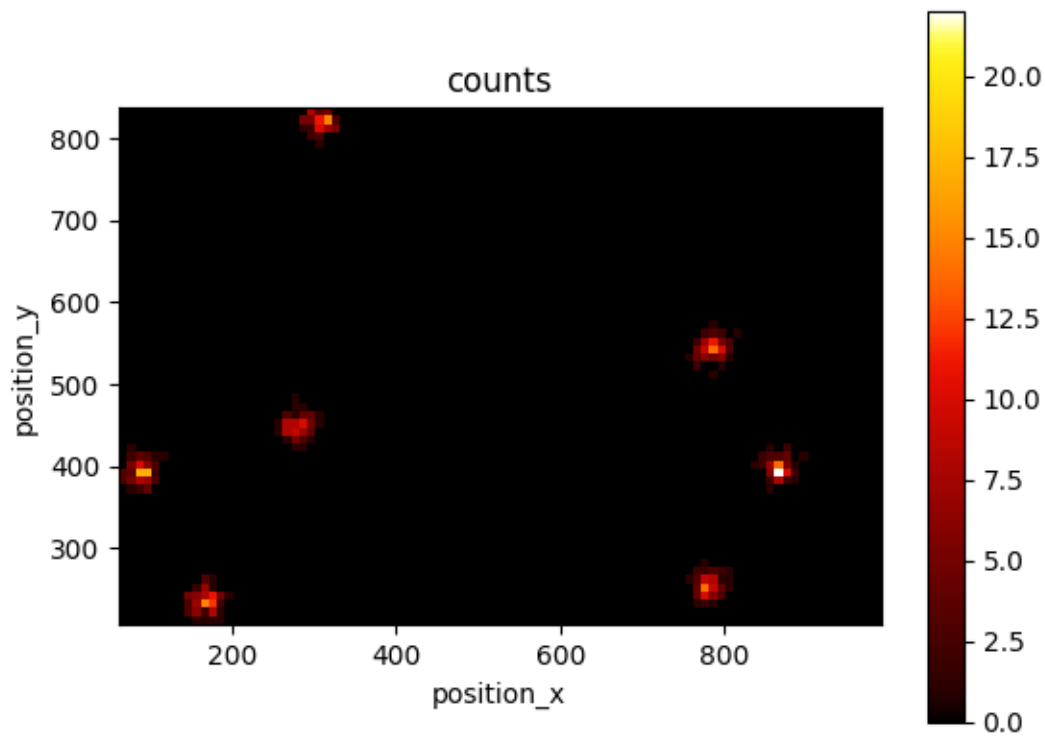


```
norm = lc.HistogramEqualization(power=1, mask=img>0)
img_new = norm(img)
plt.hist(img_new.ravel(), bins=256, range=(epsilon, 1), fc='k', ec='k');
```



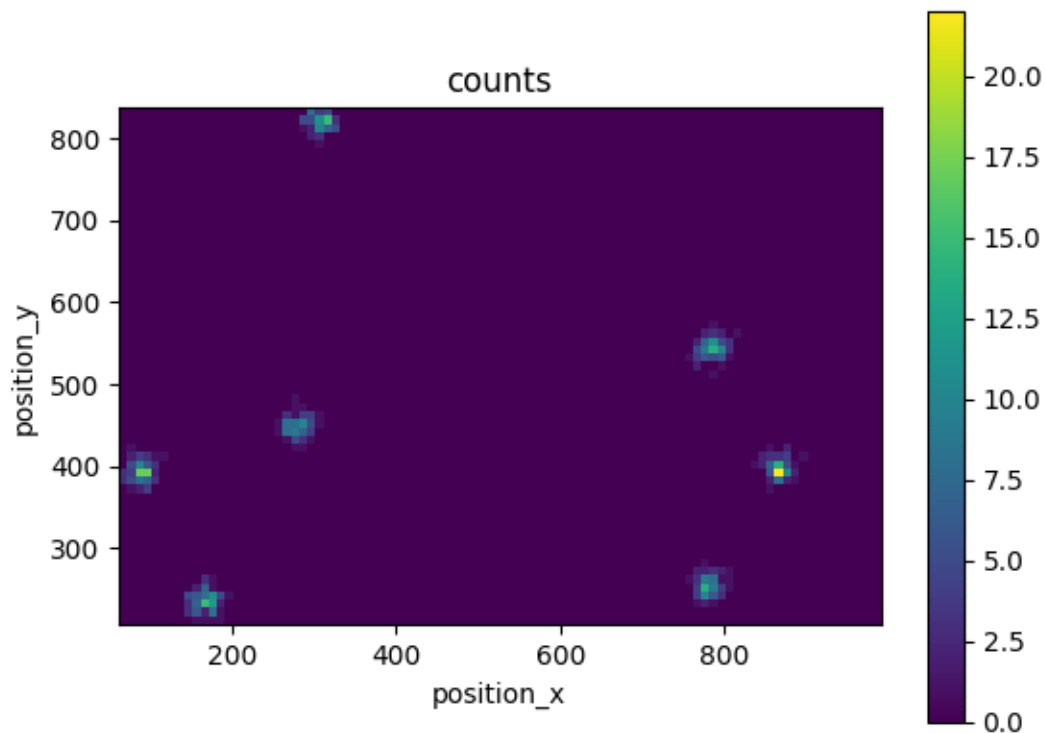
The `render_2d` method can directly provide a new figure as output.

```
lc.render_2d(locdata, bin_size=10);
```



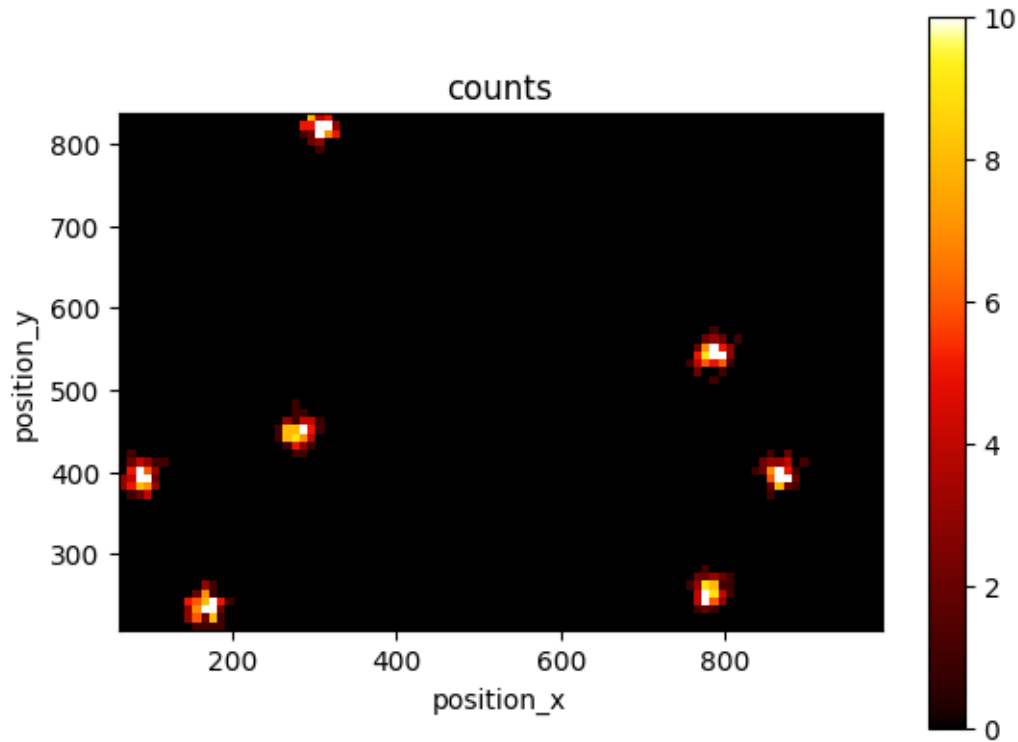
Or it can be used within the matplotlib environment.

```
fig, ax = plt.subplots(nrows=1, ncols=1)
lc.render_2d(locdata, ax = ax, bin_size=10, cmap='viridis')
plt.show()
```



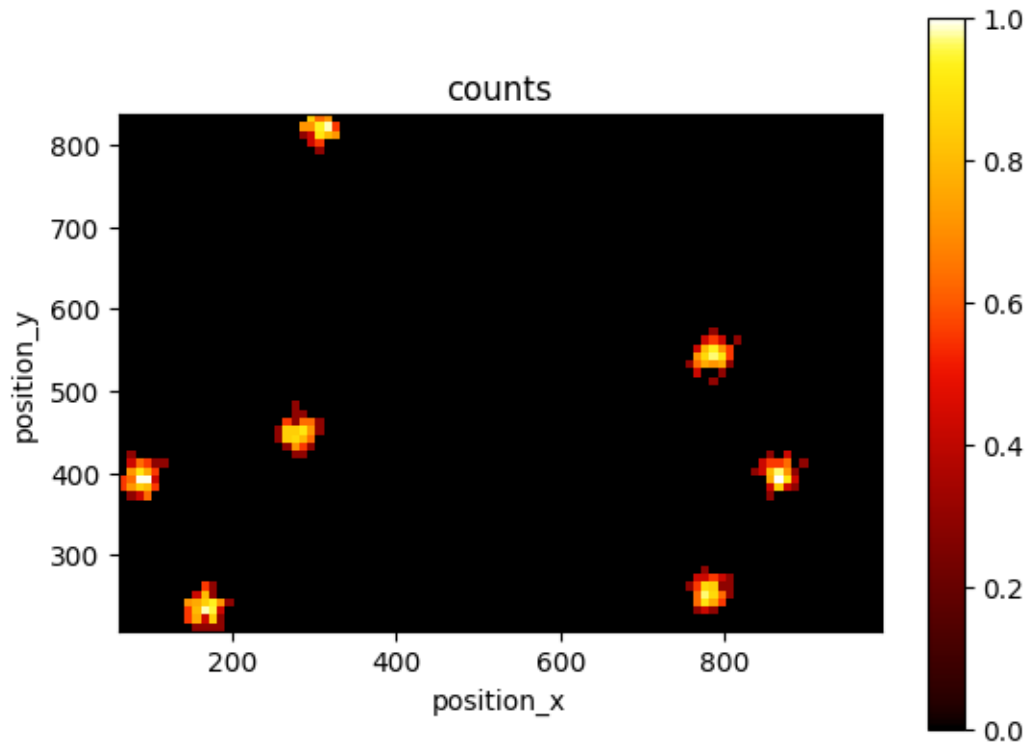
Intensity is per default scaled to the min and max intensity values but can be rescaled by applying any norm function as described for `matplotlib.imshow`:

```
norm = plt.Normalize(vmax=10)
lc.render_2d(locdata, bin_size=10, norm=norm);
```



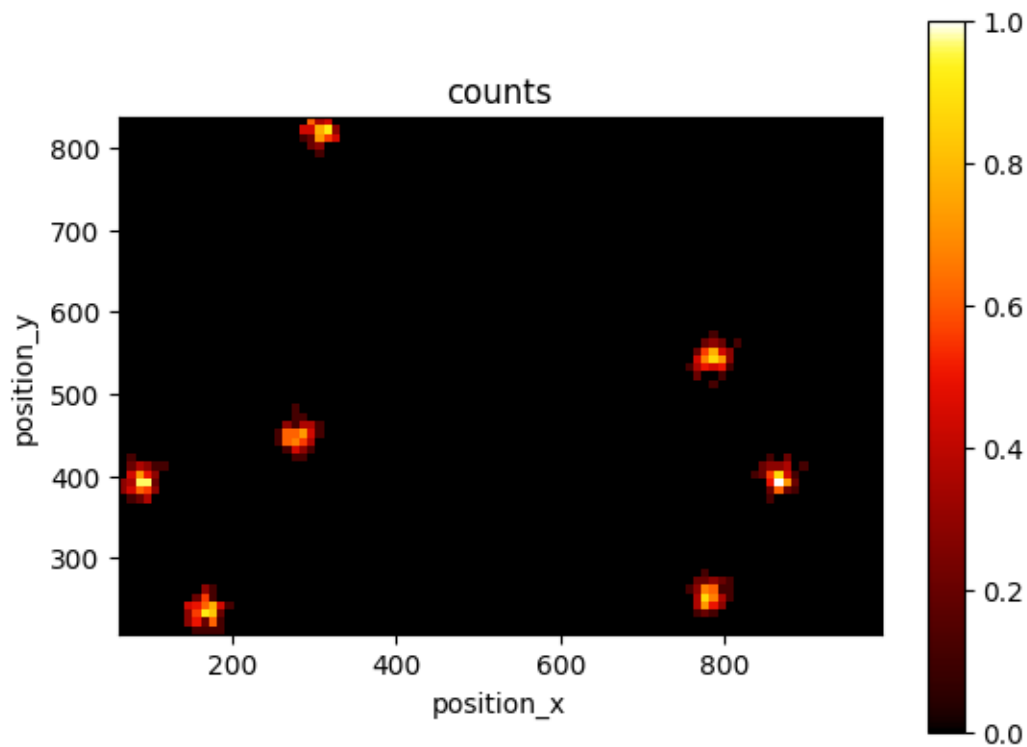
Intensity can also be rescaled by normalization functions as defined in `locan.Trafo`. Histogram equalization yields this image:

```
lc.render_2d(locdata, bin_size=10, rescale=lc.Trafo.EQUALIZE);
```



Histogram equalization with a power-intensification of  $p=0.3$  yields this image:

```
lc.render_2d(locdata, bin_size=10, rescale=lc.Trafo.EQUALIZE_0P3);
```



Any callable normalization object can be passed into the rescale kwarg:

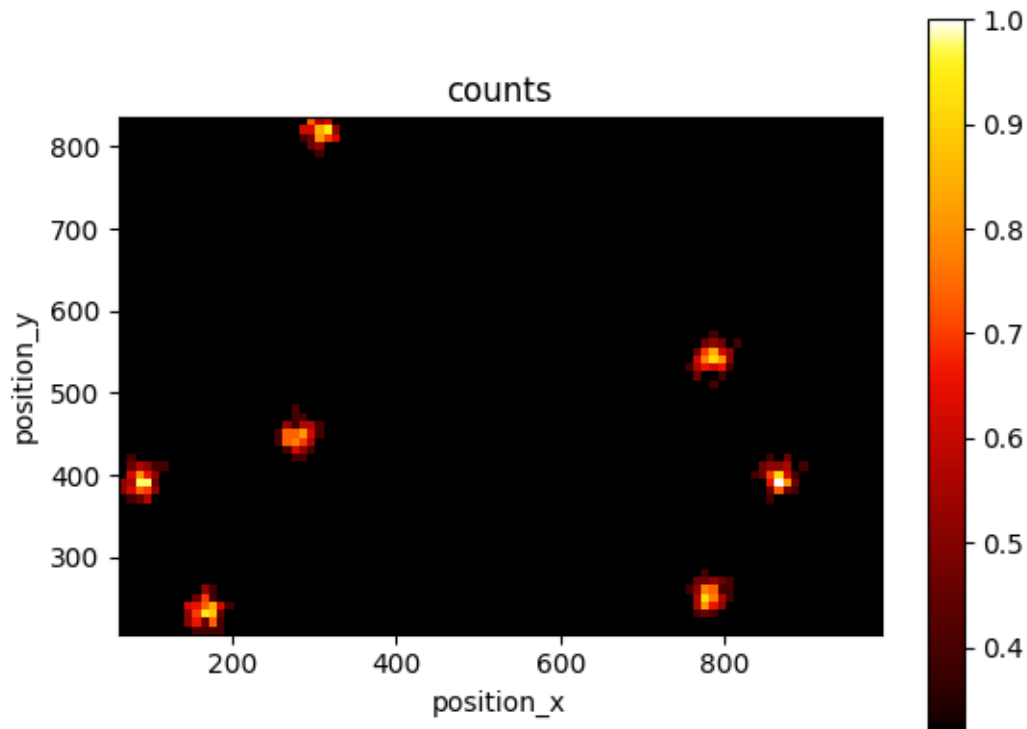
```
norm = lc.HistogramEqualization(power=0.3)
```

(continues on next page)



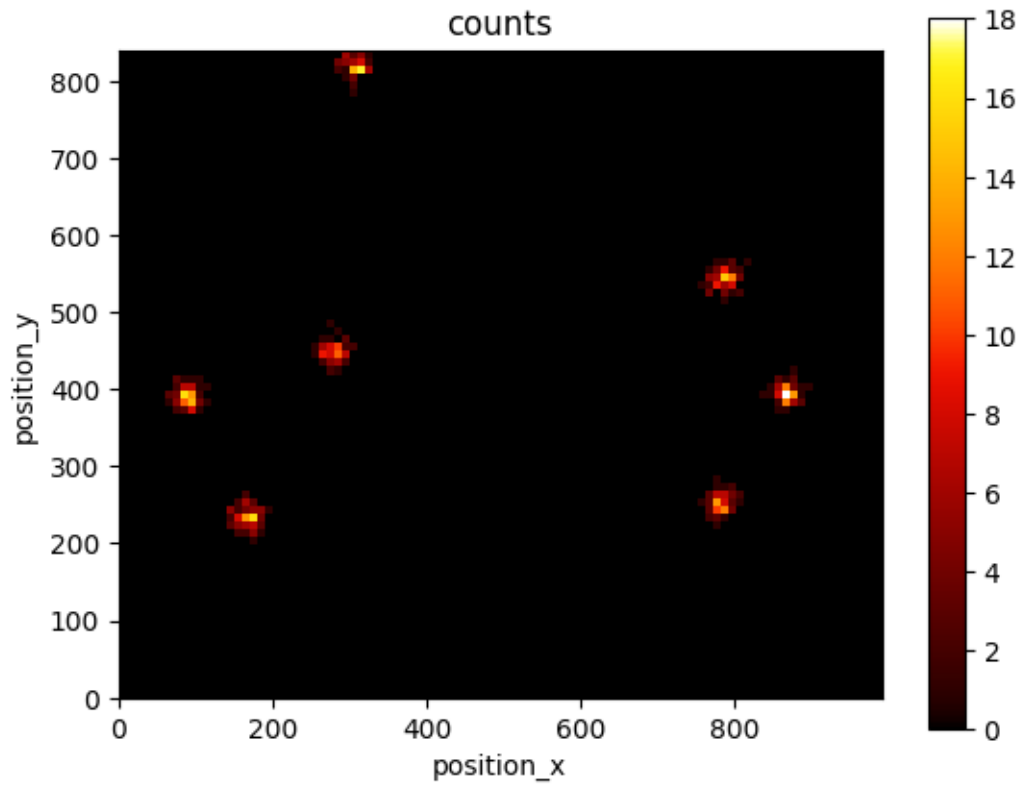
(continued from previous page)

```
lc.render_2d(locdata, bin_size=10, rescale=norm);
```

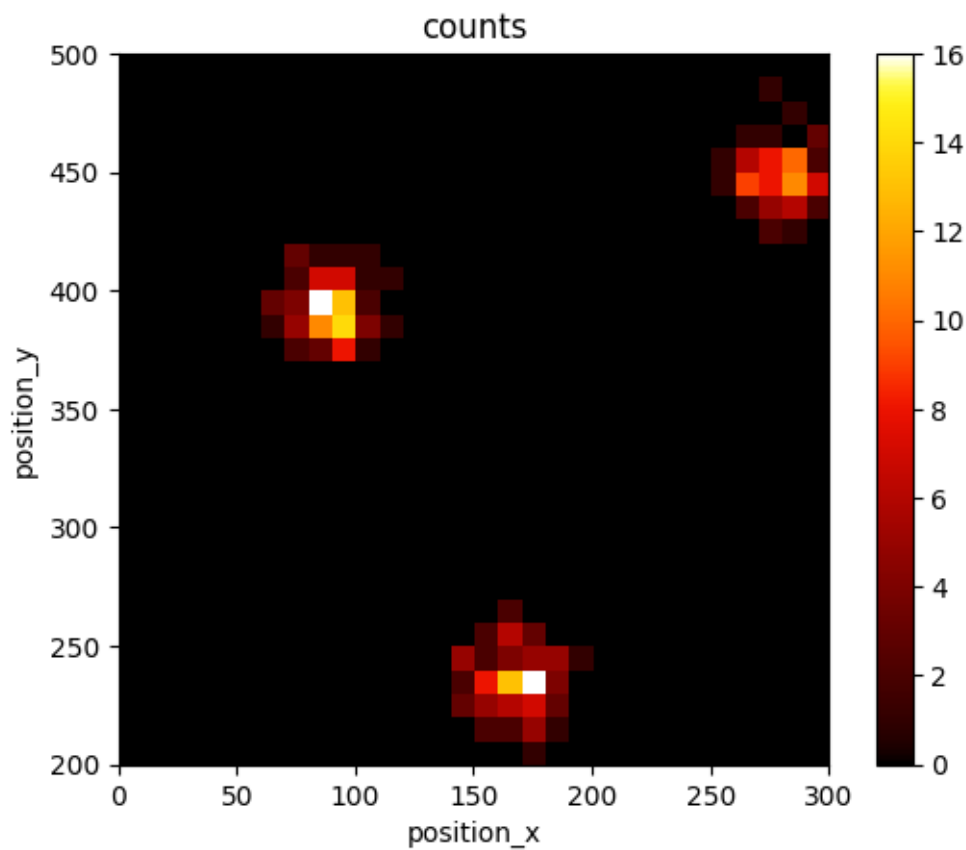


The image size is set automatically to the min and max coordinates but can be set to (0, max) or an arbitrary range.

```
lc.render_2d(locdata, bin_size=10, bin_range='zero');
```



```
lc.render_2d(locdata, bin_size=10, bin_range=((0, 300),(200, 500)));
```



### 2.8.3 Use different libraries for rendering

Rendering can also be carried out with a different render engine. Choose one of the following (MPL is the standard matplotlib):

```
list(lc.RenderEngine)
```

```
[<RenderEngine.MPL: 0>,
 <RenderEngine.MPL_SCATTER_DENSITY: 1>,
 <RenderEngine.NAPARI: 2>]
```

#### napari

As external viewer you can use **napari**.

For napari the qt gui interface has to be activated. Make sure to run the following command manually in a single cell and give enough time for this procedure to complete in the background.

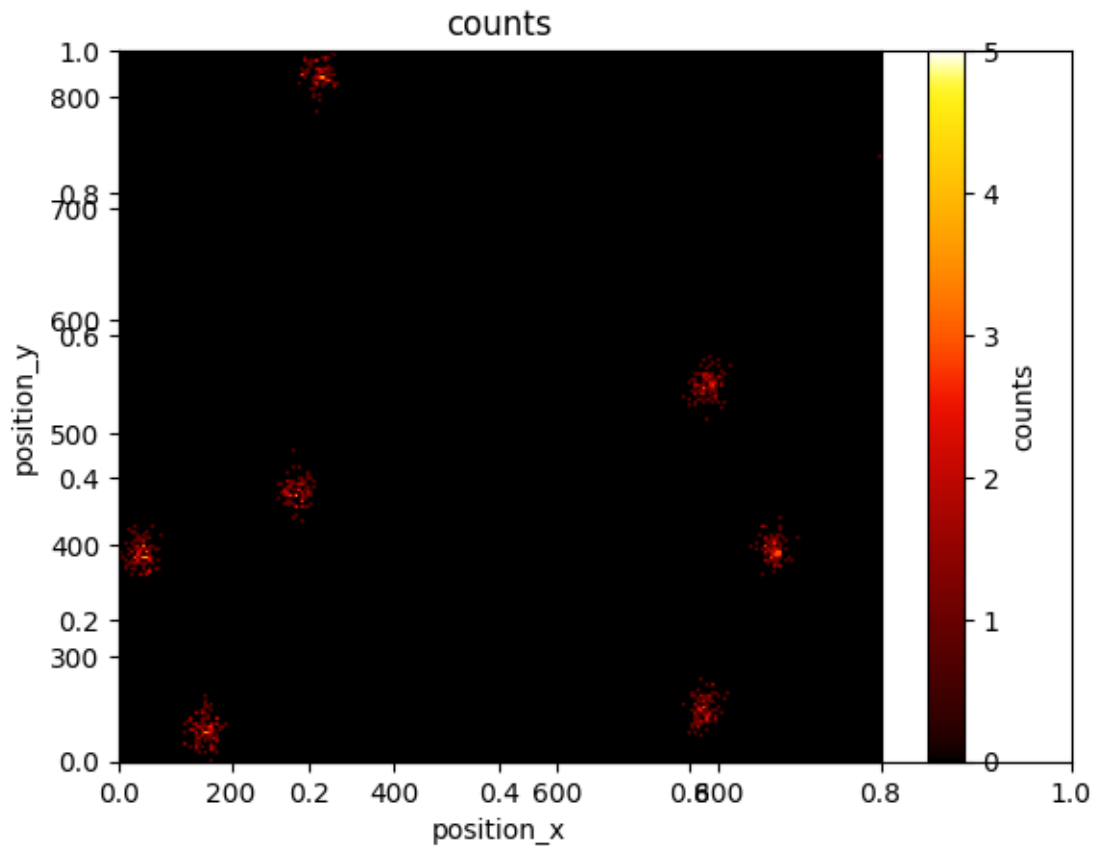
or alternatively:

#### mpl-scatter-density

For interactive viewing with variable binning, **mpl-scatter-density** is helpful. To run this the notebook has to be run with a new kernel applying the magic command `%matplotlib widget` before matplotlib is imported.

```
lc.render_2d(locdata, render_engine=lc.RenderEngine.MPL_SCATTER_DENSITY);
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/mpl_scatter_density/generic_density_artist.py:77:
↳RuntimeWarning: All-NaN slice encountered
  vmin = self._density_vmin(array)
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/mpl_scatter_density/generic_density_artist.py:82:
↳RuntimeWarning: All-NaN slice encountered
  vmax = self._density_vmax(array)
```



## 2.8.4 Choose colormaps

Colormaps can be chosen in matplotlib, napari and other visualization tools.

Colormaps are identified through specific class instances or by a name.

locan.Colormap serves as adapter class for the various visualization tools.

A Colormap instance can be created with the `get_colormap` function:

```
colormap = lc.get_colormap("viridis")
colormap.name
```

```
'viridis'
```

```
colormap.matplotlib
```



A mapping of names on Colormap instances is provided and can be extended by users. If a colormap name is provided to any rendering function, first the `colormap_registry` is searched, then the matplotlib registry and finally napari colormap names.

```
lc.colormap_registry
```

```
{'viridis': <locan.visualize.colormap.Colormap at 0x7f4028ed7750>,
'viridis_r': <locan.visualize.colormap.Colormap at 0x7f4028df7d90>,
'gray': <locan.visualize.colormap.Colormap at 0x7f4028df7d50>,
'gray_r': <locan.visualize.colormap.Colormap at 0x7f4027feab10>,
'turbo': <locan.visualize.colormap.Colormap at 0x7f4027feb190>,
'coolwarm': <locan.visualize.colormap.Colormap at 0x7f4027feacd0>,
'tab20': <locan.visualize.colormap.Colormap at 0x7f4027fead90>,
'cet_fire': <locan.visualize.colormap.Colormap at 0x7f4027feae90>,
'cet_fire_r': <locan.visualize.colormap.Colormap at 0x7f4027feaf50>,
'cet_gray': <locan.visualize.colormap.Colormap at 0x7f4027feb010>,
'cet_gray_r': <locan.visualize.colormap.Colormap at 0x7f4027feb0d0>,
'cet_coolwarm': <locan.visualize.colormap.Colormap at 0x7f4027feb250>,
'cet_glasbey_dark': <locan.visualize.colormap.Colormap at 0x7f4027feb310>}
```

Default colormaps are defined for use with locan and can be accessed through a mapping or an enum:

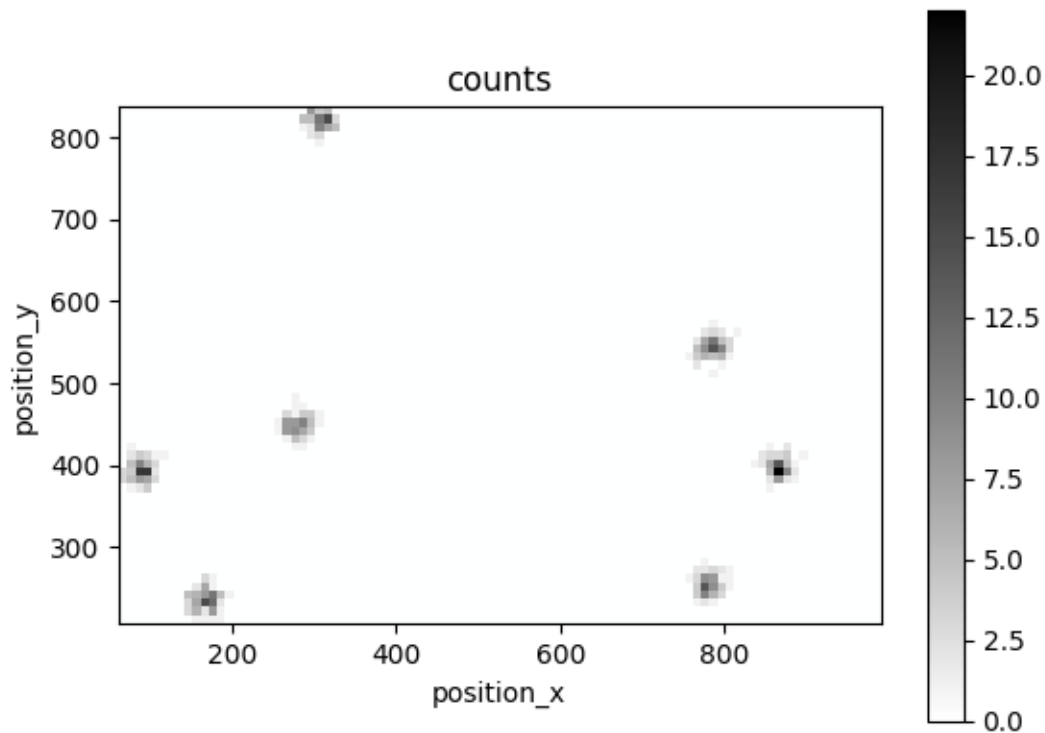
```
lc.COLORMAP_DEFAULTS
```

```
{'CONTINUOUS': 'cet_fire',
'CONTINUOUS_REVERSE': 'cet_fire_r',
'CONTINUOUS_GRAY': 'cet_gray',
'CONTINUOUS_GRAY_REVERSE': 'cet_gray_r',
'DIVERGING': 'cet_coolwarm',
'CATEGORICAL': 'cet_glasbey_dark',
'TURBO': 'turbo'}
```

```
list(lc.Colormaps)
```

```
[<Colormaps.CONTINUOUS: 'cet_fire'>,
<Colormaps.CONTINUOUS_REVERSE: 'cet_fire_r'>,
<Colormaps.CONTINUOUS_GRAY: 'cet_gray'>,
<Colormaps.CONTINUOUS_GRAY_REVERSE: 'cet_gray_r'>,
<Colormaps.DIVERGING: 'cet_coolwarm'>,
<Colormaps.CATEGORICAL: 'cet_glasbey_dark'>,
<Colormaps.TURBO: 'turbo'>]
```

```
lc.render_2d(locdata, bin_size=10, cmap=lc.Colormaps.CONTINUOUS_GRAY_REVERSE);
```



## 2.9 Tutorial about simulating localization data

Locan provides methods for simulating basic localization data sets as LocData objects.

```
from pathlib import Path
import numpy as np
import pandas as pd

%matplotlib inline

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import locan as lc
```

```
/tmp/ipykernel_1984/3554245465.py:3: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳ 54466

import pandas as pd
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

Locan:

version: 0.20.0.dev41+g755b969

Python:

version: 3.11.6

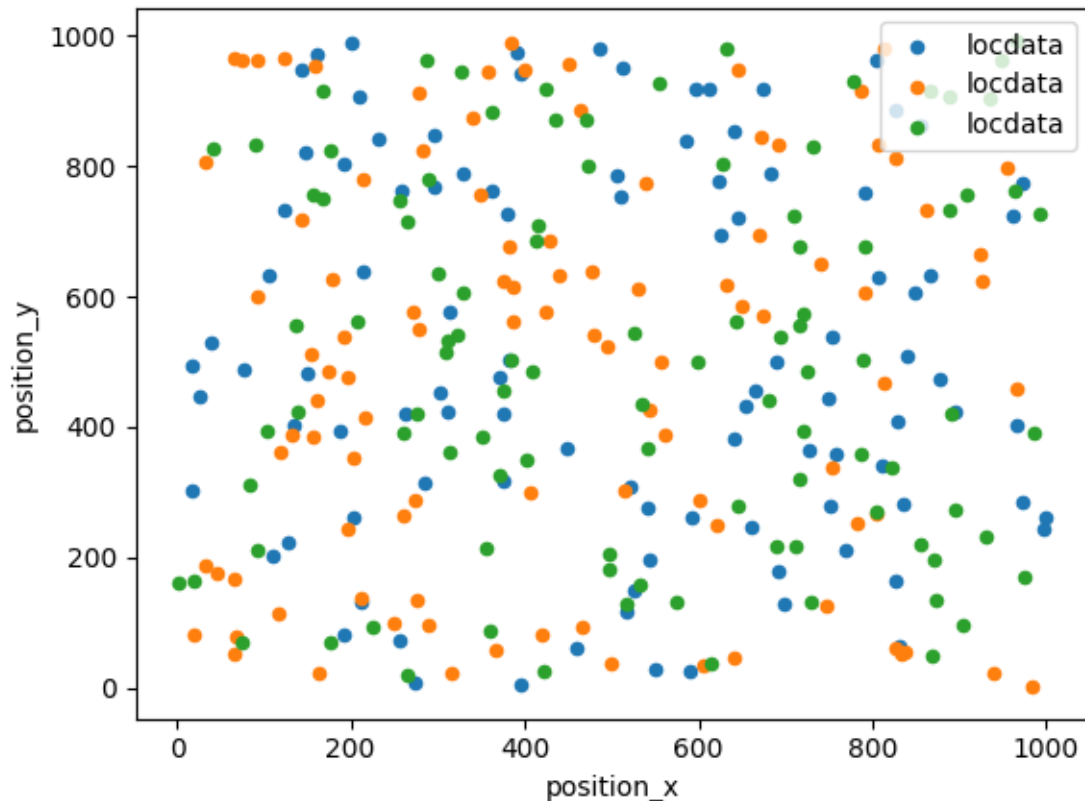
### 2.9.1 Use random number generator

In all simulations we make use of numpy routines for random number generation by instantiating `numpy.random.default_rng` and taking a seed parameter. Therefore, we recommend to set up a random number generator in every script and pass that generator instance to all simulation functions through the seed parameter.

```
rng = np.random.default_rng(seed=1)
locdatas = [lc.simulate_uniform(n_samples=100, region=((0, 1000), (0, 1000)),
↪seed=rng) for i in range(3)]
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
for i, locdata in enumerate(locdatas):
    locdata.data.plot.scatter(x='position_x', y='position_y', color=plt.cm.
↪tab10(i), ax=ax, label='locdata')
plt.show()
```



Make sure to follow the correct procedure for parallel computation as described in the numpy tutorials (<https://numpy.org/doc/stable/reference/random/parallel.html>).

## 2.9.2 Simulate localization data

Point coordinates are distributed on region as specified by Region instances or interval tuples.

### Simulate localization data that follows a uniform distribution

#### in 1d

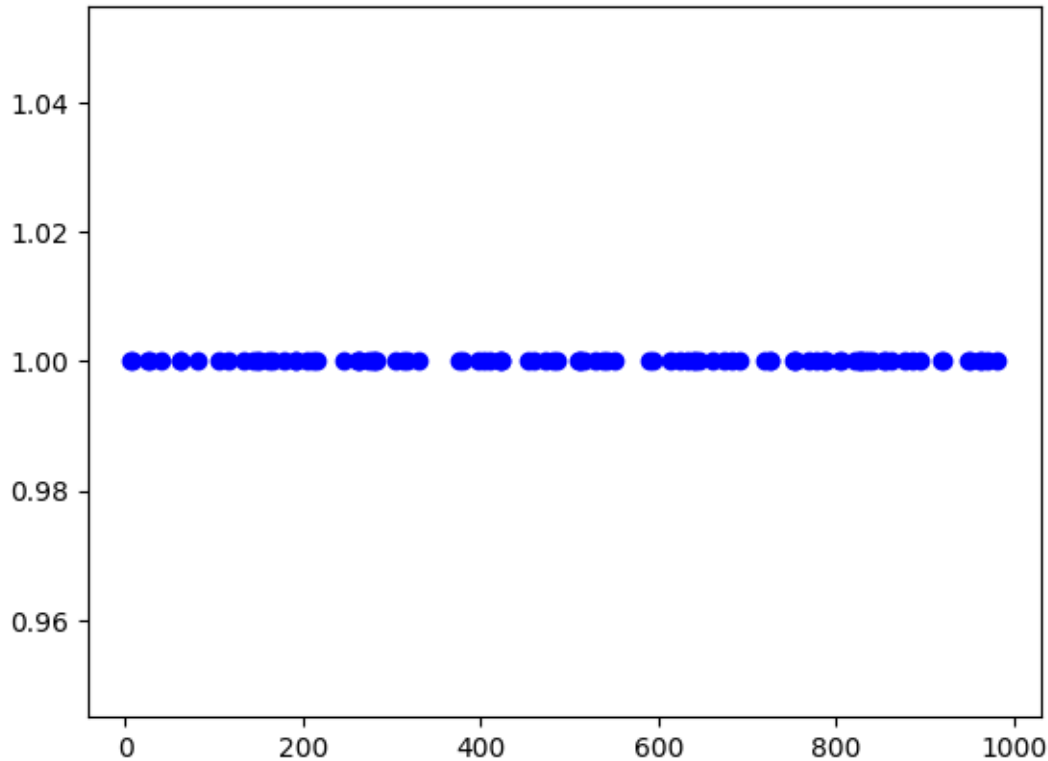
```
locdata = lc.simulate_uniform(n_samples=100, region=lc.data.region.Interval(0,
↪ 1000), seed=1)
```

```
locdata.print_summary()
```

```
identifier: "4"
comment: ""
source: SIMULATION
state: RAW
element_count: 100
frame_count: 0
creation_time {
  2024-03-14T11:09:56.501420Z
}
```



```
fig, ax = plt.subplots(nrows=1, ncols=1)
ax.plot(locdata.data.position_x, [1] * len(locdata), 'o', color='Blue', label=
→ 'locdata')
plt.show()
```



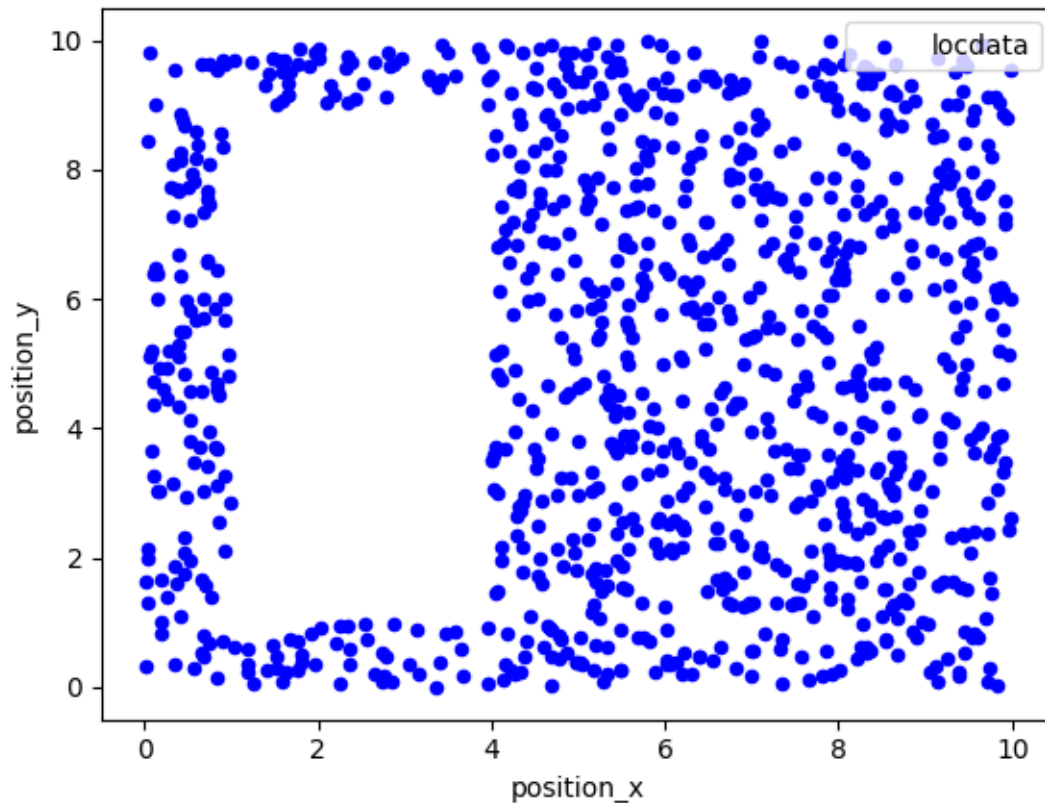
in 2d

```
points = ((0, 0), (0, 10), (10, 10), (10, 0))
holes = [((1, 1), (1, 9), (4, 9), (4, 1))]
region = lc.data.region.Polygon(points, holes)
locdata = lc.simulate_uniform(n_samples=1000, region=region, seed=1)

locdata.print_summary()
```

```
identifier: "5"
comment: ""
source: SIMULATION
state: RAW
element_count: 1000
frame_count: 0
creation_time {
  2024-03-14T11:09:56.776378Z
}
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↪ label='locdata')
plt.show()
```



Simulate localization data that follows a homogeneous (Poisson) distribution

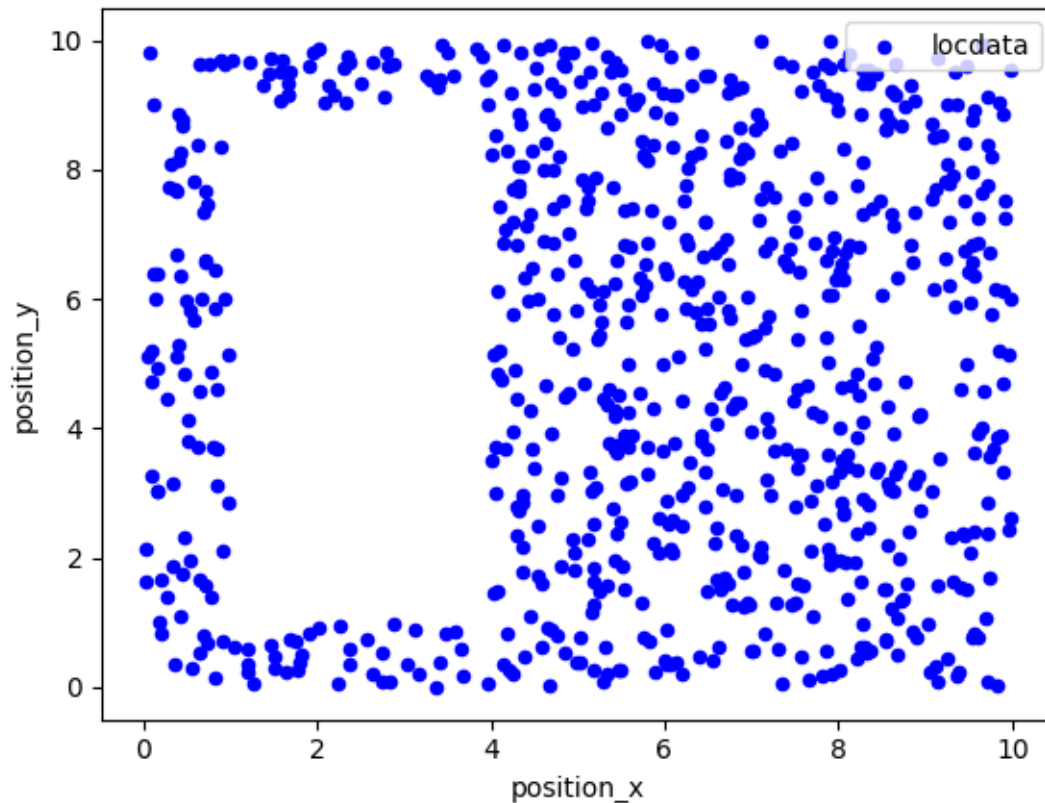
in 2d

```
points = ((0, 0), (0, 10), (10, 10), (10, 0))
holes = [((1, 1), (1, 9), (4, 9), (4, 1))]
region = lc.data.region.Polygon(points, holes)
locdata = lc.simulate_Poisson(intensity=10, region=region, seed=1)

locdata.print_summary()
```

```
identifier: "6"
comment: ""
source: SIMULATION
state: RAW
element_count: 761
frame_count: 0
creation_time {
  2024-03-14T11:09:57.133141Z
}
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↪ label='locdata')
plt.show()
```



in 3d

```
locdata = lc.simulate_Poisson(intensity=1e-4, region=((0, 100), (0, 100), (0, ↪
    ↪100))), seed=1)

locdata.print_summary()
```

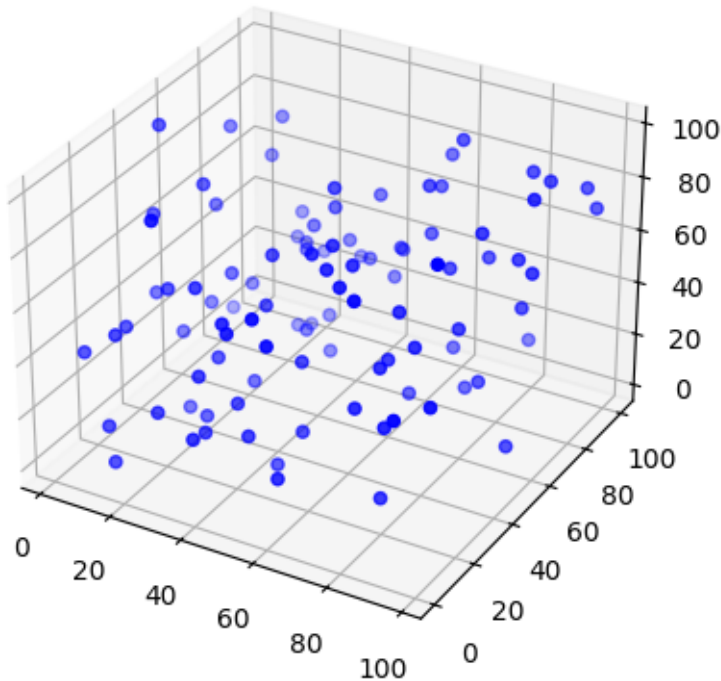
```
identifier: "7"
comment: ""
source: SIMULATION
state: RAW
element_count: 100
frame_count: 0
creation_time {
    2024-03-14T11:09:57.382497Z
}
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

(continues on next page)

(continued from previous page)

```
x,y,z = locdata.coordinates.T
ax.scatter(x, y, z, color='Blue', label='locdata')
plt.show()
```



## Neyman-Scott distribution

In a Neyman-Scott distribution parent events are homogeneously distributed with a certain region and each parent event brings about a number of offspring events distributed around the parent event. For a typical Neyman-Scott process, both the number of parent events and the number of offspring events for each cluster are Poisson distributed. It is important to note that parent events can be outside the support region. For correct simulation, the support region is expanded to distribute parent events and then clipped after offspring substitution.

## Matern distribution

In a Matern process offspring localizations are distributed homogeneously in circles of a given radius around the parent event.

```
locdata = lc.simulate_Matern(parent_intensity=1e-3, region=((0, 100), (0, 100)), cluster_mu=100, radius=10, clip=True, seed=1)
locdata_expanded = lc.simulate_Matern(parent_intensity=1e-3, region=((0, 100), (0, 100)), cluster_mu=100, radius=10, clip=False, seed=1)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
```

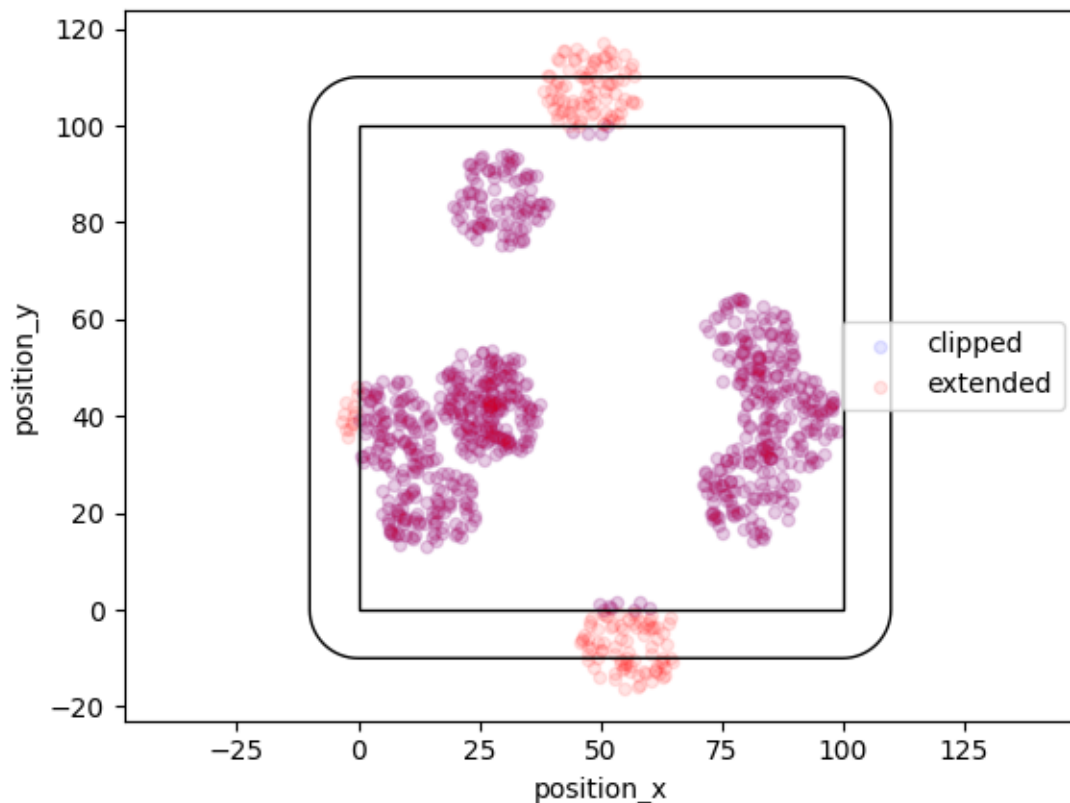
(continues on next page)

(continued from previous page)

```

↪ label='clipped', alpha=0.1)
locdata_expanded.data.plot.scatter(x='position_x', y='position_y', ax=ax,
↪ color='Red', label='extended', alpha=0.1)
ax.add_patch(locdata.region.as_artist(fill=False))
ax.add_patch(locdata_expanded.region.as_artist(fill=False))
ax.axis('equal')
plt.show()

```



More variability can be achieved by specifying arrays for radius.

```

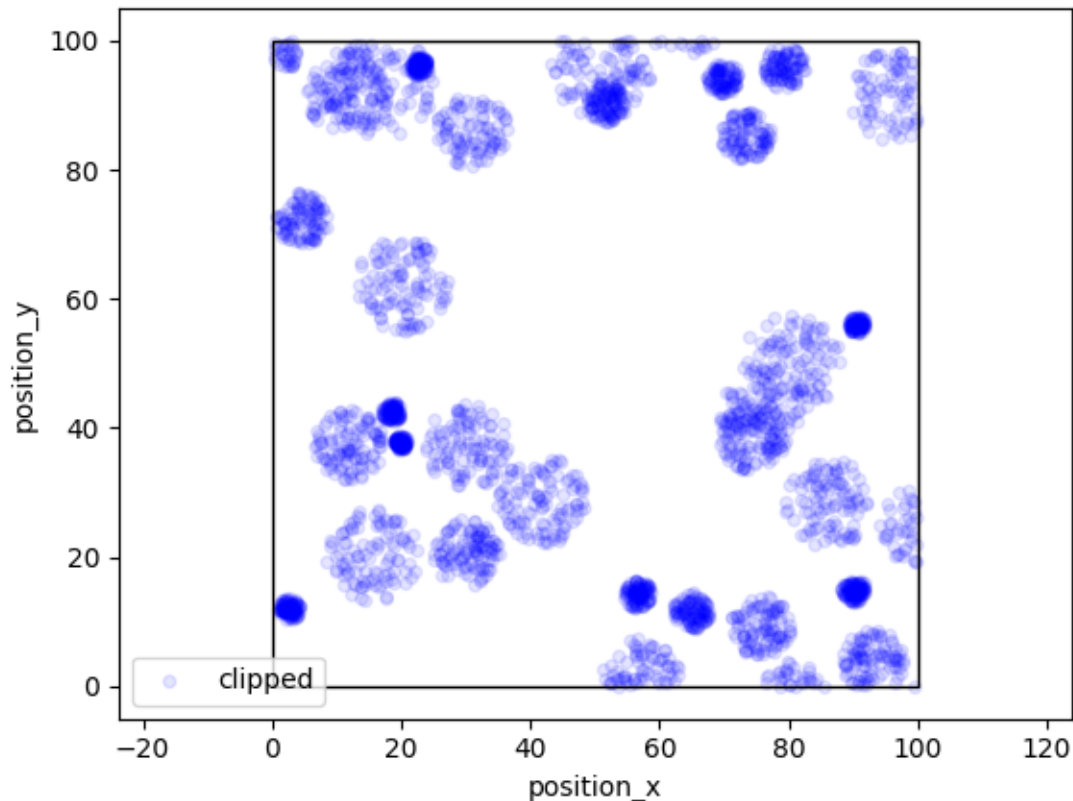
locdata = lc.simulate_Matern(parent_intensity=3e-3, region=((0, 100), (0,
↪ 100)), cluster_mu=100, radius=np.linspace(1, 30, 300), clip=True, seed=1)

```

```

fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
↪ label='clipped', alpha=0.1)
ax.add_patch(locdata.region.as_artist(fill=False))
ax.axis('equal')
plt.show()

```

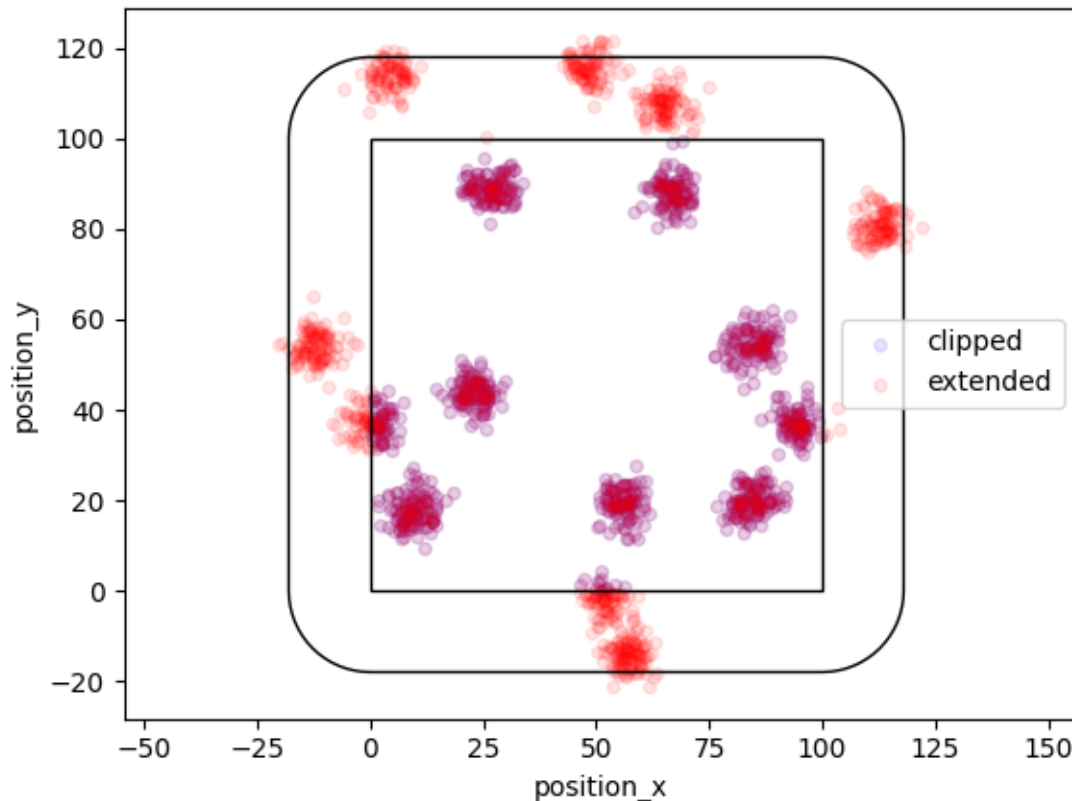


### Thomas distribution

In a Thomas process offspring localizations follow a normal distribution with center being the parent event and a given standard deviation. Here the region is expanded by a distance that equals `cluster_std * expansion_factor`.

```
locdata = lc.simulate_Thomas(parent_intensity=1e-3, region=((0, 100), (0, 100)), cluster_mu=100, cluster_std=3, clip=True, seed=1)
locdata_expanded = lc.simulate_Thomas(parent_intensity=1e-3, region=((0, 100), (0, 100)), cluster_mu=100, cluster_std=3, clip=False, seed=1)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue', label='clipped', alpha=0.1)
locdata_expanded.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Red', label='extended', alpha=0.1)
ax.add_patch(locdata.region.as_artist(fill=False))
ax.add_patch(locdata_expanded.region.as_artist(fill=False))
ax.axis('equal')
plt.show()
```



More variability can be achieved by specifying arrays of `cluster_mu` or `cluster_std`.

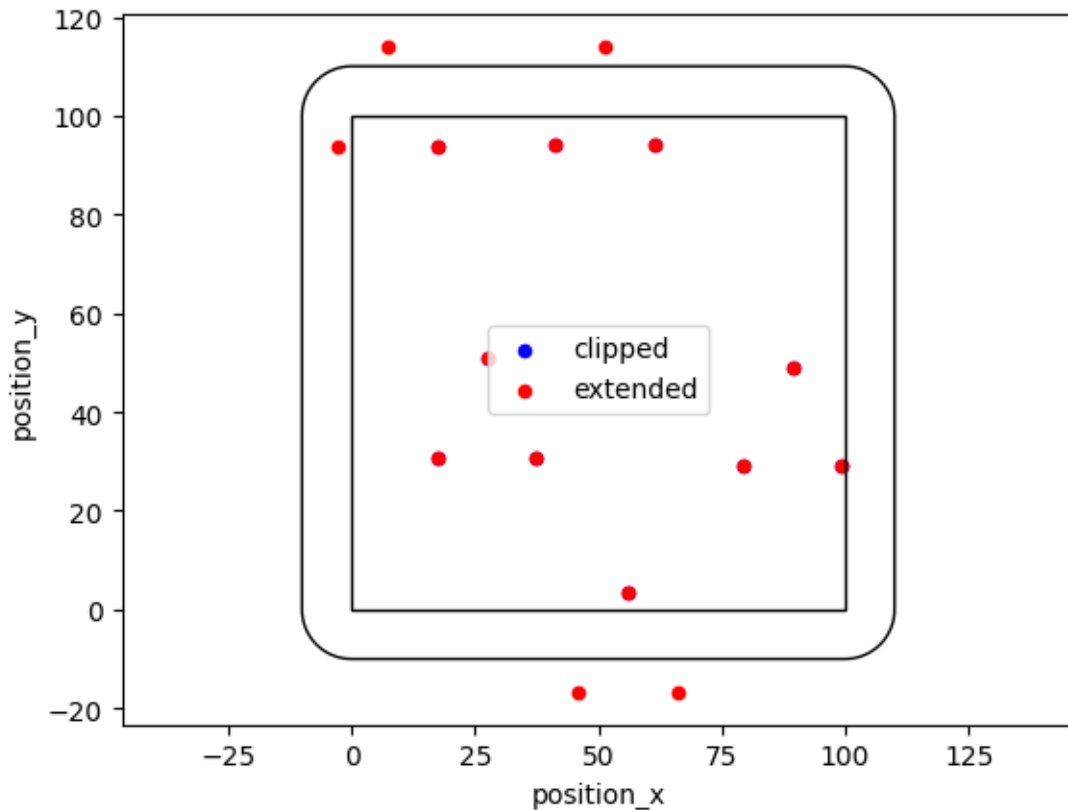
## Cluster distribution

If you need a fixed number of samples, use `simulate_cluster` and specify arbitrary offspring distributions.

```
offspring_points = [((-10, -10), (0, 10), (10, -10))] * 5

locdata = lc.simulate_cluster(centers=5, region=((0, 100), (0, 100)),
    ↳ expansion_distance=10, offspring=offspring_points, clip=True, seed=1)
locdata_expanded = lc.simulate_cluster(centers=5, region=((0, 100), (0, 100)),
    ↳ expansion_distance=10, offspring=offspring_points, clip=False, seed=1)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↳ label='clipped')
locdata_expanded.data.plot.scatter(x='position_x', y='position_y', ax=ax,
    ↳ color='Red', label='extended')
ax.add_patch(locdata.region.as_artist(fill=False))
ax.add_patch(locdata_expanded.region.as_artist(fill=False))
ax.axis('equal')
plt.show()
```

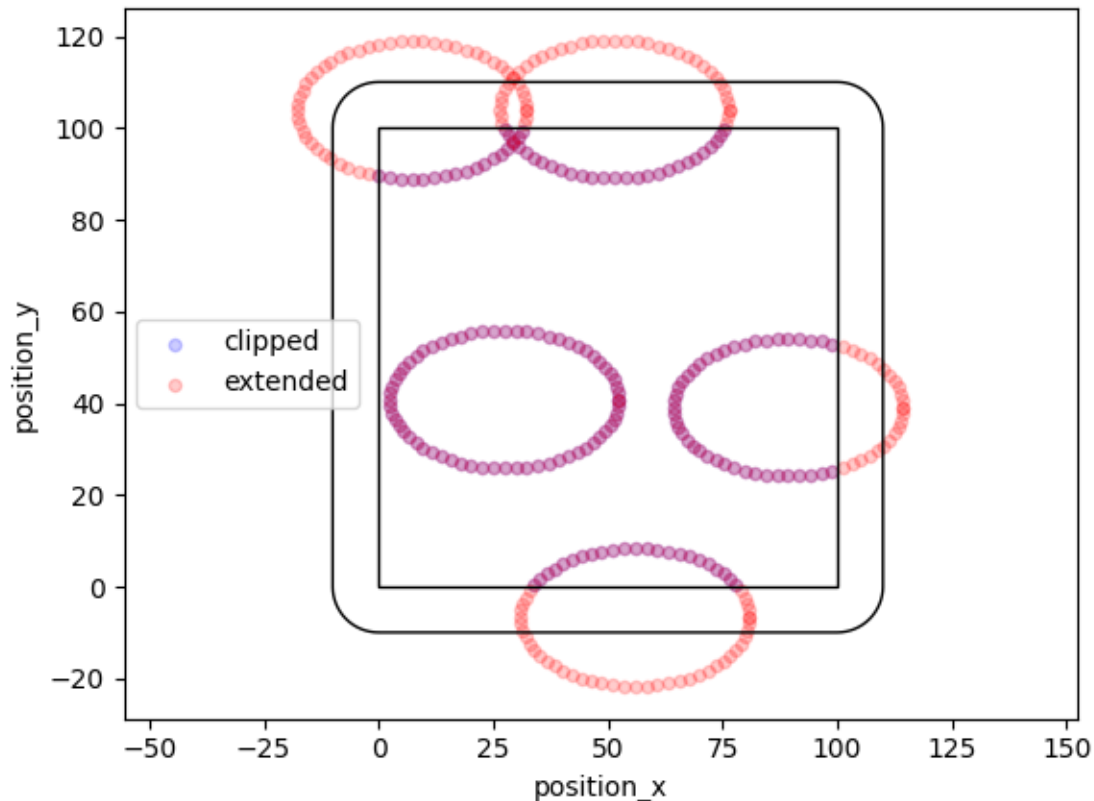


```
def offspring_points(parent):
    angles = np.linspace(0, 360, 36)
    for angle in angles:
        circle = lc.data.region.Ellipse(parent, 50, 30, angle)
    return circle.points

locdata = lc.simulate_cluster(centers=5, region=((0, 100), (0, 100)),
    ↳ expansion_distance=10, offspring=offspring_points, clip=True, seed=1)
locdata_expanded = lc.simulate_cluster(centers=5, region=((0, 100), (0, 100)),
    ↳ expansion_distance=10, offspring=offspring_points, clip=False, seed=1)
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↳ label='clipped', alpha=0.2)
locdata_expanded.data.plot.scatter(x='position_x', y='position_y', ax=ax,
    ↳ color='Red', label='extended', alpha=0.2)
ax.add_patch(locdata.region.as_artist(fill=False))
ax.add_patch(locdata_expanded.region.as_artist(fill=False))
ax.axis('equal')
plt.show()
```





### 2.9.3 Resample data

The resample function provides additional localizations for each given localizations that are Gauss distributed around the original localizations with a standard deviation given by the `uncertainty_x` property.

```
nrg = np.random.default_rng(seed=1)
n_samples = 10
dat = lc.simulate_uniform(n_samples=n_samples, region=((0, 1000), (0, 1000)),
    ↪seed=nrg)
dat.dataframe = dat.dataframe.assign(uncertainty_x= 20*nrg.random(n_samples))
dat.dataframe = dat.dataframe.assign(uncertainty_y= 20*nrg.random(n_samples))
```

```
dat_resampled = lc.resample(dat, n_samples=1000, seed=nrg)
dat_resampled.data.tail()
```

|      | original_index | position_x | position_y | uncertainty_x | uncertainty_y |
|------|----------------|------------|------------|---------------|---------------|
| 9995 | 9              | 85.522771  | 535.121307 | 16.728111     | 10.264751     |
| 9996 | 9              | 117.665446 | 532.110220 | 16.728111     | 10.264751     |
| 9997 | 9              | 78.991916  | 518.541451 | 16.728111     | 10.264751     |
| 9998 | 9              | 103.053219 | 543.823520 | 16.728111     | 10.264751     |
| 9999 | 9              | 96.072240  | 510.063092 | 16.728111     | 10.264751     |

```
fig, ax = plt.subplots(nrows=1, ncols=1)
dat_resampled.data.plot.scatter(x='position_x', y='position_y', ax=ax, color=
```

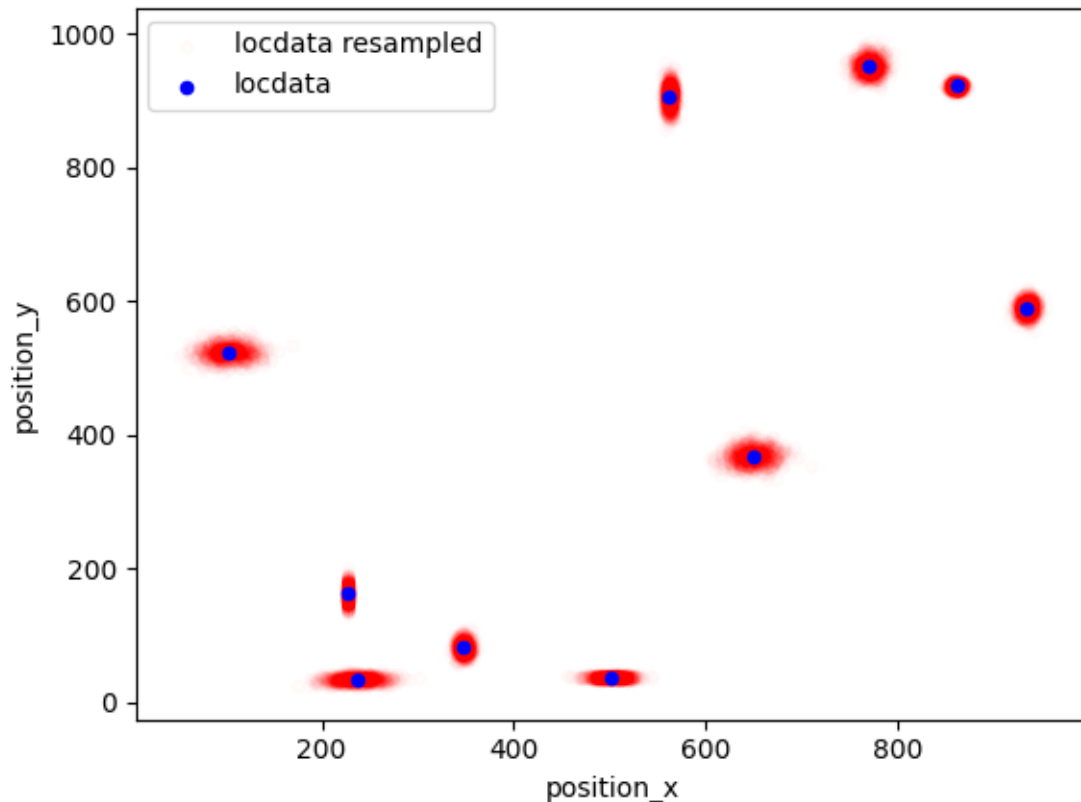
(continues on next page)

(continued from previous page)

```

↪ 'Red', label='locdata resampled', alpha=0.01)
dat.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
↪ label='locdata')
plt.show()

```



## 2.10 Tutorial about tracking LocData objects

Tracking refers to link localizations that are close in space over multiple frames and collect those localizations in individual tracks. We here make use of the trackpy package through wrapper functions to deal with LocData objects.

```

%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import locan as lc

```

```

/tmp/ipykernel_2021/3049569888.py:4: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release
↪ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↪ better interoperability with other libraries)

```

(continues on next page)

(continued from previous page)

but was not found to be installed on your system.  
 If this would cause problems for you,  
 please provide us feedback at <https://github.com/pandas-dev/pandas/issues/>  
 ↪ 54466

```
import pandas as pd
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

Locan:  
 version: 0.20.0.dev41+g755b969

Python:  
 version: 3.11.6

## 2.10.1 Synthetic data

A random dataset is created.

```
dat = lc.simulate_tracks(n_walks=5, n_steps=100, ranges=((0,1000),(0,1000)),  
                        diffusion_constant=1, seed=1)
```

```
dat.print_meta()
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.  

[Open3D INFO] WebRTC GUI backend enabled.  

[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.  

identifier: "1"  

source: SIMULATION  

state: RAW  

history {  

  name: "simulate_tracks"  

  parameter: "{\'n_walks\': 5, \'n_steps\': 100, \'ranges\': ((0, 1000), (0, ↪  

  ↪ 1000)), \'diffusion_constant\': 1, \'time_step\': 10, \'seed\': 1}"  

}  

element_count: 500  

frame_count: 100  

creation_time {  

  2024-03-14T11:10:03.274708Z  

}
```

```
dat.data.head()
```

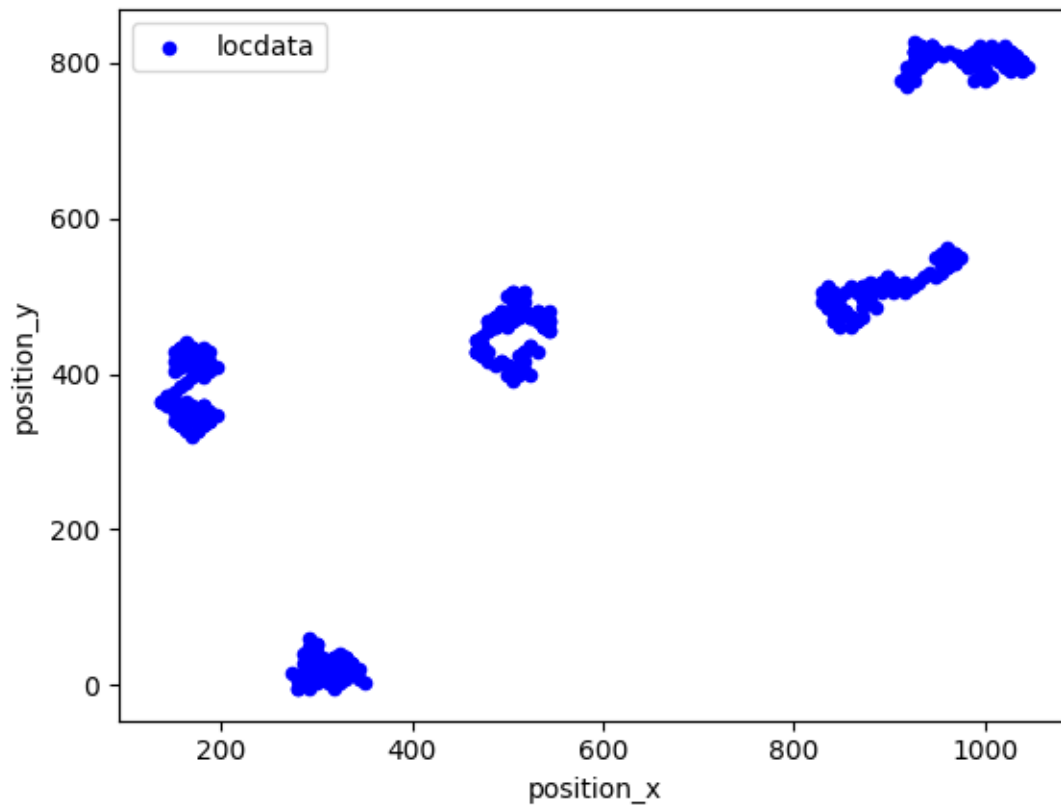
|   | position_x | position_y | frame |
|---|------------|------------|-------|
| 0 | 518.146180 | 429.651004 | 0     |
| 1 | 524.470735 | 435.975560 | 1     |
| 2 | 530.795291 | 429.651004 | 2     |

(continues on next page)

(continued from previous page)

```
3 524.470735 435.975560 3
4 518.146180 429.651004 4
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
dat.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
                      label='locdata')
plt.show()
```



## 2.10.2 Track locdata

The track function collects tracks in a new locdata object.

```
tracks, track_numbers = lc.track(dat, search_range=500)
```

```
Frame 99: 5 trajectories present.
```

```
tracks.print_summary()
```

```
identifier: "7"
comment: ""
source: DESIGN
state: RAW
element_count: 5
frame_count: 1
```

(continues on next page)

(continued from previous page)

```
creation_time {
  2024-03-14T11:10:03.783495Z
}
```

```
tracks.data
```

|   | localization_count | position_x | uncertainty_x | position_y | uncertainty_y | \ |
|---|--------------------|------------|---------------|------------|---------------|---|
| 0 | 100                | 882.368107 | 4.321878      | 504.562854 | 2.382224      |   |
| 1 | 100                | 169.710816 | 1.293090      | 381.371093 | 3.679042      |   |
| 2 | 100                | 505.497069 | 2.098581      | 456.720101 | 3.047211      |   |
| 3 | 100                | 307.151281 | 1.787996      | 19.463682  | 1.378273      |   |
| 4 | 100                | 972.599640 | 3.900272      | 805.187177 | 1.217117      |   |

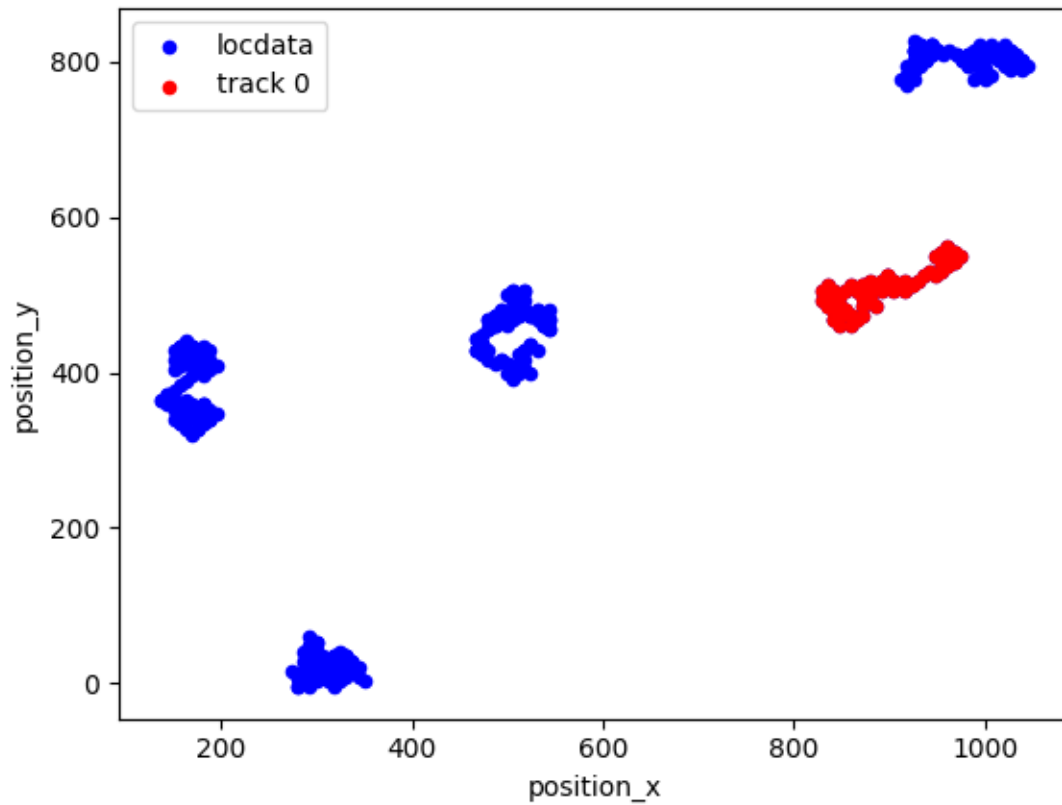
  

|   | frame | region_measure_bb | localization_density_bb | subregion_measure_bb |
|---|-------|-------------------|-------------------------|----------------------|
| 0 | 0     | 14720.0           | 0.006793                | 493.315315           |
| 1 | 0     | 6840.0            | 0.014620                | 354.175098           |
| 2 | 0     | 8640.0            | 0.011574                | 379.473319           |
| 3 | 0     | 4800.0            | 0.020833                | 278.280434           |
| 4 | 0     | 7560.0            | 0.013228                | 379.473319           |

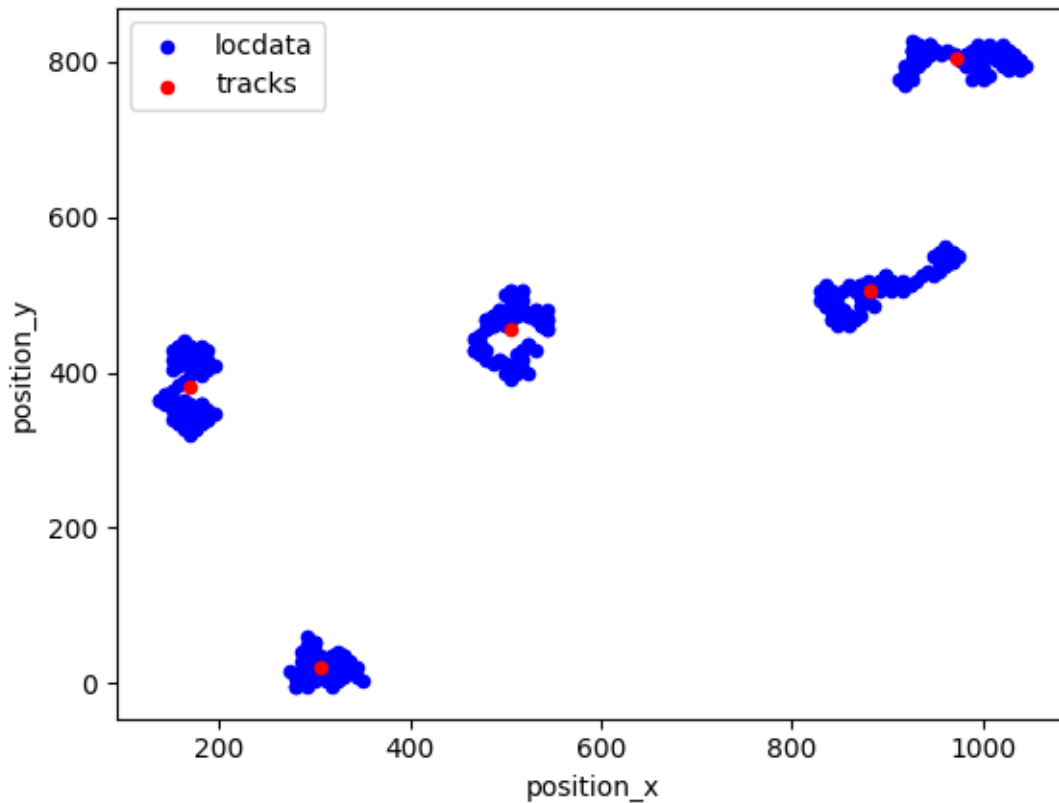
```
tracks.references[0].data.head()
```

|     | position_x | position_y | frame |
|-----|------------|------------|-------|
| 300 | 954.974002 | 543.269132 | 0     |
| 301 | 961.298558 | 549.593688 | 1     |
| 302 | 967.623113 | 555.918243 | 2     |
| 303 | 961.298558 | 549.593688 | 3     |
| 304 | 967.623113 | 543.269132 | 4     |

```
fig, ax = plt.subplots(nrows=1, ncols=1)
dat.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↪label='locdata')
tracks.references[0].data.plot.scatter(x='position_x', y='position_y', ax=ax,
    ↪color='Red', label='track 0')
plt.show()
```



```
fig, ax = plt.subplots(nrows=1, ncols=1)
dat.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
    ↪label='locdata')
tracks.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Red',
    ↪label='tracks')
plt.show()
```



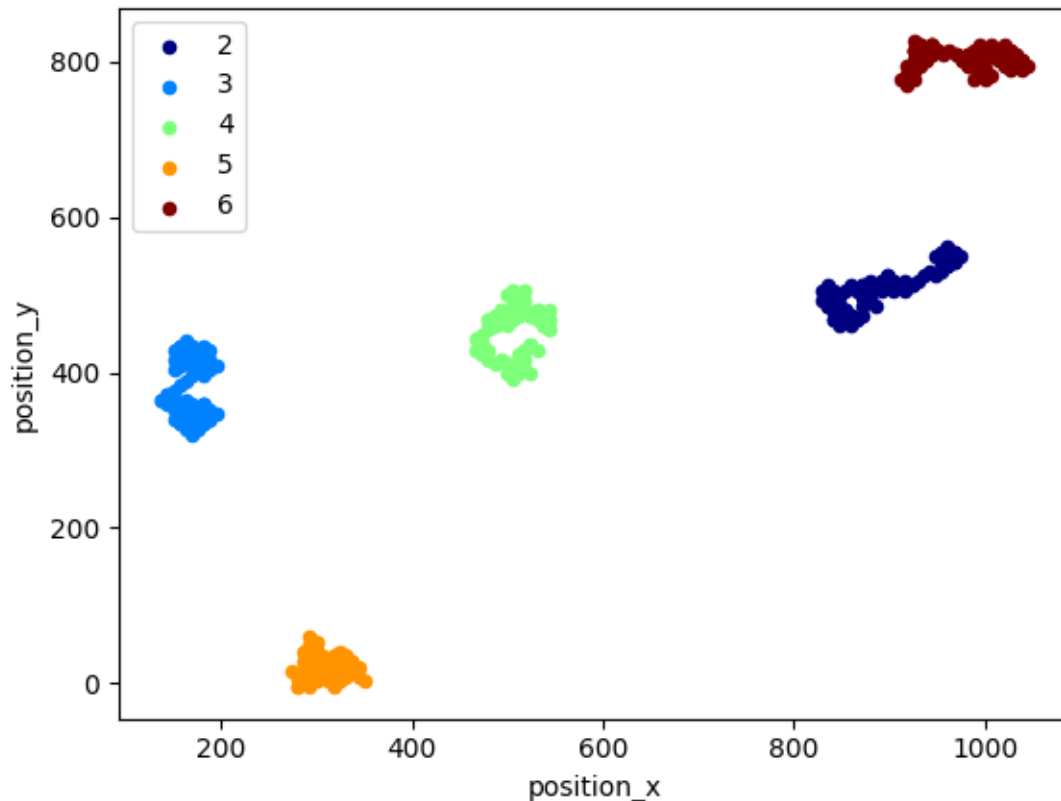
To show individual tracks make use of the reference attribute.

```
fig, ax = plt.subplots(nrows=1, ncols=1)

jet= plt.get_cmap('jet')
colors = iter(jet(np.linspace(0, 1, len(tracks))))

for ref in tracks.references:
    c=next(colors)
    ref.data.plot.scatter(x='position_x', y='position_y', ax=ax, color=(c,),
        label=ref.meta.identifier)

plt.show()
```



Alternatively to a new locdata object, a pandas DataFrame can be generated using the `link_locdata` method.

```
links = lc.link_locdata(dat, search_range=10)
```

```
Frame 99: 5 trajectories present.
```

```
links.head()
```

```
300    0
200    1
0      2
400    3
100    4
Name: track, dtype: int64
```

Use the following to add particle column to original locdata dataset.

```
dat.data.loc[links.index, 'track']=links
```

```
dat.data.head()
```

```
   position_x  position_y  frame  track
0  518.146180  429.651004      0    2.0
1  524.470735  435.975560      1    2.0
2  530.795291  429.651004      2    2.0
```

(continues on next page)



(continued from previous page)

|   |            |            |   |     |
|---|------------|------------|---|-----|
| 3 | 524.470735 | 435.975560 | 3 | 2.0 |
| 4 | 518.146180 | 429.651004 | 4 | 2.0 |

### 2.10.3 Track using trackpy with locdata.data as input

For a detailed tracking analysis you might want to use `trackpy` functions with the pandas DataFrame carrying localization data as input. The following examples will give a short illustration of using `trackpy` functions.

```
import trackpy as tp
```

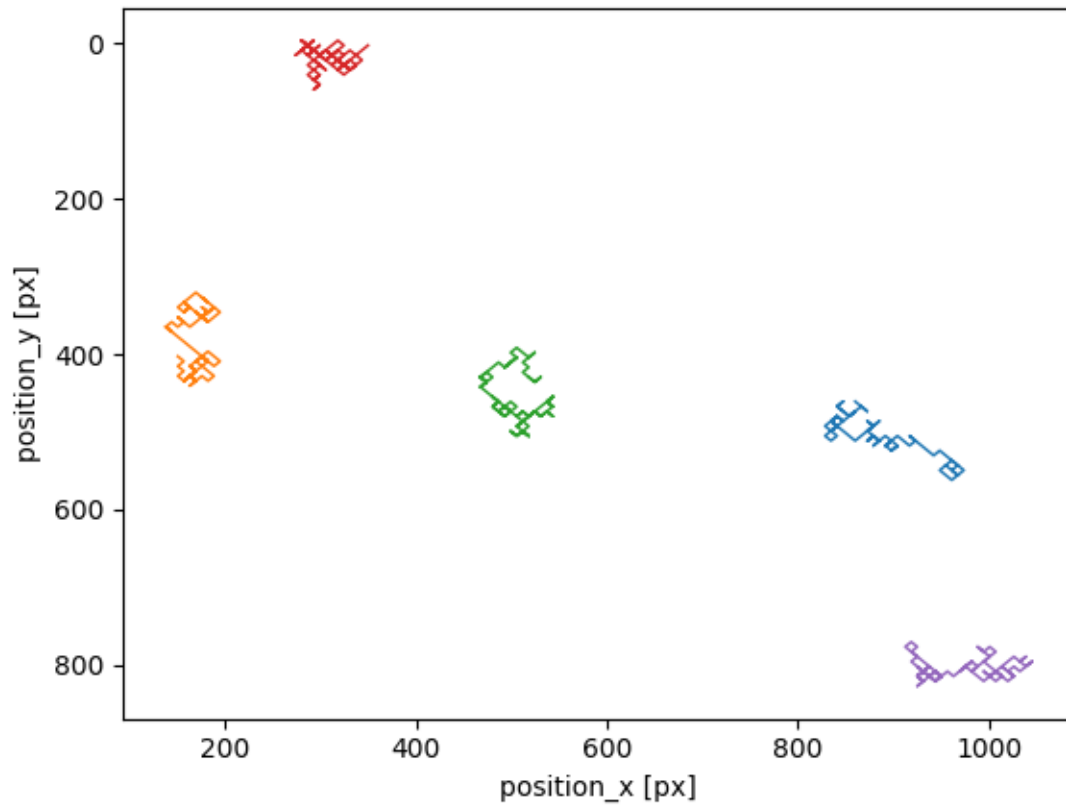
```
t = tp.link_df(dat.data, search_range=100, memory=1, pos_columns=['position_x', 'position_y'], t_column='frame')
```

```
t.head(10)
```

Frame 99: 5 trajectories present.

|     | position_x | position_y | frame | track | particle |
|-----|------------|------------|-------|-------|----------|
| 300 | 954.974002 | 543.269132 | 0     | 0.0   | 0        |
| 200 | 150.484168 | 402.874581 | 0     | 1.0   | 1        |
| 0   | 518.146180 | 429.651004 | 0     | 2.0   | 2        |
| 400 | 318.156007 | 33.883669  | 0     | 3.0   | 3        |
| 100 | 944.139141 | 821.378039 | 0     | 4.0   | 4        |
| 1   | 524.470735 | 435.975560 | 1     | 2.0   | 2        |
| 201 | 156.808723 | 409.199136 | 1     | 1.0   | 1        |
| 301 | 961.298558 | 549.593688 | 1     | 0.0   | 0        |
| 401 | 324.480563 | 27.559113  | 1     | 3.0   | 3        |
| 101 | 937.814586 | 815.053483 | 1     | 4.0   | 4        |

```
plt.figure()
tp.plot_traj(t, pos_columns=['position_x', 'position_y'], t_column='frame');
```



#### 2.10.4 filter

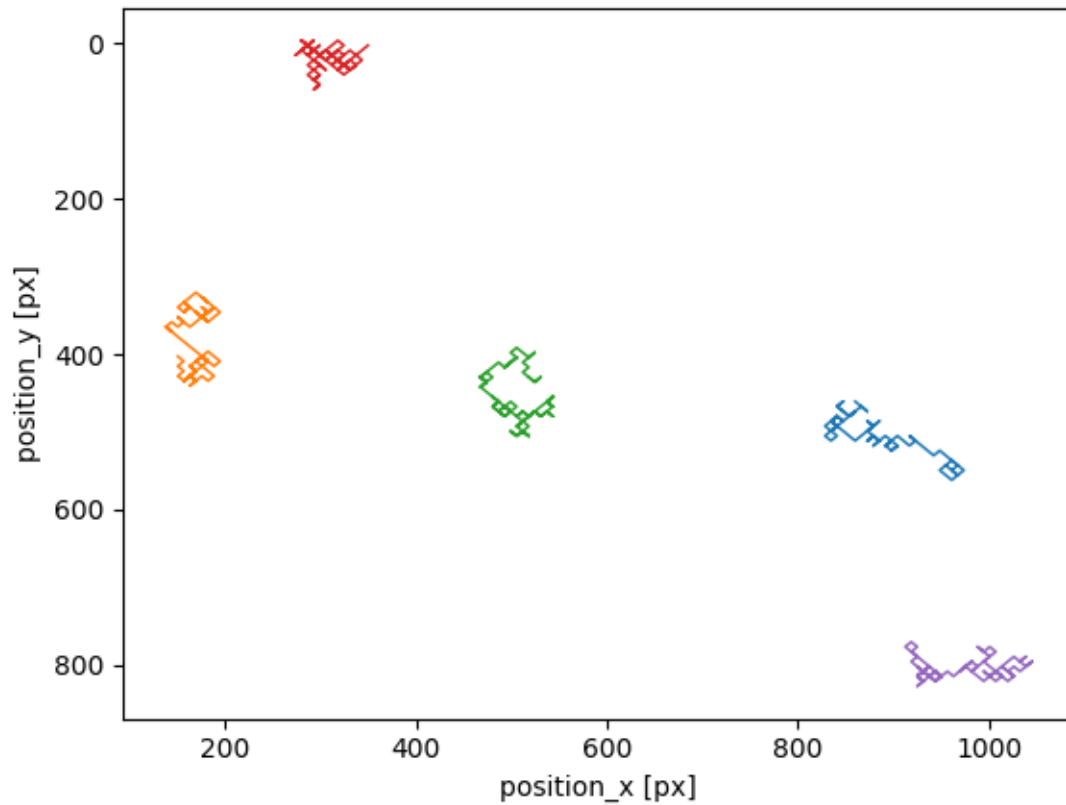
```
t.head()
```

|     | position_x | position_y | frame | track | particle |
|-----|------------|------------|-------|-------|----------|
| 300 | 954.974002 | 543.269132 | 0     | 0.0   | 0        |
| 200 | 150.484168 | 402.874581 | 0     | 1.0   | 1        |
| 0   | 518.146180 | 429.651004 | 0     | 2.0   | 2        |
| 400 | 318.156007 | 33.883669  | 0     | 3.0   | 3        |
| 100 | 944.139141 | 821.378039 | 0     | 4.0   | 4        |

```
t1 = tp.filter_stubs(t, 10).reset_index(drop=True)
len(t1)
```

```
500
```

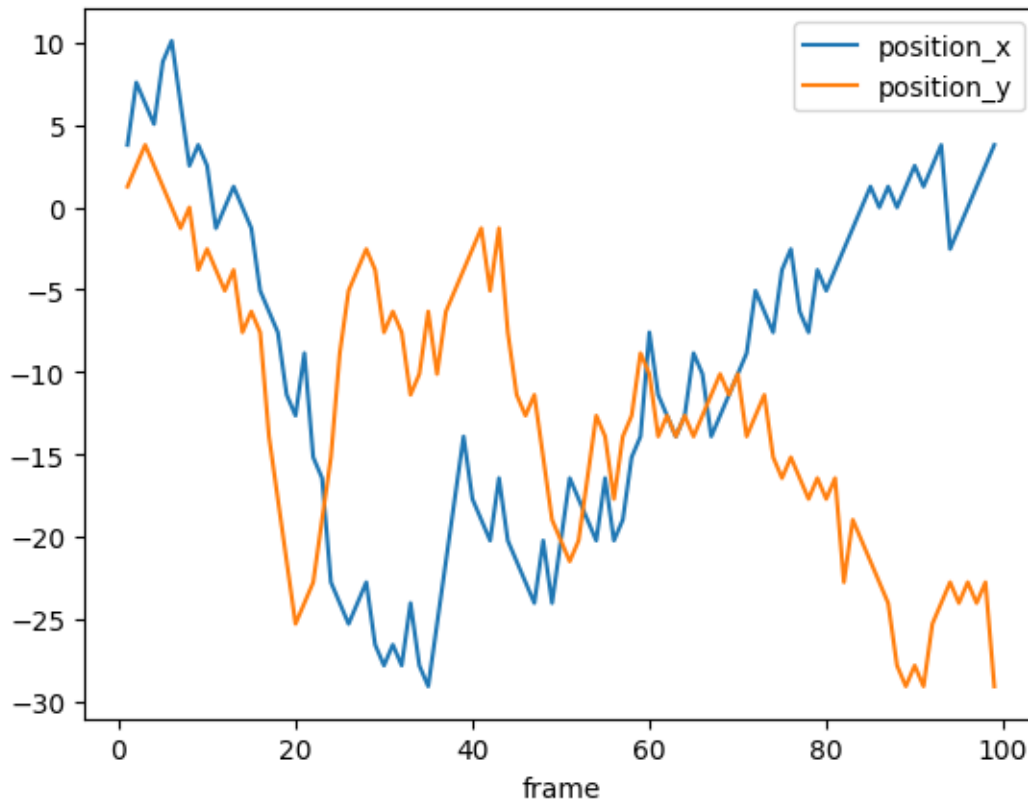
```
plt.figure()
tp.plot_traj(t1, pos_columns=['position_x', 'position_y']);
```



### 2.10.5 drift

```
d = tp.compute_drift(t1, pos_columns=['position_x', 'position_y'])
```

```
d.plot()  
plt.show()
```



### 2.10.6 Mean square displacement (msd)

```
em = tp.emsd(t1, 0.1, 100, pos_columns=['position_x', 'position_y']) # microns
↳ per pixel = 100/285., frames per second = 24
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[25], line 1
----> 1 em = tp.emsd(t1, 0.1, 100, pos_columns=['position_x', 'position_y']) #
↳ microns per pixel = 100/285., frames per second = 24

File ~/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/
↳ site-packages/trackpy/motion.py:235, in emsd(traj, mpp, fps, max_lagtime,
↳ detail, pos_columns)
    233     ids.append(pid)
    234     msds = pandas_concat(msds, keys=ids, names=['particle', 'frame'])
--> 235     results = msds.mul(msds['N'], axis=0).mean(level=1) # weighted
↳ average
    236     results = results.div(msds['N'].mean(level=1), axis=0) # weights
↳ normalized
    237     # Above, lagt is lumped in with the rest for simplicity and speed.
    238     # Here, rebuild it from the frame index.

File ~/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/
↳ site-packages/pandas/core/frame.py:11666, in DataFrame.mean(self, axis,
```

(continues on next page)

(continued from previous page)

```

↪ skipna, numeric_only, **kwargs)
11658 @doc(make_doc("mean", ndim=2))
11659 def mean(
11660     self,
11661     (...)
11662     **kwargs,
11663 ):
> 11664     result = super().mean(axis, skipna, numeric_only, **kwargs)
11665     if isinstance(result, Series):
11666         result = result.__finalize__(self, method="mean")

File ~/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/
↪ site-packages/pandas/core/generic.py:12413, in NDFrame.mean(self, axis,
↪ skipna, numeric_only, **kwargs)
12406 def mean(
12407     self,
12408     axis: Axis | None = 0,
12409     (...)
12410     **kwargs,
12411 ) -> Series | float:
> 12412     return self._stat_function(
12413         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
12414     )

File ~/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/
↪ site-packages/pandas/core/generic.py:12366, in NDFrame._stat_function(self,
↪ name, func, axis, skipna, numeric_only, **kwargs)
12355 @final
12356 def _stat_function(
12357     self,
12358     (...)
12359     **kwargs,
12360 ):
12361     assert name in ["median", "mean", "min", "max", "kurt", "skew"],
↪ name
> 12362     nv.validate_func(name, (), kwargs)
12363     validate_bool_kwarg(skipna, "skipna", none_allowed=False)
12364     return self._reduce(
12365         func, name=name, axis=axis, skipna=skipna, numeric_
↪ only=numeric_only
12366     )

File ~/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/
↪ site-packages/pandas/compat/numpy/function.py:416, in validate_func(fname,
↪ args, kwargs)
413     return validate_stat_func(args, kwargs, fname=fname)
414     validation_func = _validation_funcs[fname]
--> 415 return validation_func(args, kwargs)

```

(continues on next page)

(continued from previous page)

```
File ~/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/
↳ site-packages/pandas/compat/numpy/function.py:88, in CompatValidator.__call_
↳ (self, args, kwargs, fname, max_fname_arg_count, method)
    86     validate_kwargs(fname, kwargs, self.defaults)
    87     elif method == "both":
--> 88     validate_args_and_kwargs(
    89         fname, args, kwargs, max_fname_arg_count, self.defaults
    90     )
    91 else:
    92     raise ValueError(f"invalid validation method '{method}'")
```

```
File ~/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/
↳ site-packages/pandas/util/_validators.py:223, in validate_args_and_
↳ kwargs(fname, args, kwargs, max_fname_arg_count, compat_args)
    218         raise TypeError(
    219             f"{fname}() got multiple values for keyword argument '
↳ {key}'"
    220         )
    222 kwargs.update(args_dict)
--> 223 validate_kwargs(fname, kwargs, compat_args)
```

```
File ~/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/
↳ site-packages/pandas/util/_validators.py:164, in validate_kwargs(fname,
↳ kwargs, compat_args)
    142 """
    143 Checks whether parameters passed to the **kwargs argument in a
    144 function `fname` are valid parameters as specified in `*compat_args`
    (...)
    161 map to the default values specified in `compat_args`
    162 """
    163 kwds = kwargs.copy()
--> 164 _check_for_invalid_keys(fname, kwargs, compat_args)
    165 _check_for_default_values(fname, kwds, compat_args)
```

```
File ~/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/
↳ site-packages/pandas/util/_validators.py:138, in _check_for_invalid_
↳ keys(fname, kwargs, compat_args)
    136 if diff:
    137     bad_arg = next(iter(diff))
--> 138     raise TypeError(f"{fname}() got an unexpected keyword argument '
↳ {bad_arg}'")
```

**TypeError:** mean() got an unexpected keyword argument 'level'

```
fig, ax = plt.subplots()
ax.plot(em.index, em, 'o')
#ax.set_xscale('log')
#ax.set_yscale('log')
ax.set(ylabel=r'$\langle \Delta r^2 \rangle$ [$\mu m^2$'],
```

(continues on next page)

(continued from previous page)

```
xlabel='lag time $t$')
#ax.set(ylim=(1e-2, 10));
```

## 2.11 Tutorial about transforming LocData

Locan provides methods for transforming localization data sets into new LocData objects.

```
from pathlib import Path

%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

import locan as lc
```

```
/tmp/ipykernel_2055/1692805095.py:6: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳ 54466
```

```
import pandas as pd
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.11.1 Spatially randomize a structured set of localizations

Assume that localizations are somehow structured throughout a region. Often it is helpful to compare analysis results to a similar dataset in which localizations are homogeneously Poisson distributed. A LocData object with this kind of data can be provided by the randomize function.

```
rng = np.random.default_rng(seed=1)
```

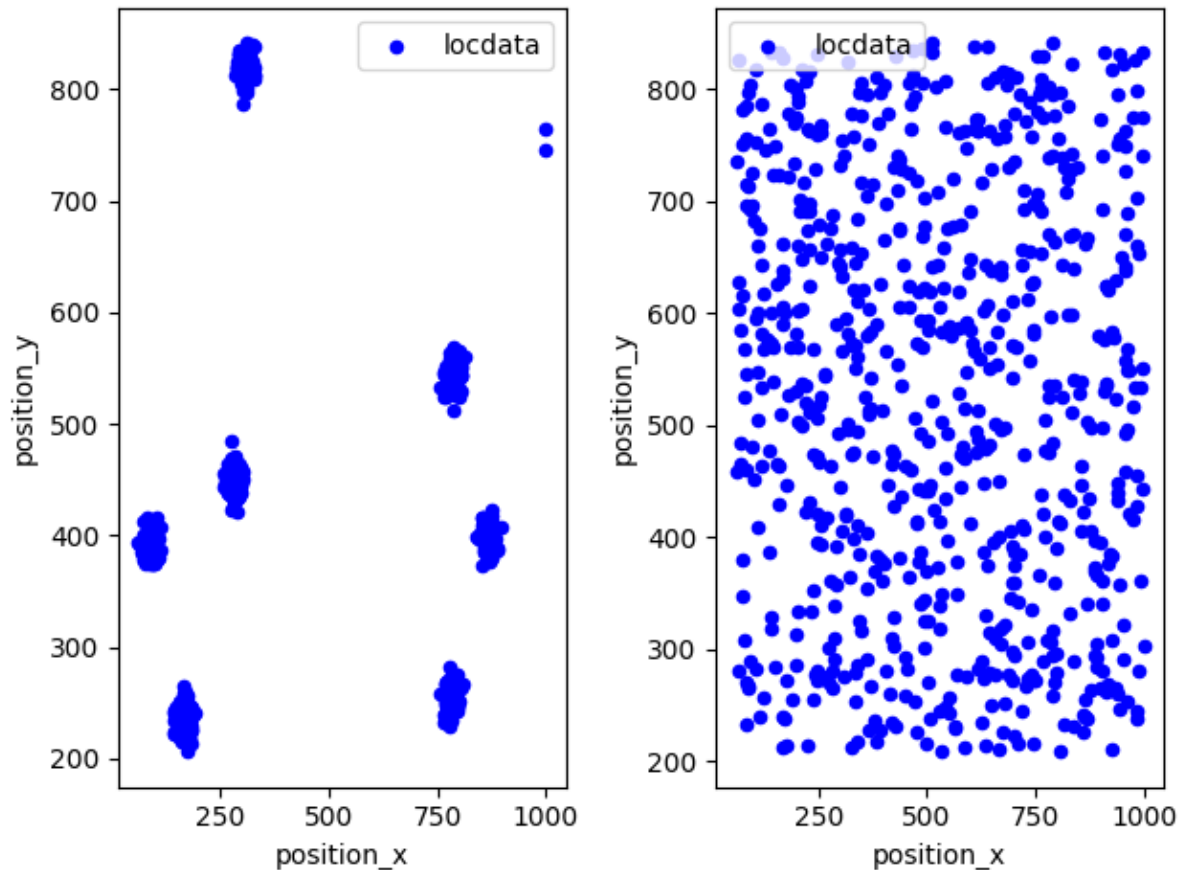
```
locdata = lc.simulate_Thomas(parent_intensity=1e-5, region=((0, 1000), (0, 1000)), cluster_mu=100, cluster_std=10, seed=rng)
locdata.print_summary()
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
identifier: "1"
comment: ""
source: SIMULATION
state: RAW
element_count: 705
frame_count: 0
creation_time {
  2024-03-14T11:10:11.039092Z
}
```

```
locdata_random = lc.randomize(locdata, hull_region='bb', seed=rng)
```

```
fig, ax = plt.subplots(nrows=1, ncols=2)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax[0], color='Blue', label='locdata')
locdata_random.data.plot.scatter(x='position_x', y='position_y', ax=ax[1], color='Blue', label='locdata')
plt.tight_layout()
plt.show()
```





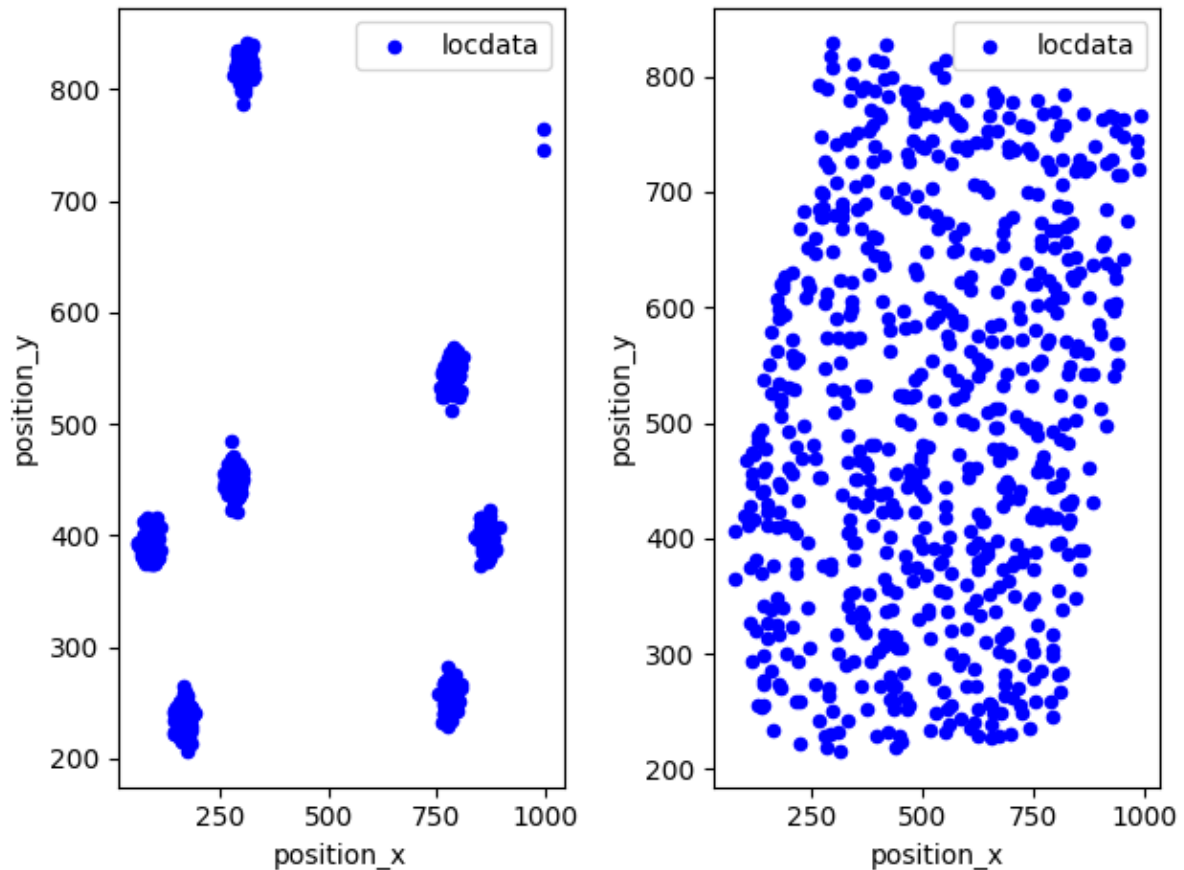
```
print('Area of bounding box for structured data: {:.0f}'.format(locdata.
↳properties['region_measure_bb']))
print('Area of bounding box for randomized data: {:.0f}'.format(locdata_
↳random.properties['region_measure_bb']))
print('Ratio: {:.4f}'.format(locdata_random.properties['region_measure_bb'] /
↳locdata.properties['region_measure_bb']))
```

```
Area of bounding box for structured data: 595088
Area of bounding box for randomized data: 592451
Ratio: 0.9956
```

Regions other from bounding box can be specified as RoiRegion instance.

```
region = lc.ConvexHull(locdata.coordinates).region
locdata_random = lc.randomize(locdata, hull_region=region)
```

```
fig, ax = plt.subplots(nrows=1, ncols=2)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax[0], color=
↳'Blue', label='locdata')
locdata_random.data.plot.scatter(x='position_x', y='position_y', ax=ax[1],
↳color='Blue', label='locdata')
plt.tight_layout()
plt.show()
```



```
print('Area of bounding box for structured data: {:.0f}'.format(locdata.
↳properties['region_measure_bb']))
print('Area of bounding box for randomized data: {:.0f}'.format(locdata_
↳random.properties['region_measure_bb']))
print('Ratio: {:.4f}'.format(locdata_random.properties['region_measure_bb'] /
↳locdata.properties['region_measure_bb']))
```

```
Area of bounding box for structured data: 595088
Area of bounding box for randomized data: 558256
Ratio: 0.9381
```

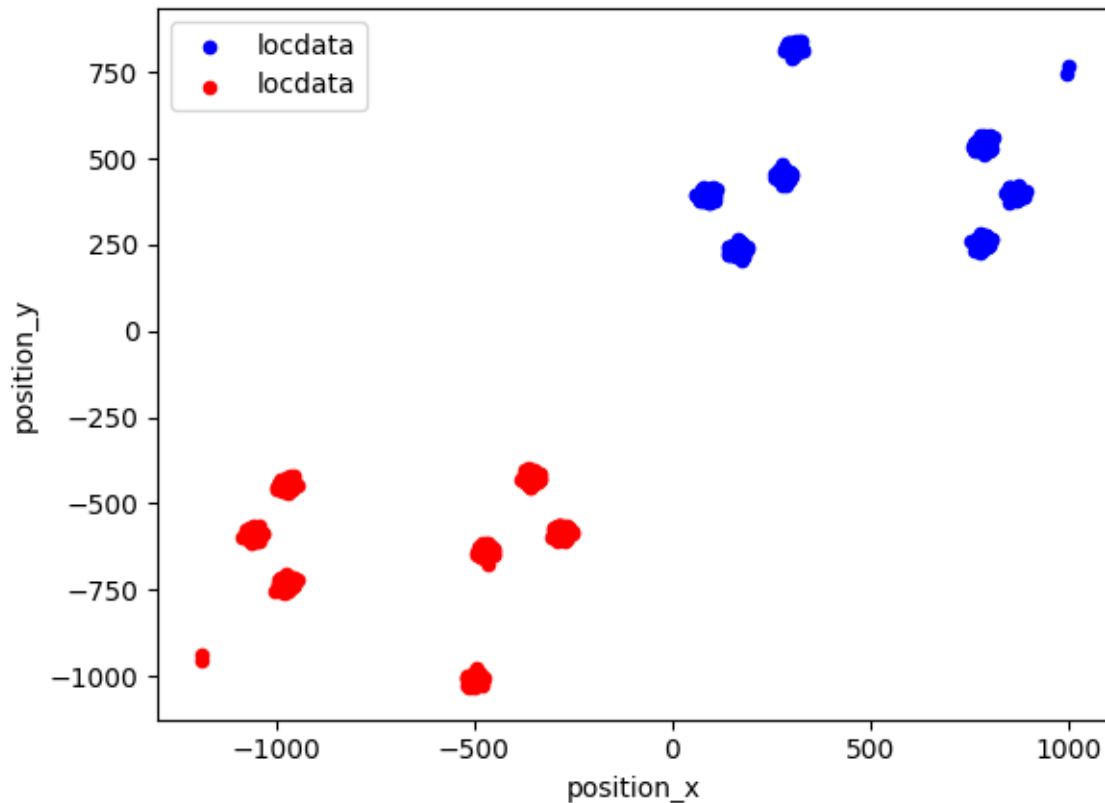
### 2.11.2 Apply an affine transformation to localization coordinates

A wrapper function provides affine transformations based on either numpy or open3d methods.

```
matrix = ((-1, 0), (0, -1))
offset = (10, 10)
pre_translation = (100, 100)

locdata_transformed = lc.transform_affine(locdata, matrix, offset, pre_
↳translation, method='numpy')
```

```
fig, ax = plt.subplots()
locdata.data.plot.scatter(x='position_x', y='position_y', color='Blue', label=
↳ 'locdata', ax=ax)
locdata_transformed.data.plot.scatter(x='position_x', y='position_y', color=
↳ 'Red', label='locdata', ax=ax);
```



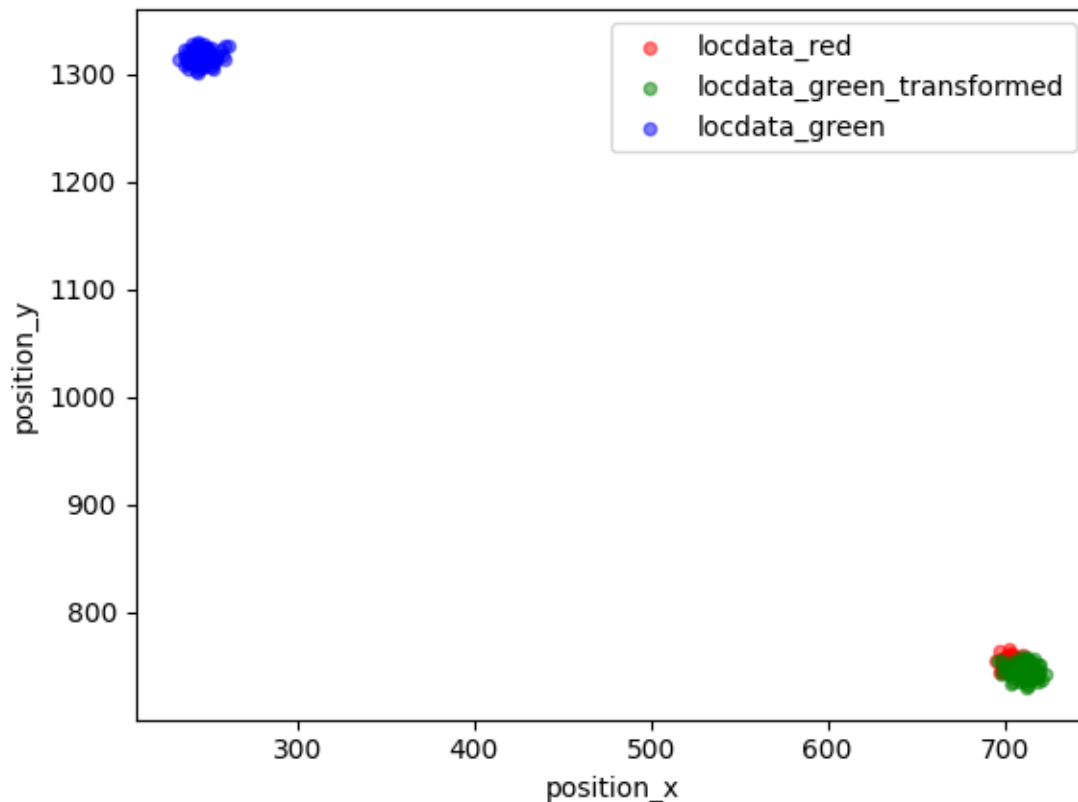
### 2.11.3 Apply a BunwarpJ transformation to localization coordinates

Often a transformation matrix was computed using ImageJ. The `bunwarp` function allows applying a transformation from the raw matrix of the ImageJ/Fiji plugin BunwarpJ. Here we show a very small region with a single fluorescent bead that is recorded on a red and a green dSTORM channel.

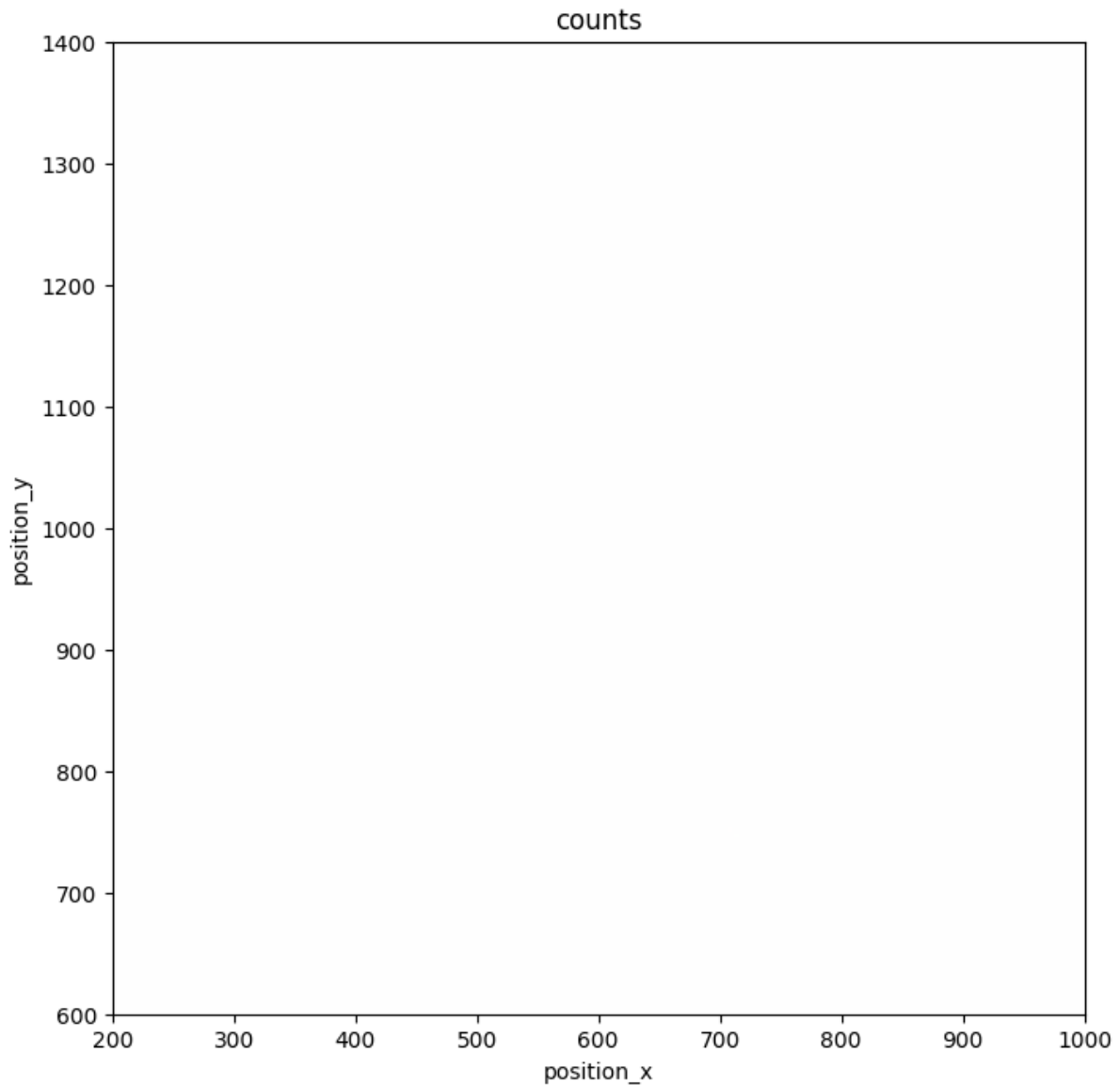
```
matrix_path = lc.ROOT_DIR / 'tests/test_data/transform/BunwarpJ_
↳ transformation_raw_green.txt'
locdata_green = lc.load_asdf_file(path=lc.ROOT_DIR /
                                  'tests/test_data/transform/rapidSTORM_beads_
↳ green.asdf')
locdata_red = lc.load_asdf_file(path=lc.ROOT_DIR /
                                 'tests/test_data/transform/rapidSTORM_beads_
↳ red.asdf')

locdata_green_transformed = lc.bunwarp(locdata=locdata_green, matrix_
↳ path=matrix_path, pixel_size=(10, 10), flip=True)
```

```
fig, ax = plt.subplots()
locdata_red.data.plot.scatter(x='position_x', y='position_y', color='Red',
    ↪ label='locdata_red', alpha=0.5, ax=ax)
locdata_green_transformed.data.plot.scatter(x='position_x', y='position_y',
    ↪ color='Green', label='locdata_green_transformed', alpha=0.5, ax=ax)
locdata_green.data.plot.scatter(x='position_x', y='position_y', color='Blue',
    ↪ label='locdata_green', alpha=0.5, ax=ax);
```



```
fig, ax = plt.subplots(figsize=(10, 8))
lc.render_2d_rgb_mpl([locdata_red, locdata_green_transformed, locdata_green],
    ↪ bin_size=5, bin_range=((200, 1000), (600, 1400)), rescale='equal', ax=ax);
```



## 2.12 Tutorial about localizations per frame

```
from pathlib import Path

%matplotlib inline

import matplotlib.pyplot as plt

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969
```

(continues on next page)

(continued from previous page)

```
Python:
  version: 3.11.6
```

### 2.12.1 Load rapidSTORM data file

Identify some data in the test\_data directory and provide a path using pathlib.Path

```
path = lc.ROOT_DIR / 'tests/test_data/rapidSTORM_dstorm_data.txt'
print(path, '\n')
```

```
dat = lc.load_rapidSTORM_file(path=path, nrows=1000)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.txt
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

Print information about the data:

```
print(dat.data.head(), '\n')
print('Summary:')
dat.print_summary()
print('Properties:')
print(dat.properties)
```

|   | position_x | position_y | frame | intensity | chi_square | local_background |
|---|------------|------------|-------|-----------|------------|------------------|
| 0 | 9657.40    | 24533.5    | 0     | 33290.10  | 1192250.0  | 767.732971       |
| 1 | 16754.90   | 18770.0    | 0     | 21275.40  | 2106810.0  | 875.460999       |
| 2 | 14457.60   | 18582.6    | 0     | 20748.70  | 526031.0   | 703.369995       |
| 3 | 6820.58    | 16662.8    | 0     | 8531.77   | 3179190.0  | 852.789001       |
| 4 | 19183.20   | 22907.2    | 0     | 14139.60  | 448631.0   | 662.770020       |

```
Summary:
identifier: "1"
comment: ""
source: EXPERIMENT
state: RAW
element_count: 14
frame_count: 1
file {
  type: RAPIDSTORM
  path: "/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/
lib/python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.
txt"
}
creation_time {
```

(continues on next page)

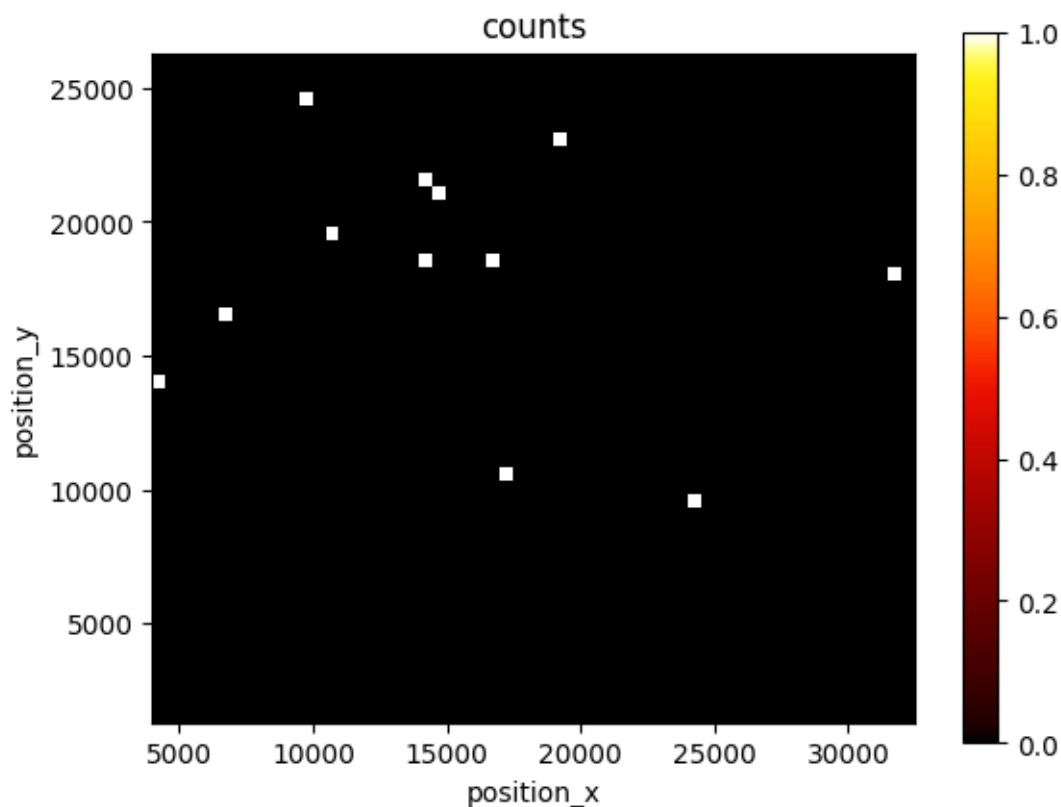
(continued from previous page)

```
2024-03-14T11:07:44.308119Z
}

Properties:
{'localization_count': 14, 'position_x': 15873.847142857145, 'uncertainty_x': 2361.4490857013648, 'position_y': 17403.909285714286, 'uncertainty_y': 1803.9975262697349, 'intensity': 183987.669999999998, 'local_background': 675.0614, 'frame': 0, 'region_measure_bb': 730882123.3259, 'localization_density_bb': 1.915493559521281e-08, 'subregion_measure_bb': 108337.2}
```

## 2.12.2 Visualization

```
lc.render_2d(dat, bin_size=500, rescale=lc.Trafo.EQUALIZE_0P3);
```



### Analyze localizations per frame

We have a look at the number of localizations that were detected in each frame.

The analysis class `Localizations_per_frame` provides numerical results, a plot of results versus frame, and a density graph (histogram).

```
lpf = lc.LocalizationsPerFrame()
lpf.compute(dat)
```

```
LocalizationsPerFrame(norm=None, time_delta=integration_time, resample=None)
```

```
lpf.results
```

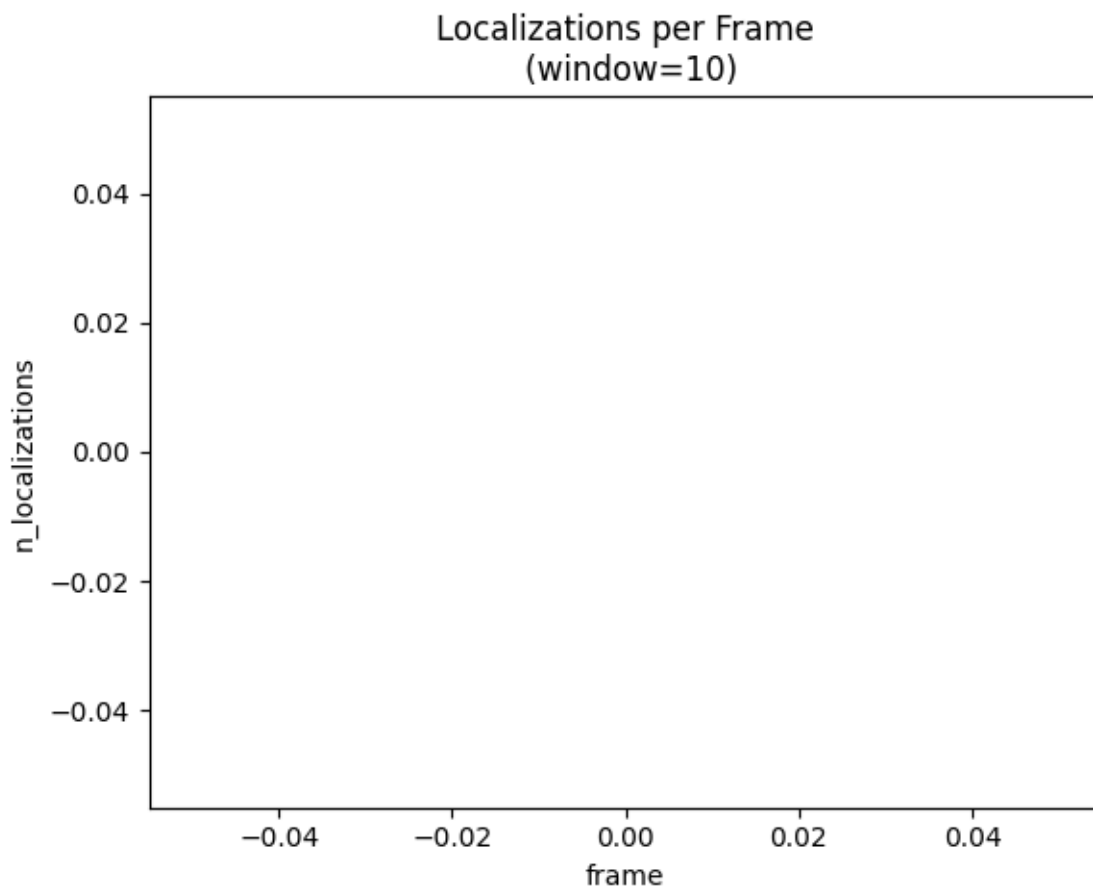
```
<locan.analysis.localizations_per_frame._Results at 0x7f0a1c93c990>
```

```
print(lpf.results.time_series.head())
```

```
frame
0    14.0
Name: n_localizations, dtype: float64
```

The plot shows results smoothed by a running average according to the specified window.

```
lpf.plot(window=10);
```



The histogram per default provides automatic bins, is normalized to show a probability density function and estimates a normal distribution.

```
lpf.hist();
```

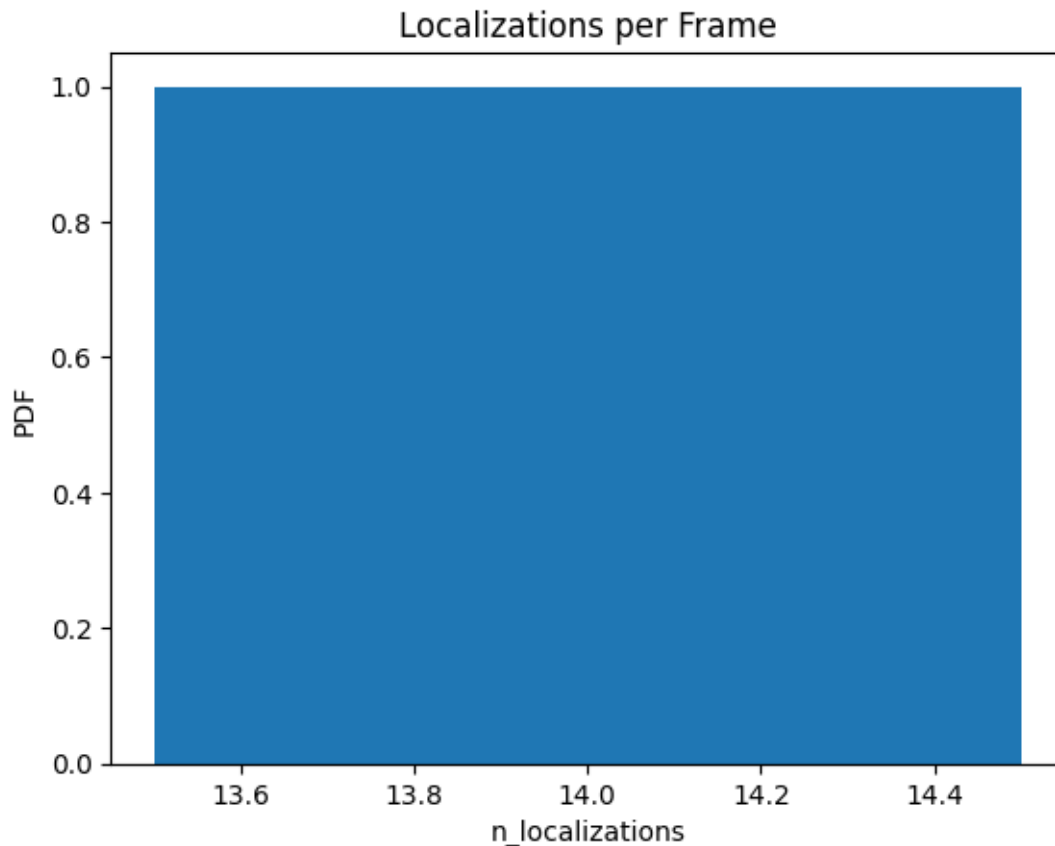
```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/scipy/stats/_distn_infrastructure.py:2244: ␣
```

(continues on next page)



(continued from previous page)

```
↳RuntimeWarning: invalid value encountered in multiply
    lower_bound = _a * scale + loc
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/scipy/stats/_distn_infrastructure.py:2245:↳
↳RuntimeWarning: invalid value encountered in multiply
    upper_bound = _b * scale + loc
```



The accumulation time is the time at which fraction of the cumulative intensity is reached.

```
lpf.results.accumulation_time()
```

```
0
```

### 2.12.3 Plot normalized values

We can normalize the number of localizations to any other LocData property.

```
dat.properties
```

```
{'localization_count': 14,
 'position_x': 15873.847142857145,
 'uncertainty_x': 2361.4490857013648,
 'position_y': 17403.909285714286,
```

(continues on next page)

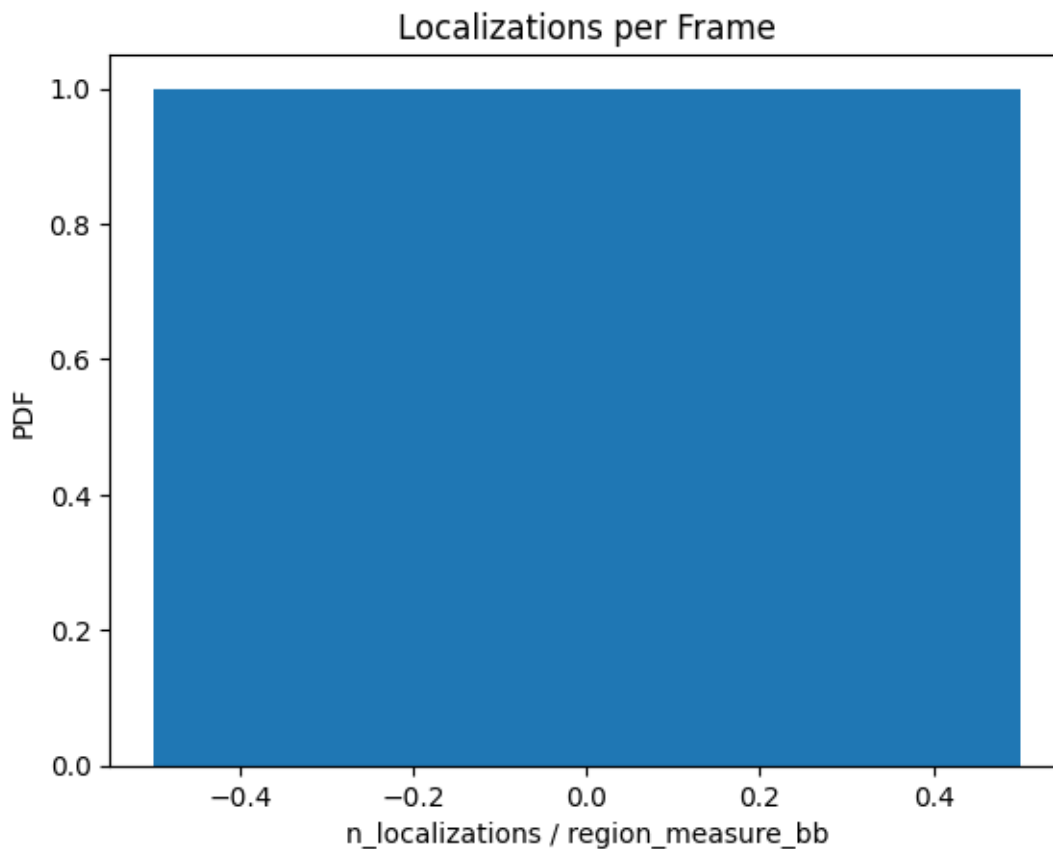
(continued from previous page)

```
'uncertainty_y': 1803.9975262697349,
'intensity': 183987.66999999998,
'local_background': 675.0614,
'frame': 0,
'region_measure_bb': 730882123.3259,
'localization_density_bb': 1.915493559521281e-08,
'subregion_measure_bb': 108337.2}
```

```
lpf = lc.LocalizationsPerFrame(norm='region_measure_bb').compute(dat)
```

```
lpf.hist();
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/scipy/stats/_distn_infrastructure.py:2244:
RuntimeWarning: invalid value encountered in multiply
    lower_bound = _a * scale + loc
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/scipy/stats/_distn_infrastructure.py:2245:
RuntimeWarning: invalid value encountered in multiply
    upper_bound = _b * scale + loc
```



## 2.13 Tutorial about localization precision

Localization precision is determined from consecutive localizations that are identified within a certain search radius. The results include the distances between localization pairs, the position deltas for each coordinate and the corresponding frame of the first localizations.

```
from pathlib import Path

%matplotlib inline

import matplotlib.pyplot as plt

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.13.1 Load SMLM data file

Identify some data in the test\_data directory and provide a path using `pathlib.Path` (returned by `lc.ROOT_DIR`)

```
path = lc.ROOT_DIR / 'tests/test_data/npc_gp210.asdf'
print(path, '\n')
dat = lc.load_locdata(path=path, file_type=lc.FileType.ASDF)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/locan/tests/test_data/npc_gp210.asdf
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

Print information about the data:

```
print(dat.data.head(), '\n')
print('Summary:')
dat.print_summary()
print('Properties:')
print(dat.properties)
```

|   | position_x  | position_y  | frame | intensity    | two_kernel_improvement | \   |
|---|-------------|-------------|-------|--------------|------------------------|-----|
| 0 | 5631.709961 | 6555.850098 | 24    | 21800.699219 |                        | 0.0 |

(continues on next page)

(continued from previous page)

|   |             |             |     |              |     |
|---|-------------|-------------|-----|--------------|-----|
| 1 | 5642.899902 | 6550.990234 | 25  | 23352.900391 | 0.0 |
| 2 | 5610.319824 | 6546.819824 | 26  | 5007.509766  | 0.0 |
| 3 | 5713.299805 | 6611.950195 | 141 | 6437.140137  | 0.0 |
| 4 | 5699.750000 | 6627.390137 | 142 | 19998.300781 | 0.0 |

|   | chi_square | local_background |
|---|------------|------------------|
| 0 | 1426930.0  | 1188.489990      |
| 1 | 983634.0   | 1135.199951      |
| 2 | 577707.0   | 1107.219971      |
| 3 | 616255.0   | 1118.430054      |
| 4 | 903412.0   | 1144.709961      |

**Summary:**

identifier: "5"

comment: ""

source: EXPERIMENT

state: MODIFIED

element\_count: 202

frame\_count: 202

file {

type: ASDF

path: "/home/docs/checkouts/readthedocs.org/user\_builds/locan/envs/latest/  
→lib/python3.11/site-packages/locan/tests/test\_data/npc\_gp210.asdf"

}

creation\_time {

2024-01-23T08:32:31.518779Z

}

modification\_time {

2024-01-23T08:32:31.518779Z

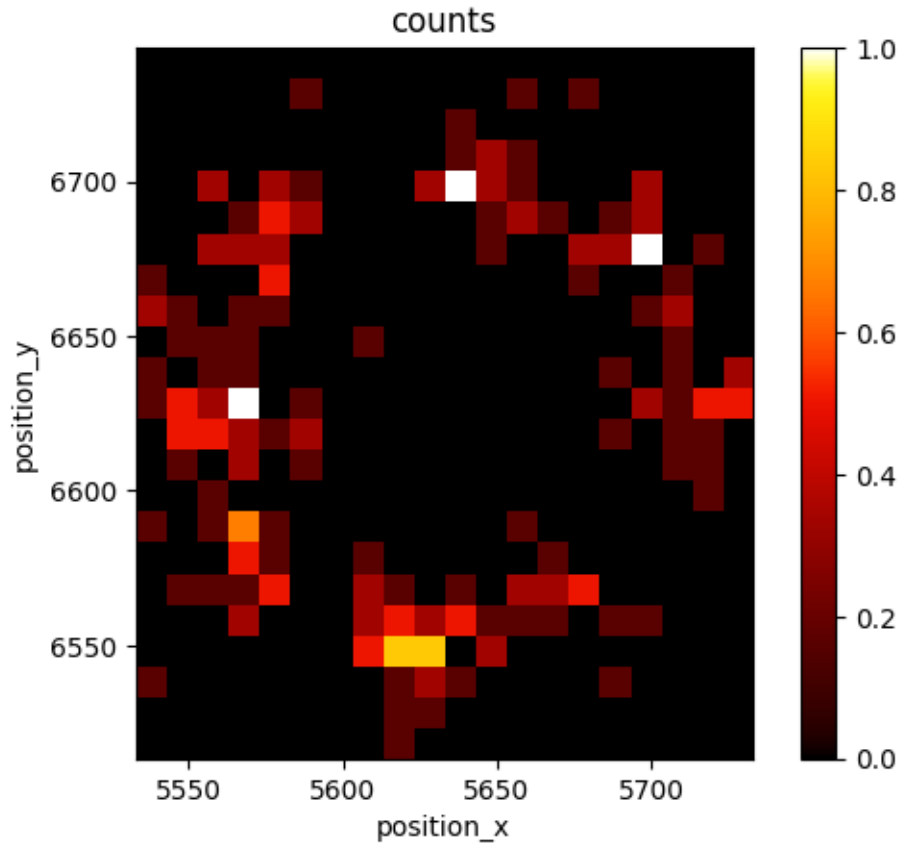
}

**Properties:**

```
{'localization_count': 202, 'position_x': 5623.488892810179, 'uncertainty_x': 3.9700846636030356, 'position_y': 6625.534602703435, 'uncertainty_y': 3.999303432482808, 'intensity': 3944778.0, 'local_background': 1131.3207, 'frame': 24, 'region_measure_bb': 47134.543, 'localization_density_bb': 0.00428560429946091, 'subregion_measure_bb': 870.98046875}
```

## 2.13.2 Visualization

```
lc.render_2d(dat, bin_size=10, rescale=(0,100));
```



### 2.13.3 Analyze localization precision

```
lp = lc.LocalizationPrecision(radius=50)
```

```
lp.compute(dat)
lp.results.head()
```

```
Processed frames::  0%|          | 0/24884 [00:00<?, ?it/s]
```

```
Processed frames::  6%|          | 1401/24884 [00:00<00:01, 13892.00it/s]
```

```
Processed frames:: 16%|          | 3906/24884 [00:00<00:01, 20010.85it/s]
```

```
Processed frames:: 25%|          | 6245/24884 [00:00<00:00, 21481.83it/s]
```

```
Processed frames:: 36%|          | 8880/24884 [00:00<00:00, 23094.35it/s]
```

```
Processed frames:: 64%|          | 15900/24884 [00:00<00:00, 39841.49it/s]
```

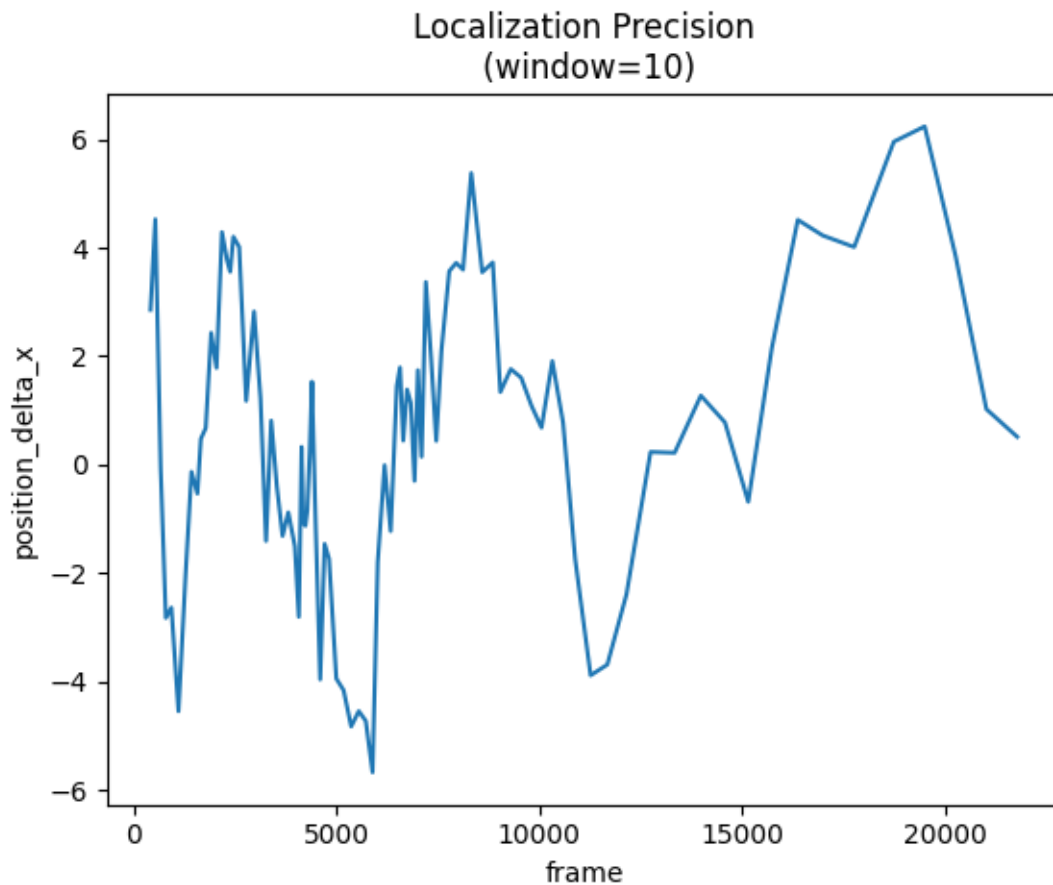
```
Processed frames:: 99%|| 24643/24884 [00:00<00:00, 55869.86it/s]
```

Processed frames:: 100%|| 24884/24884 [00:00<00:00, 40163.68it/s]

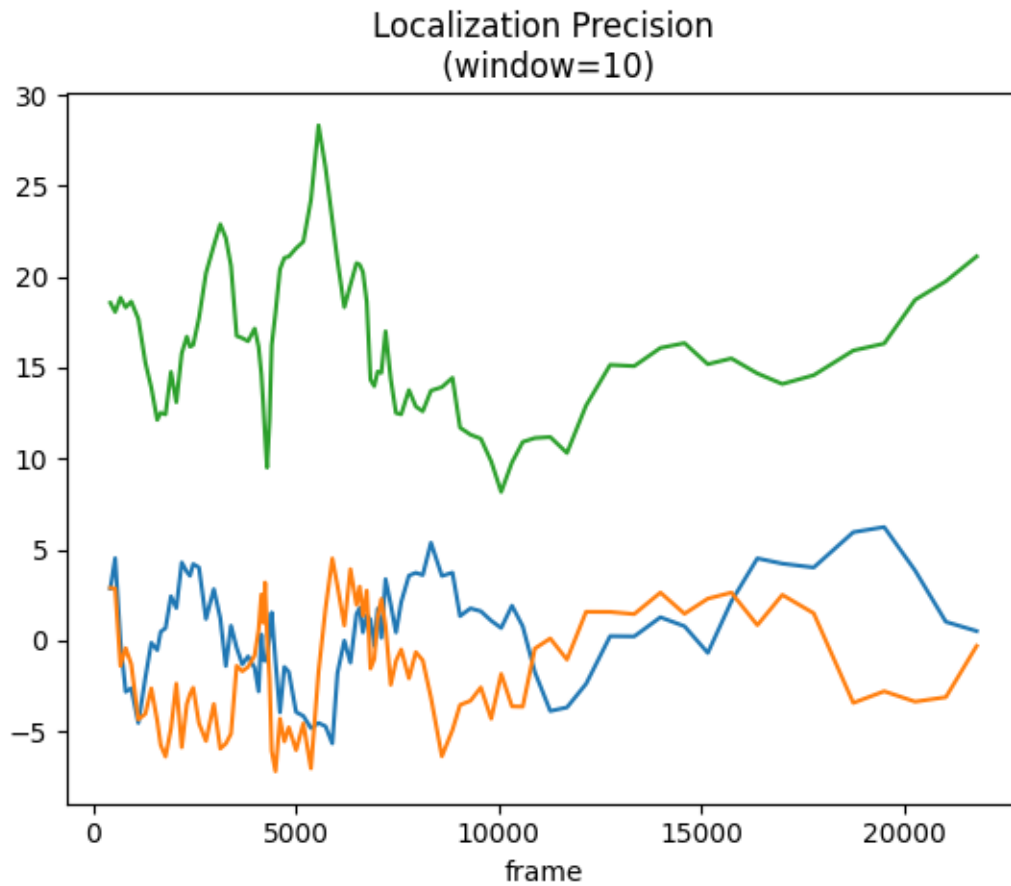
|   | position_delta_x | position_delta_y | position_distance | frame |
|---|------------------|------------------|-------------------|-------|
| 0 | -11.189941       | 4.859863         | 12.199716         | 24    |
| 1 | 32.580078        | 4.170410         | 32.845909         | 25    |
| 2 | 13.549805        | -15.439941       | 20.542370         | 141   |
| 3 | 4.669922         | 3.010254         | 5.556060          | 142   |
| 4 | 20.469727        | 14.750000        | 25.230383         | 239   |

The plot

```
lp.plot(loc_property='position_delta_x', window=10);
```

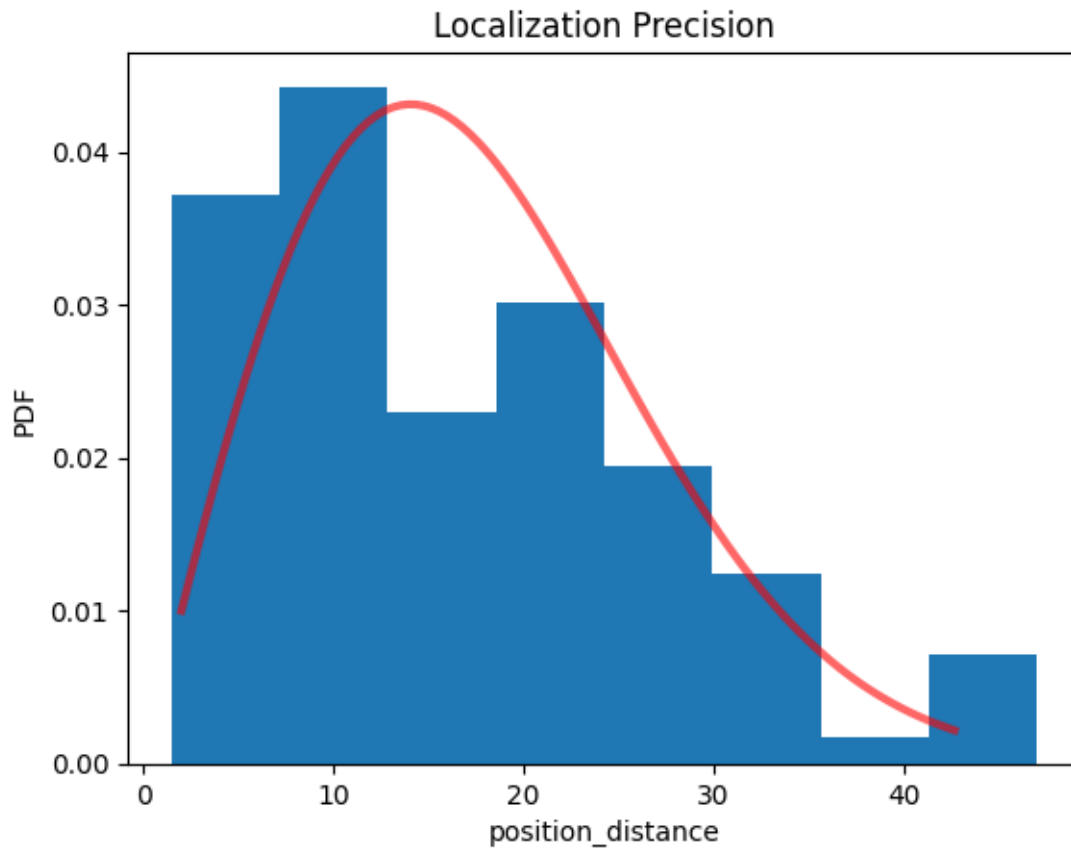


```
lp.plot(window=10);
```



The histogram for the distances per default includes a fit to a distribution expected for normal distributed localizations.  $\text{Sigma} / \sqrt{2}$  is the localization precision.

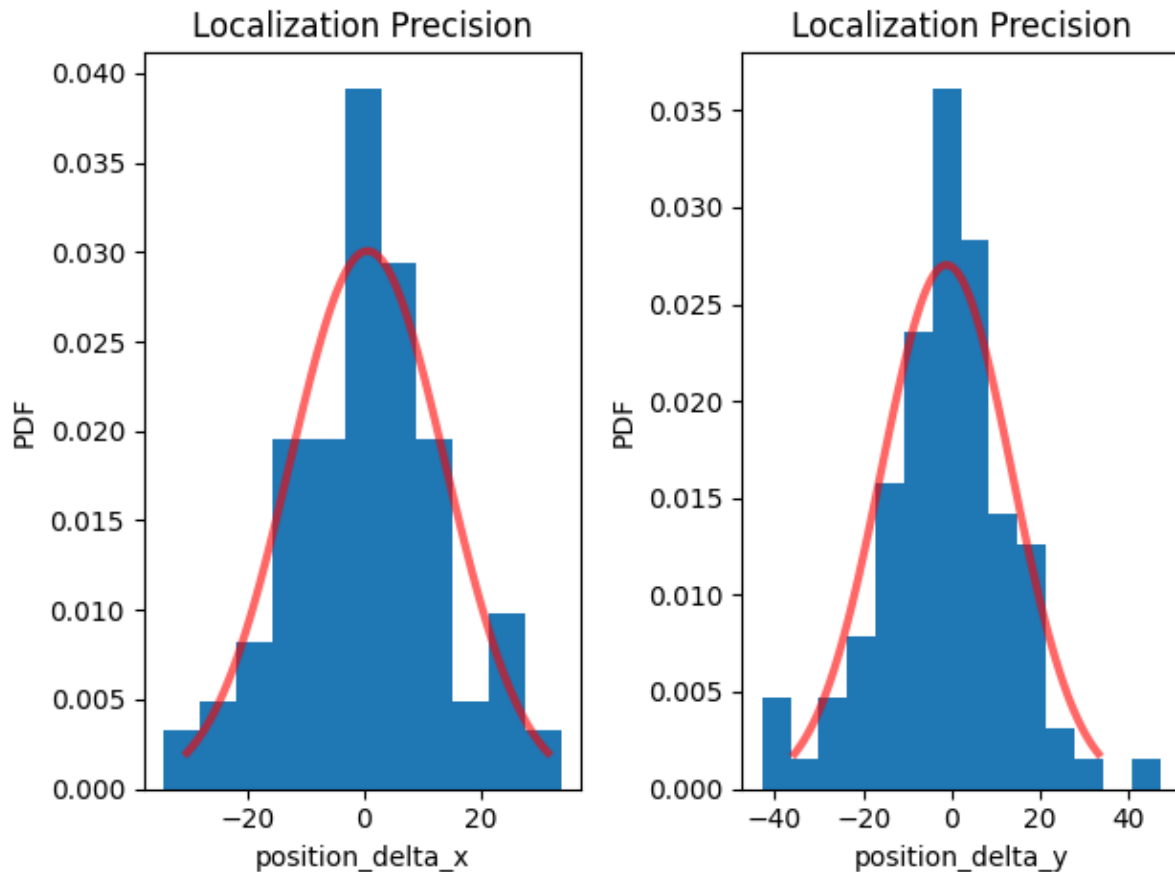
```
lp.hist();
```



Alternatively the position deltas can be histogrammed.

```
fig, ax = plt.subplots(nrows=1, ncols=2)
lp.hist(ax=ax[0], loc_property='position_delta_x')
lp.hist(ax=ax[1], loc_property='position_delta_y')
plt.tight_layout()
plt.show()
```





#### 2.13.4 Fit distributions and show parameters

Appropriate distribution functions are fitted to the data either by calling the `hist` function or by running:

```
lp.fit_distributions()
```

The estimated fit parameters are provided under the `distribution_statistics` attribute

```
lp.distribution_statistics.parameter_dict()
```

```
{'position_delta_x_loc': 0.6029139,
 'position_delta_x_scale': 13.2682,
 'position_delta_y_loc': -1.227687,
 'position_delta_y_scale': 14.76087,
 'position_distance_sigma': 14.067675781250028,
 'position_distance_loc': 0,
 'position_distance_scale': 1}
```

Remember: localization precision is typically defined as the standard deviation for the distances between localizations and their center position (the true dye position). Therefore, the estimated sigmas have to be divided by  $\sqrt{2}$  to yield localization precision.

## 2.14 Tutorial about analyzing localization properties

```
from pathlib import Path

%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.14.1 Load rapidSTORM data file

Identify some data in the test\_data directory and provide a path using `pathlib.Path` (returned by `lc.ROOT_DIR`)

```
path = lc.ROOT_DIR / 'tests/test_data/rapidSTORM_dstorm_data.txt'
print(path, '\n')

dat = lc.load_rapidSTORM_file(path=path, nrows=1000)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.txt
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

Print information about the data:

```
print(dat.data.head(), '\n')
print('Summary:')
dat.print_summary()
print('Properties:')
print(dat.properties)
```

|   | position_x | position_y | frame | intensity | chi_square | local_background |
|---|------------|------------|-------|-----------|------------|------------------|
| 0 | 9657.40    | 24533.5    | 0     | 33290.10  | 1192250.0  | 767.732971       |
| 1 | 16754.90   | 18770.0    | 0     | 21275.40  | 2106810.0  | 875.460999       |
| 2 | 14457.60   | 18582.6    | 0     | 20748.70  | 526031.0   | 703.369995       |

(continues on next page)

(continued from previous page)

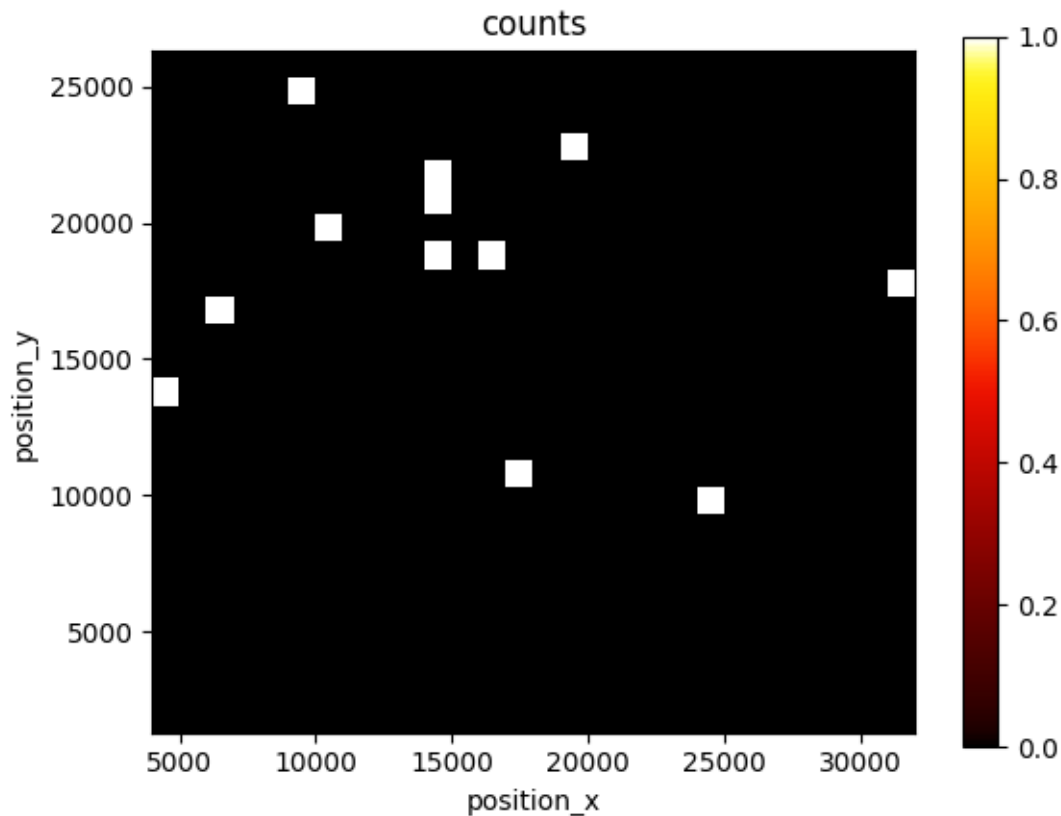
|   |          |         |   |          |           |            |
|---|----------|---------|---|----------|-----------|------------|
| 3 | 6820.58  | 16662.8 | 0 | 8531.77  | 3179190.0 | 852.789001 |
| 4 | 19183.20 | 22907.2 | 0 | 14139.60 | 448631.0  | 662.770020 |

```
Summary:
identifier: "1"
comment: ""
source: EXPERIMENT
state: RAW
element_count: 14
frame_count: 1
file {
  type: RAPIDSTORM
  path: "/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/
↳ lib/python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.
↳ txt"
}
creation_time {
  2024-03-14T11:07:36.400872Z
}
```

```
Properties:
{'localization_count': 14, 'position_x': 15873.847142857145, 'uncertainty_x':
↳ 2361.4490857013648, 'position_y': 17403.909285714286, 'uncertainty_y': 1803.
↳ 9975262697349, 'intensity': 183987.669999999998, 'local_background': 675.
↳ 0614, 'frame': 0, 'region_measure_bb': 730882123.3259, 'localization_
↳ density_bb': 1.915493559521281e-08, 'subregion_measure_bb': 108337.2}
```

## 2.14.2 Visualization

```
lc.render_2d(dat, bin_size=1000, rescale=(0,100));
```



### Analyze a localization property

We have a look at a certain localization property in locdata.

The analysis class `LocalizationProperty` provides a dataframe with the property as function of another property (index), and a plot or histogram of this property.

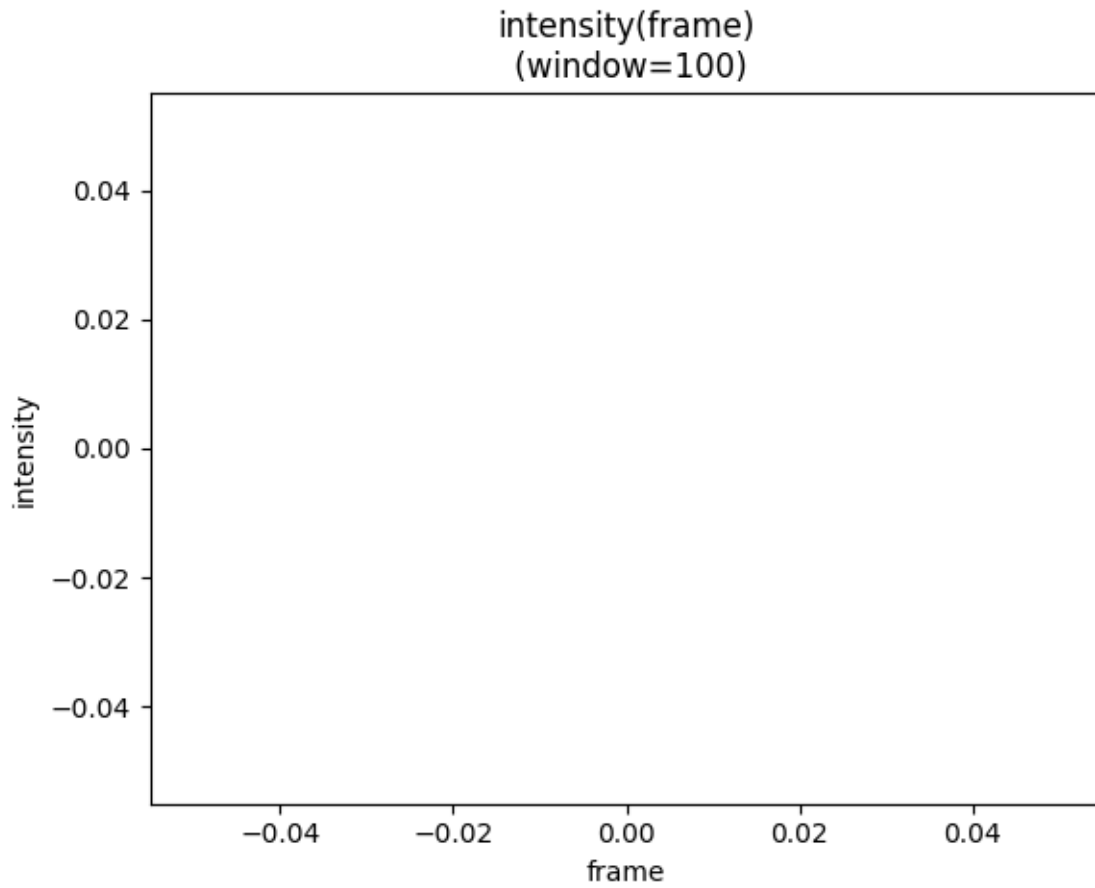
```
lprop = lc.LocalizationProperty(loc_property='intensity', index='frame')
```

```
lprop.compute(dat)
print(lprop.results.head())
```

|       | intensity |
|-------|-----------|
| frame |           |
| 0     | 33290.10  |
| 0     | 21275.40  |
| 0     | 20748.70  |
| 0     | 8531.77   |
| 0     | 14139.60  |

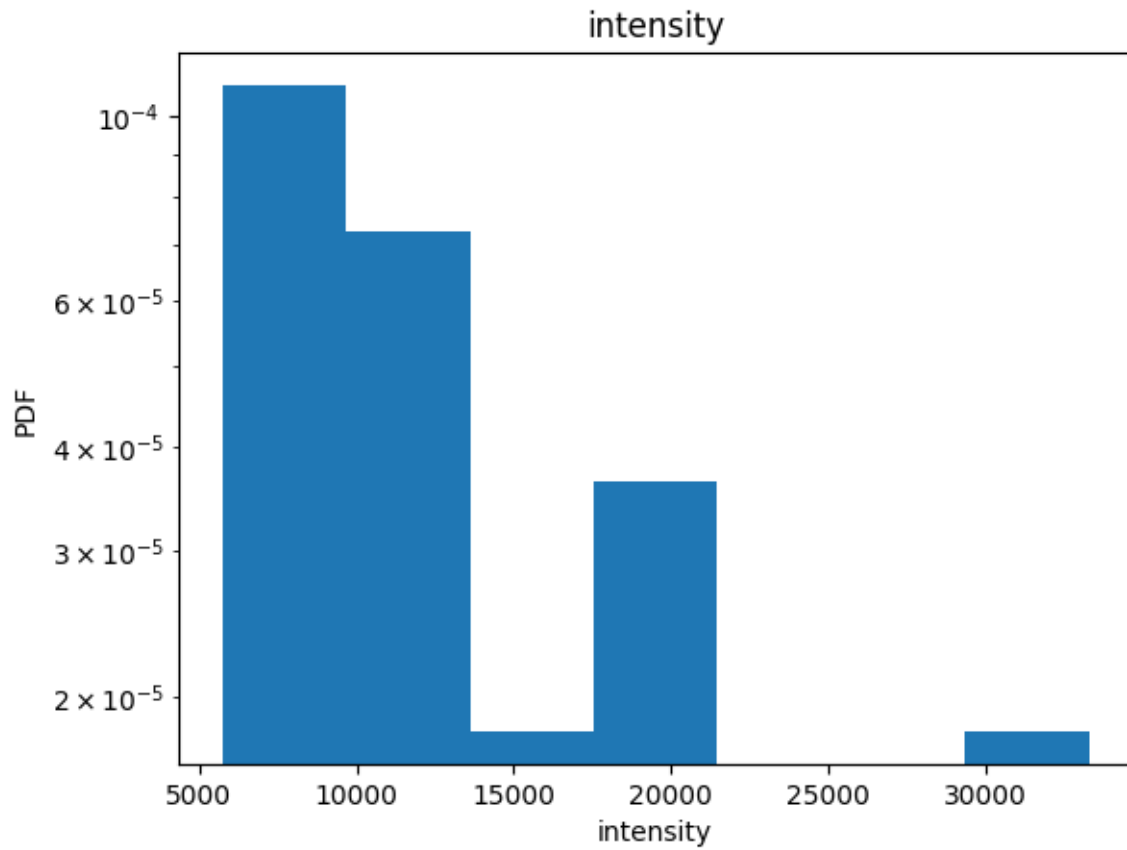
The plot shows results smoothed by a running average according to the specified window.

```
lprop.plot(window=100);
```



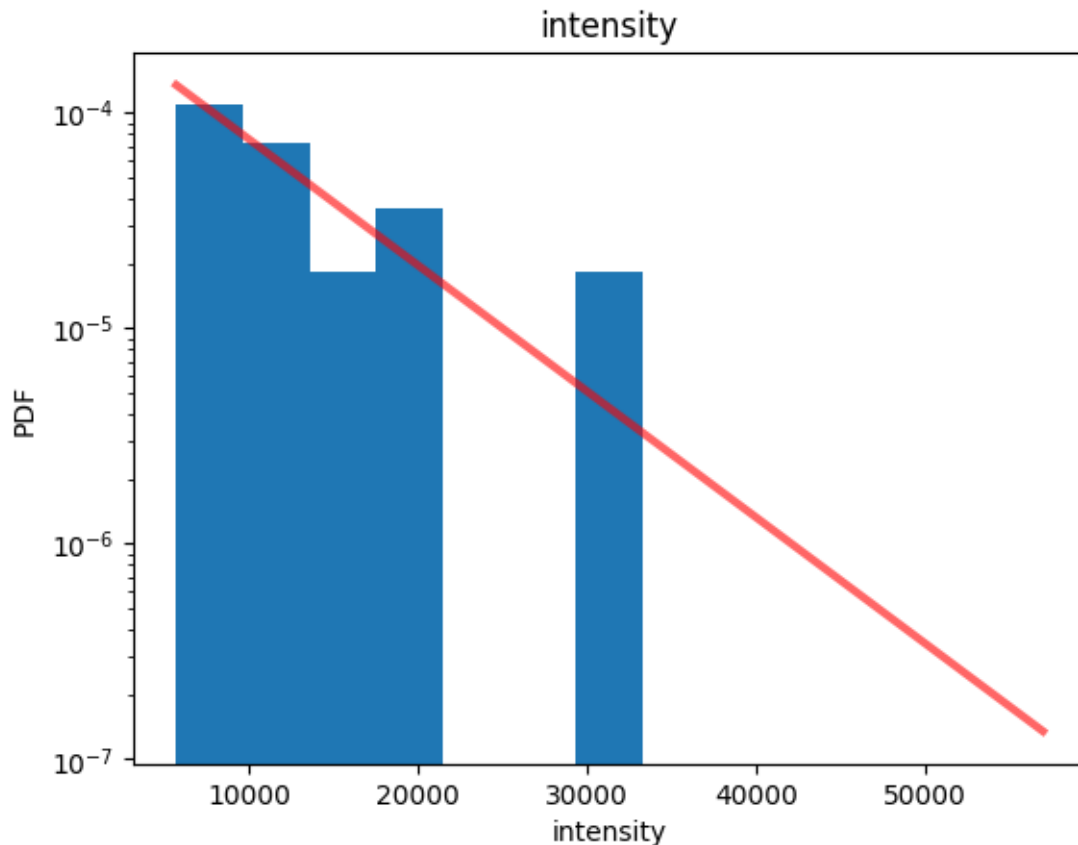
The histogram shows the probability density function of results.

```
lprop.hist(fit=False);
```



Per default the distribution is fitted to an exponential decay.

```
lprop.hist();
```



Fit results (as derived using the lmfit library) are provided in the `distribution_statistics` attribute.

```
lprop.distribution_statistics.parameter_dict()
```

```
{'intensity_loc': 5730.75, 'intensity_scale': 7411.226428571428}
```

```
lprop.results.min()
```

```
intensity    5730.75
dtype: float64
```

### 2.14.3 Fitting different distribution models

Per default the `'with_constraints'` flag is `True` to apply standard fit constraints. This can be set to `false` and other parameters can be passed to the fit function.

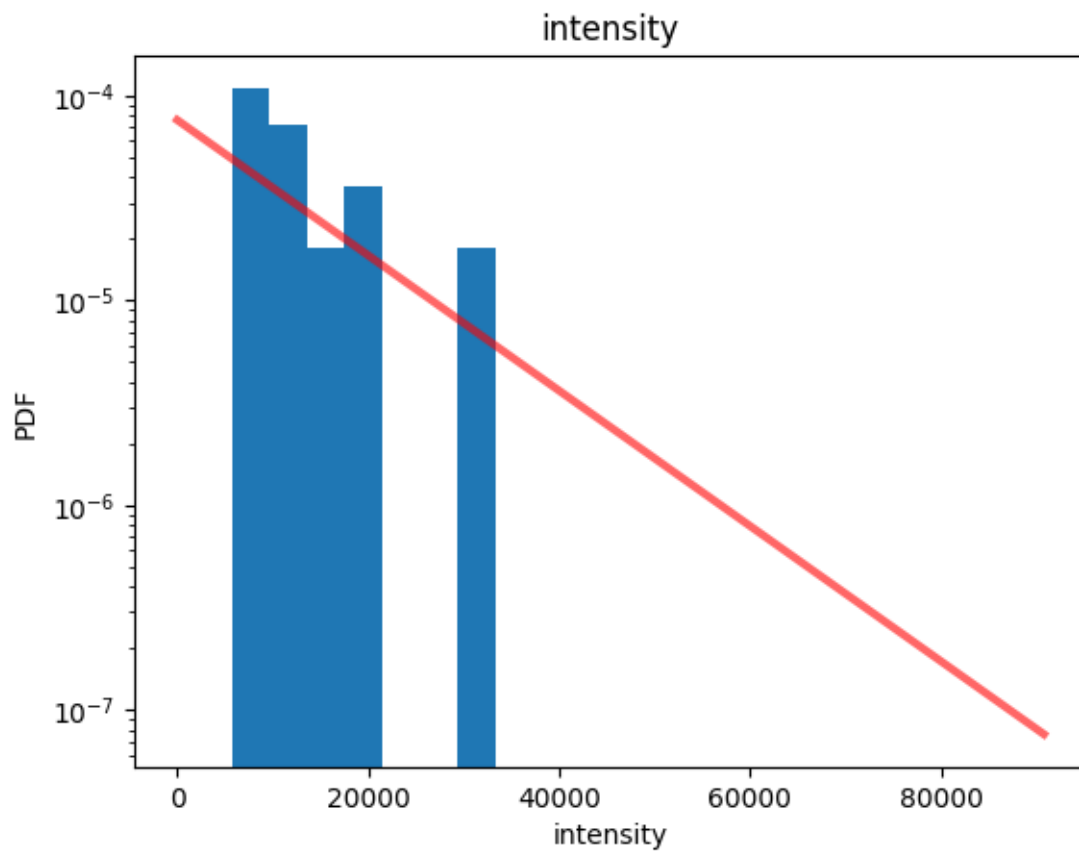
```
lprop.fit_distributions(with_constraints=False, floc=0)
```

```
lprop.distribution_statistics.parameter_dict()
```

```
{'intensity_loc': 0.0, 'intensity_scale': 13141.976428571428}
```

```
lprop.hist(fit=True)
print(lprop.distribution_statistics.parameter_dict())
```

```
{'intensity_loc': 0.0, 'intensity_scale': 13141.976428571428}
```



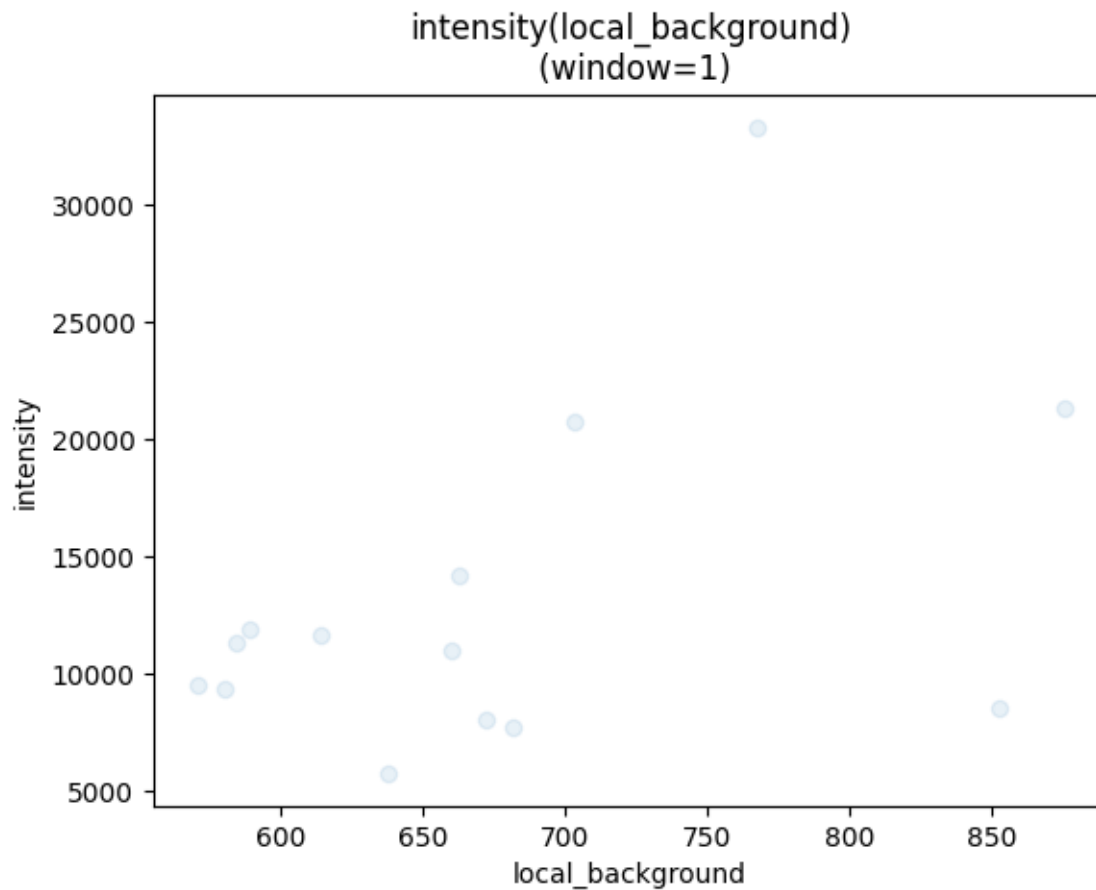
#### 2.14.4 Showing correlations between two properties

By setting the index to another localization property correlations can be shown.

```
lprop = lc.LocalizationProperty(loc_property='intensity', index='local_
↪background').compute(dat)
```

```
lprop.plot(marker='o', linestyle="", alpha=0.1);
```



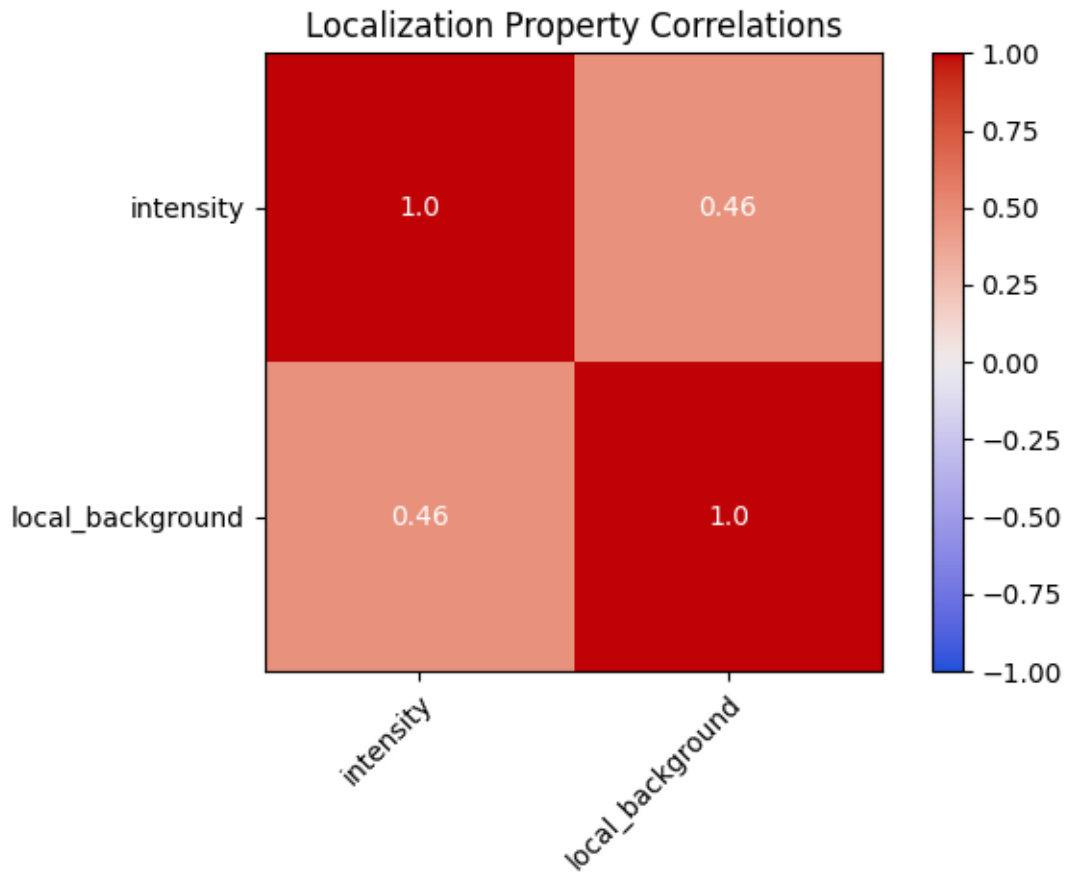


Correlation coefficients can be investigated in more detail using the `LocalizationPropertyCorrelation` class that is just a visualization of `pandas.DataFrame.corr()`.

```
lpcorr = lc.LocalizationPropertyCorrelations(loc_properties=['intensity',  
↪ 'local_background']).compute(dat)  
lpcorr
```

```
LocalizationPropertyCorrelations(loc_properties=['intensity', 'local_  
↪ background'])
```

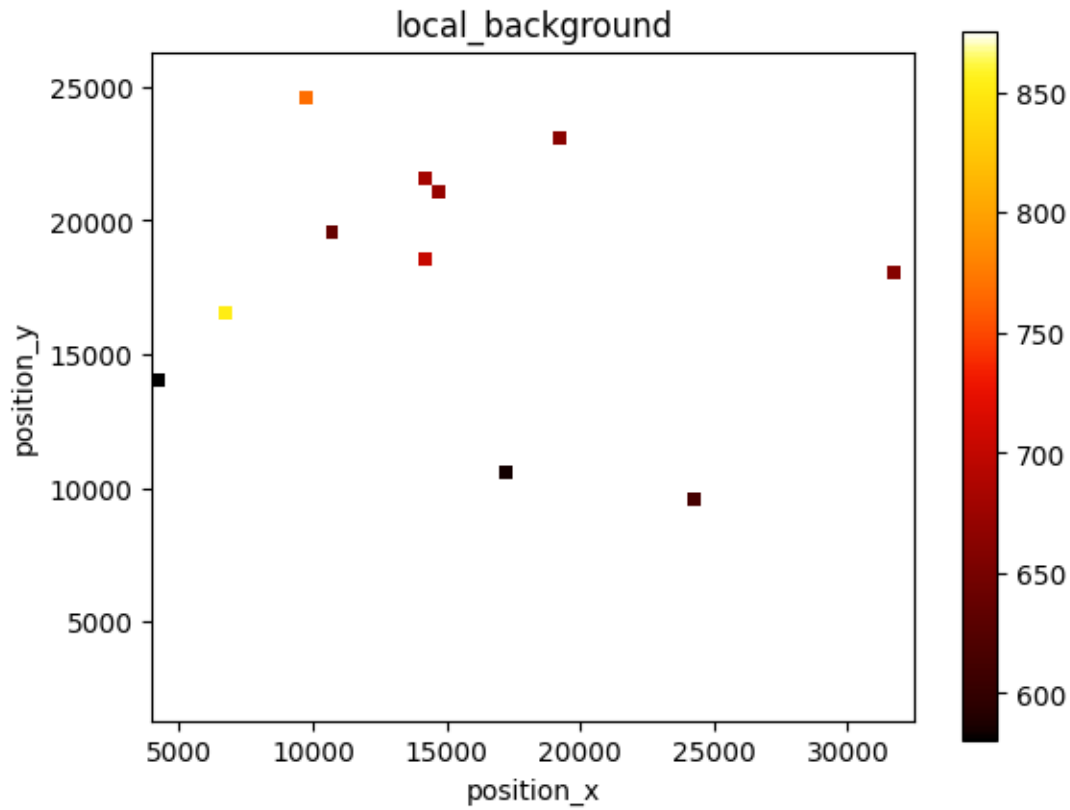
```
lpcorr.plot();
```



#### 2.14.5 2-dimensional distribution of localization properties

In order to investigate a certain localization property in 2D you can just print the image with a color code representing the mean value of the chosen localization property in each bin.

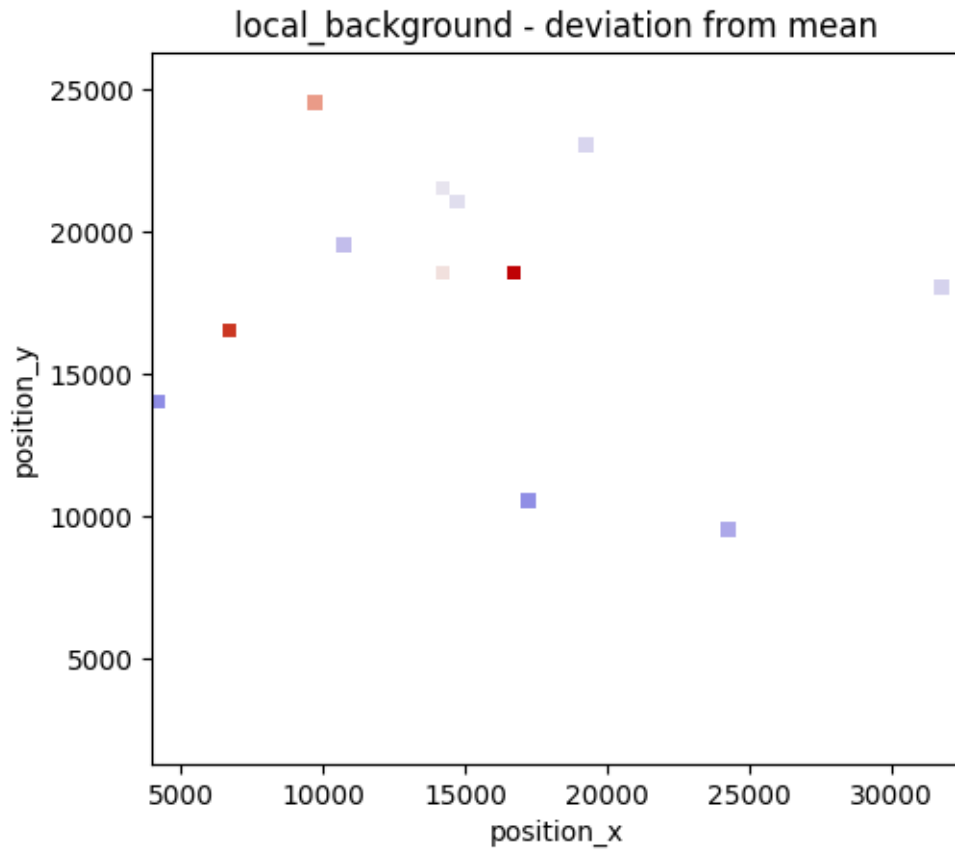
```
lc.render_2d_mpl(dat, other_property='local_background', bin_size=500);
```



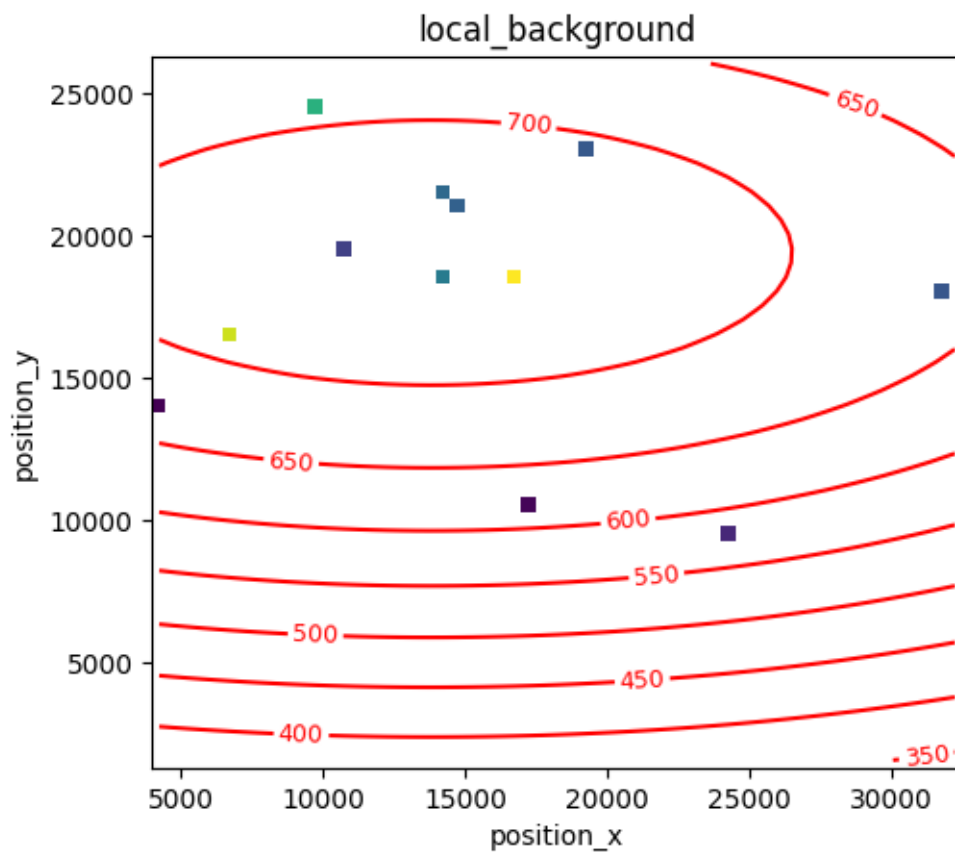
Otherwise use a specific class to analyse localization properties in 2d. Per default a bimodal normal distribution is fitted. This can e.g. help to check on even illumination during the recording.

```
lprop2d = lc.LocalizationProperty2d(loc_properties=None, other_property=  
↪ 'local_background', bin_size=500).compute(dat)
```

```
lprop2d.plot_deviation_from_mean();
```



```
lprop2d.plot(colors="r");
```



```
lprop2d.report()
```

Fit results for:

```
[[Model]]
  Model(model_function)
[[Fit Statistics]]
  # fitting method   = leastsq
  # function evals   = 43
  # data points      = 12
  # variables        = 5
  chi-square         = 75179.6882
  reduced chi-square = 10739.9555
  Akaike info crit   = 114.912757
  Bayesian info crit = 117.337290
  R-squared           = 0.25569453
[[Variables]]
  amplitude: 732.468389 +/- 47.0045236 (6.42%) (init = 875.461)
  center_x:  13822.4142 +/- 14800.9733 (107.08%) (init = 18245.59)
  center_y:  19401.6868 +/- 3825.17126 (19.72%) (init = 13803.61)
  sigma_x:   42169.8059 +/- 52664.9262 (124.89%) (init = 7125)
  sigma_y:   15484.6869 +/- 8624.45209 (55.70%) (init = 6250)
[[Correlations]] (unreported correlations are < 0.250)
  C(center_y, sigma_y) = +0.7364
  C(center_x, sigma_x) = -0.6514
  C(amplitude, sigma_y) = -0.6030
  C(amplitude, sigma_x) = -0.5552
  C(amplitude, center_y) = -0.4075
  C(center_y, sigma_x) = +0.3845
  C(amplitude, center_x) = +0.3313
  C(sigma_x, sigma_y) = +0.3140
  C(center_x, sigma_y) = -0.2882
  C(center_x, center_y) = -0.2544
Maximum fit value in image: 731.334
Minimum fit value in image: 580.351
Fit value variation over image range: 0.21
```

## 2.15 Tutorial about analyzing blink statistics

SMLM depends critically on the fluorescence intermittency or, in other words, the blinking of fluorescence dyes. To characterize blinking properties you can compute on- and off-periods from clustered localizations assuming that they originate from the same fluorophore.

```
from pathlib import Path

%matplotlib inline

import numpy as np
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt
import scipy.stats as stats

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.15.1 Synthetic data

We use synthetic data that represents localizations from a single fluorophore being normal distributed in space and emitting at a constant intensity. We assume that the on- and off-times in units of frames are distributed like a geometric distribution with a mean on\_period `mean_on` and a mean off\_period `mean_off`. Typically a geometric distribution is parameterized by a variable `p` with  $p = 1 / \text{mean}$ .

```
rng = np.random.default_rng(seed=1)
```

```
n_samples = 10_000
mean_on = 5
mean_off = 20

on_periods = stats.geom.rvs(p=1/mean_on, size=n_samples, random_state=rng)
off_periods = stats.geom.rvs(p=1/mean_off, size=n_samples, random_state=rng)
```

On- and off-times are converted in a series of frame numbers at which a localization was detected.

```
def periods_to_frames(on_periods, off_periods):
    """
    Convert on- and off-periods into a series of increasing frame values.
    """
    on_frames = np.arange(np.sum(on_periods))
    cumsums = np.r_[0, np.cumsum(off_periods)[:len(on_periods)]]
    add_on = np.repeat(cumsums, on_periods)
    frames = on_frames + add_on
    return frames[:len(on_periods)]
```

```
frames = periods_to_frames(on_periods, off_periods)
```

```
offspring = [rng.normal(loc=0, scale=10, size=(n_samples, 2))]
locdata = lc.simulate_cluster(centers=[(50, 50)], region=[(0, 100), (0, 100)],
    ↳ offspring=offspring, clip=False, shuffle=False, seed=rng)
locdata.dataframe['intensity'] = 1
locdata.dataframe['frame'] = frames
```

(continues on next page)

(continued from previous page)

```
locdata = lc.LocData.from_dataframe(dataframe=locdata.data)

print('Data head:')
print(locdata.data.head(), '\n')
print('Summary:')
locdata.print_summary()
print('Properties:')
print(locdata.properties)
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

Data head:

|   | position_x | position_y | cluster_label | intensity | frame |
|---|------------|------------|---------------|-----------|-------|
| 0 | 50.507149  | 36.698297  | 0             | 1         | 0     |
| 1 | 44.539062  | 27.969248  | 0             | 1         | 1     |
| 2 | 56.876954  | 42.152826  | 0             | 1         | 2     |
| 3 | 38.878363  | 43.999842  | 0             | 1         | 3     |
| 4 | 61.001644  | 54.231150  | 0             | 1         | 4     |

Summary:

identifier: "2"

comment: ""

source: DESIGN

state: RAW

element\_count: 10000

frame\_count: 10000

creation\_time {

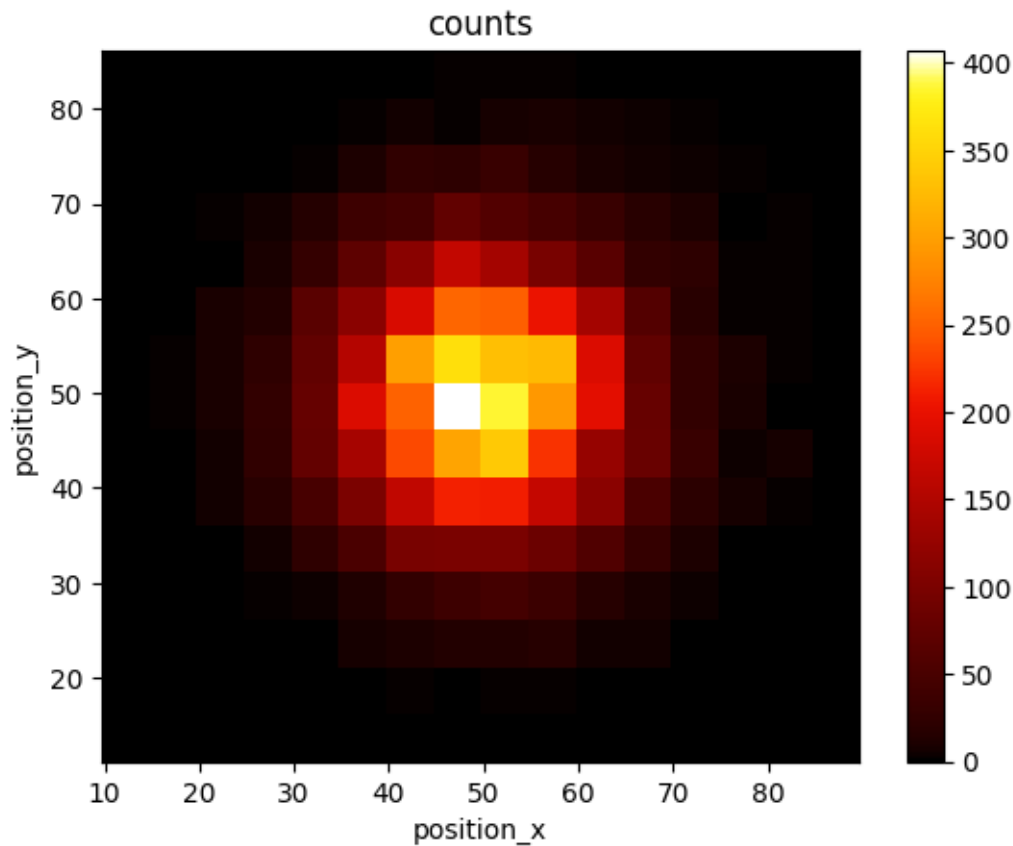
2024-03-14T11:04:50.982889Z

}

Properties:

```
{'localization_count': 10000, 'position_x': 49.986893963647944, 'uncertainty_x'
↪: 0.09881426457266232, 'position_y': 49.84659129517861, 'uncertainty_y': 0.
↪10007238629130723, 'intensity': 10000, 'frame': 0, 'region_measure_bb':
↪6466.041774468444, 'localization_density_bb': 1.5465411992056104,
↪'subregion_measure_bb': 322.017777257221}
```

```
lc.render_2d(locdata, bin_size=5);
```



### 2.15.2 Blinking statistics

To determine on- and off-times for the observed blink events use the analysis class `BlinkStatistics`.

```
bs = lc.BlinkStatistics(memory=0, remove_heading_off_periods=False).
    ↪ compute(locdata)
bs
```

```
BlinkStatistics(memory=0, remove_heading_off_periods=False)
```

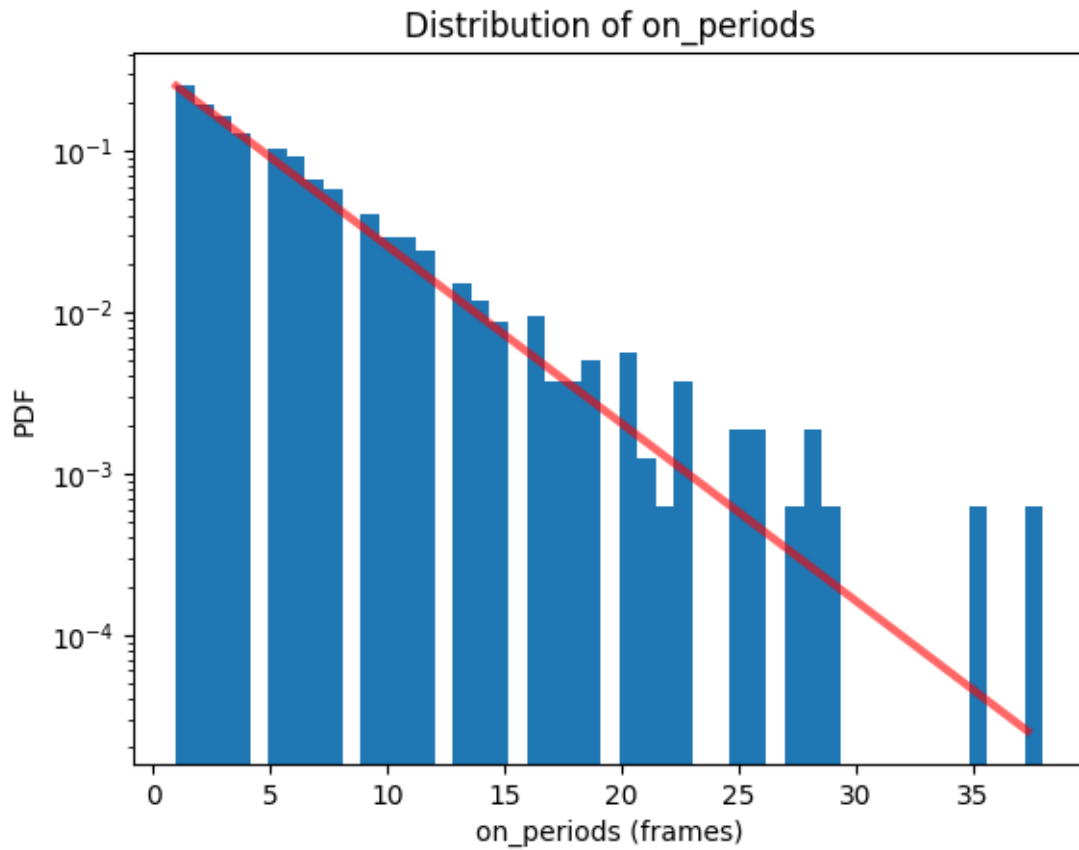
```
bs.results.keys()
```

```
dict_keys(['on_periods', 'on_periods_frame', 'off_periods', 'off_periods_frame',
    ↪ 'on_periods_indices'])
```

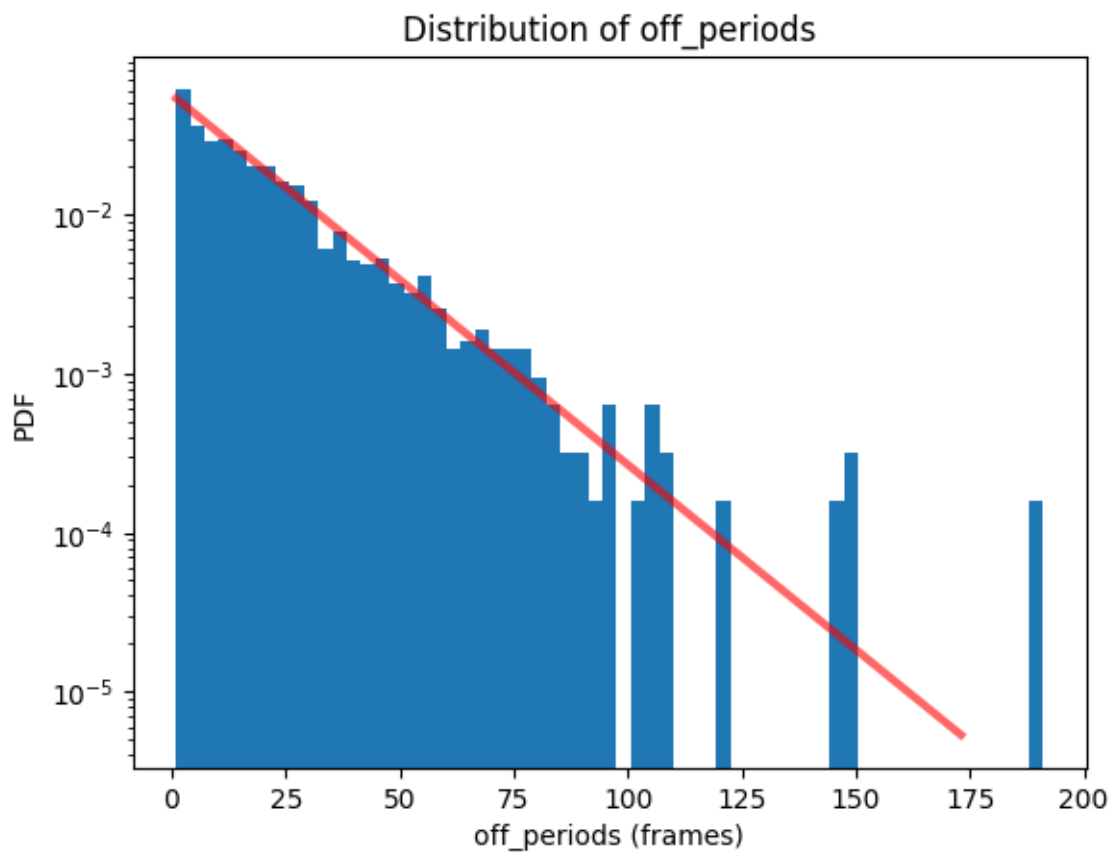
When plotting the histogram an exponential distribution is fitted by default.

```
bs.hist(data_identifier='on_periods');
```





```
bs.hist(data_identifier='off_periods');
```



```
bs.distribution_statistics
```

```
{'on_periods': _DistributionFits(analysis_class=BlinkStatistics,
↪distribution=expon_gen, data_identifier=on_periods),
'off_periods': _DistributionFits(analysis_class=BlinkStatistics,
↪distribution=expon_gen, data_identifier=off_periods)}
```

The fit results provide loc and scale parameter (see `scipy.stats` documentation). For `loc = 0`, `scale` describes the mean of the distribution..

```
bs.distribution_statistics['on_periods'].parameter_dict()
```

```
{'on_periods_loc': 1.0, 'on_periods_scale': 3.9455984174085064}
```

```
bs.distribution_statistics['off_periods'].parameter_dict()
```

```
{'off_periods_loc': 1.0, 'off_periods_scale': 18.682830282038594}
```

Due to the default setting for the scaling parameter `loc` the mean on\_period is `on_periods_scale + on_periods_loc` in agreement with our input value.

### 2.15.3 Geometric distribution

We can compare this with a geometric distribution that is estimated from the observed mean on\_period `on_periods_mean`.

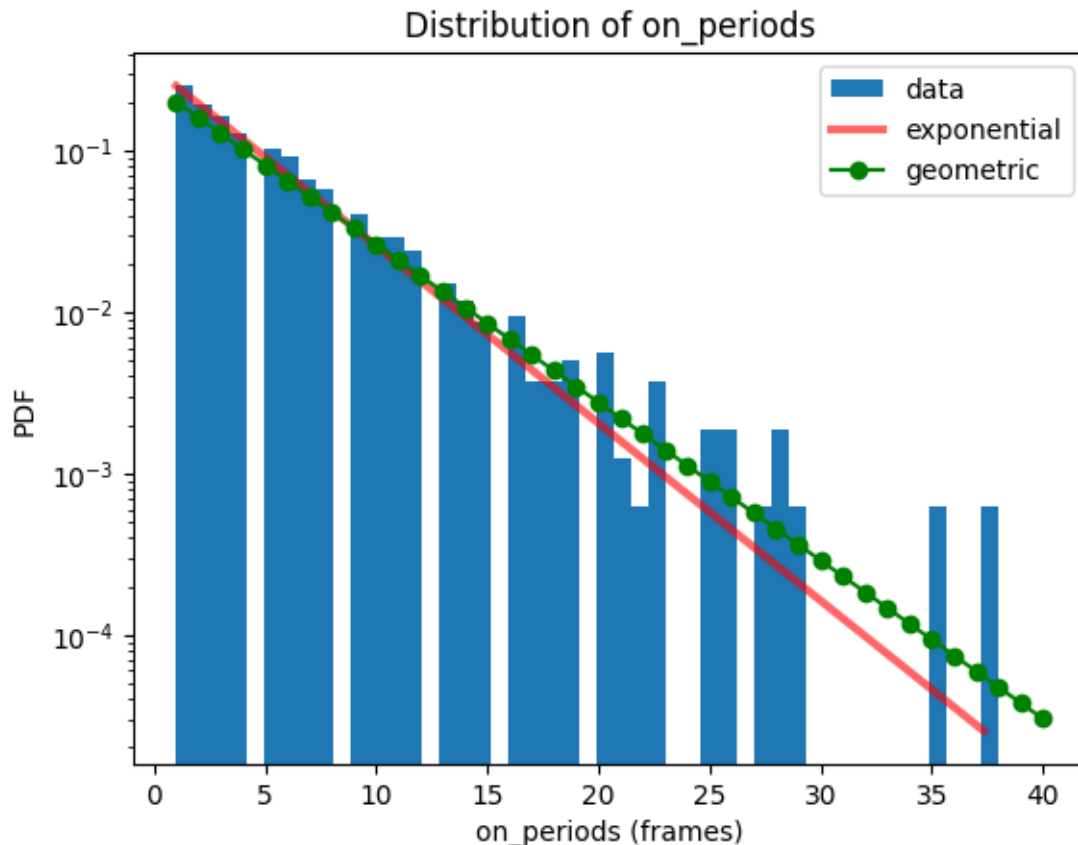
```
on_periods_mean = bs.results['on_periods'].mean()
on_periods_mean.round(2)
```

```
4.95
```

```
off_periods_mean = bs.results['off_periods'].mean()
off_periods_mean.round(2)
```

```
19.68
```

```
# test result
x = np.arange(stats.geom.ppf(0.01, 1/on_periods_mean), stats.geom.ppf(0.9999,
↪1/on_periods_mean))
y = stats.geom.pmf(x, 1/on_periods_mean)
fig, ax = plt.subplots()
bs.hist(data_identifier='on_periods', fit=False, label='data')
bs.distribution_statistics['on_periods'].plot(label='exponential')
ax.plot(x, y, '-go', label='geometric')
ax.set_yscale('log')
ax.legend(loc='best')
plt.show()
```



## 2.16 Tutorial about coordinate-based colocalization

Coordinate-based colocalization of two selections is computed as introduced by Malkusch et al. (1). A local density of locdata A is compared with the local density of locdata B within a varying radius for each localization in selection A. Local densities at various radii are compared by Spearman-rank-correlation and weighted by an exponential factor including the Euclidean distance to the nearest neighbor for each localization. The colocalization coefficient can take a value between -1 and 1 with -1 indicating anti-correlation (i.e. nearby localizations of selection B), 0 no colocalization, 1 strong colocalization.

The colocalization coefficient is provided as property for each localization within the corresponding dataset. The property key refers to colocalizing locdata A with locdata B.

```
from pathlib import Path

%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as colors

import locan as lc
```

```
/tmp/ipykernel_979/1945727434.py:6: DeprecationWarning:
```

(continues on next page)

(continued from previous page)

```

Pyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳ 54466

import pandas as pd

```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```

Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6

```

### 2.16.1 Synthetic data

Provide synthetic data made of clustered localizations that are normal distributed around their center positions.

```
rng = np.random.default_rng(seed=1)
```

```

def set_centers(n_centers_1d=3, feature_range = (0, 1000)):
    dist = (feature_range[1] - feature_range[0])/(n_centers_1d + 1)
    centers = np.mgrid[feature_range[0] + dist : feature_range[1] : dist,
↳ feature_range[0] + dist : feature_range[1] : dist].reshape(2, -1).T
    return centers

```

```

n_centers_1d = 3
feature_range = (0, 1000)
centers = set_centers(n_centers_1d, feature_range)

```

```

offspring = rng.normal(loc=0, scale=20, size=(len(centers), 100, 2))
locdata = lc.simulate_cluster(centers=centers, region=[feature_range] * 2,
↳ offspring=offspring, clip=True, shuffle=True, seed=rng)

locdata.print_summary()

```

```

Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
identifier: "1"
comment: ""

```

(continues on next page)

(continued from previous page)

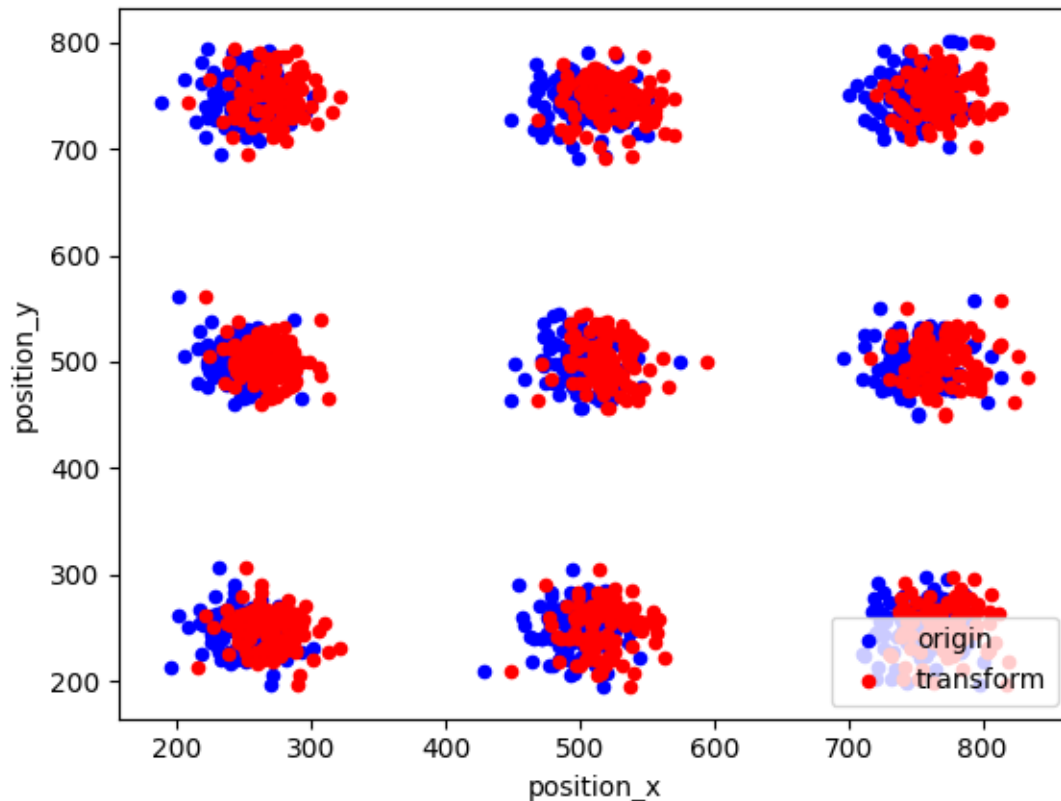
```
source: SIMULATION
state: RAW
element_count: 900
frame_count: 0
creation_time {
  2024-03-14T11:05:27.455710Z
}
```

A second dataset is provided by shifting the first dataset by an offset.

```
locdata_trans = lc.transform_affine(locdata, offset=(20,0))
locdata_trans.print_summary()
```

```
identifier: "2"
comment: ""
source: SIMULATION
state: MODIFIED
element_count: 900
frame_count: 0
creation_time {
  2024-03-14T11:05:27.455710Z
}
modification_time {
  2024-03-14T11:05:27.470967Z
}
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
locdata.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Blue',
↪ label='origin')
locdata_trans.data.plot.scatter(x='position_x', y='position_y', ax=ax, color=
↪ 'Red', label='transform')
plt.show()
```



### 2.16.2 CBC computation

```
cbc = lc.CoordinateBasedColocalization(radius=100, n_steps=10).
    ↪ compute(locdata=locdata, other_locdata=locdata_trans)
```

```
cbc.results.head()
```

```
colocalization_cbc_2
0          0.382530
1          0.280571
2          0.036730
3          0.152874
4          0.385790
```

### 2.16.3 Results

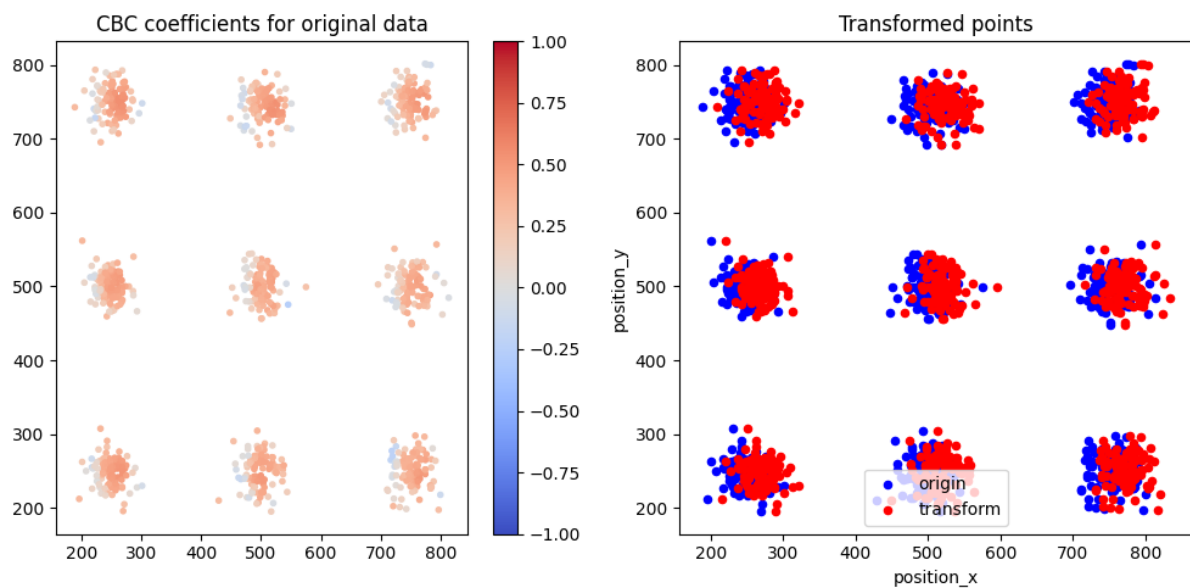
```
points = locdata.coordinates
color = cbc.results['colocalization_cbc_2'].values

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
cax = axes[0].scatter(x=points[:,0], y=points[:,1], marker='.', c=color, cmap=
    ↪ 'coolwarm', norm= colors.Normalize(-1., 1.), label='points')
axes[0].set_title('CBC coefficients for original data')
```

(continues on next page)

(continued from previous page)

```
# axes[1].scatter(x=points[:,0], y=points[:,1], marker='.', color='Blue', label=
    ↪ 'points')
# axes[1].scatter(x=points_trans[:,0], y=points_trans[:,1], marker='o', color=
    ↪ 'Red', label='transformed points')
locdata.data.plot.scatter(x='position_x', y='position_y', ax=axes[1], color=
    ↪ 'Blue', label='origin')
locdata_trans.data.plot.scatter(x='position_x', y='position_y', ax=axes[1],
    ↪ color='Red', label='transform')
axes[1].set_title('Transformed points')
plt.colorbar(cax, ax=axes[0])
plt.tight_layout()
plt.show()
```



## 2.16.4 CBC for various shifts

```
n_centers_1d = 3
feature_range = (0, 2000)
centers = set_centers(n_centers_1d, feature_range)
```

```
offspring = rng.normal(loc=0, scale=20, size=(len(centers), 100, 2))
locdata = lc.simulate_cluster(centers=centers, region=[feature_range] * 2,
    ↪ offspring=offspring, clip=True, shuffle=True, seed=rng)
```

```
locdata.print_summary()
```

```
identifier: "3"
comment: ""
source: SIMULATION
state: RAW
```

(continues on next page)

(continued from previous page)

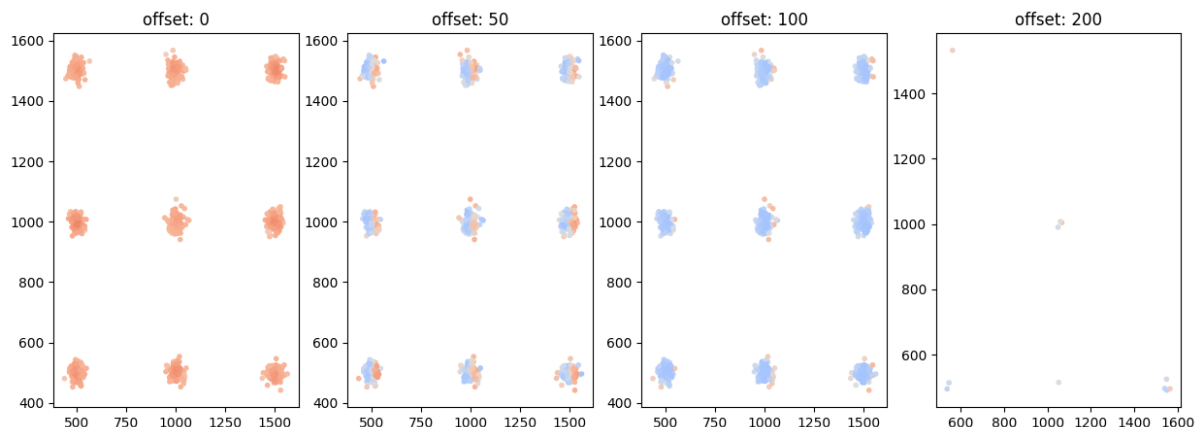
```
element_count: 900
frame_count: 0
creation_time {
  2024-03-14T11:05:29.468484Z
}
```

```
offsets = [0, 50, 100, 200]
locdata_trans_list = [lc.transform_affine(locdata, offset=(offset, 0)) for
↳ offset in offsets]
```

```
cbc_list = [lc.CoordinateBasedColocalization(radius=100, n_steps=10).
↳ compute(locdata=locdata, other_locdata=other_locdata) for other_locdata in
↳ locdata_trans_list]
```

```
points = locdata.coordinates
n_rows = 1
n_cols = 4

fig = plt.figure(figsize=(15, 5))
for n, (cbc, offset) in enumerate(zip(cbc_list, offsets)):
    ax = fig.add_subplot(n_rows, n_cols, n+1)
    color = cbc.results.iloc[:, 0].values
    ax.scatter(x=points[:,0], y=points[:,1], marker='.', c=color, cmap=
↳ 'coolwarm', norm=colors.Normalize(-1., 1.))
    ax.set_title(f'offset: {offset}')
plt.show()
```





### 2.16.5 CBC on various length scales (for small cluster)

```
n_centers_1d = 3
feature_range = (0, 2000)
centers = set_centers(n_centers_1d, feature_range)
```

```
offspring = rng.normal(loc=0, scale=20, size=(len(centers), 100, 2))
locdata = lc.simulate_cluster(centers=centers, region=[feature_range] * 2,
↳offspring=offspring, clip=True, shuffle=True, seed=rng)
```

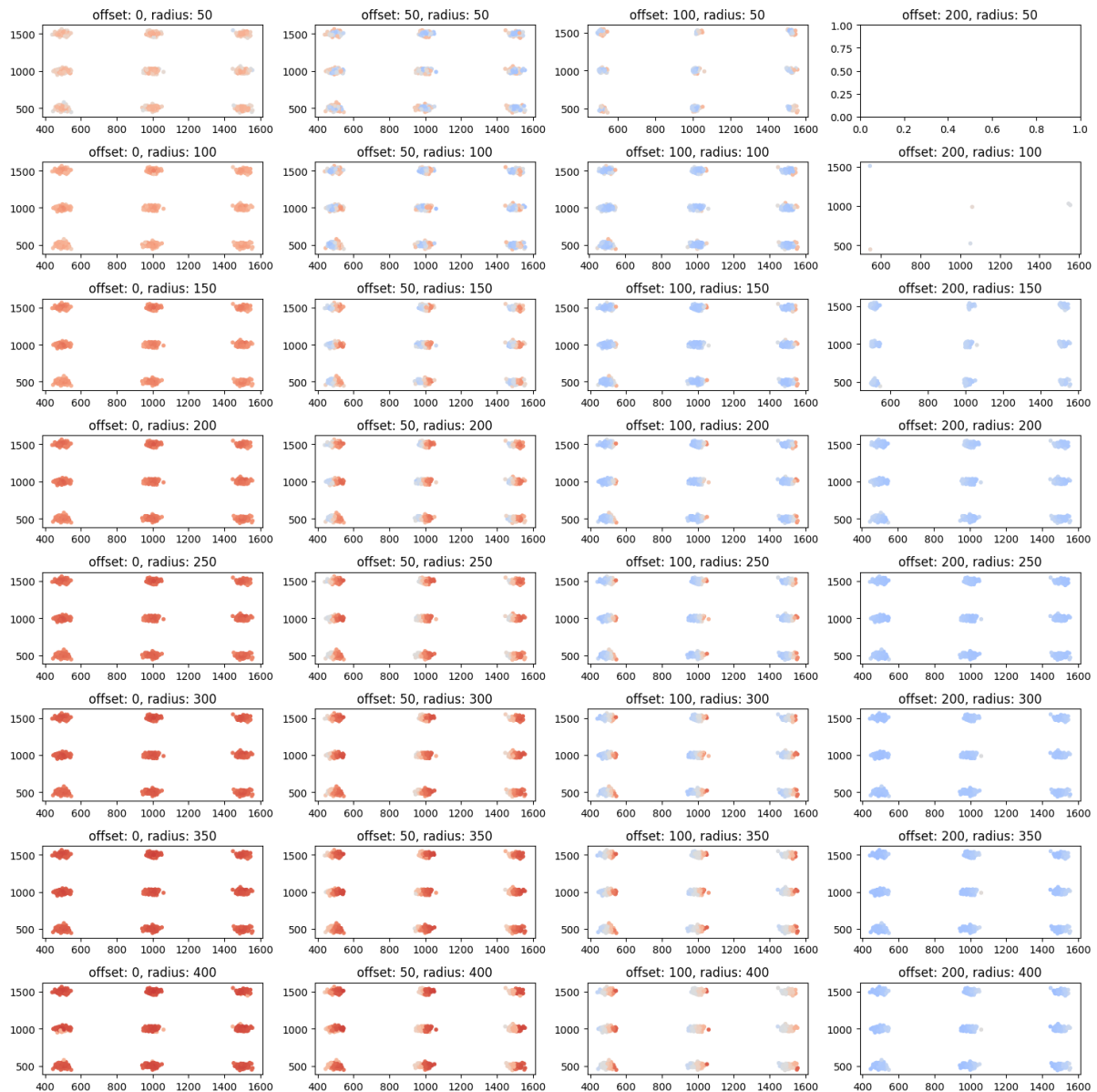
```
offsets = [0, 50, 100, 200]
locdata_trans_list = [lc.transform_affine(locdata, offset=(offset,0)) for
↳offset in offsets]
```

```
radii = [50, 100, 150, 200, 250, 300, 350, 400]
cbc_list = [lc.CoordinateBasedColocalization(radius=radius, n_steps=10).
↳compute(locdata=locdata, other_locdata=other_locdata) for radius in radii
    for other_locdata in locdata_trans_list
]
```

```
points = locdata.coordinates
params = [(radius, offset) for radius in radii for offset in offsets]

n_rows = len(radii)
n_cols = len(offsets)

fig = plt.figure(figsize=(15, 15))
for n, (cbc, (radius, offset)) in enumerate(zip(cbc_list, params)):
    ax = fig.add_subplot(n_rows, n_cols, n+1)
    color = cbc.results.iloc[:, 0].values
    if not all(np.isnan(color)):
        ax.scatter(x=points[:,0], y=points[:,1], marker='.', c=color, cmap=
↳'coolwarm', norm=colors.Normalize(-1., 1.))
    ax.set_title(f'offset: {offset}, radius: {radius}')
plt.tight_layout()
plt.show()
```



### 2.16.6 CBC on various length scales (for larger cluster)

```
n_centers_1d = 3
feature_range = (0, 10_000)
centers = set_centers(n_centers_1d, feature_range)
```

```
offspring = rng.normal(loc=0, scale=100, size=(len(centers), 100, 2))
locdata = lc.simulate_cluster(centers=centers, region=[feature_range] * 2,
    ↳offspring=offspring, clip=True, shuffle=True, seed=rng)
```

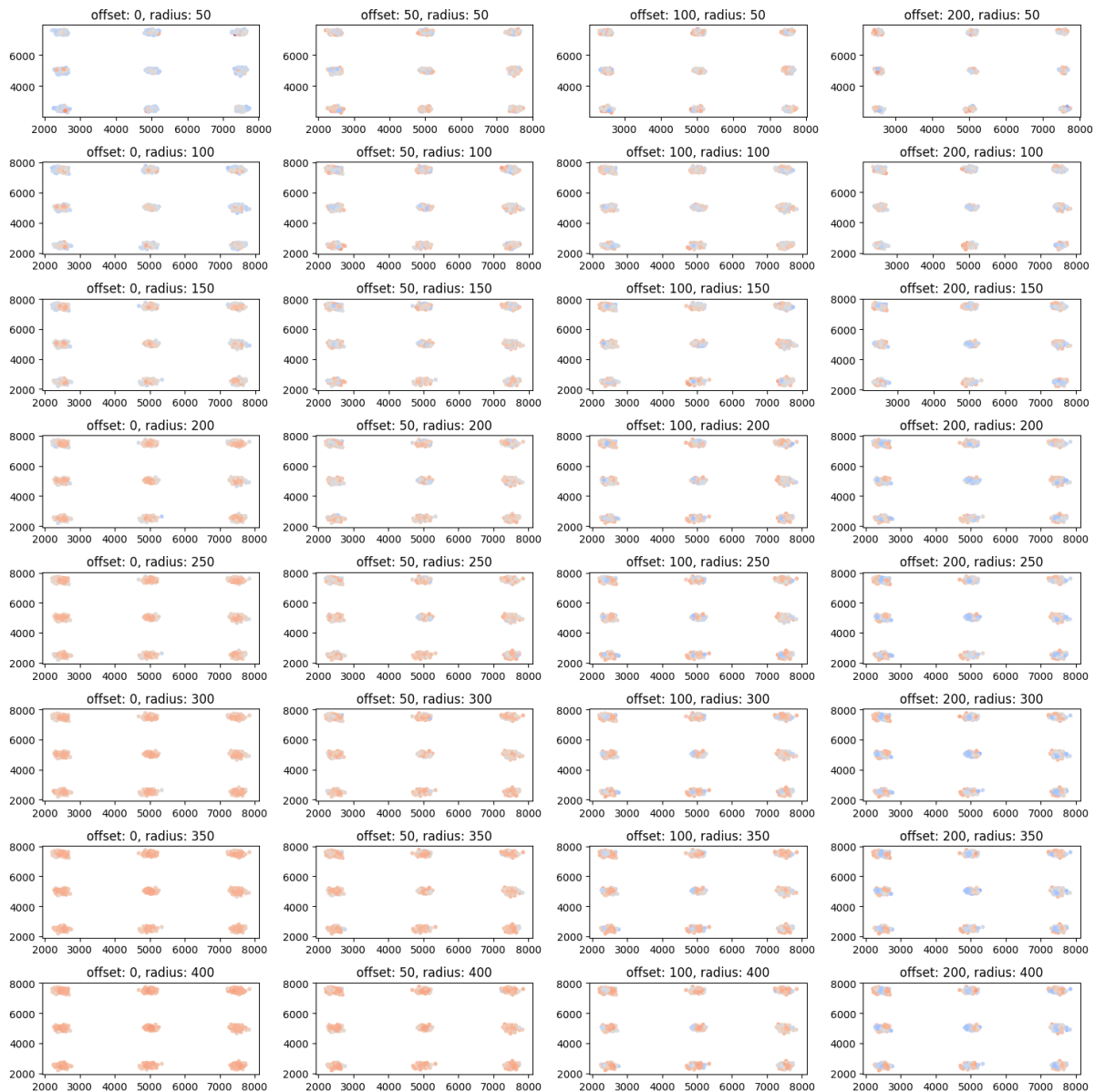
```
offsets = [0, 50, 100, 200]
locdata_trans_list = [lc.transform_affine(locdata, offset=(offset,0)) for
    ↳offset in offsets]
```

```
radii = [50, 100, 150, 200, 250, 300, 350, 400]
cbc_list = [lc.CoordinateBasedColocalization(radius=radius, n_steps=10).
    ↪compute(locdata=locdata, other_locdata=other_locdata) for radius in radii
            for other_locdata in locdata_trans_list
            ]
```

```
points = locdata.coordinates
params = [(radius, offset) for radius in radii for offset in offsets]

n_rows = len(radii)
n_cols = len(offsets)

fig = plt.figure(figsize=(15, 15))
for n, (cbc, (radius, offset)) in enumerate(zip(cbc_list, params)):
    ax = fig.add_subplot(n_rows, n_cols, n+1)
    color = cbc.results.iloc[:, 0].values
    ax.scatter(x=points[:,0], y=points[:,1], marker='.', c=color, cmap=
    ↪'coolwarm', norm=colors.Normalize(-1., 1.))
    ax.set_title(f'offset: {offset}, radius: {radius}')
plt.tight_layout()
plt.show()
```



## 2.17 Tutorial about drift analysis and correction

Lateral drift correction is useful in most SMLM experiments. To determine the amount of drift a method based on image cross-correlation or an iterative closest point algorithm can be applied.

We demonstrate drift analysis and correction on simulated data.

```
from pathlib import Path

%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

Locan:

version: 0.20.0.dev41+g755b969

Python:

version: 3.11.6

### 2.17.1 Synthetic data

We use synthetic data that follows a Neyman-Scott spatial distribution (blobs). The intensity values are exponentially distributed and the number of localizations per frame follows a Poisson distribution:

```
rng = np.random.default_rng(seed=1)
```

```
intensity_mean = 1000
```

```
localizations_per_frame_mean = 3
```

```
dat_blob = lc.simulate_Thomas(parent_intensity=1e-4, region=((0, 1000), (0, 1000)), cluster_mu=1000, cluster_std=10, seed=rng)
```

```
dat_blob.dataframe['intensity'] = stats.expon.rvs(scale=intensity_mean, size=len(dat_blob), loc=500)
```

```
dat_blob.dataframe['frame'] = lc.simulate_frame_numbers(n_samples=len(dat_blob), lam=localizations_per_frame_mean, seed=rng)
```

```
dat_blob = lc.LocData.from_dataframe(dataframe=dat_blob.data)
```

```
print('Data head:')
```

```
print(dat_blob.data.head(), '\n')
```

```
print('Summary:')
```

```
dat_blob.print_summary()
```

```
print('Properties:')
```

```
print(dat_blob.properties)
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
```

```
[Open3D INFO] WebRTC GUI backend enabled.
```

```
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

Data head:

|   | position_x | position_y | cluster_label | intensity   | frame |
|---|------------|------------|---------------|-------------|-------|
| 0 | 915.763326 | 465.421770 | 28            | 3371.713793 | 0     |
| 1 | 729.786200 | 81.672520  | 56            | 814.920825  | 1     |
| 2 | 869.708004 | 9.266125   | 41            | 3385.119344 | 1     |
| 3 | 521.743493 | 54.508814  | 14            | 1397.009488 | 2     |
| 4 | 430.972011 | 375.461311 | 61            | 1879.050494 | 3     |

Summary:

identifier: "2"

(continues on next page)

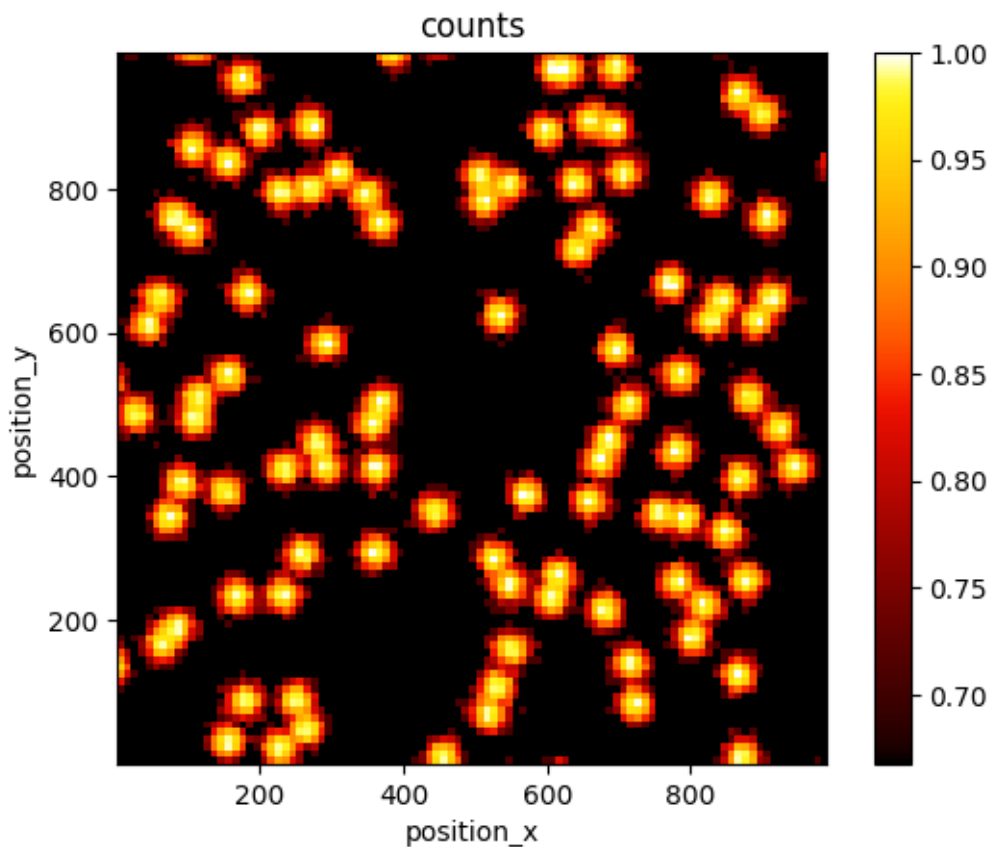
(continued from previous page)

```
comment: ""
source: DESIGN
state: RAW
element_count: 98201
frame_count: 31016
creation_time {
  2024-03-14T11:04:58.105764Z
}
```

**Properties:**

```
{'localization_count': 98201, 'position_x': 495.13588743804684, 'uncertainty_x'
→': 0.8971814298938875, 'position_y': 507.65355920347866, 'uncertainty_y': 0.
→8964573205917536, 'intensity': 146845730.2327197, 'frame': 0, 'region_
→measure_bb': 999959.2361869529, 'localization_density_bb': 0.
→09820500321039116, 'subregion_measure_bb': 3999.918472062067}
```

```
lc.render_2d(dat_blob, bin_size=10, rescale='equal');
```

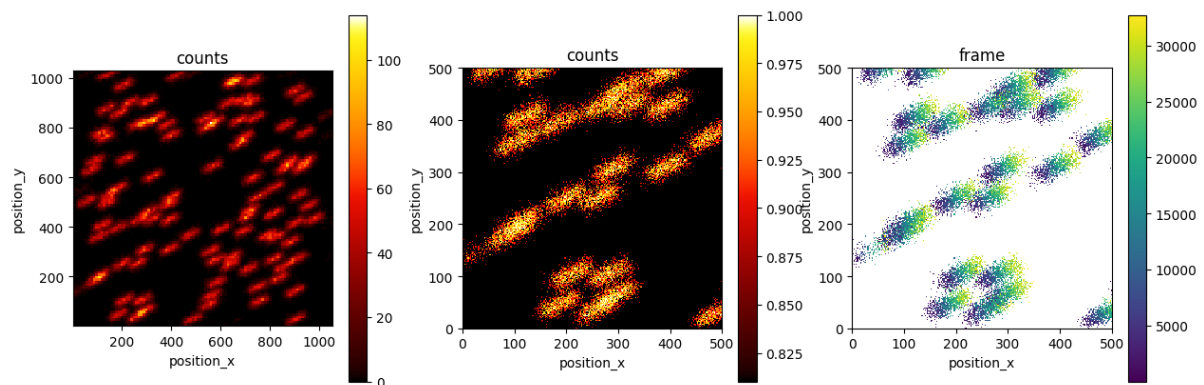


### 2.17.2 Add linear drift

We add linear drift with a velocity given in length units per frame.

```
dat_blob_with_drift = lc.add_drift(dat_blob, velocity=(0.002, 0.001),
    ↪ seed=rng)
```

```
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
lc.render_2d(dat_blob_with_drift, ax=axes[0], bin_size=10);
lc.render_2d(dat_blob_with_drift, ax=axes[1], bin_size=2, rescale='equal',
    ↪ bin_range=((0, 500),(0, 500)));
lc.render_2d_mpl(dat_blob_with_drift, ax=axes[2], other_property='frame', bin_
    ↪ size=2, bin_range=((0, 500),(0, 500)), cmap='viridis');
```



### 2.17.3 Estimate RMS errors

Knowing the ground truth, you can define a root mean squared error between the original localization coordinates and those after drift and later after correction.

```
def rmse(locdata, other_locdata):
    return np.sqrt(np.mean(np.square(np.subtract(locdata.coordinates, other_
    ↪ locdata.coordinates)), axis=0))
```

```
rmse(dat_blob, dat_blob_with_drift).round(2)
```

```
array([37.76, 18.88])
```

### 2.17.4 Estimate drift

Drift can be estimated by comparing different chunks of successive localizations using either an “iterative closest point” algorithm or a “cross-correlation” algorithm. Per default, the icp algorithm is applied.

```
%%time
drift = lc.Drift(chunk_size=10_000, target='first', method='icp').compute(dat_
    ↪ blob_with_drift)
```

```
CPU times: user 7.81 s, sys: 31.2 ms, total: 7.84 s
Wall time: 4.35 s
```

Transformations to register the different data chunks are represented by a transformation matrix and a transformation offset that together specify an affine transformation. The transformation parameters are kept under the `transformations` attribute.

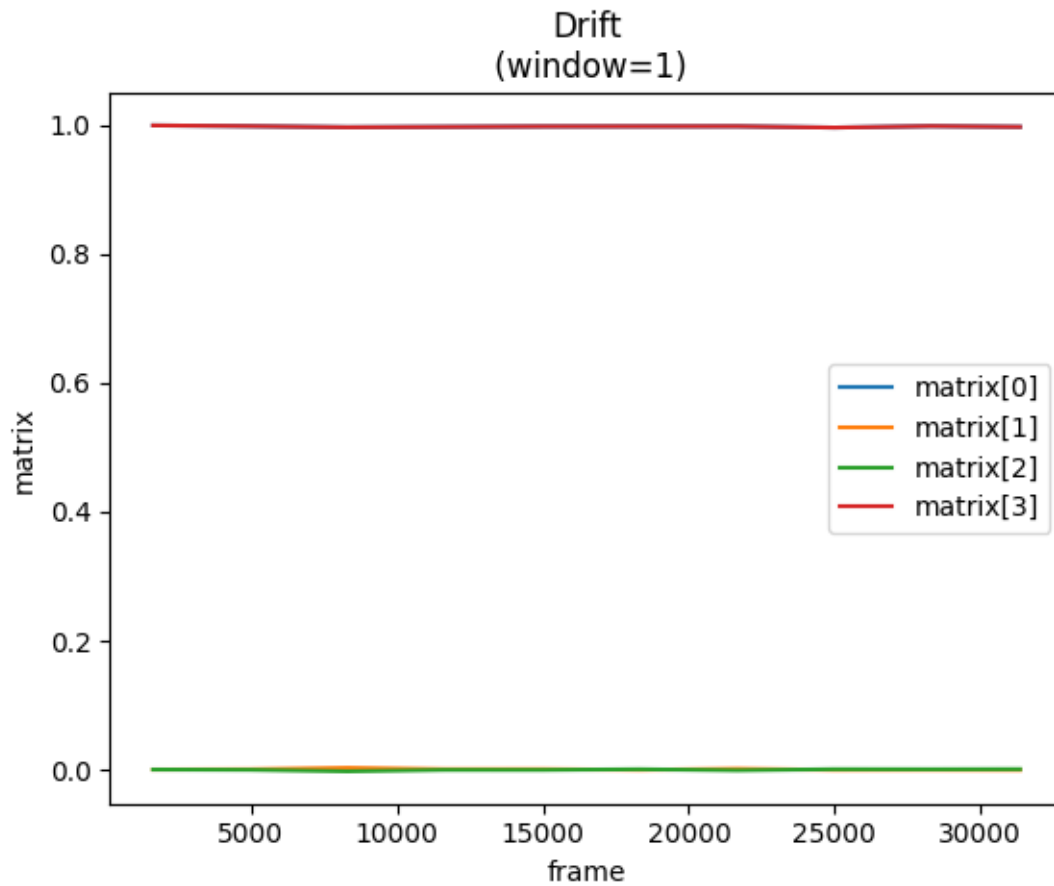
```
drift.transformations
```

```
[Transformation(matrix=array([[1., 0.],
                               [0., 1.]]), offset=array([0., 0.])),
 Transformation(matrix=array([[ 9.98792443e-01,  4.43801557e-04],
                               [-4.43801557e-04,  9.98792443e-01]]), offset=array([-6.56553239, -2.
↪7171172 ])),
 Transformation(matrix=array([[ 0.99716405,  0.00268322],
                               [-0.00268322,  0.99716405]]), offset=array([-13.15345296, -4.
↪05470652])),
 Transformation(matrix=array([[ 9.97764214e-01,  5.81845172e-04],
                               [-5.81845172e-04,  9.97764214e-01]]), offset=array([-19.46087507, -8.
↪60939708])),
 Transformation(matrix=array([[ 9.98398817e-01,  6.25680278e-04],
                               [-6.25680278e-04,  9.98398817e-01]]), offset=array([-25.94756073, -12.
↪08151622])),
 Transformation(matrix=array([[ 9.98495460e-01, -4.65982241e-04],
                               [ 4.65982241e-04,  9.98495460e-01]]), offset=array([-32.71859007, -15.
↪76861207])),
 Transformation(matrix=array([[ 0.99857889,  0.00120287],
                               [-0.00120287,  0.99857889]]), offset=array([-39.95058187, -18.
↪65026698])),
 Transformation(matrix=array([[ 9.96629116e-01, -5.37378964e-04],
                               [ 5.37378964e-04,  9.96629116e-01]]), offset=array([-44.18251022, -22.
↪15203275])),
 Transformation(matrix=array([[ 9.98851114e-01, -4.33292180e-04],
                               [ 4.33292180e-04,  9.98851114e-01]]), offset=array([-52.9063275 , -26.
↪00989829])),
 Transformation(matrix=array([[ 9.97771654e-01, -6.47144209e-04],
                               [ 6.47144209e-04,  9.97771654e-01]]), offset=array([-58.14458854, -28.
↪33975215]))]
```

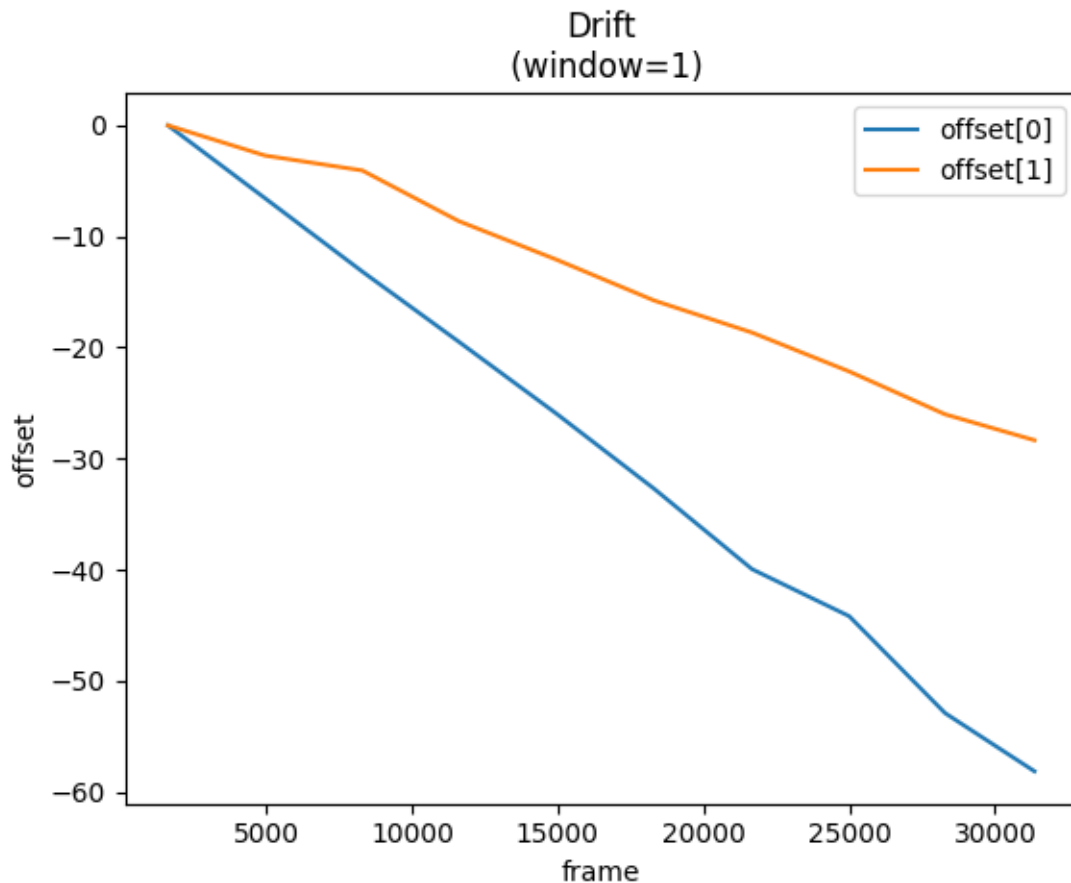
The parameters can be visualized using the `plot` function. The matrix in this case is close to the unit matrix.

```
drift.plot(transformation_component='matrix', element=None);
plt.legend();
```





```
drift.plot(transformation_component='offset', element=None)  
plt.legend();
```

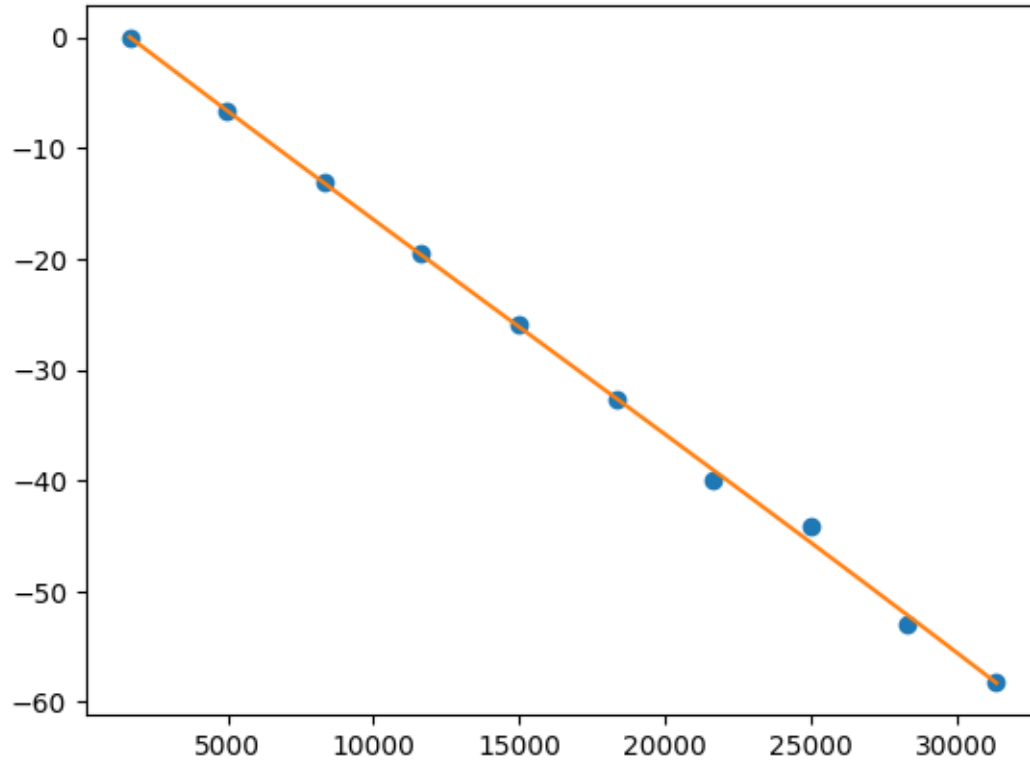


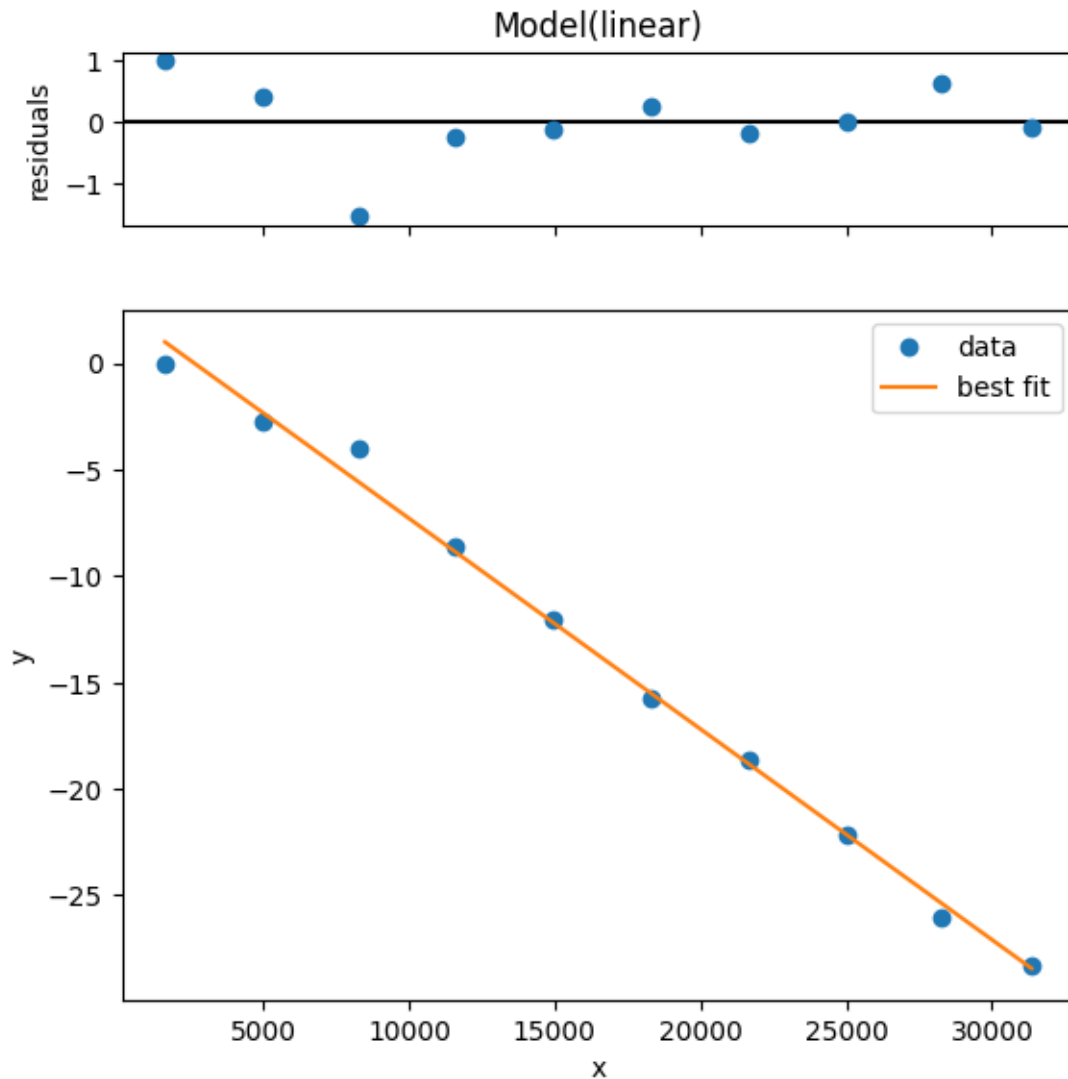
### 2.17.5 Model drift

A continuous transformation model as function of frame number is estimated by fitting the individual transformation components with the specified fit models. Fit models can be provided as `DriftComponent` or by a string representing standard model functions.

```
from lmfit.models import ConstantModel, LinearModel, PolynomialModel

drift.fit_transformations(slice_data=slice(None), offset_models=(lc.
↳ DriftComponent('spline', s=100), 'linear'), verbose=True);
```





The fit models are represented as `DriftComponent` and can be accessed through the `transformation_models` attribute.

```
drift.transformation_models
```

```
{'matrix': None,
 'offset': [<locan.analysis.drift.DriftComponent at 0x7f88c21cd3d0>,
            <locan.analysis.drift.DriftComponent at 0x7f88bca74690>]}
```

```
drift.transformation_models['offset'][0].type
```

```
'spline'
```

```
drift.transformation_models['offset'][0].eval(0)
```

```
array(3.33056629)
```

Each `DriftModel` carries detailed information about the fit under the `model_result` attribute. In most cases, except splines, this will be a `lmfit.ModelResult` object.

```
drift.transformation_models['offset'][0].model_result
```

```
(array([ 1639.0867    , 1639.0867    , 1639.0867    , 1639.0867    ,
        31356.39897573, 31356.39897573, 31356.39897573, 31356.39897573]),
 array([ 3.39564863e-02, -1.98083279e+01, -3.82432449e+01, -5.82808309e+01,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00]),
 3)
```

```
drift.transformation_models['offset'][1].type
```

```
'linear'
```

```
drift.transformation_models['offset'][1].model_result
```

```
<lmfit.model.ModelResult at 0x7f88bc0eb750>
```

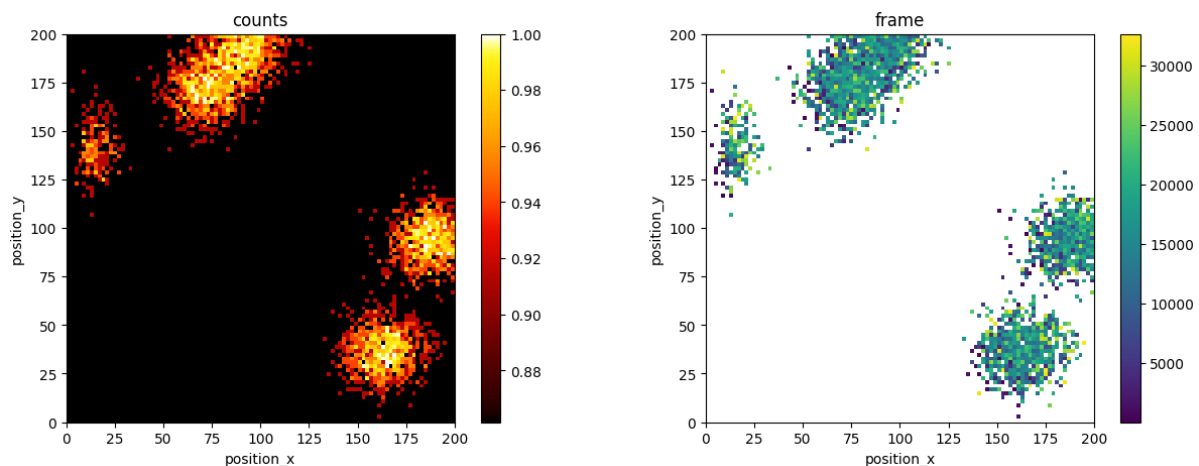
## 2.17.6 Drift correction

The estimated drift is corrected by applying a transformation on the localization chunks (from\_model=False).

```
%%time
drift.apply_correction(from_model=False);
```

```
CPU times: user 416 ms, sys: 4.03 ms, total: 420 ms
Wall time: 418 ms
```

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))
lc.render_2d(drift.locdata_corrected, ax=axes[0], bin_size=2, rescale='equal',
    ↪ bin_range=((0, 200),(0, 200)));
lc.render_2d_mpl(drift.locdata_corrected, ax=axes[1], other_property='frame',
    ↪ bin_size=2, bin_range=((0, 200),(0, 200)), cmap='viridis');
```



```
rmse(dat_blob, drift.locdata_corrected).round(2)
```

```
array([9.55, 4.82])
```

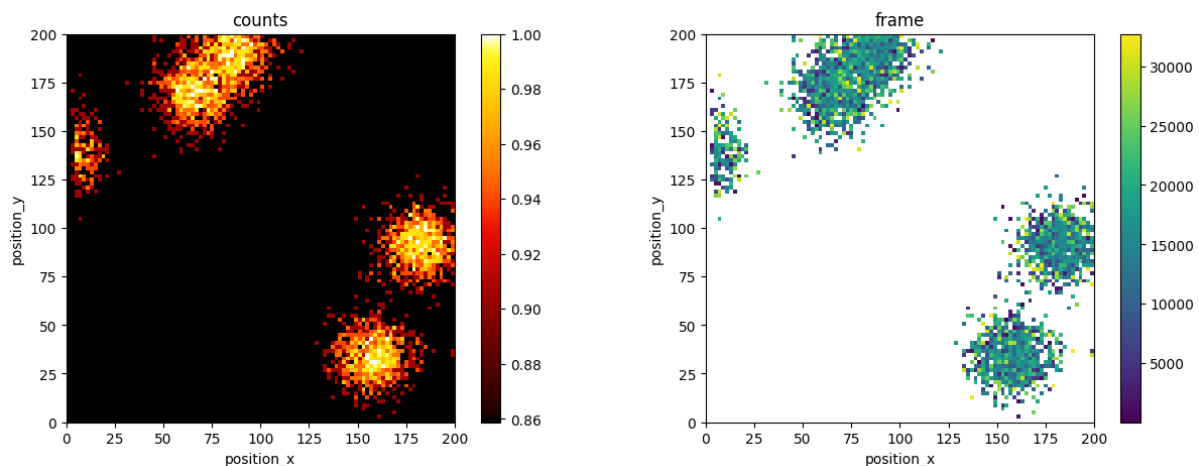
Or the estimated drift is corrected by applying a transformation on each individual localization using the drift models (from\_model=True).

```
%%time
drift.apply_correction(from_model=True)
```

```
CPU times: user 64.1 ms, sys: 8 ms, total: 72.1 ms
Wall time: 72 ms
```

```
Drift(chunks=None, chunk_size=10000, n_chunks=None, target=first, method=icp,
      ↪kwargs_chunk=None, kwargs_register=None)
```

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))
lc.render_2d(drift.locdata_corrected, ax=axes[0], bin_size=2, rescale='equal',
  ↪bin_range=((0, 200),(0, 200)));
lc.render_2d_mpl(drift.locdata_corrected, ax=axes[1], other_property='frame',
  ↪bin_size=2, bin_range=((0, 200),(0, 200)), cmap='viridis');
```



```
rmse(dat_blob, drift.locdata_corrected).round(2)
```

```
array([3.93, 2.77])
```

```
drift.locdata_corrected.meta
```

```
identifier: "26"
source: DESIGN
state: MODIFIED
history {
  name: "LocData.from_dataframe"
}
history {
```

(continues on next page)

(continued from previous page)

```

    name: "add_drift"
    parameter: "{\'locdata\': <locan.data.locdata.LocData object at 0x7f88c7558a90>, \\'diffusion_constant\': None, \\'velocity\': (0.002, 0.001), \\'seed\': Generator(PCG64) at 0x7F88DCCC90E0}"
}
history {
    name: "apply_correction"
    parameter: "{\'self\': Drift(chunks=None, chunk_size=10000, n_chunks=None, target=first, method=icp, kwargs_chunk=None, kwargs_register=None), \\'locdata\': None, \\'from_model\': True}"
}
ancestor_identifiers: "2"
ancestor_identifiers: "3"
element_count: 98201
frame_count: 31016
creation_time {
    seconds: 1710414298
    nanos: 105764000
}
modification_time {
    seconds: 1710414306
    nanos: 259044000
}

```

### 2.17.7 Drift analysis by a cross-correlation algorithm

The same kind of drift estimation and correction can be applied using the image cross-correlation algorithm.

```

%%time
drift = lc.Drift(chunk_size=10_000, target='first', method='cc').\
    compute(dat_blob_with_drift).\
    fit_transformations(slice_data=slice(None), offset_
↳models=(LinearModel(), LinearModel()), verbose=True).\
    apply_correction(from_model=True);

```

```

/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/analysis/drift.py:265: UserWarning: The
↳function register_cc has been refactored. The kwargs max_offset and verbose
↳are deprecated . It now calls _register_cc_skimage. Use _register_cc_
↳picasso for legacy behavior.
    transformation = register_cc(
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/analysis/drift.py:265: UserWarning: The
↳function register_cc has been refactored. The kwargs max_offset and verbose
↳are deprecated . It now calls _register_cc_skimage. Use _register_cc_
↳picasso for legacy behavior.
    transformation = register_cc(

```

(continues on next page)

(continued from previous page)

```

/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/analysis/drift.py:265: UserWarning: The
↳function register_cc has been refactored. The kwargs max_offset and verbose
↳are deprecated . It now calls _register_cc_skimage. Use _register_cc_
↳picasso for legacy behavior.
    transformation = register_cc(
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/analysis/drift.py:265: UserWarning: The
↳function register_cc has been refactored. The kwargs max_offset and verbose
↳are deprecated . It now calls _register_cc_skimage. Use _register_cc_
↳picasso for legacy behavior.
    transformation = register_cc(
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/analysis/drift.py:265: UserWarning: The
↳function register_cc has been refactored. The kwargs max_offset and verbose
↳are deprecated . It now calls _register_cc_skimage. Use _register_cc_
↳picasso for legacy behavior.
    transformation = register_cc(
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/analysis/drift.py:265: UserWarning: The
↳function register_cc has been refactored. The kwargs max_offset and verbose
↳are deprecated . It now calls _register_cc_skimage. Use _register_cc_
↳picasso for legacy behavior.
    transformation = register_cc(
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/analysis/drift.py:265: UserWarning: The
↳function register_cc has been refactored. The kwargs max_offset and verbose
↳are deprecated . It now calls _register_cc_skimage. Use _register_cc_
↳picasso for legacy behavior.
    transformation = register_cc(
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/analysis/drift.py:265: UserWarning: The
↳function register_cc has been refactored. The kwargs max_offset and verbose
↳are deprecated . It now calls _register_cc_skimage. Use _register_cc_
↳picasso for legacy behavior.
    transformation = register_cc(

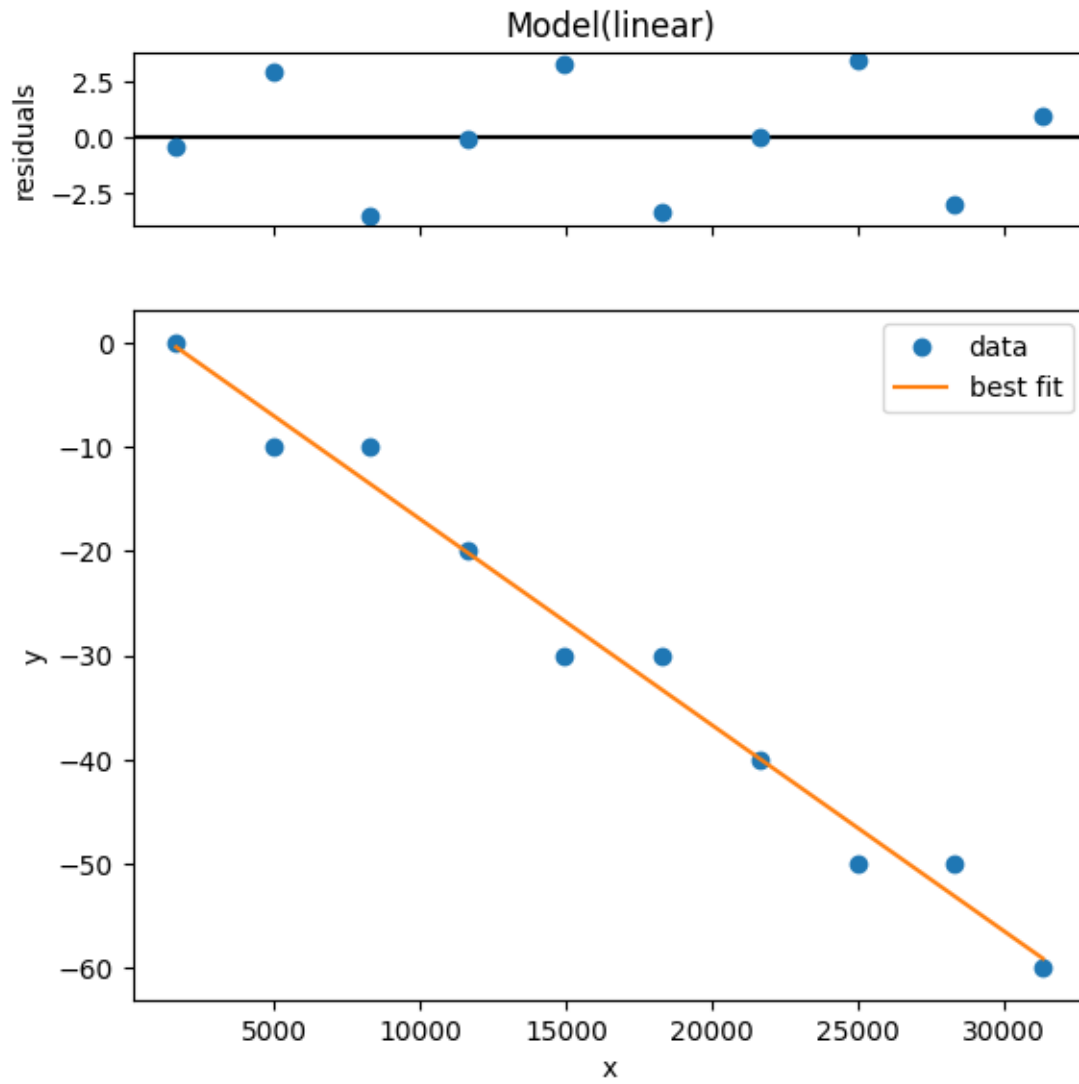
```

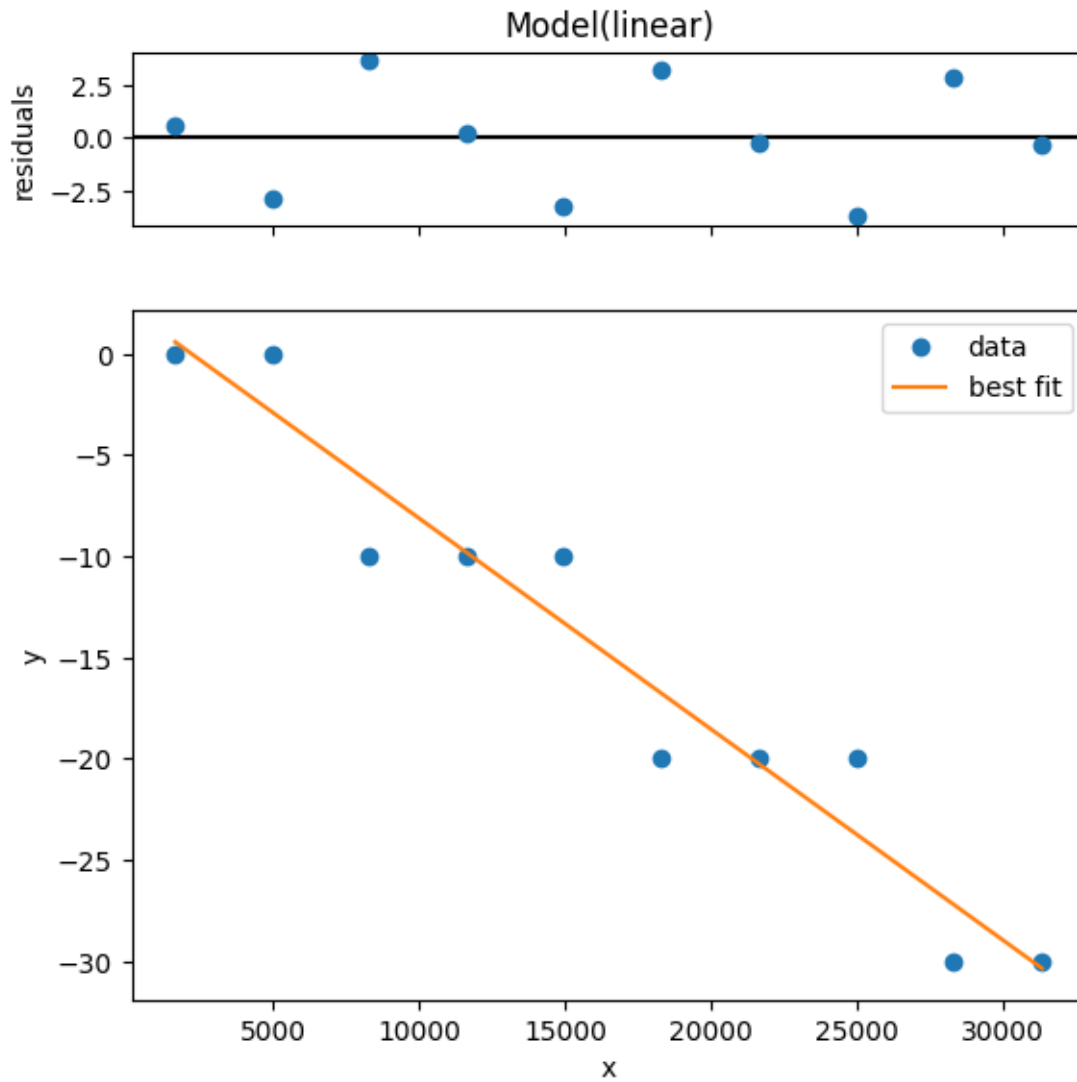
```

CPU times: user 1.03 s, sys: 128 ms, total: 1.16 s
Wall time: 950 ms

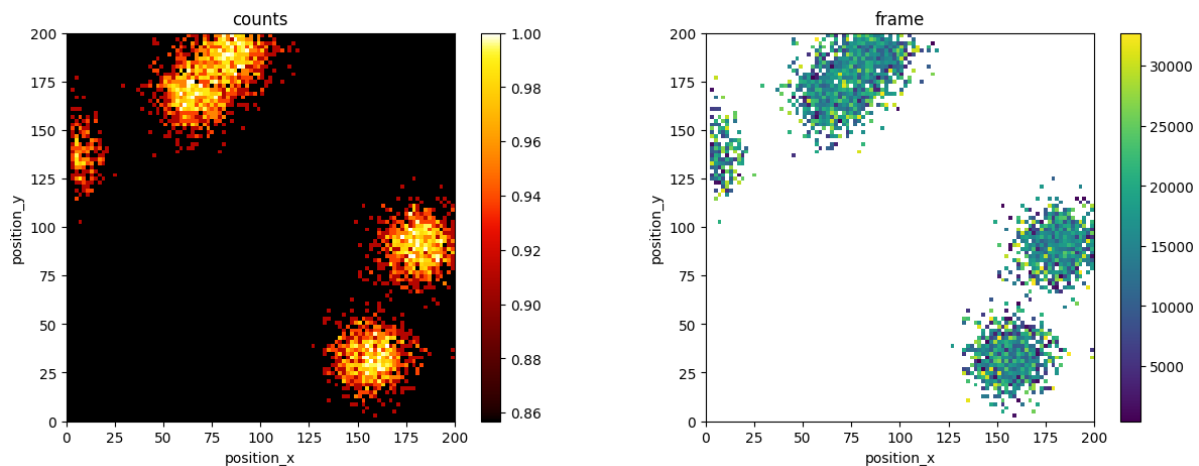
```







```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))
lc.render_2d(drift.locdata_corrected, ax=axes[0], bin_size=2, rescale='equal',
    ↪ bin_range=((0, 200),(0, 200)));
lc.render_2d_mpl(drift.locdata_corrected, ax=axes[1], other_property='frame',
    ↪ bin_size=2, bin_range=((0, 200),(0, 200)), cmap='viridis');
```



```
rmse(dat_blob, drift.locdata_corrected)
```

```
array([3.22658567, 1.66731276])
```

## 2.18 Tutorial about nearest neighbor distances

```
from pathlib import Path

%matplotlib inline

import matplotlib.pyplot as plt

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.18.1 Load rapidSTORM data file

Identify some data in the test\_data directory and provide a path using pathlib.Path

```
path = lc.ROOT_DIR / 'tests/test_data/rapidSTORM_dstorm_data.txt'
print(path, '\n')

dat = lc.load_rapidSTORM_file(path=path, nrows=1000)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/locan/tests/test_data/rapidSTORM_dstorm_data.txt
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

Print information about the data:

```
print(dat.data.head(), '\n')
print('Summary:')
dat.print_summary()
print('Properties:')
print(dat.properties)
```

|   | position_x | position_y | frame | intensity | chi_square | local_background |
|---|------------|------------|-------|-----------|------------|------------------|
| 0 | 9657.40    | 24533.5    | 0     | 33290.10  | 1192250.0  | 767.732971       |
| 1 | 16754.90   | 18770.0    | 0     | 21275.40  | 2106810.0  | 875.460999       |
| 2 | 14457.60   | 18582.6    | 0     | 20748.70  | 526031.0   | 703.369995       |
| 3 | 6820.58    | 16662.8    | 0     | 8531.77   | 3179190.0  | 852.789001       |
| 4 | 19183.20   | 22907.2    | 0     | 14139.60  | 448631.0   | 662.770020       |

#### Summary:

identifier: "1"

comment: ""

source: EXPERIMENT

state: RAW

element\_count: 14

frame\_count: 1

file {

type: RAPIDSTORM

path: "/home/docs/checkouts/readthedocs.org/user\_builds/locan/envs/latest/

↪lib/python3.11/site-packages/locan/tests/test\_data/rapidSTORM\_dstorm\_data.

↪txt"

}

creation\_time {

2024-03-14T11:07:49.827896Z

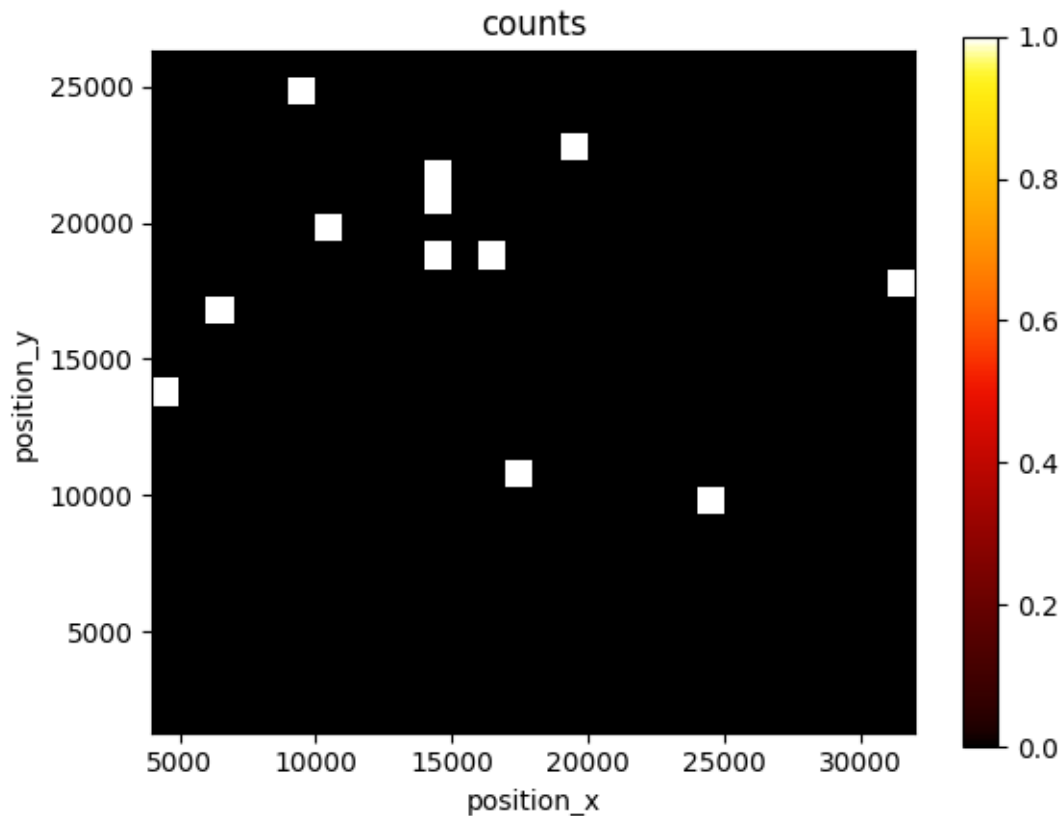
}

#### Properties:

```
{'localization_count': 14, 'position_x': 15873.847142857145, 'uncertainty_x': 2361.4490857013648, 'position_y': 17403.909285714286, 'uncertainty_y': 1803.9975262697349, 'intensity': 183987.66999999998, 'local_background': 675.0614, 'frame': 0, 'region_measure_bb': 730882123.3259, 'localization_density_bb': 1.915493559521281e-08, 'subregion_measure_bb': 108337.2}
```

## 2.18.2 Visualization

```
lc.render_2d(dat, bin_size=1000, rescale=(0,100));
```



### 2.18.3 Analyze nearest neighbor distances

The k-nearest neighbor distances can be analyzed for all localizations or a random subset.

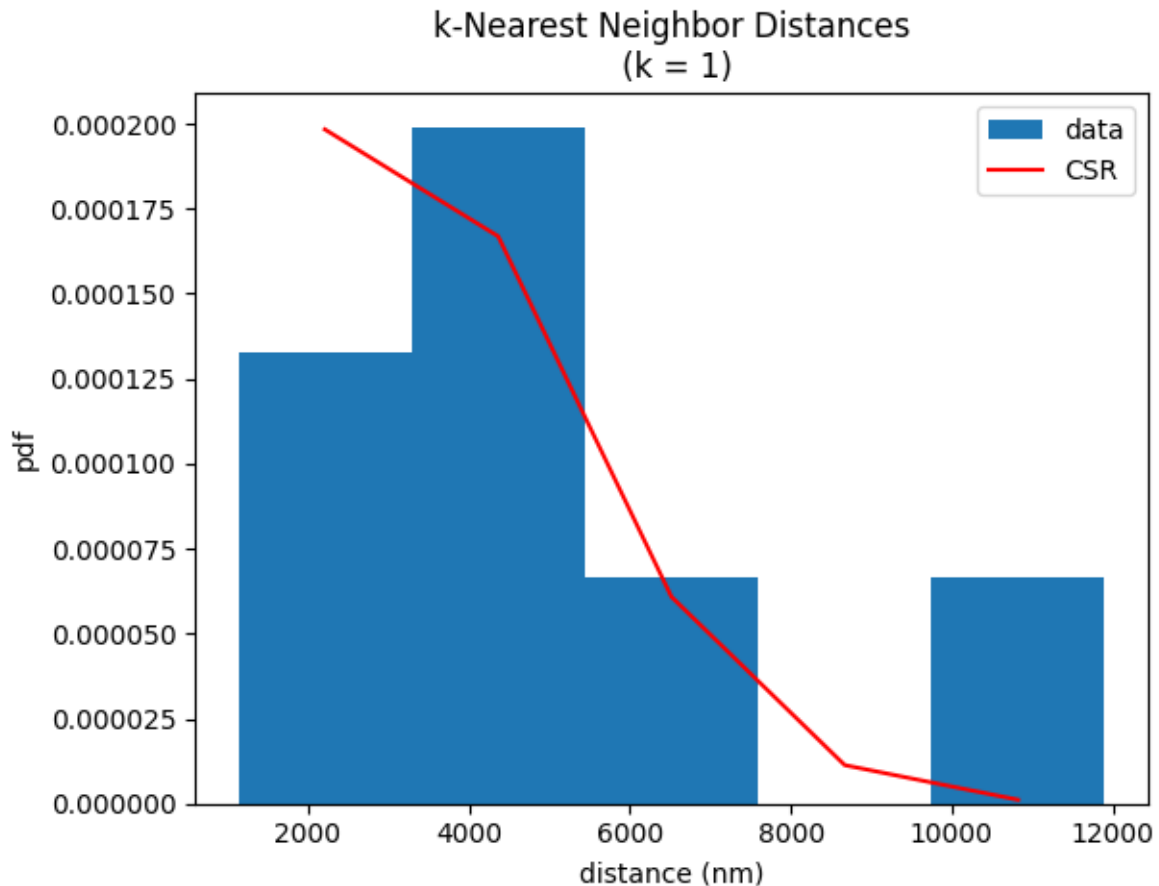
```
nn = lc.NearestNeighborDistances()
```

```
nn.compute(dat)
nn.results.head()
```

|   | nn_distance | nn_index |
|---|-------------|----------|
| 0 | 4219.766477 | 12       |
| 1 | 2304.930812 | 2        |
| 2 | 2291.302095 | 9        |
| 3 | 3974.974397 | 10       |
| 4 | 4797.193422 | 1        |

The histogram shows the experimental distribution and for comparison the expectation for a spatial distribution of complete spatial randomness given the experimental localization density (per default relative to the minimum bounding box region).

```
nn.hist();
```



The `localization_density` relative to the minimum bounding box region is retweeted by the `localization_density` attribute.

```
nn.localization_density
```

```
1.915493559521281e-08
```

## 2.18.4 Meta data for the analysis procedure

```
nn.meta
```

```
identifier: "1"
method {
  name: "NearestNeighborDistances"
  parameter: "{\'k\': 1}"
}
creation_time {
  seconds: 1710414470
  nanos: 710585000
}
```

## 2.19 Tutorial about Ripley's k function

```
from pathlib import Path

%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.19.1 Synthetic data

We simulate localization data that is homogeneously Poisson distributed (also described as complete spatial randomness, csr).

```
rng = np.random.default_rng(seed=1)
```

```
locdata_csr = lc.simulate_Poisson(intensity=1e-3, region=((0,1000), (0,1000)),
→ seed=rng)

print('Data head:')
print(locdata_csr.data.head(), '\n')
print('Summary:')
locdata_csr.print_summary()
print('Properties:')
print(locdata_csr.properties)
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
Data head:
   position_x  position_y
0  144.159613  948.649447
1  311.831452  423.326449
2  827.702594  409.199136
3  549.593688   27.559113
4  753.513109  538.143313

Summary:
```

(continues on next page)

(continued from previous page)

```

identifier: "1"
comment: ""
source: SIMULATION
state: RAW
element_count: 1001
frame_count: 0
creation_time {
  2024-03-14T11:05:13.014271Z
}

Properties:
{'localization_count': 1001, 'position_x': 497.1480704304259, 'uncertainty_x
↪': 8.846806436903796, 'position_y': 503.49401443075675, 'uncertainty_y': 9.
↪309630174812341, 'region_measure_bb': 997049.9168748705, 'localization_
↪density_bb': 0.001003961770677952, 'subregion_measure_bb': 3994.
↪0982202393607, 'region_measure': 1000000, 'localization_density': 0.001001,
↪'subregion_measure': 4000}

```

We also simulate data that follows a Neyman-Scott distribution (blobs):

```

locdata_blob = lc.simulate_Thomas(parent_intensity=1e-4, region=((0, 1000),
↪((0, 1000))), cluster_mu=100, cluster_std=5, seed=rng)

print('Data head:')
print(locdata_blob.data.head(), '\n')
print('Summary:')
locdata_blob.print_summary()
print('Properties:')
print(locdata_blob.properties)

```

```

Data head:
  position_x  position_y  cluster_label
0  639.830857  587.034592             13
1  499.121280   3.258363             15
2  347.348453  532.577203             62
3  563.963921  711.136973             61
4  258.865522   24.737190             47

```

```

Summary:
identifier: "2"
comment: ""
source: SIMULATION
state: RAW
element_count: 9860
frame_count: 0
creation_time {
  2024-03-14T11:05:13.061376Z
}

```

(continues on next page)



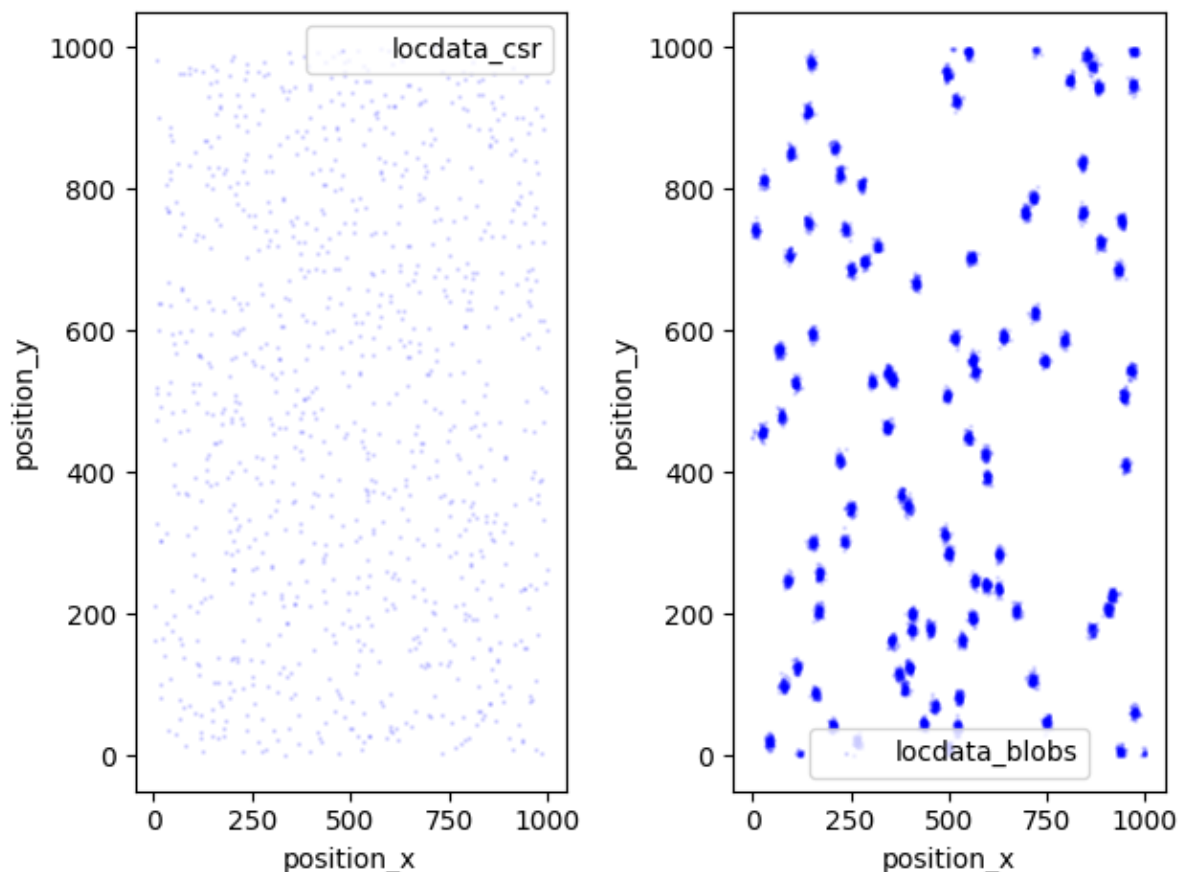
(continued from previous page)

Properties:

```
{'localization_count': 9860, 'position_x': 486.6445632771076, 'uncertainty_x': 2.851385115124155, 'position_y': 467.1399336108736, 'uncertainty_y': 3.0359994421575136, 'region_measure_bb': 999385.4817400454, 'localization_density_bb': 0.009866062875791034, 'subregion_measure_bb': 3998.7709412537642, 'region_measure': 1000000, 'localization_density': 0.00986, 'subregion_measure': 4000}
```

## Scatter plot

```
fig, ax = plt.subplots(nrows=1, ncols=2)
locdata_csr.data.plot.scatter(x='position_x', y='position_y', ax=ax[0], color='Blue', s=1, alpha=0.1, label='locdata_csr')
locdata_blob.data.plot.scatter(x='position_x', y='position_y', ax=ax[1], color='Blue', s=1, alpha=0.1, label='locdata_blobs')
plt.tight_layout()
plt.show()
```



## 2.19.2 Analyze Ripley's h function

We have a look at the Ripley's h function from all localizations in locdata.

The analysis class `Ripley_h_function` provides numerical results, and a plot of results versus radii.

```
rhf_csr = lc.RipleysHFunction(radii=np.linspace(0, 200, 50))
rhf_csr.compute(locdata_csr)
rhf_csr.results.head()
```

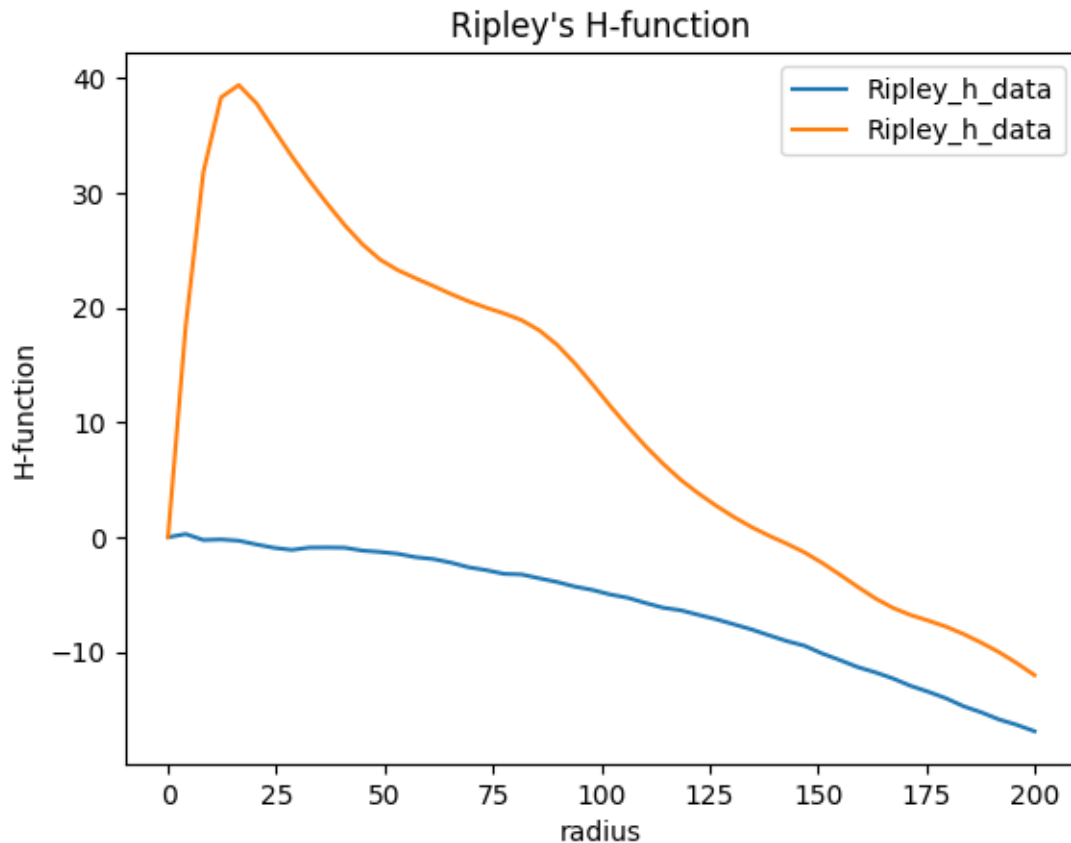
| Ripley_h_data |           |
|---------------|-----------|
| radius        |           |
| 0.000000      | 0.000000  |
| 4.081633      | 0.279930  |
| 8.163265      | -0.240094 |
| 12.244898     | -0.194560 |
| 16.326531     | -0.301127 |

```
rhf_blob = lc.RipleysHFunction(radii=np.linspace(0, 200, 50))
rhf_blob.compute(locdata_blob)
rhf_blob.results.head()
```

| Ripley_h_data |           |
|---------------|-----------|
| radius        |           |
| 0.000000      | 0.000000  |
| 4.081633      | 18.353371 |
| 8.163265      | 31.787540 |
| 12.244898     | 38.295378 |
| 16.326531     | 39.369222 |

The plot reflects the amount of clustering. For homogeneous distributed data it decreases towards negative values since edge effects are not taken into account.

```
rhf_csr.plot()
rhf_blob.plot();
```



The maximum of the computed H-function is provided by the attribute.

```
rhf_blob.Ripley_h_maximum
```

|               | radius    | Ripley_h_maximum |
|---------------|-----------|------------------|
| Ripley_h_data | 16.326531 | 39.369222        |

### 2.19.3 Estimate Ripley's h function

We can speed up the computation of an estimated Ripley's k function by providing a subset of the original localizations as test points.

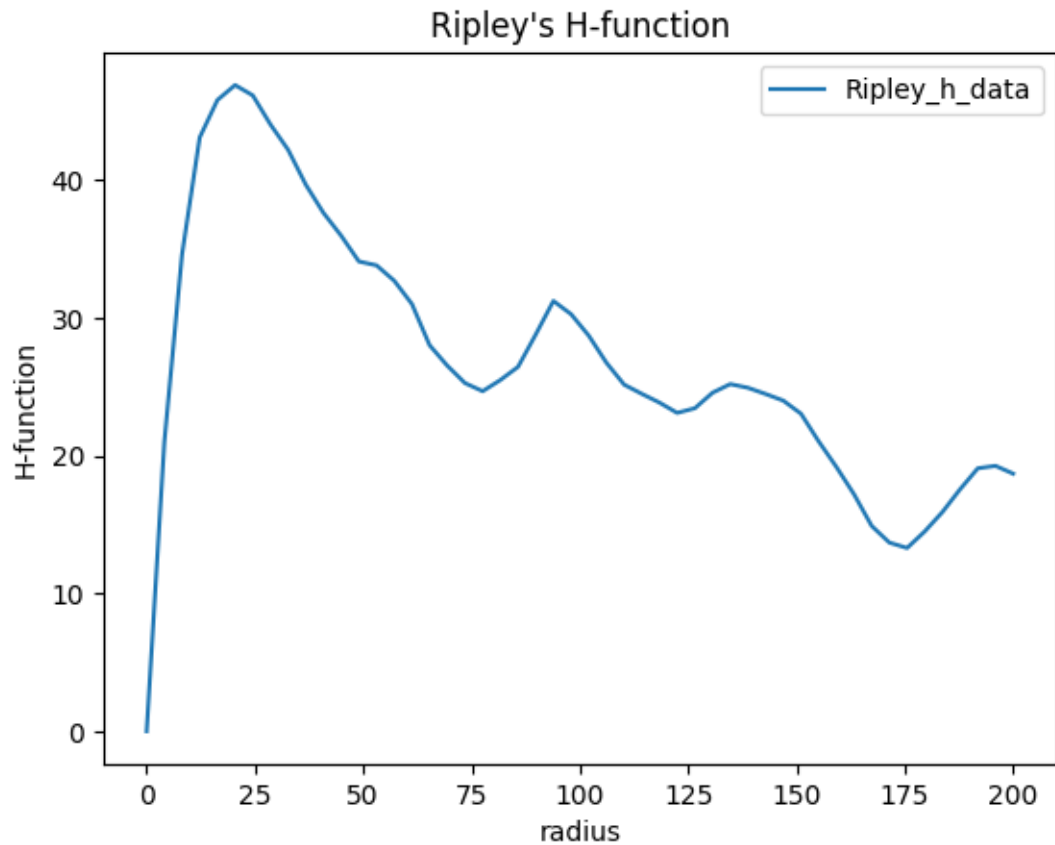
We first take a random subset of the original localizations as test data. Here we provide 10 shuffled data sets.

```
from locan.data.filter import random_subset
subsets = [lc.random_subset(locdata_blob, n_points=5, seed=rng) for i in
    range(10)]
```

We then compute the estimated Ripley's h function'

```
rhf_estimate = lc.RipleysHFunction(radii=np.linspace(0, 200, 50)).
    compute(locdata_blob, other_locdata=subsets[0])
```

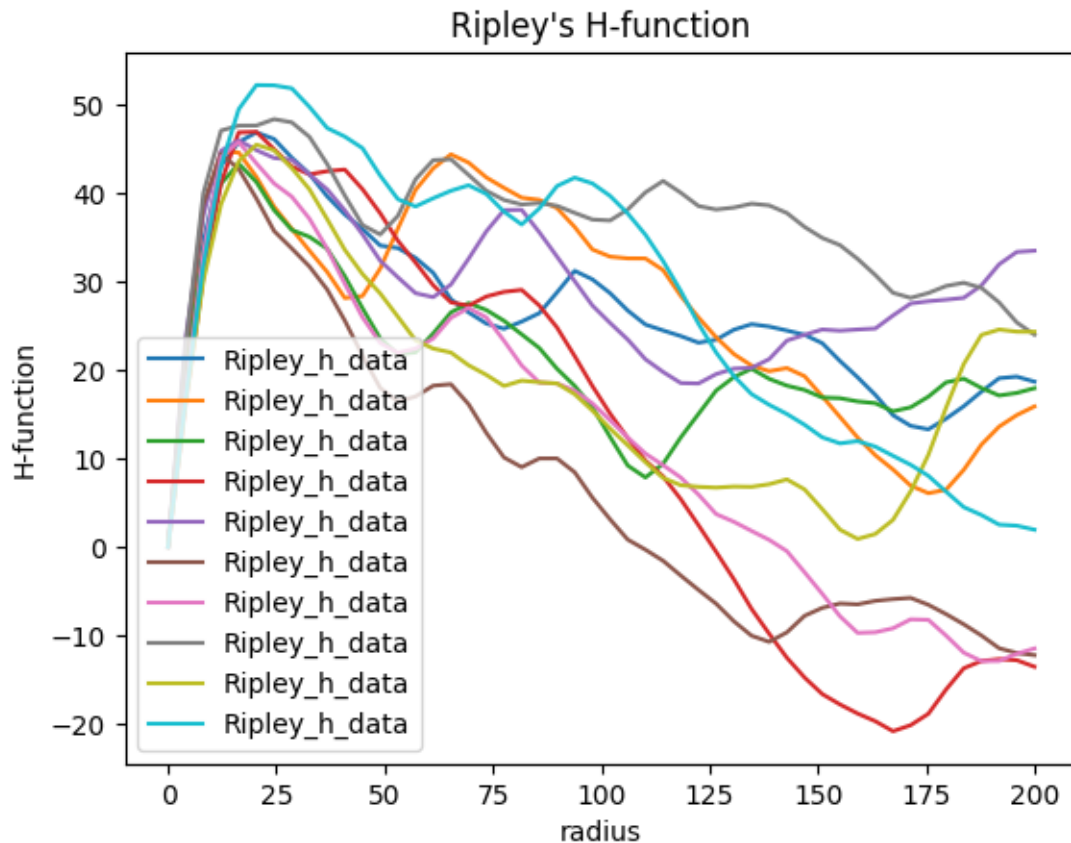
```
rhf_estimate.plot();
```



We can do the same for all subsets

```
rhf_estimates = [lc.RipleysHFunction(radii=np.linspace(0, 200, 50)).
    ↪compute(locdata_blob, other_locdata=subset) for subset in subsets]
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
for estimate in rhf_estimates:
    estimate.plot(ax=ax)
plt.show()
```

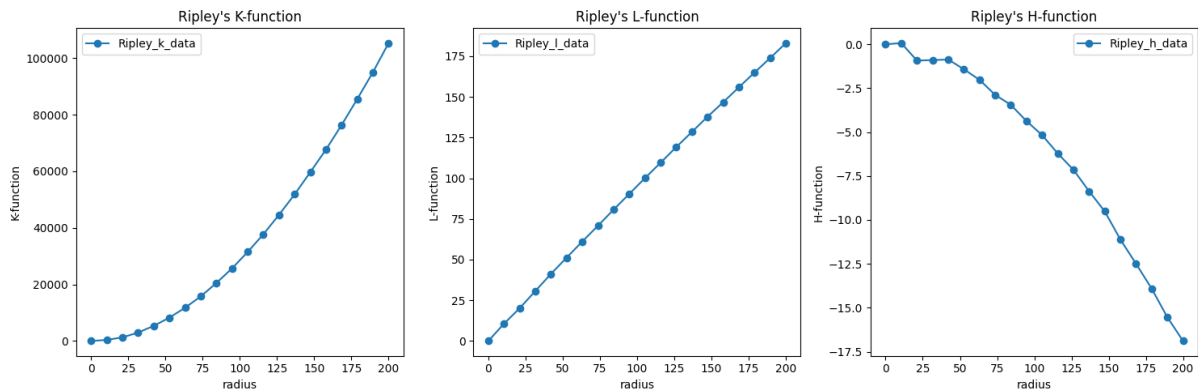


#### 2.19.4 Compute Ripley's k, l and h function

We can compute Ripley's k, l and h function

```
rkf_csr = lc.RipleysKFunction(radii=np.linspace(0, 200, 20)).compute(locdata_
    ↳csr)
rlf_csr = lc.RipleysLFunction(radii=np.linspace(0, 200, 20)).compute(locdata_
    ↳csr)
rhf_csr = lc.RipleysHFunction(radii=np.linspace(0, 200, 20)).compute(locdata_
    ↳csr)
```

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 5))
for estimate, ax in zip([rkf_csr, rlf_csr, rhf_csr], axes.ravel()):
    estimate.plot(marker='o', ax=ax)
plt.tight_layout()
plt.show()
```



### 2.19.5 Estimate Ripley's h function for 3D data

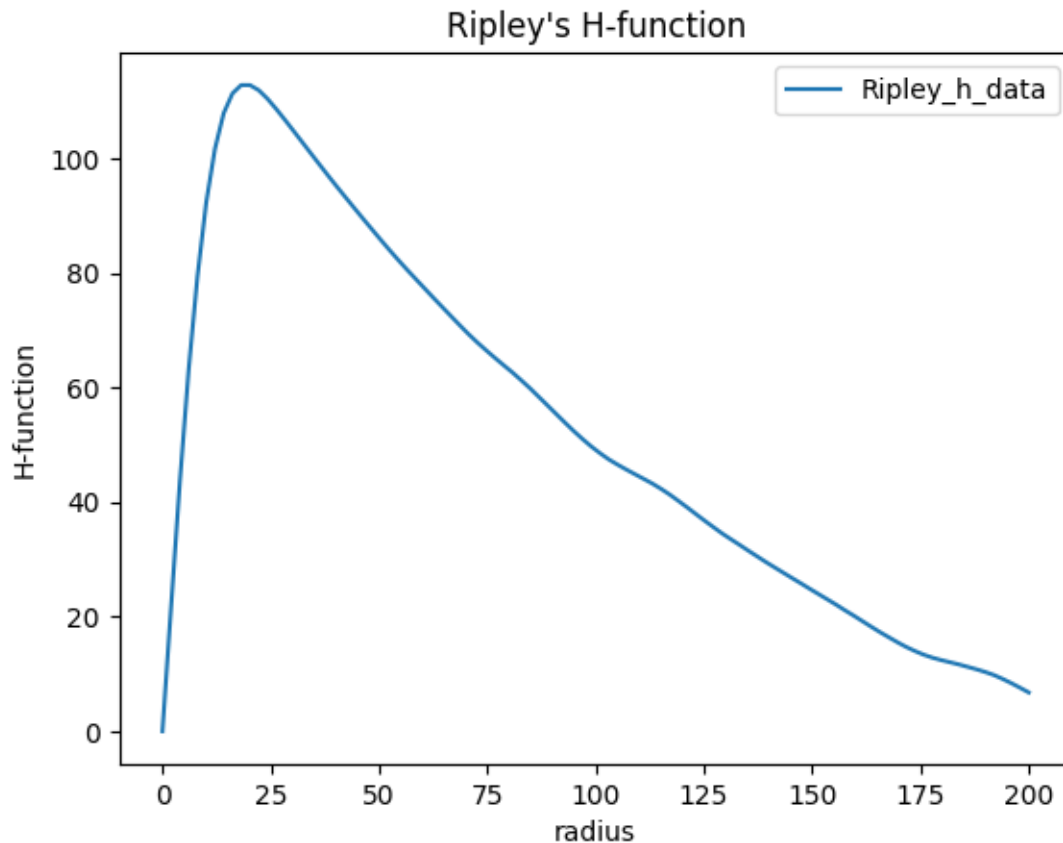
```
dat_blob_3D = lc.simulate_Thomas(parent_intensity=1e-7, region=((0, 1000), (0,
↪ 1000), (0, 1000)), cluster_mu=100, cluster_std=5, seed=rng)
dat_blob_3D.print_summary()
```

```
identifier: "23"
comment: ""
source: SIMULATION
state: RAW
element_count: 9394
frame_count: 0
creation_time {
  2024-03-14T11:05:22.095819Z
}
```

```
sub = lc.random_subset(dat_blob_3D, n_points=1000, seed=rng)
```

```
rhf_3D = lc.RipleysHFunction(radii=np.linspace(0, 200, 100)).compute(dat_blob_
↪ 3D, other_locdata=sub)
```

```
rhf_3D.plot();
```



## 2.20 Tutorial about analyzing grouped cluster properties

Localization properties vary within clusters. Analyzing such variations can help to characterize cluster populations. Here, we show examples for variations in convex hull properties or coordinate variances.

```
from pathlib import Path

%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from colorcet import m_fire, m_gray, m_dimgray

import locan as lc
```

```
/tmp/ipykernel_1019/3459984724.py:6: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
```

(continues on next page)

(continued from previous page)

`↪54466``import pandas as pd``lc.show_versions(system=False, dependencies=False, verbose=False)`

Locan:

version: 0.20.0.dev41+g755b969

Python:

version: 3.11.6

## 2.20.1 Synthetic data

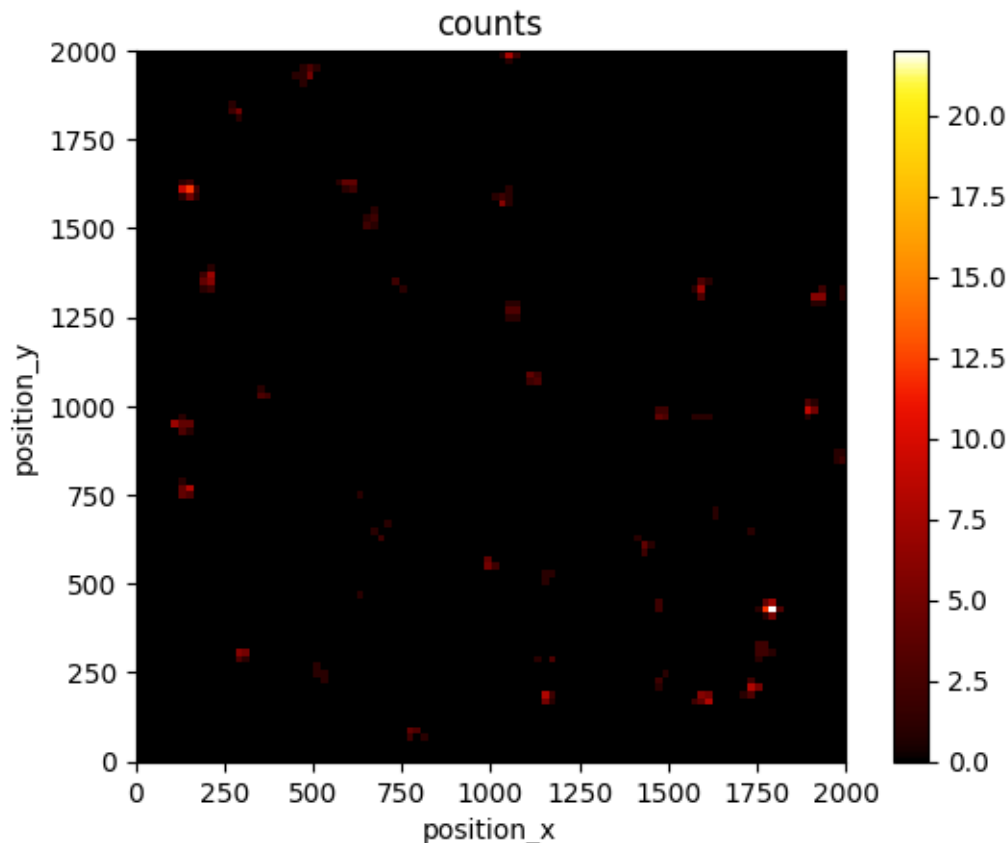
We simulate localization data that follows a Neyman-Scott distribution in 2D:

`rng = np.random.default_rng(seed=1)``locdata = lc.simulate_dstorm(parent_intensity=1e-5, region=((0, 10_000), (0, ↪10_000))), cluster_mu=10, cluster_std=10, seed=rng)``locdata.print_summary()`

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
identifier: "1"
comment: ""
source: SIMULATION
state: RAW
element_count: 10380
frame_count: 0
creation_time {
  2024-03-14T11:06:09.138715Z
}
```

`bin_range=((0, 2_000), (0, 2_000))``lc.render_2d_mpl(locdata, bin_size=20, bin_range=bin_range)``<Axes: title={'center': 'counts'}, xlabel='position_x', ylabel='position_y'>`





## 2.20.2 True clusters

First we look at ground truth clusters.

```
grouped = locdata.data.groupby("cluster_label")
clust = lc.LocData.from_collection([lc.LocData.from_selection(locdata,
↪ indices=group.index) for name, group in grouped])
```

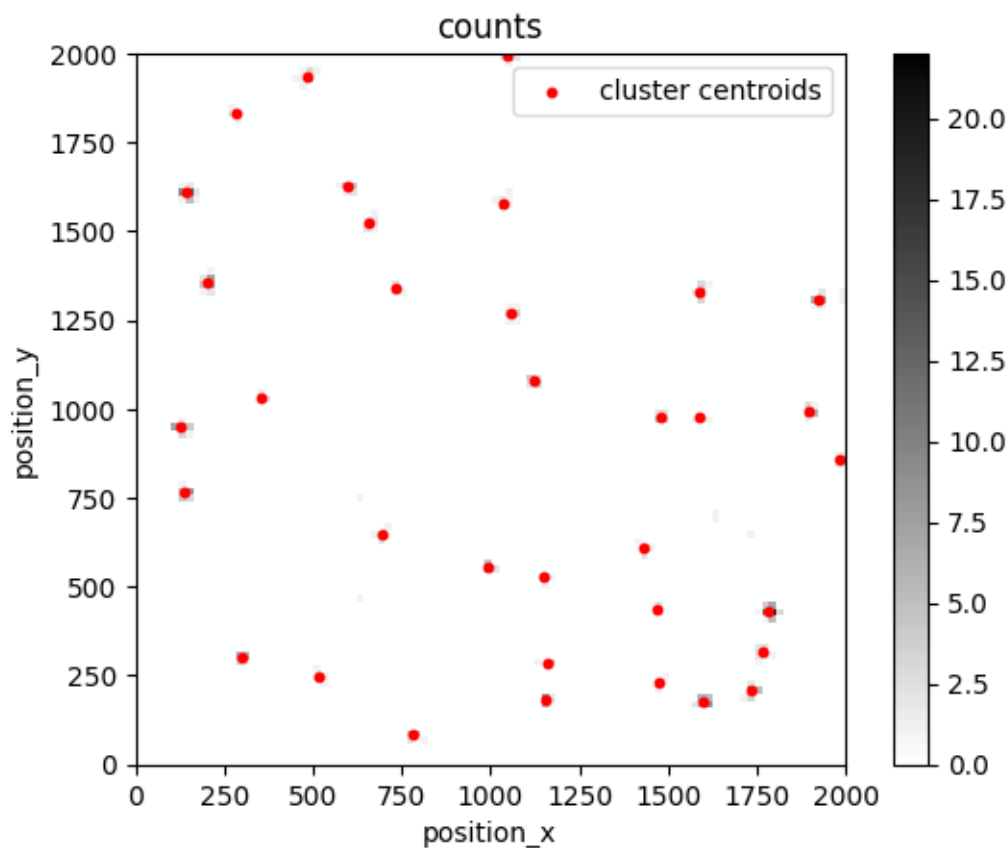
Filter out clusters with less than 3 localizations since on convex hull can be computed for such clusters.

```
clust_selection = lc.select_by_condition(clust, condition="2 < localization_
↪ count")
references_ = [clust.references[i] for i in clust_selection.indices]
clust_selection.reduce()
clust_selection.references = references_
clust = clust_selection
```

```
n_clustered_loc = np.sum([ref.properties['localization_count'] for ref in
↪ clust.references])
print(f"Number of clusters: {clust.properties['localization_count']}")
print(f"Number of clustered localizations: {n_clustered_loc}")
print(f"Ratio cluster to noise localizations: {n_clustered_loc /
↪ len(locdata):.3}")
```

```
Number of clusters: 771
Number of clustered localizations: 10172
Ratio cluster to noise localizations: 0.98
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
lc.render_2d_mpl(locdata, bin_size=20, bin_range=bin_range, cmap=m_gray.
    reversed())
clust.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Red',
    s=10, label='cluster centroids', xlim=bin_range[0], ylim=bin_range[1])
plt.show()
```



```
clust.data.head()
```

|   | localization_count | position_x  | uncertainty_x | position_y  | uncertainty_y |
|---|--------------------|-------------|---------------|-------------|---------------|
| 0 | 24                 | 1399.549521 | 1.877356      | 9539.052779 | 1.853978      |
| 2 | 3                  | 8303.079096 | 10.889812     | 4081.419445 | 3.852366      |
| 3 | 7                  | 5507.141828 | 2.444467      | 224.387484  | 2.132102      |
| 4 | 4                  | 7561.723474 | 3.515554      | 5385.880528 | 5.842642      |
| 5 | 7                  | 3279.865197 | 5.284643      | 7918.206789 | 2.848332      |

|   | region_measure_bb | localization_density_bb | subregion_measure_bb |
|---|-------------------|-------------------------|----------------------|
| 0 | 1025.930045       | 0.023393                | 128.121678           |
| 2 | 456.241155        | 0.006575                | 96.259339            |

(continues on next page)

(continued from previous page)

| 3 | 336.278485     | 0.020816             | 73.989892         |
|---|----------------|----------------------|-------------------|
| 4 | 443.103490     | 0.009027             | 87.512555         |
| 5 | 774.964110     | 0.009033             | 117.649012        |
|   | region_measure | localization_density | subregion_measure |
| 0 | 1000000000     | 2.400000e-07         | 40000             |
| 2 | 1000000000     | 3.000000e-08         | 40000             |
| 3 | 1000000000     | 7.000000e-08         | 40000             |
| 4 | 1000000000     | 4.000000e-08         | 40000             |
| 5 | 1000000000     | 7.000000e-08         | 40000             |

```
clust.properties
```

```
{'localization_count': 771,
 'position_x': 5078.461816893804,
 'uncertainty_x': 138.598117645907,
 'position_y': 5571.31780554545,
 'uncertainty_y': 421.90449657593706,
 'region_measure_bb': 99518129.6386362,
 'localization_density_bb': 7.7473320971727e-06,
 'subregion_measure_bb': 39903.55763820844}
```

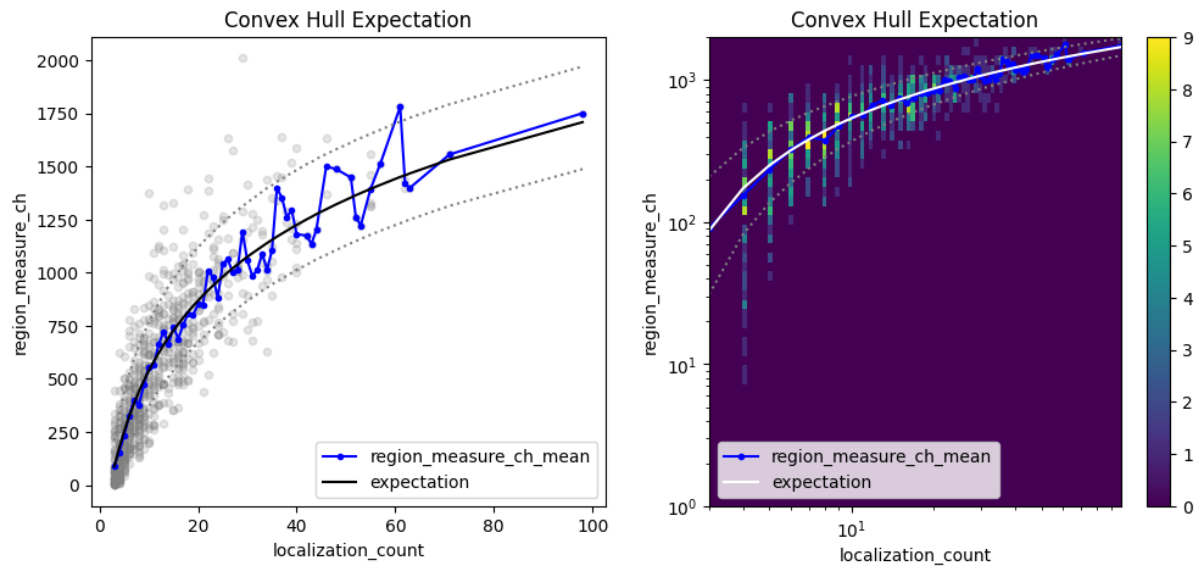
### 2.20.3 Investigate the convex hull areas

Localization clusters can be analyzed with respect to their convex hull region properties as function of localization\_count as outlined in Ebert et al. (<https://doi:10.1093/bioinformatics/btac700>).

```
che = lc.ConvexHullExpectation(convex_hull_property='region_measure_ch',
    ↪expected_variance=10**2).compute(locdata=clust)
che.results
```

```
<locan.analysis.convex_hull_expectation.ConvexHullExpectationResults at
    ↪0x7fd35a649490>
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))
che.plot(ax=axes[0])
che.hist(ax=axes[1]);
```



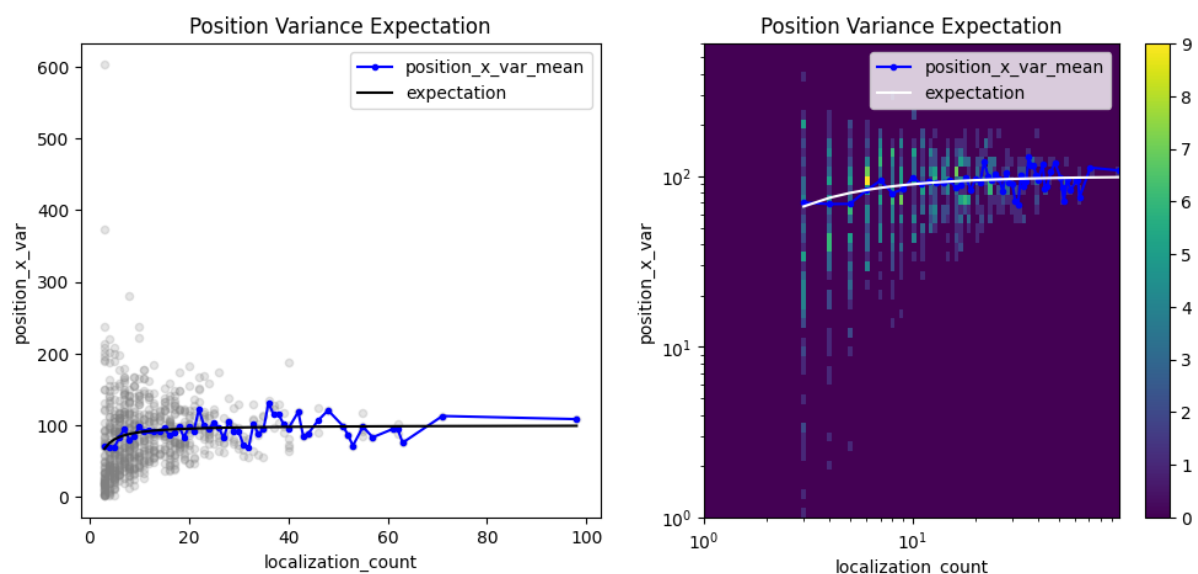
## 2.20.4 Investigate the position variances

Localization coordinates in localization clusters come with a certain variance. The variance is related to the localization precision or other localization properties but also varies with localization\_count if determined as biased sample variance (i.e. without Bessel's correction).

```
pve_biased = lc.PositionVarianceExpectation(loc_property="position_x",
→ expectation=10**2, biased=True).compute(locdata=clust)
pve_biased.results
```

```
<locan.analysis.position_variance_expectation.
→ PositionVarianceExpectationResults at 0x7fd35a4af190>
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))
pve_biased.plot(ax=axes[0])
pve_biased.hist(ax=axes[1]);
```

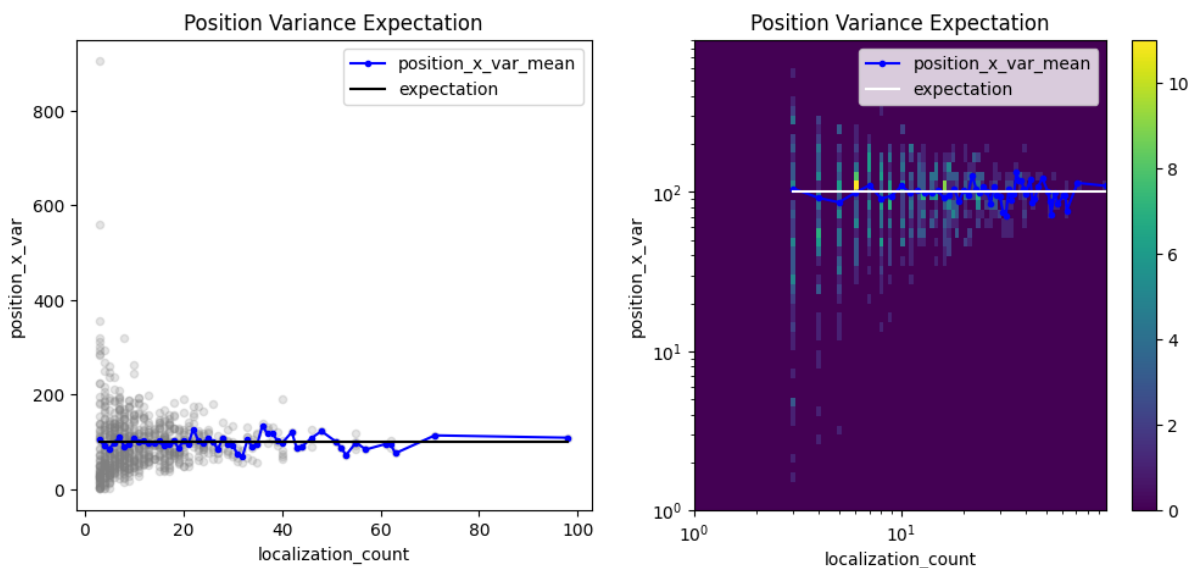


A similar analysis can be performed with unbiased variances in which Bessel's correction is applied.

```
pve = lc.PositionVarianceExpectation(loc_property="position_x",
    expectation=10**2, biased=False).compute(locdata=clust)
pve.results
```

```
<locan.analysis.position_variance_expectation.
    PositionVarianceExpectationResults at 0x7fd359e616d0>
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))
pve.plot(ax=axes[0])
pve.hist(ax=axes[1], log=True);
```



## 2.20.5 Investigate any grouped property

A similar analysis can be carried out with any LocData property. For instance, let's check the coordinate uncertainties of cluster centroids. The uncertainty in one dimension should follow the square root of the biased position variance for clusters with variable number of localizations.

It is important to consider the differences between variance and standard deviation. Position uncertainties are usually given as standard deviation with units equal to position units. Converting the ground truth for the coordinate standard deviation in each cluster (std as used in the simulations above) requires a Bessel correction on the squared std being the variance). In addition the coordinate uncertainties of cluster centroids should scale with the inverse square root of the number of localizations per cluster.

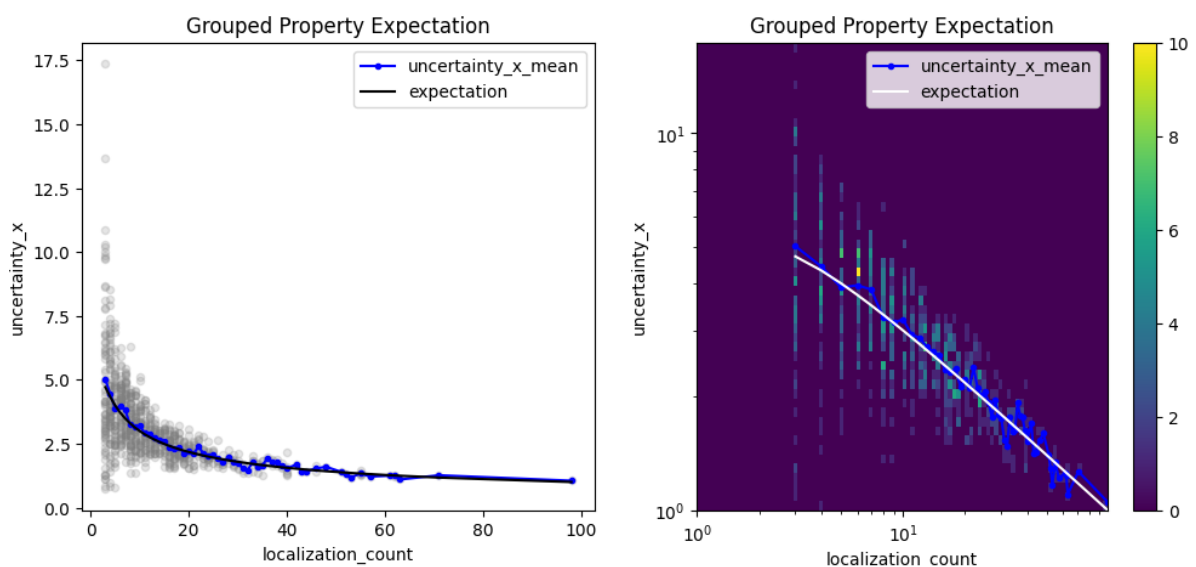
```
n_locs = np.arange(1, 1000)
ground_truth_std = 10
ground_truth_variance = ground_truth_std**2
biased_variance = ground_truth_variance * (1 - 1 / n_locs)
biased_uncertainty = np.sqrt(biased_variance)
expected_uncertainty = biased_uncertainty / np.sqrt(n_locs)
expectation = pd.Series(data=expected_uncertainty, index=n_locs)
expectation;
```

```
loc_property = "uncertainty_x"
other_loc_property = "localization_count"
```

```
gpe = lc.GroupedPropertyExpectation(loc_property=loc_property, other_loc_
↳property=other_loc_property, expectation=expectation).compute(locdata=clust)
gpe.results
```

```
<locan.analysis.grouped_property_expectation.
↳GroupedPropertyExpectationResults at 0x7fd353e70310>
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))
gpe.plot(ax=axes[0])
gpe.hist(ax=axes[1]);
```



## 2.20.6 Cluster localizations by dbscan

When clustering data by dbscan slight deviations appear between expectation and computed properties.

```
noise, clust = lc.cluster_dbscan(locdata, eps=20, min_samples=3)
```

```
n_clustered_loc = np.sum([ref.properties['localization_count'] for ref in
↳clust.references])
print(f"Number of clusters: {clust.properties['localization_count']}")
print(f"Number of noise localizations: {noise.properties['localization_count']
↳}")
print(f"Number of clustered localizations: {n_clustered_loc}")
print(f"Ratio cluster to noise localizations: {n_clustered_loc / (n_clustered_
↳loc + noise.properties['localization_count']):.3}")
```

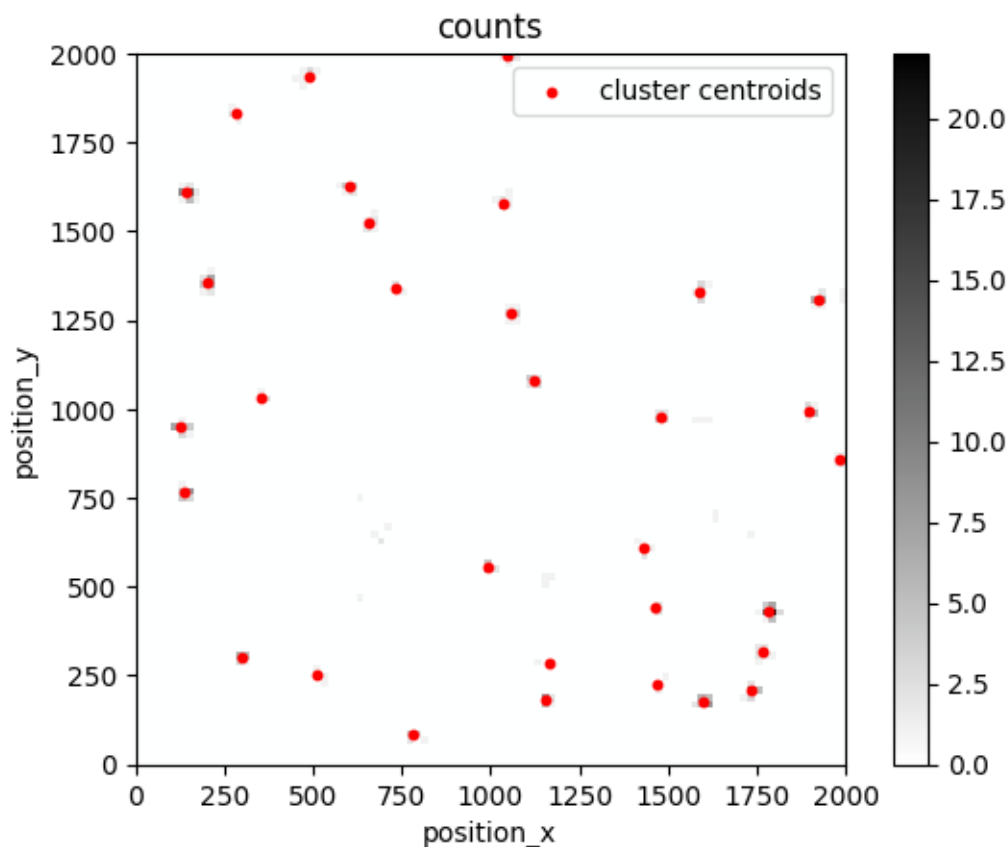
```
Number of clusters: 725
Number of noise localizations: 415
```

(continues on next page)

(continued from previous page)

```
Number of clustered localizations: 9965
Ratio cluster to noise localizations: 0.96
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
lc.render_2d_mpl(locdata, bin_size=20, bin_range=bin_range, cmap=m_gray.
    ↪reversed())
clust.data.plot.scatter(x='position_x', y='position_y', ax=ax, color='Red',
    ↪s=10, label='cluster centroids', xlim=bin_range[0], ylim=bin_range[1])
plt.show()
```

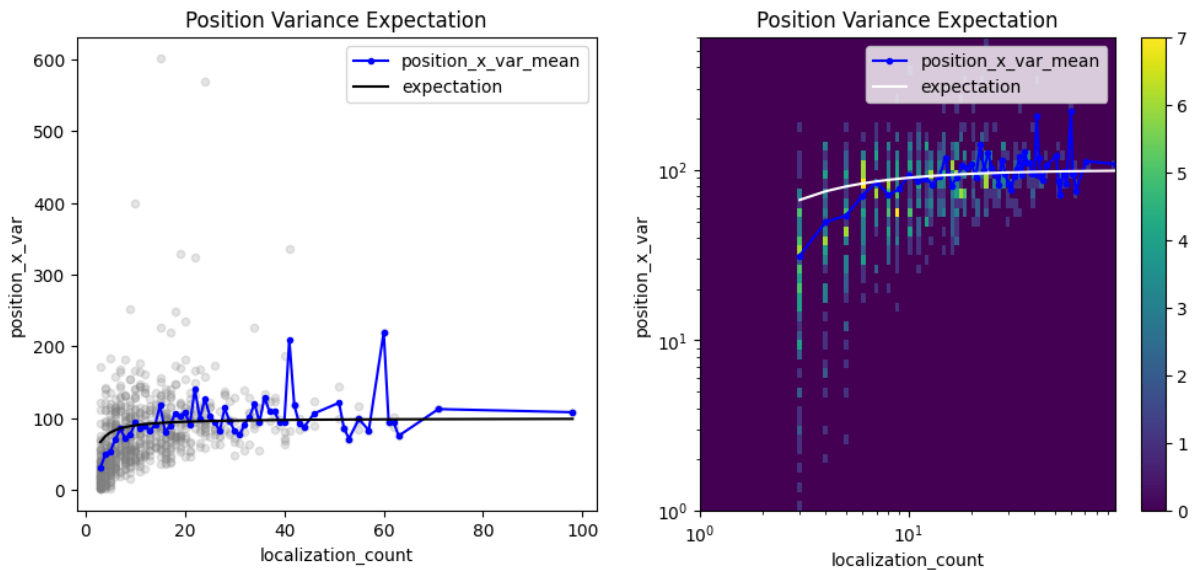


## 2.20.7 Investigate the position variances

```
pve_biased = lc.PositionVarianceExpectation(loc_property="position_x",
    ↪expectation=10**2, biased=True).compute(locdata=clust)
pve_biased.results
```

```
<locan.analysis.position_variance_expectation.
    ↪PositionVarianceExpectationResults at 0x7fd353ac8c10>
```

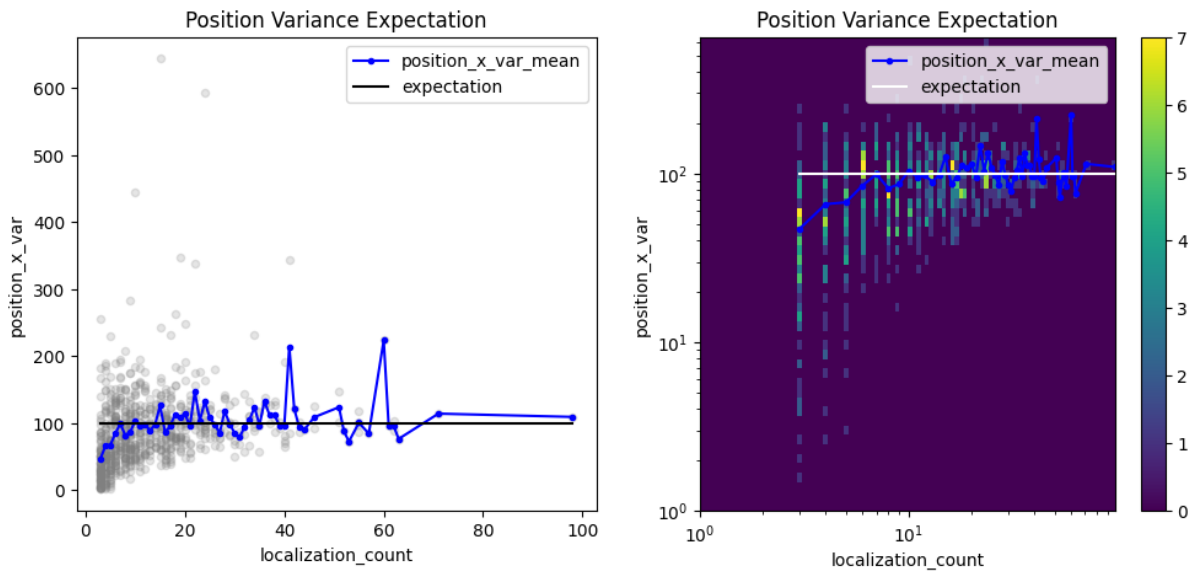
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))
pve_biased.plot(ax=axes[0])
pve_biased.hist(ax=axes[1]);
```



```
pve = lc.PositionVarianceExpectation(loc_property="position_x",
    expectation=10**2, biased=False).compute(locdata=clust)
pve.results
```

```
<locan.analysis.position_variance_expectation.
    PositionVarianceExpectationResults at 0x7fd35e16b8d0>
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))
pve.plot(ax=axes[0])
pve.hist(ax=axes[1], log=True);
```



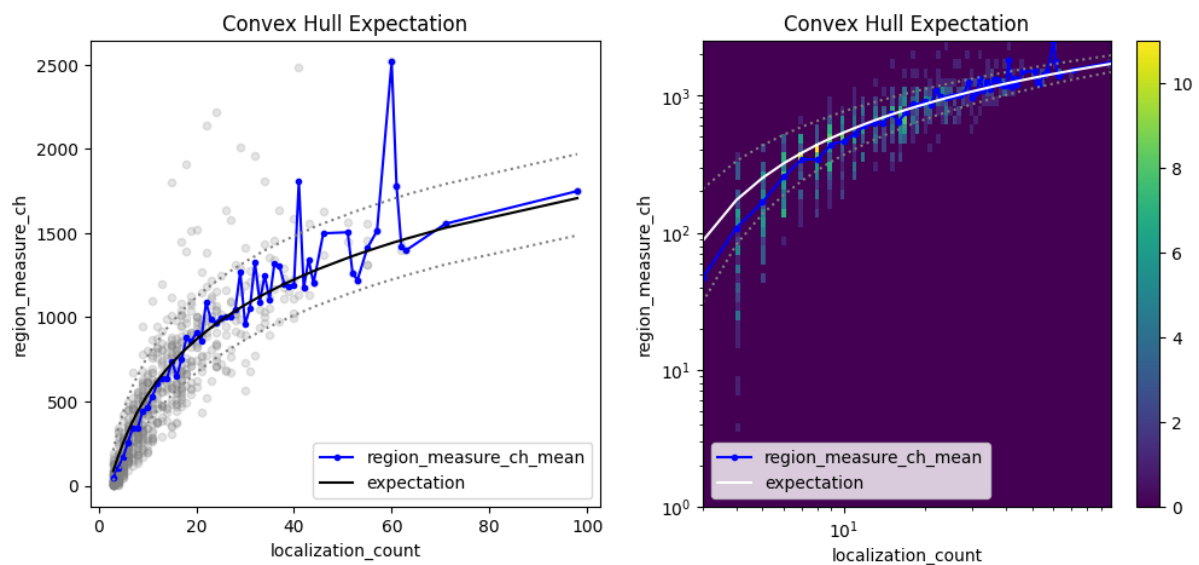


## 2.20.8 Investigate the convex hull areas

```
che = lc.ConvexHullExpectation(convex_hull_property='region_measure_ch',
    expected_variance=10**2).compute(locdata=clust)
che.results
```

```
<locan.analysis.convex_hull_expectation.ConvexHullExpectationResults at
0x7fd3539916d0>
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))
che.plot(ax=axes[0])
che.hist(ax=axes[1]);
```



## 2.20.9 Investigate the uncertainties for cluster centroids

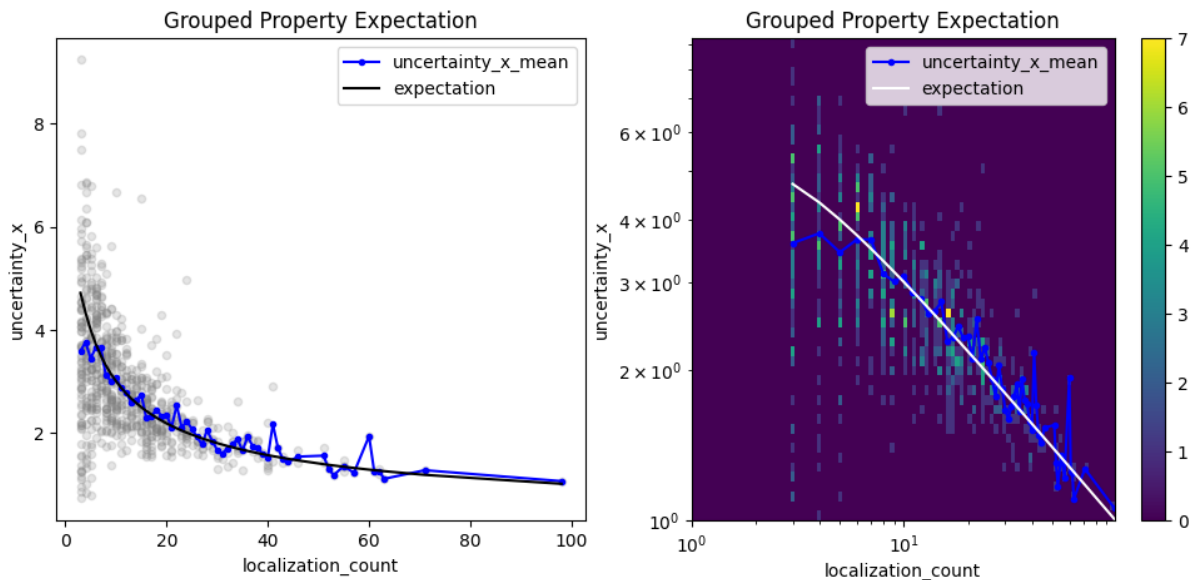
```
n_locs = np.arange(1, 1000)
ground_truth_std = 10
ground_truth_variance = ground_truth_std**2
biased_variance = ground_truth_variance * (1 - 1 / n_locs)
biased_uncertainty = np.sqrt(biased_variance)
expected_uncertainty = biased_uncertainty / np.sqrt(n_locs)
expectation = pd.Series(data=expected_uncertainty, index=n_locs)
expectation;
```

```
loc_property = "uncertainty_x"
other_loc_property = "localization_count"
```

```
gpe = lc.GroupedPropertyExpectation(loc_property=loc_property, other_loc_
    property=other_loc_property, expectation=expectation).compute(locdata=clust)
gpe.results
```

```
<locan.analysis.grouped_property_expectation.  
↳ GroupedPropertyExpectationResults at 0x7fd359c0c890>
```

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))  
gpe.plot(ax=axes[0])  
gpe.hist(ax=axes[1]);
```



## 2.21 Tutorial about how to use a standard Analysis class

```
from pathlib import Path  
  
%matplotlib inline  
  
import matplotlib.pyplot as plt  
  
import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:  
  version: 0.20.0.dev41+g755b969
```

```
Python:  
  version: 3.11.6
```

### 2.21.1 Load SMLM data file

Identify some data in the test\_data directory and provide a path using pathlib.Path

```
path = lc.ROOT_DIR / 'tests/test_data/npc_gp210.asdf'
print(path, '\n')
dat = lc.load_locdata(path=path, file_type=lc.FileType.ASDF)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
↳python3.11/site-packages/locan/tests/test_data/npc_gp210.asdf
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

```
print(dat.data.head(), '\n')
print('Summary:')
dat.print_summary()
print('Properties:')
print(dat.properties)
```

|   | position_x  | position_y  | frame | intensity    | two_kernel_improvement | \ |
|---|-------------|-------------|-------|--------------|------------------------|---|
| 0 | 5631.709961 | 6555.850098 | 24    | 21800.699219 | 0.0                    |   |
| 1 | 5642.899902 | 6550.990234 | 25    | 23352.900391 | 0.0                    |   |
| 2 | 5610.319824 | 6546.819824 | 26    | 5007.509766  | 0.0                    |   |
| 3 | 5713.299805 | 6611.950195 | 141   | 6437.140137  | 0.0                    |   |
| 4 | 5699.750000 | 6627.390137 | 142   | 19998.300781 | 0.0                    |   |

|   | chi_square | local_background |
|---|------------|------------------|
| 0 | 1426930.0  | 1188.489990      |
| 1 | 983634.0   | 1135.199951      |
| 2 | 577707.0   | 1107.219971      |
| 3 | 616255.0   | 1118.430054      |
| 4 | 903412.0   | 1144.709961      |

```
Summary:
identifier: "5"
comment: ""
source: EXPERIMENT
state: MODIFIED
element_count: 202
frame_count: 202
file {
  type: ASDF
  path: "/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/
↳lib/python3.11/site-packages/locan/tests/test_data/npc_gp210.asdf"
}
creation_time {
  2024-01-23T08:32:31.518779Z
}
```

(continues on next page)

(continued from previous page)

```
modification_time {
  2024-01-23T08:32:31.518779Z
}
```

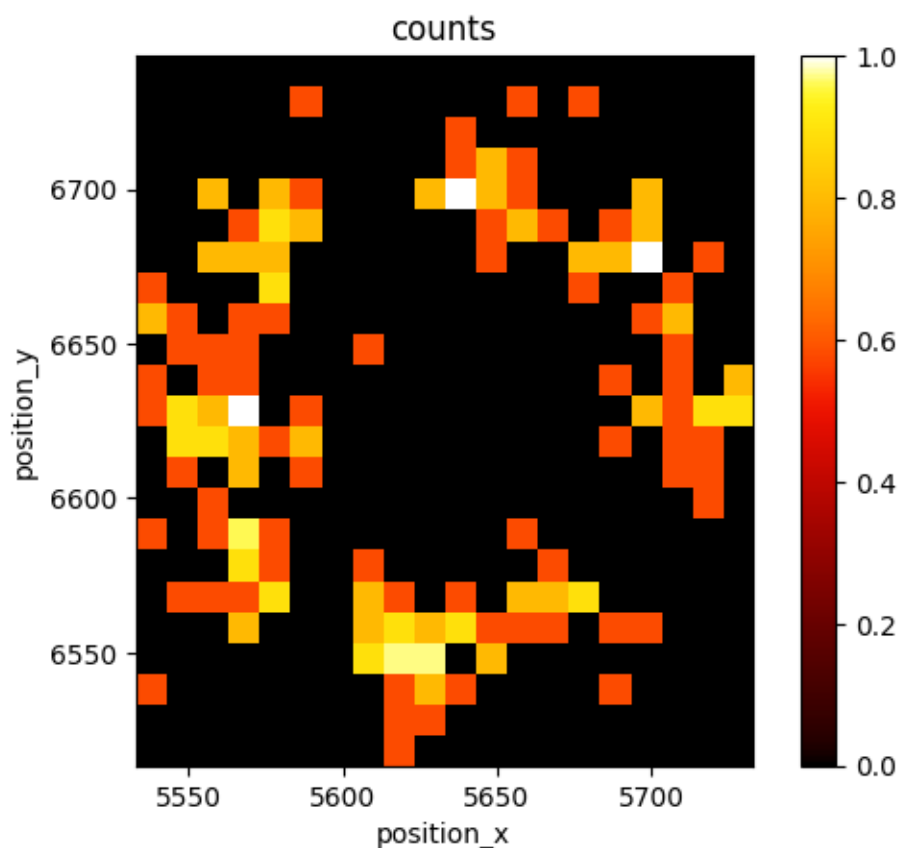
Properties:

```
{'localization_count': 202, 'position_x': 5623.488892810179, 'uncertainty_x': 3.9700846636030356, 'position_y': 6625.534602703435, 'uncertainty_y': 3.999303432482808, 'intensity': 3944778.0, 'local_background': 1131.3207, 'frame': 24, 'region_measure_bb': 47134.543, 'localization_density_bb': 0.00428560429946091, 'subregion_measure_bb': 870.98046875}
```

## 2.21.2 Visualization

For visualizing the data use one of the rendering methods.

```
lc.render_2d_mpl(dat, bin_size=10, rescale=lc.Trafo.EQUALIZE);
```



### 2.21.3 A simple analysis procedure: localization precision

#### Instantiation

Create an instance of the analysis class. By doing this you set all parameters in the parameter attribute. To start the actual computation you have to call instance.compute().

```
lp = lc.LocalizationPrecision(radius=50)
```

Each analysis class provides some attributes and methods for the most common interactions with the computed results.

```
attributes = [x for x in dir(lp) if not x.startswith('_')]
attributes
```

```
['compute',
 'count',
 'distribution_statistics',
 'fit_distributions',
 'hist',
 'meta',
 'parameter',
 'plot',
 'report',
 'results']
```

#### The results attribute

A standard analysis class has an attribute *results* to hold the most fundamental results as number, numpy array or pandas series or dataframe.

```
lp.compute(dat)
print('type of lpf.results: ', type(lp.results), '\n')
print(lp.results.head())
```

```
Processed frames::  0%|          | 0/24884 [00:00<?, ?it/s]
```

```
Processed frames::  6%|          | 1401/24884 [00:00<00:01, 13854.30it/s]
```

```
Processed frames:: 16%|          | 3906/24884 [00:00<00:01, 20010.91it/s]
```

```
Processed frames:: 25%|          | 6245/24884 [00:00<00:00, 21306.66it/s]
```

```
Processed frames:: 36%|          | 8879/24884 [00:00<00:00, 23229.79it/s]
```

```
Processed frames:: 59%|          | 14723/24884 [00:00<00:00, 35796.48it/s]
```

```
Processed frames:: 98%|| 24376/24884 [00:00<00:00, 56172.32it/s]
```

```
Processed frames:: 100%|| 24884/24884 [00:00<00:00, 39835.13it/s]
```

```
type of lpf.results: <class 'pandas.core.frame.DataFrame'>
```

|   | position_delta_x | position_delta_y | position_distance | frame |
|---|------------------|------------------|-------------------|-------|
| 0 | -11.189941       | 4.859863         | 12.199716         | 24    |
| 1 | 32.580078        | 4.170410         | 32.845909         | 25    |
| 2 | 13.549805        | -15.439941       | 20.542370         | 141   |
| 3 | 4.669922         | 3.010254         | 5.556060          | 142   |
| 4 | 20.469727        | 14.750000        | 25.230383         | 239   |

Results can be saved by specifying a path

```
from pathlib import Path
temp_directory = Path('.') / 'temp'
temp_directory.mkdir(parents=True, exist_ok=True)

path = temp_directory / 'results.txt'
path
```

```
PosixPath('temp/results.txt')
```

and saving the data using numpy or pandas routines

```
lp.results.to_csv(path, sep='\t')
```

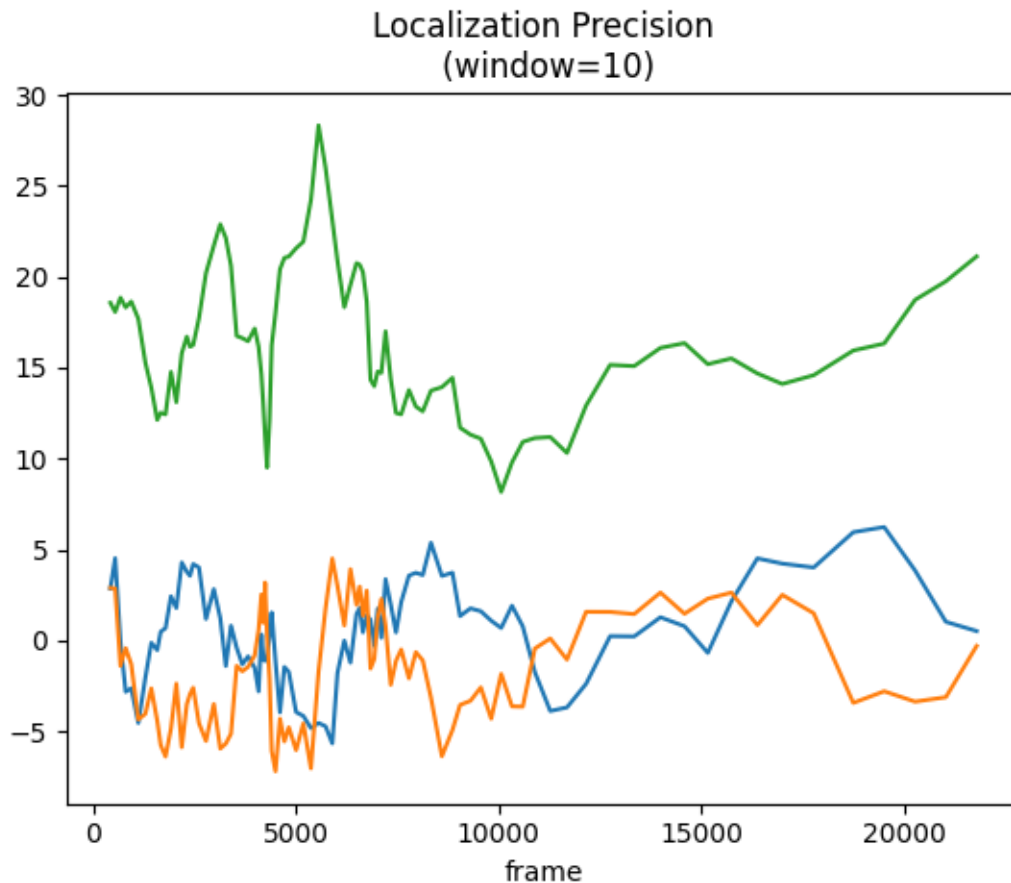
Delete the file and empty directory

```
path.unlink()
temp_directory.rmdir()
```

## A simple standardized plot

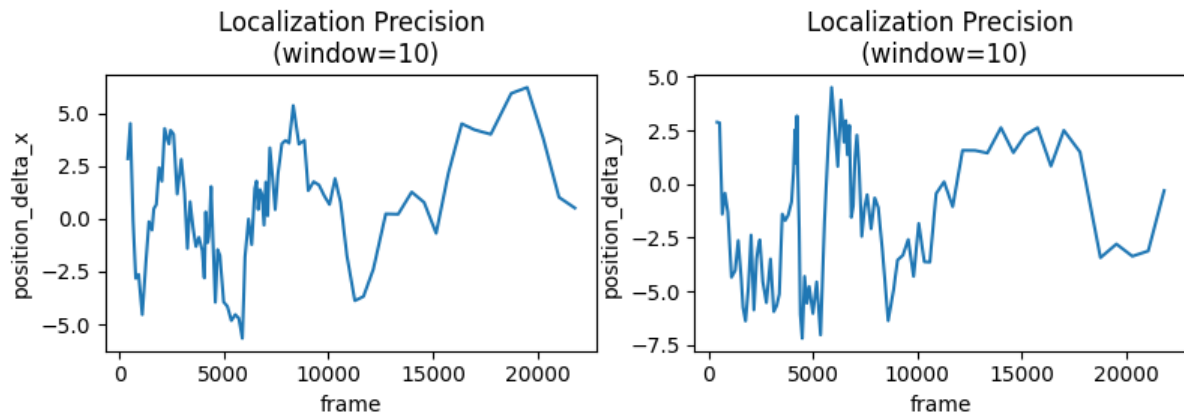
Most likely the results should be inspected by looking at a typical plot. In this case the plot shows results smoothed by a running average according to the specified window.

```
lp.plot(window=10);
```



For more advanced plotting schemes use the matplotlib framework.

```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8,3))
lp.plot(ax=ax[0], window=10, loc_property='position_delta_x')
lp.plot(ax=ax[1], window=10, loc_property='position_delta_y')
plt.tight_layout()
plt.show()
```

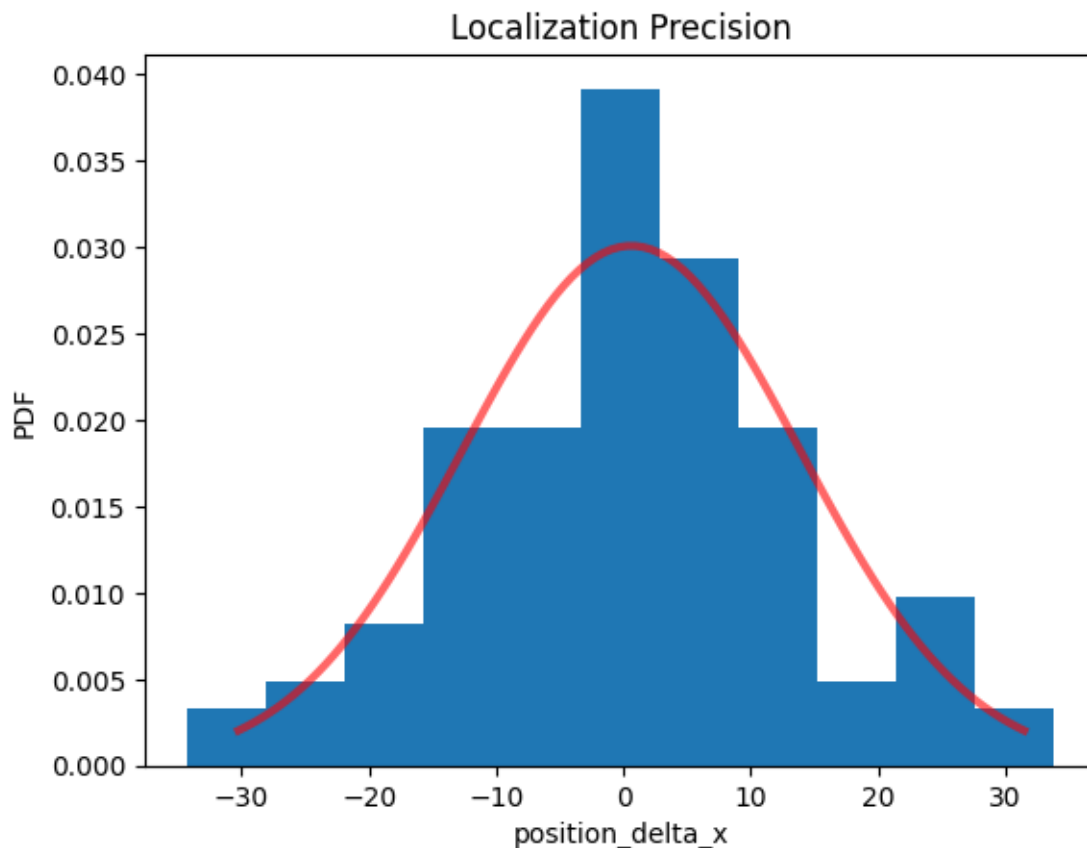


### A simple standardized histogram

Quite often the results are best presented as a histogram. The histogram for the distances per default includes a fit to a distribution expected for normal distributed localizations. Sigma is the localization precision.

The histogram per default provides automatic bins and is normalized to show a probability density function.

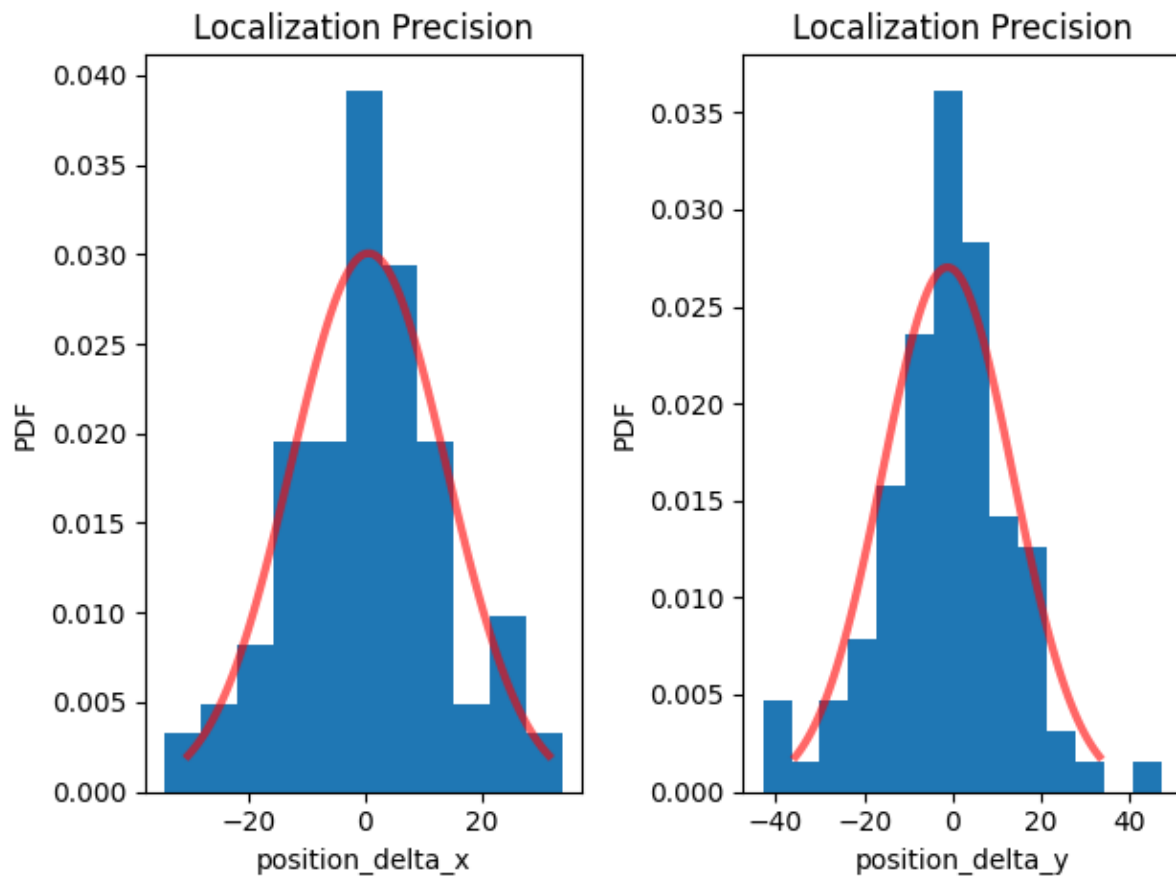
```
lp.hist(loc_property='position_delta_x');
```



Alternatively the position deltas can be histogrammed.



```
fig, ax = plt.subplots(nrows=1, ncols=2)
lp.hist(ax=ax[0], loc_property='position_delta_x')
lp.hist(ax=ax[1], loc_property='position_delta_y')
plt.tight_layout()
plt.show()
```



## Secondary results

Secondary results are e.g. fit parameter derived from analyzing the distribution of *results* values. Secondary results are different for each analysis routine. They are included as additional attributes.

Localization precision can e.g. be derived from fitting the position distances to an appropriate distribution and estimating the sigma parameter.

```
lp.distribution_statistics.parameter_dict()
```

```
{'position_delta_x_loc': 0.6029139,
 'position_delta_x_scale': 13.2682,
 'position_delta_y_loc': -1.227687,
 'position_delta_y_scale': 14.76087,
 'position_distance_sigma': 14.067675781250028,
 'position_distance_loc': 0,
 'position_distance_scale': 1}
```

```
print('position_distance_sigma: ', lp.distribution_statistics.parameter_
↪dict()['position_distance_sigma'])
```

```
position_distance_sigma: 14.067675781250028
```

## Metadata

Each analysis class is supplied with meta data. The main purpose is to (i) capture methods and parameters that were supplied in each instantiation and (ii) provide information on the dataset on which the particular analysis was carried out. Metadata is structured using protocol buffers.

```
lp.meta
```

```
identifier: "1"
method {
  name: "LocalizationPrecision"
  parameter: "{\'radius\': 50}"
}
creation_time {
  seconds: 1710414431
  nanos: 981314000
}
```

You can add some user-defined key-value pairs:

```
lp.meta.map['some key'] = 'some value'
lp.meta.map
```

```
{'some key': 'some value'}
```

### 2.21.4 Metadata can be used to rerun the analysis with the same parameter.

```
lp.meta.method.name
```

```
'LocalizationPrecision'
```

```
lp.meta.method.parameter
```

```
"{'radius': 50}"
```

```
import ast
import locan.analysis
params = ast.literal_eval(lp.meta.method.parameter)
print(params)
lp_2 = getattr(locan.analysis, lp.meta.method.name)(**params)
lp_2.compute(dat)
lp_2.results.head()
```

```
{'radius': 50}
```

```
Processed frames:: 0%|          | 0/24884 [00:00<?, ?it/s]
```

```
Processed frames:: 9%|          | 2315/24884 [00:00<00:01, 22329.52it/s]
```

```
Processed frames:: 18%|          | 4548/24884 [00:00<00:00, 20478.09it/s]
```

```
Processed frames:: 29%|          | 7174/24884 [00:00<00:00, 22633.81it/s]
```

```
Processed frames:: 43%|          | 10816/24884 [00:00<00:00, 27816.48it/s]
```

```
Processed frames:: 74%|          | 18343/24884 [00:00<00:00, 44441.31it/s]
```

```
Processed frames:: 100%|| 24884/24884 [00:00<00:00, 41875.98it/s]
```

|   | position_delta_x | position_delta_y | position_distance | frame |
|---|------------------|------------------|-------------------|-------|
| 0 | -11.189941       | 4.859863         | 12.199716         | 24    |
| 1 | 32.580078        | 4.170410         | 32.845909         | 25    |
| 2 | 13.549805        | -15.439941       | 20.542370         | 141   |
| 3 | 4.669922         | 3.010254         | 5.556060          | 142   |
| 4 | 20.469727        | 14.750000        | 25.230383         | 239   |

## 2.22 Tutorial about setting up an analysis pipeline and batch processing

Quite often you experiment with various analysis routines and appropriate parameters and come up with an analysis pipeline. A pipeline procedure then is a script defining analysis steps for a single locdata object (or a single group of corresponding locdatas as for instance used in 2-color measurements).

The Pipeline class can be used to combine the pipeline code, metadata and analysis results in a single pickleable object (meaning it can be serialized by the python pickle module).

This pipeline might then be applied to a number of similar datasets. A batch process is such a procedure for running a pipeline over multiple locdata objects and collecting and combining results.

```
from pathlib import Path

%matplotlib inline

import matplotlib.pyplot as plt

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

## 2.22.1 Apply a pipeline of different analysis routines

### Load rapidSTORM data file

```
path = lc.ROOT_DIR / 'tests/test_data/npc_gp210.asdf'
print(path, '\n')
dat = lc.load_locdata(path=path, file_type=lc.FileType.ASDF)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/
python3.11/site-packages/locan/tests/test_data/npc_gp210.asdf
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

```
dat.properties
```

```
{'localization_count': 202,
 'position_x': 5623.488892810179,
 'uncertainty_x': 3.9700846636030356,
 'position_y': 6625.534602703435,
 'uncertainty_y': 3.999303432482808,
 'intensity': 3944778.0,
 'local_background': 1131.3207,
 'frame': 24,
 'region_measure_bb': 47134.543,
 'localization_density_bb': 0.00428560429946091,
 'subregion_measure_bb': 870.98046875}
```

### Set up an analysis procedure

First define the analysis procedure (pipeline) in form of a computation function. Make sure the first parameter is the self referring to the Pipeline object. Add arbitrary keyword arguments thereafter. When finishing with return self the compute method can easily be called with instantiation.

```
def computation(self, locdata, n_localizations_min=4):

    # import required modules
```

(continues on next page)

(continued from previous page)

```
from locan.analysis import LocalizationPrecision

# prologue
self.file_indicator = locdata.meta.file.path
self.locdata = locdata

# check requirements
if len(locdata) <= n_localizations_min:
    return None

# compute localization precision
self.lp = LocalizationPrecision().compute(self.locdata)

return self
```

## Run the analysis procedure

Instantiate a Pipeline object and run compute():

```
pipe = lc.Pipeline(computation=computation, locdata=dat, n_localizations_
    ↪ min=4).compute()
pipe.meta
```

```
Processed frames:: 0%|          | 0/24884 [00:00<?, ?it/s]
```

```
Processed frames:: 5%|          | 1317/24884 [00:00<00:01, 13169.23it/s]
```

```
Processed frames:: 16%|         | 3904/24884 [00:00<00:01, 20363.74it/s]
```

```
Processed frames:: 25%|         | 6166/24884 [00:00<00:00, 21082.94it/s]
```

```
Processed frames:: 33%|         | 8270/24884 [00:00<00:00, 21065.24it/s]
```

```
Processed frames:: 56%|        | 13998/24884 [00:00<00:00, 33719.92it/s]
```

```
Processed frames:: 96%|| 23938/24884 [00:00<00:00, 55707.04it/s]
```

```
Processed frames:: 100%|| 24884/24884 [00:00<00:00, 39111.39it/s]
```

```
identifier: "1"
method {
  name: "Pipeline"
  parameter: "{\'computation\': <function computation at 0x7f677e5874c0>, \
    ↪ \'locdata\': <locan.data.locdata.LocData object at 0x7f677f7e78d0>, \'n_
```

(continues on next page)

(continued from previous page)

```
↪localizations_min\': 4}"
}
creation_time {
  seconds: 1710414475
  nanos: 844844000
}
```

Results are available from Pipeline object in form of attributes defined in the compute function:

```
[attr for attr in dir(pipe) if not attr.startswith('__') and not attr.
↪endswith('__')]
```

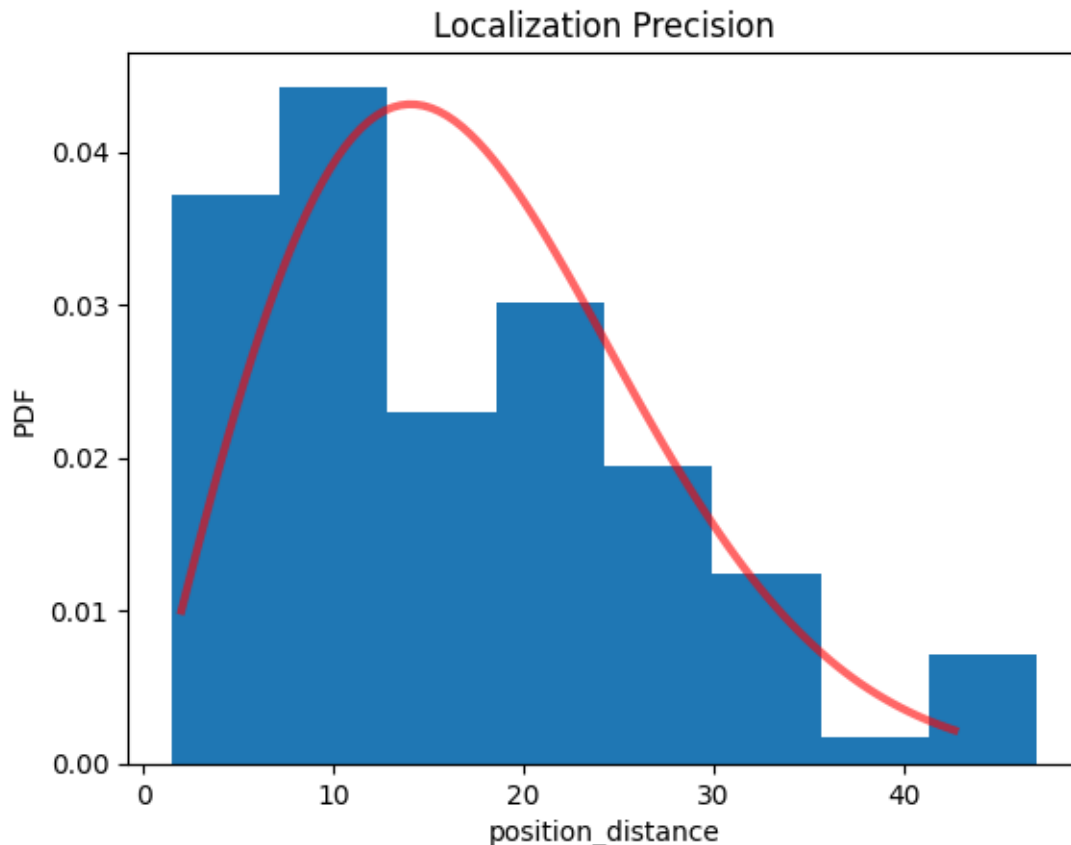
```
['_get_parameters',
 '_init_meta',
 '_update_meta',
 'computation',
 'computation_as_string',
 'compute',
 'count',
 'file_indicator',
 'kwargs',
 'locdata',
 'lp',
 'meta',
 'parameter',
 'report',
 'results',
 'save_computation']
```

```
pipe.lp.results.head()
```

|   | position_delta_x | position_delta_y | position_distance | frame |
|---|------------------|------------------|-------------------|-------|
| 0 | -11.189941       | 4.859863         | 12.199716         | 24    |
| 1 | 32.580078        | 4.170410         | 32.845909         | 25    |
| 2 | 13.549805        | -15.439941       | 20.542370         | 141   |
| 3 | 4.669922         | 3.010254         | 5.556060          | 142   |
| 4 | 20.469727        | 14.750000        | 25.230383         | 239   |

```
pipe.lp.hist();
print(pipe.lp.distribution_statistics.parameter_dict())
```

```
{'position_delta_x_loc': 0.6029139, 'position_delta_x_scale': 13.2682,
↪'position_delta_y_loc': -1.227687, 'position_delta_y_scale': 14.76087,
↪'position_distance_sigma': 14.067675781250028, 'position_distance_loc': 0,
↪'position_distance_scale': 1}
```



You can recover the computation procedure:

```
pipe.computation_as_string()
```

```
'def computation(self, locdata, n_localizations_min=4):\n    \n    # import_\n    ↪required modules\n    from locan.analysis import LocalizationPrecision\n    ↪\n    # prologue\n    self.file_indicator = locdata.meta.file.path\n    ↪\n    self.locdata = locdata\n    \n    # check requirements\n    if len(locdata)\n    ↪<=n_localizations_min:\n        return None\n    \n    # compute_\n    ↪localization precision\n    self.lp = LocalizationPrecision().compute(self.\n    ↪locdata)\n    \n    return self\n'
```

or save it as text protocol:

The Pipeline object is pickleable and can thus be saved for revisits.

## 2.22.2 Apply the pipeline on multiple datasets - a batch process

Let's create multiple datasets:

```
path = lc.ROOT_DIR / 'tests/test_data/npc_gp210.asdf'\nprint(path, '\\n')\ndat = lc.load_locdata(path=path, file_type=lc.FileType.ASDF)\n\nlocdatas = [lc.select_by_condition(dat, f'{min}<index<{max}') for min, max in_\n
```

(continues on next page)

(continued from previous page)

```
↪((0,100), (101,202))]  
locdatas
```

```
/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/  
↪python3.11/site-packages/locan/tests/test_data/npc_gp210.asdf
```

```
[<locan.data.locdata.LocData at 0x7f677aafa710>,  
 <locan.data.locdata.LocData at 0x7f677a3209d0>]
```

Run the analysis pipeline as batch process

```
pipes = [lc.Pipeline(computation=computation, locdata=dat).compute() for dat_  
↪in locdatas]
```

```
Processed frames:: 0%|          | 0/6164 [00:00<?, ?it/s]
```

```
Processed frames:: 18%|         | 1113/6164 [00:00<00:00, 10409.72it/s]
```

```
Processed frames:: 35%|        | 2154/6164 [00:00<00:00, 8618.86it/s]
```

```
Processed frames:: 49%|        | 3032/6164 [00:00<00:00, 8644.71it/s]
```

```
Processed frames:: 63%|       | 3906/6164 [00:00<00:00, 8622.60it/s]
```

```
Processed frames:: 77%|       | 4774/6164 [00:00<00:00, 7723.44it/s]
```

```
Processed frames:: 100%|| 6164/6164 [00:00<00:00, 9060.59it/s]
```

```
Processed frames:: 0%|          | 0/18639 [00:00<?, ?it/s]
```

```
Processed frames:: 4%|          | 721/18639 [00:00<00:02, 6777.17it/s]
```

```
Processed frames:: 11%|         | 2005/18639 [00:00<00:01, 9743.37it/s]
```

```
Processed frames:: 19%|        | 3632/18639 [00:00<00:01, 12480.74it/s]
```

```
Processed frames:: 32%|        | 5936/18639 [00:00<00:00, 16008.57it/s]
```

```
Processed frames:: 56%|       | 10512/18639 [00:00<00:00, 26183.89it/s]
```

```
Processed frames:: 73%|       | 13645/18639 [00:00<00:00, 27881.09it/s]
```



```
Processed frames:: 100%|| 18639/18639 [00:00<00:00, 33936.77it/s]
```

```
Processed frames:: 100%|| 18639/18639 [00:00<00:00, 25282.72it/s]
```

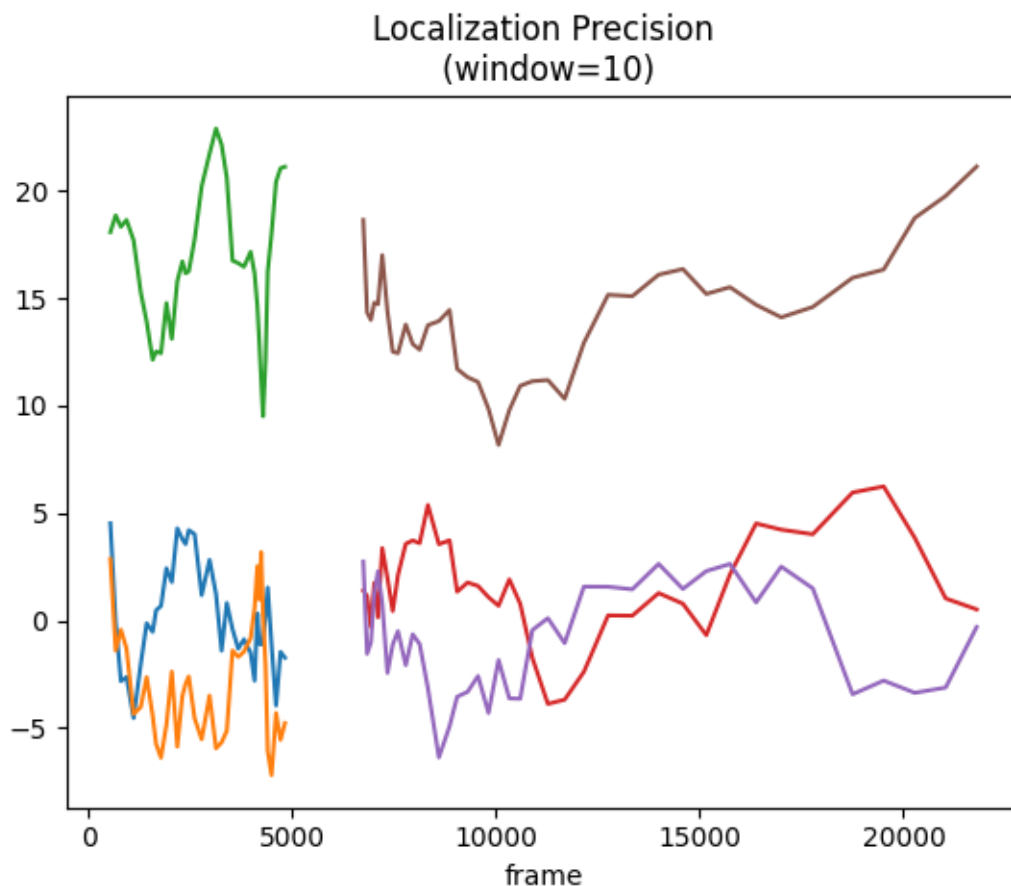
As long as the batch procedure runs in a single computer process, the identifier increases with every instantiation.

```
[pipe.meta.identifier for pipe in pipes]
```

```
['2', '3']
```

### Visualize the combined results

```
fig, ax = plt.subplots(nrows=1, ncols=1)
for pipe in pipes:
    pipe.lp.plot(ax=ax, window=10)
plt.show()
```



## 2.23 Tutorial about managing files in batch processing

When analysing a set of experiments you want to collect, match and group files according to information content and experimental conditions.

The Files class will help you.

```
from pathlib import Path
import tempfile

%matplotlib inline

import matplotlib.pyplot as plt

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6
```

### 2.23.1 Some file structure to be analysed

```
directory = Path(tempfile.mkdtemp())
subdirectory = directory.joinpath("sub_directory")
subdirectory.mkdir()
```

```
files = [
    directory / "sub_directory" / "file_group_a_0.data",
    directory / "sub_directory" / "file_group_a_1.data",
    directory / "sub_directory" / "file_group_b_2.data",
    directory / "sub_directory" / "corresponding_file_0.data",
    directory / "metadata.meta",
]
for file_ in files:
    file_.touch()
```

```
list(directory.glob("**/*.data"))
```

```
[PosixPath('/tmp/tmp_hvjxqz3/metadata.meta'),
 PosixPath('/tmp/tmp_hvjxqz3/sub_directory/file_group_b_2.data'),
 PosixPath('/tmp/tmp_hvjxqz3/sub_directory/file_group_a_1.data'),
 PosixPath('/tmp/tmp_hvjxqz3/sub_directory/file_group_a_0.data'),
 PosixPath('/tmp/tmp_hvjxqz3/sub_directory/corresponding_file_0.data')]
```

## 2.23.2 The Files class

```
lc.Files?
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

## 2.23.3 Identify files

```
files = lc.Files.from_glob(
    directory=directory,
    pattern="**/*.*"
)
files.df
```

```

                                file_path
0                                /tmp/tmp_hvjxqz3/metadata.meta
1  /tmp/tmp_hvjxqz3/sub_directory/file_group_b_2....
2  /tmp/tmp_hvjxqz3/sub_directory/file_group_a_1....
3  /tmp/tmp_hvjxqz3/sub_directory/file_group_a_0....
4  /tmp/tmp_hvjxqz3/sub_directory/corresponding_f...
```

For each file a Path object is stored:

```
files.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/1019267881.py:1: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
  files.df.applymap(lambda x: x.name)
```

```

                                file_path
0                                metadata.meta
1                                file_group_b_2.data
2                                file_group_a_1.data
3                                file_group_a_0.data
4  corresponding_file_0.data
```

```
files.print_summary()
```

```
Number of files: 5
Base directory: /tmp/tmp_hvjxqz3
Columns: Index(['file_path'], dtype='object')
      file_path
count          5
unique          5
```

### 2.23.4 Exclude files

```
files = lc.Files.from_glob(
    directory=directory,
    pattern="**/*.*"
)

files.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/641744908.py:6: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
    files.df.applymap(lambda x: x.name)
```

```

          file_path
0      metadata.meta
1    file_group_b_2.data
2    file_group_a_1.data
3    file_group_a_0.data
4  corresponding_file_0.data
```

```
stoplist = lc.Files.concatenate([
    lc.Files.from_glob(directory=files.directory, pattern="**/*.meta"),
    lc.Files.from_glob(directory=files.directory, pattern="**/*group_b*.*")
])
stoplist.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/823210556.py:5: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
    stoplist.df.applymap(lambda x: x.name)
```

```

          file_path
0      metadata.meta
1  file_group_b_2.data
```

```
files.exclude(stoplist=stoplist)
files.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/3083961993.py:2: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
    files.df.applymap(lambda x: x.name)
```

```

          file_path
2    file_group_a_1.data
3    file_group_a_0.data
4  corresponding_file_0.data
```

### 2.23.5 Match corresponding files

```
files = lc.Files.from_glob(
    directory=directory,
    pattern="**/*.*",
    regex="group_a_0"
)
files.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/3682074927.py:6: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
files.df.applymap(lambda x: x.name)
```

```
file_path
0 file_group_a_0.data
```

```
corresponding_files = lc.Files.from_glob(
    directory=directory,
    pattern="**/*.*",
    regex="corresponding"
)
corresponding_files.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/257078577.py:6: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
corresponding_files.df.applymap(lambda x: x.name)
```

```
file_path
0 corresponding_file_0.data
```

```
files.match_files(files=corresponding_files.df)
files.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/1564609699.py:2: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
files.df.applymap(lambda x: x.name)
```

```
file_path      other_file_path
0 file_group_a_0.data  corresponding_file_0.data
```

### 2.23.6 Match metadata files

```
files = lc.Files.from_glob(
    directory=directory,
    pattern="**/*.meta",
    regex="group_a_0"
)
files.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/3682074927.py:6: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
    files.df.applymap(lambda x: x.name)
```

```
      file_path
0  file_group_a_0.data
```

```
files.match_file_upstream(pattern="*.meta")
files.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/745566580.py:2: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
    files.df.applymap(lambda x: x.name)
```

```
      file_path      metadata
0  file_group_a_0.data  metadata.meta
```

### 2.23.7 Group files

```
files = lc.Files.from_glob(
    directory=directory,
    pattern="**/file*.data"
)
files.df.applymap(lambda x: x.name)
```

```
/tmp/ipykernel_1373/1768514306.py:5: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
    files.df.applymap(lambda x: x.name)
```

```
      file_path
0  file_group_b_2.data
1  file_group_a_1.data
2  file_group_a_0.data
```

```
files.set_group_identifier(name="A", pattern="group_a")
files.df
```

```

                                file_path group
0  /tmp/tmp_hvjxqz3/sub_directory/file_group_b_2....  NaN
1  /tmp/tmp_hvjxqz3/sub_directory/file_group_a_1....   A
2  /tmp/tmp_hvjxqz3/sub_directory/file_group_a_0....   A

```

```
files.set_group_identifier(name="B", pattern="group_b")
files.df
```

```

                                file_path group
0  /tmp/tmp_hvjxqz3/sub_directory/file_group_b_2....   B
1  /tmp/tmp_hvjxqz3/sub_directory/file_group_a_1....   A
2  /tmp/tmp_hvjxqz3/sub_directory/file_group_a_0....   A

```

```
files.group_identifiers()
```

```
Index(['A', 'B'], dtype='object')
```

```
grouped = files.grouped()
grouped.groups
```

```
{'A': [1, 2], 'B': [0]}
```

### 2.23.8 Indexing and iterating over files

```
files = lc.Files.from_glob(
    directory=directory,
    pattern="**/file*.data"
)
files.df.applymap(lambda x: x.name)
```

```

/tmp/ipykernel_1373/1768514306.py:5: FutureWarning: DataFrame.applymap has
↳ been deprecated. Use DataFrame.map instead.
files.df.applymap(lambda x: x.name)

```

```

                                file_path
0  file_group_b_2.data
1  file_group_a_1.data
2  file_group_a_0.data

```

Slicing Files yield a new Files instance:

```
files[0:3]
```

```
<locan.locan_io.files.Files at 0x7f740224e510>
```

Indexing Files yields a Series with the selected row:

```
files[0]
```

```
file_path    /tmp/tmp_hvjxqz3/sub_directory/file_group_b_2....
Name: 0, dtype: object
```

Iterating over Files yields a namedtuple for each row:

```
for file in files:
    print(file)
    print(file.file_path)
```

```
Files(Index=0, file_path=PosixPath('/tmp/tmp_hvjxqz3/sub_directory/file_group_
↳ b_2.data'))
/tmp/tmp_hvjxqz3/sub_directory/file_group_b_2.data
Files(Index=1, file_path=PosixPath('/tmp/tmp_hvjxqz3/sub_directory/file_group_
↳ a_1.data'))
/tmp/tmp_hvjxqz3/sub_directory/file_group_a_1.data
Files(Index=2, file_path=PosixPath('/tmp/tmp_hvjxqz3/sub_directory/file_group_
↳ a_0.data'))
/tmp/tmp_hvjxqz3/sub_directory/file_group_a_0.data
```

## 2.24 Tutorial about multiprocessing using ray

We will describe how to set up an analysis pipeline to process multiple datasets in parallel using the framework `ray`.

```
import sys
import logging

%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import ray

import locan as lc
```

```
/tmp/ipykernel_1620/3782818269.py:7: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳ 54466
```

(continues on next page)



(continued from previous page)

```
import pandas as pd
```

```
lc.show_versions(dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6

System:
python-bits: 64
  system: Linux
  release: 5.19.0-1028-aws
  version: #29~22.04.1-Ubuntu SMP Tue Jun 20 19:12:11 UTC 2023
  machine: x86_64
processor: x86_64
byteorder: little
  LC_ALL: None
  LANG: C.UTF-8
  LOCALE: {'language-code': 'en_US', 'encoding': 'UTF-8'}
```

### 2.24.1 Activate logging

```
logging.basicConfig(stream=sys.stdout, level=logging.INFO, format='
↳%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger()
```

For changing the configuration logging has to be reloaded or the kernel be restarted.

### 2.24.2 Synthetic data

Simulate 3 datasets of localization data that is homogeneously Poisson distributed and treat them as files.

```
rng = np.random.default_rng(seed=1)
```

```
locdatas = [lc.simulate_Poisson(intensity=1e-3, region=((0,1000), (0,1000)),
↳seed=rng) for _ in range(3)]
files = locdatas

print("Element_counts:", [locdata.meta.element_count for locdata in locdatas])
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
Element_counts: [1001, 994, 978]
```

### 2.24.3 Analysis pipeline

Define an analysis pipeline. Typically a pipeline processes a single file, which in this example will be an element of locdatas.

Within the analysis procedure there will be more random number generation involved. Therefore a correctly generated seed has to be passed.

```
def computation(self, file, seed):
    logging.basicConfig(level=logging.INFO)
    logger.info(f'computation started for file: {file}')

    rng = np.random.default_rng(seed=seed)

    other_locdata = lc.simulate_Poisson(intensity=1e-3, region=((0,1000), (0,
↪1000))), seed=rng)
    self.nn = lc.NearestNeighborDistances().compute(locdata=file, other_
↪locdata=other_locdata)

    return self
```

### 2.24.4 Run analysis in parallel

```
ray.init()
# ray.init(num_cpus = 4)
```

```
2024-03-14 11:09:23,446      WARNING services.py:1996 -- WARNING: The
↪object store is using /tmp instead of /dev/shm because /dev/shm has only
↪67108864 bytes available. This will harm performance! You may be able to
↪free up space by deleting files in /dev/shm. If you are inside a Docker
↪container, you can increase /dev/shm size by passing '--shm-size=1.76gb' to
↪'docker run' (or add it to the run_options list in a Ray cluster config).
↪Make sure to set this to more than 30% of available RAM.
```

```
2024-03-14 11:09:24,601      INFO worker.py:1715 -- Started a local Ray
↪instance. View the dashboard at http://127.0.0.1:8265
```

```
RayContext(dashboard_url='127.0.0.1:8265', python_version='3.11.6', ray_
↪version='2.9.2', ray_commit='fce7a361807580953364e2da964f9498f3123bf9',
↪protocol_version=None)
```

```
%%time
@ray.remote
def worker(file, seed):
    pipe = lc.Pipeline(computation=computation, file=file, seed=seed).
↪compute()
    return pipe

n_processes = len(files)
```

(continues on next page)

(continued from previous page)

```
ss = np.random.SeedSequence()
child_seeds = ss.spawn(n_processes)

futures = [worker.remote(file=file, seed=seed) for file, seed in zip(locdatas,
↪ child_seeds)]
pipes = ray.get(futures)
```

```
CPU times: user 785 ms, sys: 138 ms, total: 923 ms
Wall time: 6.72 s
```

## 2.24.5 Visualize the combined results

```
[pipe.meta for pipe in pipes]
```

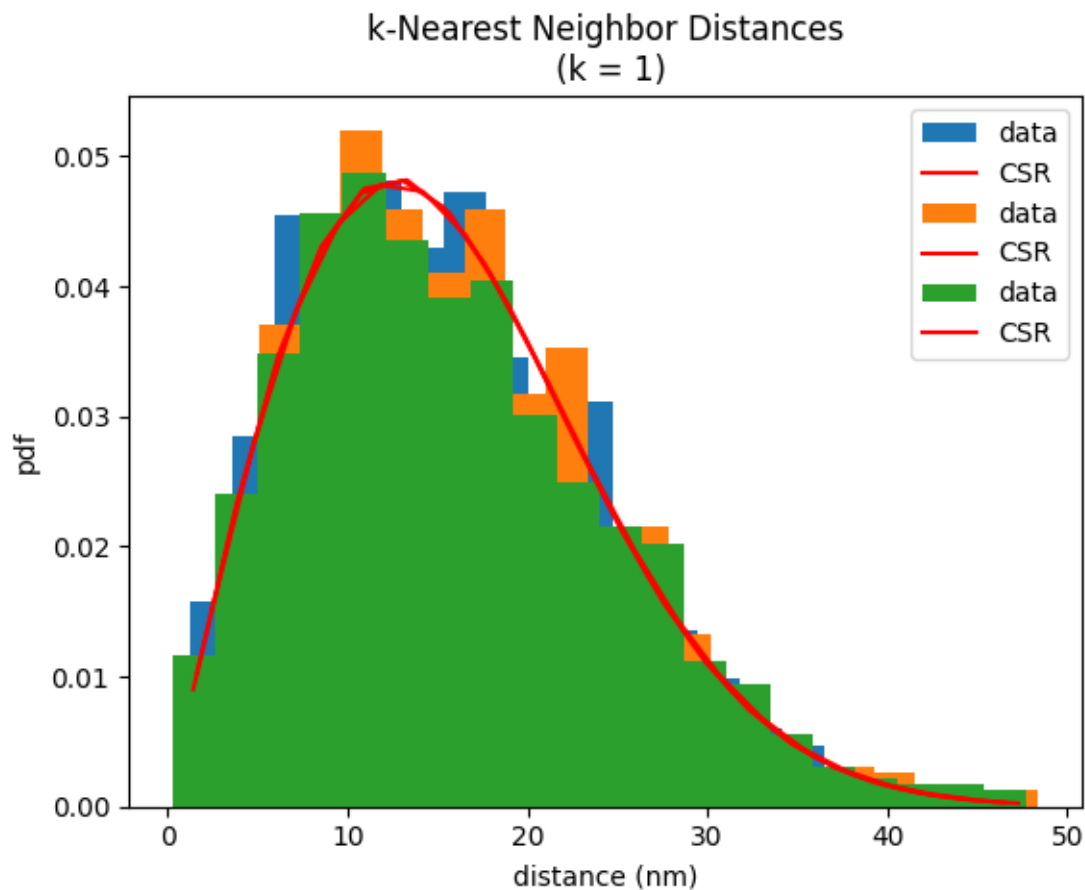
```
[identifier: "1"
  method {
    name: "Pipeline"
    parameter: "{\'computation\': <function computation at 0x7fa4bfe9efc0>, \
↪ \'file\': <locan.data.locdata.LocData object at 0x7fa4bfeb3890>, \\'seed\': \
↪ SeedSequence(\n    entropy=333947175116642173729758860773302131004,\n    \
↪ spawn_key=(0,),\n)}"
  }
  creation_time {
    seconds: 1710414571
    nanos: 809359000
  },
  identifier: "1"
  method {
    name: "Pipeline"
    parameter: "{\'computation\': <function computation at 0x7fcc5171efc0>, \
↪ \'file\': <locan.data.locdata.LocData object at 0x7fcc246f2b50>, \\'seed\': \
↪ SeedSequence(\n    entropy=333947175116642173729758860773302131004,\n    \
↪ spawn_key=(1,),\n)}"
  }
  creation_time {
    seconds: 1710414571
    nanos: 888556000
  },
  identifier: "1"
  method {
    name: "Pipeline"
    parameter: "{\'computation\': <function computation at 0x7fa4bfe9efc0>, \
↪ \'file\': <locan.data.locdata.LocData object at 0x7fa3f1be3b10>, \\'seed\': \
↪ SeedSequence(\n    entropy=333947175116642173729758860773302131004,\n    \
↪ spawn_key=(2,),\n)}"
  }
  creation_time {
```

(continues on next page)

(continued from previous page)

```
seconds: 1710414571
nanos: 837072000
}]
```

```
fig, ax = plt.subplots(nrows=1, ncols=1)
for pipe in pipes:
    pipe.nn.hist(ax=ax)
plt.show()
```



## 2.25 Tutorial about example datasets

```
import sys
from pathlib import Path
import shutil
import logging

import requests

import locan as lc
```

```
lc.show_versions(system=False, dependencies=False, verbose=False)
```

Locan:

```
version: 0.20.0.dev41+g755b969
```

Python:

```
version: 3.11.6
```

```
logging.basicConfig(stream=sys.stdout, level=logging.INFO, format='
↳%(asctime)s - %(name)s - %(levelname)s - %(message)s')
logger = logging.getLogger()
```

### 2.25.1 Load SMLM data from ShareLoc.XYZ

SMLM data can e.g. be found on [ShareLoc.XYZ](#), an open platform for sharing single-molecule localization microscopy data.

Copy a specific download link for downloading a smlm-file.

```
url = "https://zenodo.org/records/7182242/files/UniWue_Tubulin_AF647_3/data.
↳smlm"
url
```

```
'https://zenodo.org/records/7182242/files/UniWue_Tubulin_AF647_3/data.smlm'
```

```
response = requests.get(url)
print("Response is ok: ", response.status_code == requests.codes.ok)
```

```
Response is ok: True
```

```
file_path = Path.home() / Path(url).name

with open(file_path, 'wb') as file:
    for chunk in response.iter_content(chunk_size=128):
        file.write(chunk)

file_path
```

```
PosixPath('/home/docs/data.smlm')
```

## Load data and visualize

```
locdata = lc.load_SMLM_file(file_path)
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
```

Print information about the data:

```
print('Data head:')
print(locdata.data.head(), '\n')
print('Summary:')
locdata.print_summary()
print('Properties:')
print(locdata.properties)
```

```
Data head:
  original_index  position_x  local_background  chi_square  intensity \
0              1  1653.339966      733.021973   6224440.0  13324.799805
1              2  5672.879883      798.614990   2066740.0  10348.000000
2              3  7117.830078      909.901978   1787550.0   6864.439941
3              4  3707.459961      804.187012   2542450.0   9038.549805
4              5 14038.200195      815.343994   1842030.0   8774.750000
```

```
   frame  position_y
0      0  8879.339844
1      0  9851.900391
2      0 11888.099609
3      0 15224.799805
4      0  5597.879883
```

```
Summary:
identifier: "1"
comment: ""
source: EXPERIMENT
state: RAW
element_count: 4582761
frame_count: 45000
file {
  type: SMLM
  path: "/home/docs/data.smlm"
}
creation_time {
  2024-03-14T11:08:25.215283Z
}
```

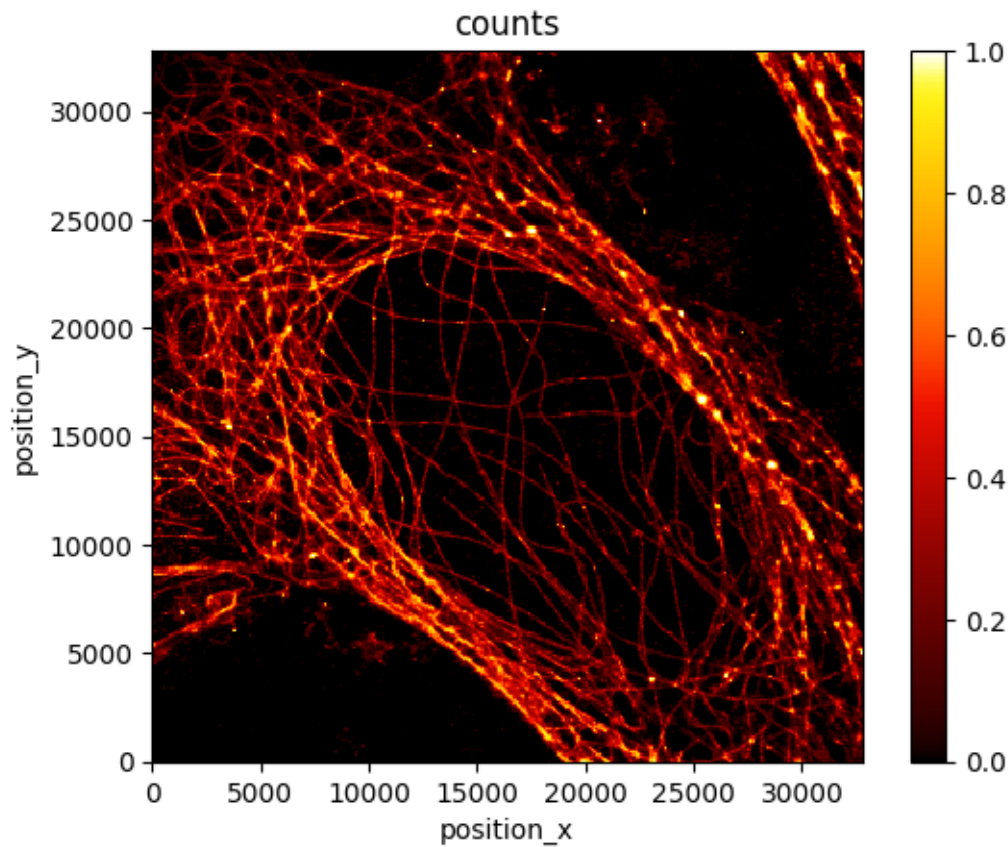
```
Properties:
{'localization_count': 4582761, 'position_x': 15788.771019381189,
  ↪ 'uncertainty_x': 4.646714958816508, 'position_y': 16751.12161338221,
```

(continues on next page)

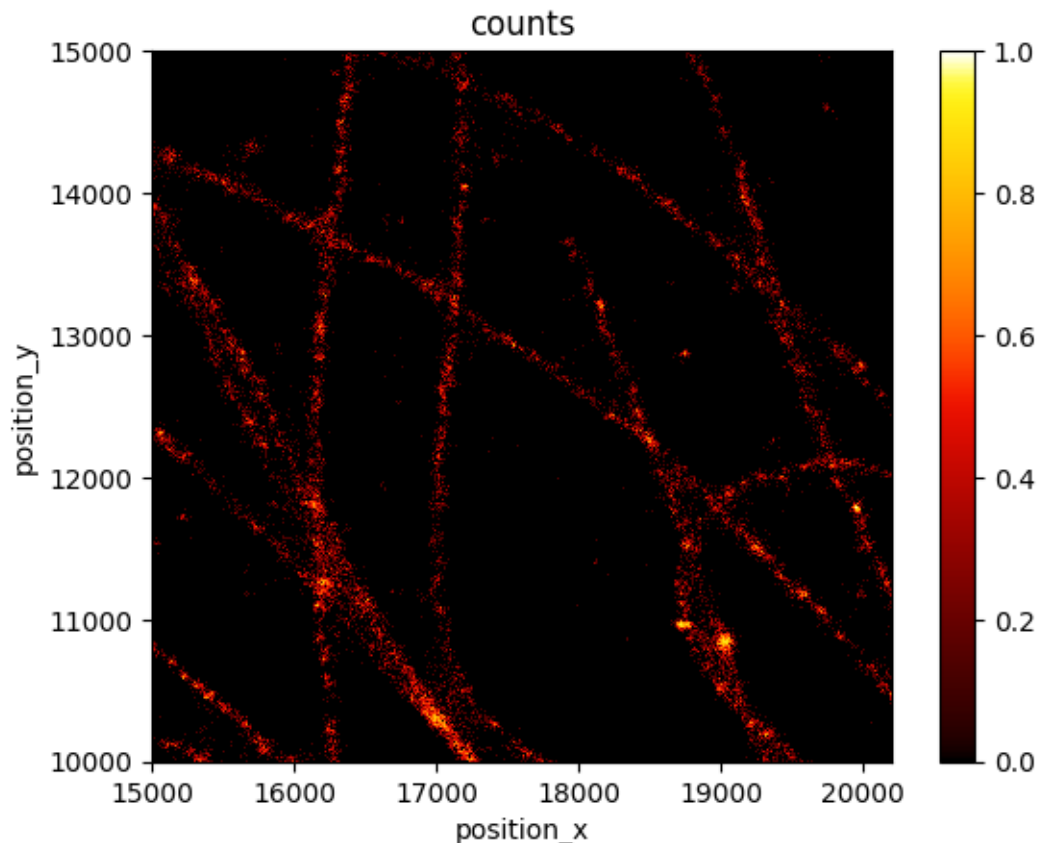
(continued from previous page)

```
↪ 'uncertainty_y': 4.13308495156872, 'intensity': 10823655000.0, 'local_
↪ background': 380.9028, 'frame': 0, 'region_measure_bb': 1078728300.0,
↪ 'localization_density_bb': 0.004248299516230371, 'subregion_measure_bb': 1
↪ 131376.0}
```

```
lc.render_2d(locdata, bin_size=100, rescale=lc.Trafo.EQUALIZE_OP3);
```



```
lc.render_2d(locdata, bin_size=10, rescale=lc.Trafo.EQUALIZE_OP3,
              bin_range=((15_000, 20_200), (10_000, 15_000)));
```



### 2.25.2 Load SMLM data from LocanDatasets

Selected example datasets are provided in a separate directory (repository) called LocanDatasets.

These datasets can be loaded by ready-to-go utility functions.

#### Set up a datasets directory

```
lc.DATASETS_DIR = Path.home() / 'LocanDatasets'
lc.DATASETS_DIR.mkdir(exist_ok=True)
```

#### Load dSTORM data of nuclear pore complexes

This is a rather large 2D dataset with > 2 mio localizations.

```
url = "https://raw.githubusercontent.com/super-resolution/LocanDatasets/main/smlm_data/
↪npc_gp210.asdf"
```

```
response = requests.get(url)
print("Response is ok: ", response.status_code == requests.codes.ok)
```

```
Response is ok: True
```



```
file_path = lc.DATASETS_DIR / 'npc_gp210.asdf'

with open(file_path, 'wb') as file:
    for chunk in response.iter_content(chunk_size=128):
        file.write(chunk)

file_path
```

```
PosixPath('/home/docs/LocanDatasets/npc_gp210.asdf')
```

```
dat = lc.load_npc()
```

Print information about the data:

```
print('Data head:')
print(dat.data.head(), '\n')
print('Summary:')
dat.print_summary()
print('Properties:')
print(dat.properties)
```

```
Data head:
   position_x  position_y  frame  intensity  two_kernel_improvement  \
0  5768.129883  20242.400391     0  83745.398438                0.0
1  21402.800781  18154.599609     0  67648.296875                0.0
2  11410.700195   3155.639893     0  73358.398438                0.0
3  15570.599609  15854.599609     0  65827.898438                0.0
4  22235.500000  12840.900391     0  56347.398438                0.0

   chi_square  local_background
0  4355630.0      1511.459961
1  8383720.0      1637.000000
2  2420450.0      1480.380005
3  2539930.0      1540.670044
4  16152800.0     1572.040039

Summary:
identifier: "13"
comment: ""
source: EXPERIMENT
state: RAW
element_count: 2285189
frame_count: 24999
file {
  type: ASDF
  path: "/home/docs/LocanDatasets/npc_gp210.asdf"
}
creation_time {
  2024-03-14T11:08:30.030663Z
```

(continues on next page)

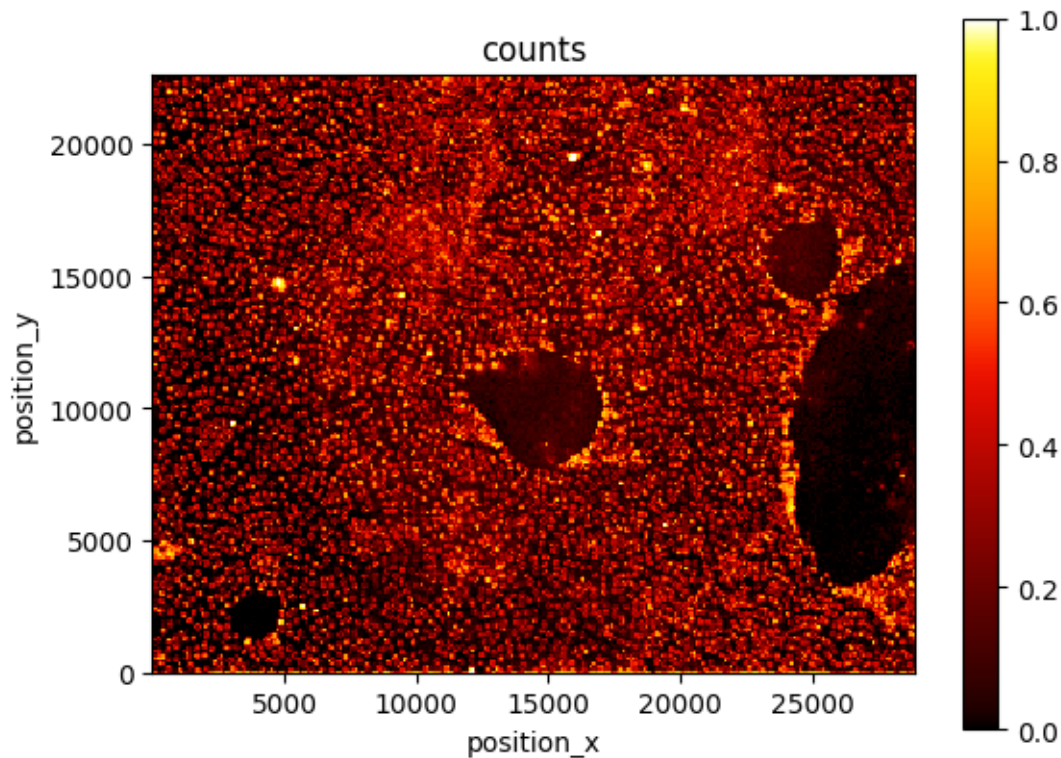
(continued from previous page)

```
}
```

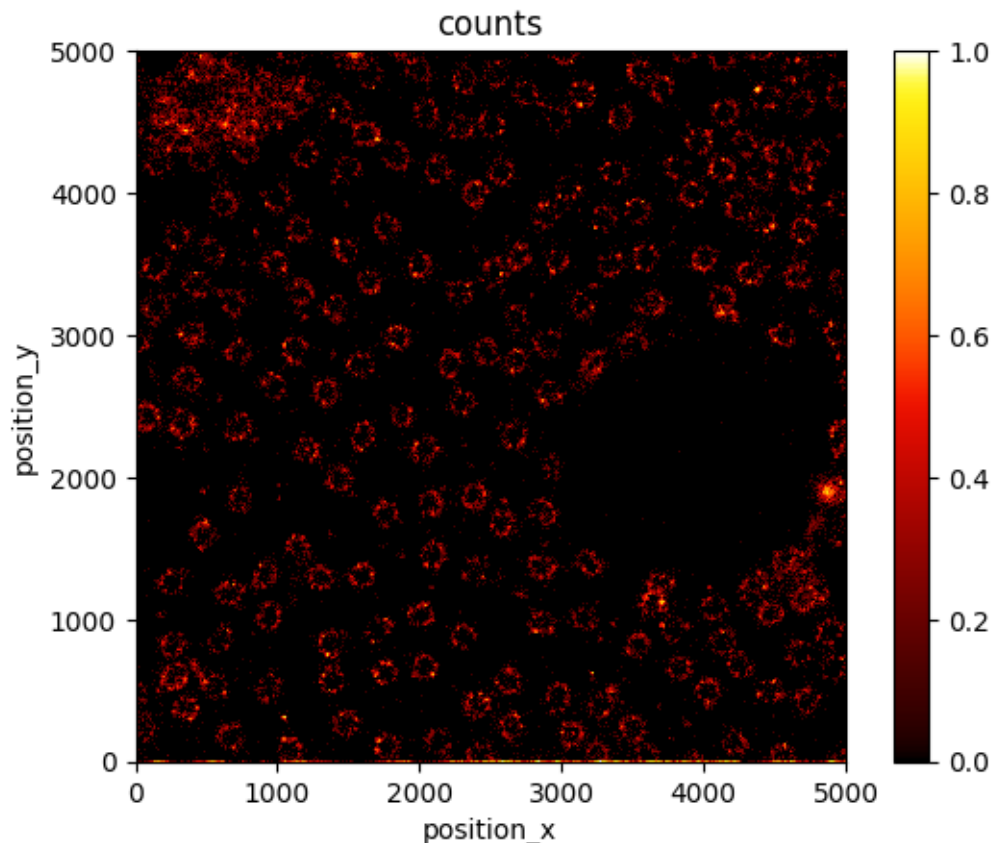
Properties:

```
{'localization_count': 2285189, 'position_x': 14570.820174404367,  
  ↪ 'uncertainty_x': 5.195636170641695, 'position_y': 12028.219687728053,  
  ↪ 'uncertainty_y': 4.523940296807998, 'intensity': 30375694000.0, 'local_  
  ↪ background': 1124.8801, 'frame': 0, 'region_measure_bb': 655392800.0,  
  ↪ 'localization_density_bb': 0.0034867473545268047, 'subregion_measure_bb':  
  ↪ 103168.0}
```

```
lc.render_2d(dat, bin_size=100, rescale=lc.Trafo.EQUALIZE_OP3);
```



```
lc.render_2d(dat, bin_size=10, rescale=lc.Trafo.EQUALIZE_OP3,  
  bin_range=((0, 5000), (0, 5000)));
```



### Load dSTORM data of microtubules

This is a rather large 2D dataset with about 1.5 mio localizations.

```
url = "https://raw.githubusercontent.com/super-resolution/LocanDatasets/main/smlm_data/
↳tubulin_cos7.asdf"
```

```
response = requests.get(url)
print("Response is ok: ", response.status_code == requests.codes.ok)
```

```
Response is ok: True
```

```
file_path = lc.DATASETS_DIR / 'tubulin_cos7.asdf'

with open(file_path, 'wb') as file:
    for chunk in response.iter_content(chunk_size=128):
        file.write(chunk)

file_path
```

```
PosixPath('/home/docs/LocanDatasets/tubulin_cos7.asdf')
```

```
dat = lc.load_tubulin()
```

Print information about the data:

```
print('Data head:')
print(dat.data.head(), '\n')
print('Summary:')
dat.print_summary()
print('Properties:')
print(dat.properties)
```

Data head:

|   | position_x   | position_y   | frame | intensity    | chi_square \ |
|---|--------------|--------------|-------|--------------|--------------|
| 0 | 9937.400391  | 16751.300781 | 0     | 40501.601562 | 3744920.0    |
| 1 | 9998.709961  | 12022.799805 | 0     | 36280.300781 | 14295400.0   |
| 2 | 9566.769531  | 8078.229980  | 0     | 29984.000000 | 12302200.0   |
| 3 | 15492.500000 | 10120.400391 | 0     | 38488.300781 | 3219820.0    |
| 4 | 6381.459961  | 16057.700195 | 0     | 37093.300781 | 1620450.0    |

local\_background

|   |            |
|---|------------|
| 0 | 709.413025 |
| 1 | 800.455017 |
| 2 | 890.807007 |
| 3 | 495.067993 |
| 4 | 476.035004 |

Summary:

identifier: "7"

comment: ""

source: EXPERIMENT

state: RAW

element\_count: 1506568

frame\_count: 74969

file {

type: ASDF

path: "/home/docs/LocanDatasets/tubulin\_cos7.asdf"

}

creation\_time {

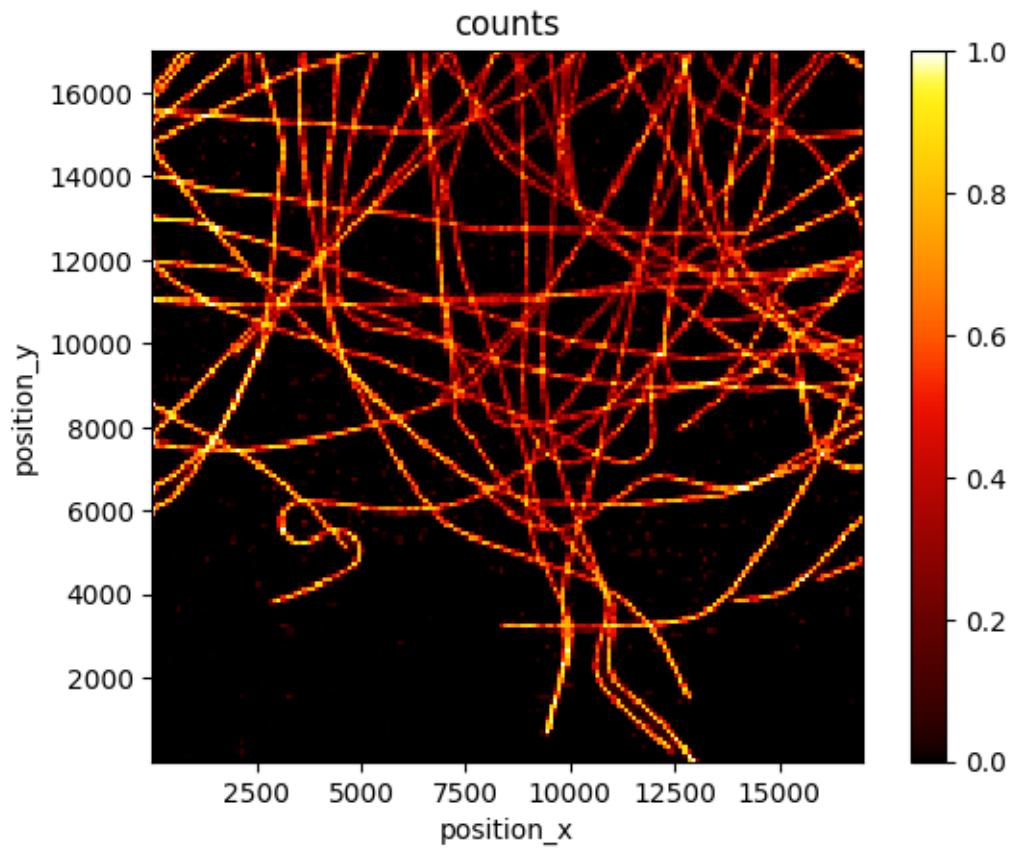
2024-03-14T11:08:36.248853Z

}

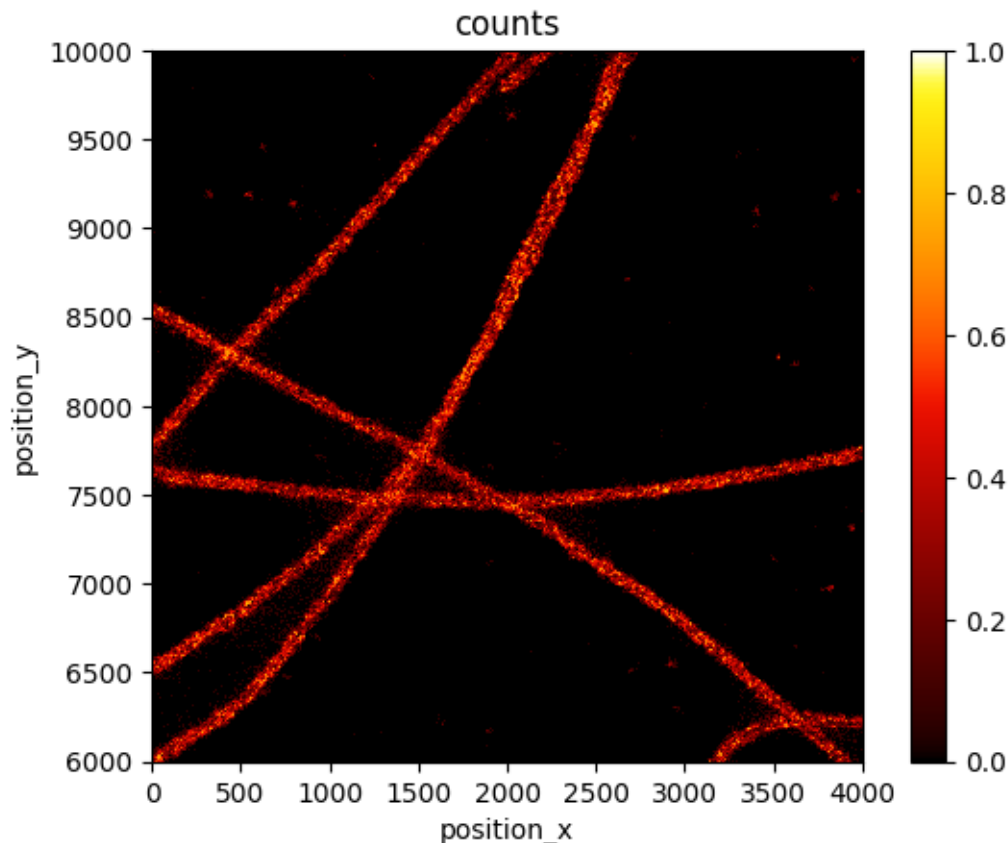
Properties:

```
{'localization_count': 1506568, 'position_x': 8833.605738466662, 'uncertainty_
↪x': 4.053253266247036, 'position_y': 10446.826432598386, 'uncertainty_y': 3.
↪278383909327508, 'intensity': 19965508000.0, 'local_background': 284.22076,
↪'frame': 0, 'region_measure_bb': 289612320.0, 'localization_density_bb': 0.
↪005202016267816231, 'subregion_measure_bb': 68072.0}
```

```
lc.render_2d(dat, bin_size=100, rescale=lc.Trafo.EQUALIZE_0P3);
```



```
lc.render_2d(dat, bin_size=10, rescale=lc.Trafo.EQUALIZE_OP3,  
             bin_range=((0, 4000), (6000, 10_000)));
```



## 2.26 Tutorial about logging

Logging as supplied by the python standard library can be used.

We make use of the standard logging levels DEBUG, INFO, WARNING, ERROR, CRITICAL.

```
import logging
import sys
from pathlib import Path

import numpy as np
import pandas as pd

import locan as lc
```

```
/tmp/ipykernel_1549/2523231786.py:6: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release,
↳ of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and
↳ better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/
↳ 54466
```

(continues on next page)

(continued from previous page)

```
import pandas as pd
```

```
lc.show_versions(dependencies=False, verbose=False)
```

```
Locan:
  version: 0.20.0.dev41+g755b969

Python:
  version: 3.11.6

System:
python-bits: 64
  system: Linux
  release: 5.19.0-1028-aws
  version: #29~22.04.1-Ubuntu SMP Tue Jun 20 19:12:11 UTC 2023
  machine: x86_64
processor: x86_64
byteorder: little
  LC_ALL: None
  LANG: C.UTF-8
  LOCALE: {'language-code': 'en_US', 'encoding': 'UTF-8'}
```

### 2.26.1 Activate logging

In any script or notebook logging has to be enabled e.g. for streaming to stdout.

For changing the configuration logging has to be reloaded or the kernel be restarted.

```
logging.basicConfig(stream=sys.stdout, level=logging.INFO, format='
↳%(asctime)s - %(name)s - %(levelname)s - %(message)s')
```

A top-level logger has to be instantiated to process any log messages from the library.

```
logger = logging.getLogger()
logger
```

```
<RootLogger root (INFO)>
```

Further log messages can be added:

```
logger.info("test")
```

```
2024-03-14 11:09:05,926 - root - INFO - test
```

## Handling locan.logger

To change the filter level of locan log records, use an instance of the locan logger identified by its module name.

```
locan_logger = logging.getLogger('locan')
locan_logger.setLevel(logging.INFO)
locan_logger
```

```
<Logger locan (INFO)>
```

### 2.26.2 Logging in locan

Many functions provide warnings if some unusual behavior occurs:

```
locdata = lc.LocData.from_coordinates([(0, 0), (1, 2), (2, 1), (5, 5)])
locdata.region = lc.Rectangle((0, 0), 2, 2, 0)
```

```
Jupyter environment detected. Enabling Open3D WebVisualizer.
[Open3D INFO] WebRTC GUI backend enabled.
[Open3D INFO] WebRTCWindowSystem: HTTP handshake server disabled.
2024-03-14 11:09:08,042 - locan.data.locdata - WARNING - Not all coordinates
↪are within region.
```

Changing the level of the locan logger to logging.WARNING or higher, will switch off most locan log records.

```
locan_logger.setLevel(logging.ERROR)

locdata = lc.LocData.from_coordinates([(0, 0), (1, 2), (2, 1), (5, 5)])
locdata.region = lc.Rectangle((0, 0), 2, 2, 0)
```

```
locan_logger.setLevel(logging.INFO)

locdata = lc.LocData.from_coordinates([(0, 0), (1, 2), (2, 1), (5, 5)])
locdata.region = lc.Rectangle((0, 0), 2, 2, 0)
```

```
2024-03-14 11:09:08,061 - locan.data.locdata - WARNING - Not all coordinates
↪are within region.
```

Levels can be set for selected loggers by specifying the module or function name:

```
logger_locan_data = logging.getLogger('locan.data')
logger_locan_data.setLevel(logging.ERROR)

locdata = lc.LocData.from_coordinates([(0, 0), (1, 2), (2, 1), (5, 5)])
locdata.region = lc.Rectangle((0, 0), 2, 2, 0)
```



```
logger_locan_data.setLevel(logging.INFO)

locdata = lc.LocData.from_coordinates([(0, 0), (1, 2), (2, 1), (5, 5)])
locdata.region = lc.Rectangle((0, 0), 2, 2, 0)
```

```
2024-03-14 11:09:08,078 - locan.data.locdata - WARNING - Not all coordinates
↳are within region.
```

## Logging in a pipeline

```
def computation(self, file):
    logger.info(f'computation started for file: {file}')
    return self
```

```
pipes = [lc.Pipeline(computation=computation, file=file).compute() for file
↳in range(3)]
```

```
2024-03-14 11:09:08,745 - root - INFO - computation started for file: 0
```

```
2024-03-14 11:09:08,746 - root - INFO - computation started for file: 1
```

```
2024-03-14 11:09:08,747 - root - INFO - computation started for file: 2
```

Another example how to use logging in analysis pipelines is given by the `computation_test` function.

```
pipes = [lc.Pipeline(computation=lc.analysis.pipeline.computation_test,
↳locdata=file).compute() for file in range(3)]
```

```
2024-03-14 11:09:08,754 - locan.analysis.pipeline - INFO - computation
↳finished for locdata: 0
```

```
2024-03-14 11:09:08,755 - locan.analysis.pipeline - WARNING - An exception
↳occurred for locdata: 0
```

```
2024-03-14 11:09:08,756 - locan.analysis.pipeline - INFO - computation
↳finished for locdata: 1
```

```
2024-03-14 11:09:08,757 - locan.analysis.pipeline - WARNING - An exception
↳occurred for locdata: 1
```

```
2024-03-14 11:09:08,758 - locan.analysis.pipeline - INFO - computation
↳finished for locdata: 2
```

```
2024-03-14 11:09:08,759 - locan.analysis.pipeline - WARNING - An exception
↳occurred for locdata: 2
```

```
print(pipes[0].computation_as_string())
```

```
def computation_test(
    self: T_Pipeline,
    locdata: LocData | None = None,
    parameter: str = "test",
) -> T_Pipeline:
    """A pipeline definition for testing."""
    self.locdata = locdata # type: ignore
    something = "changed_value"
    logger.debug(f"something has a : {something}")
    self.test = parameter # type: ignore
    logger.info(f"computation finished for locdata: {locdata}")

    try:
        raise NotImplementedError
    except NotImplementedError:
        logger.warning(f"An exception occurred for locdata: {locdata}")

    return self
```

## Logging in multiprocessing with ray

To enable logging in multiprocessing using ray you need to include a default configuration in the computation function: `logging.basicConfig(level=logging.INFO)`.

```
if False:
    def computation(self, file):
        logging.basicConfig(level=logging.INFO)
        logger.info(f'computation started for file: {file}')
        return self
```

```
if False:
    import ray

    ray.init()
    # ray.init(num_cpus = 4)
```

```
%%time
if False:
    @ray.remote
    def worker(file):
        pipe = lc.Pipeline(computation=computation, file=file).compute()
        return pipe

    futures = [worker.remote(file) for file in range(3)]
    pipes = ray.get(futures)
    len(pipes)
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.72 µs
```

### 2.26.3 Logging in locan - third party libraries

Some third-party libraries provide their own logging system. Typically the individual loggers can be imported and modified.

```
import trackpy as tr
```

```
tr.logger
```

```
<Logger trackpy (INFO)>
```

alternatively

```
trackpy_logger = logging.getLogger('trackpy')
trackpy_logger
```

```
<Logger trackpy (INFO)>
```

Depending on the library various methods can be used to change the logging level. All of the following can be used.

```
trackpy_logger.setLevel(logging.WARN)
```

```
tr.logger.setLevel(logging.WARN)
```

```
tr.quiet()
```

```
tr.ignore_logging() # this switches off the trackpy logging system and forwards all logs up the
logger hierarchy.
```

```
dat = lc.simulate_tracks(n_walks=1, n_steps=100, ranges=((0,1000),(0,1000)),
                        diffusion_constant=1, seed=1)
```

```
dat.print_meta()
```

```
identifier: "6"
source: SIMULATION
state: RAW
history {
  name: "simulate_tracks"
  parameter: "{\n'n_walks\\': 1, \\n_steps\\': 100, \\n_ranges\\': ((0, 1000), (0,
↪1000)), \\n_diffusion_constant\\': 1, \\n_time_step\\': 10, \\n_seed\\': 1}"
}
element_count: 100
frame_count: 100
creation_time {
  2024-03-14T11:09:08.812727Z
}
```

```
locdata_new, track_series = lc.track(dat, search_range=5)
```

```
Frame 99: 1 trajectories present.
```

```
2024-03-14 11:09:10,848 - locan.data.locdata_utils - WARNING - Zero_
↳uncertainties occurred resulting in nan for weighted_mean and weighted_
↳variance.
```

```
2024-03-14 11:09:10,850 - locan.data.locdata_utils - WARNING - Zero_
↳uncertainties occurred resulting in nan for weighted_mean and weighted_
↳variance.
```

```
trackpy_logger.setLevel(logging.WARN)
locdata_new, track_series = lc.track(dat, search_range=5)
```

```
2024-03-14 11:09:12,731 - locan.data.locdata_utils - WARNING - Zero_
↳uncertainties occurred resulting in nan for weighted_mean and weighted_
↳variance.
```

```
2024-03-14 11:09:12,732 - locan.data.locdata_utils - WARNING - Zero_
↳uncertainties occurred resulting in nan for weighted_mean and weighted_
↳variance.
```



## 2.27 Notebook execution table

| Document  | Modified         | Method | Run Time (s) | Status               |
|---|------------------|--------|--------------|----------------------|
| <a href="#">tutorials/notebooks/Analysis_Blinking</a>         | 2024-03-14 11:04 | force  | 7.37         |                      |
| <a href="#">tutorials/notebooks/Analysis_Drift</a>            | 2024-03-14 11:05 | force  | 15.12        |                      |
| <a href="#">tutorials/notebooks/Analysis_Ripley</a>           | 2024-03-14 11:05 | force  | 14.32        |                      |
| <a href="#">tutorials/notebooks/Analysis_coordinate_base</a>  | 2024-03-14 11:06 | force  | 41.61        |                      |
| <a href="#">tutorials/notebooks/Analysis_grouped_cluster</a>  | 2024-03-14 11:07 | force  | 61.96        |                      |
| <a href="#">tutorials/notebooks/Analysis_how_to_use_an</a>    | 2024-03-14 11:07 | force  | 11.13        |                      |
| <a href="#">tutorials/notebooks/Analysis_localization_pre</a> | 2024-03-14 11:07 | force  | 14.09        |                      |
| <a href="#">tutorials/notebooks/Analysis_localization_pro</a> | 2024-03-14 11:07 | force  | 7.86         |                      |
| <a href="#">tutorials/notebooks/Analysis_localizations_pe</a> | 2024-03-14 11:07 | force  | 5.64         |                      |
| <a href="#">tutorials/notebooks/Analysis_nearest_neighbo</a>  | 2024-03-14 11:07 | force  | 5.14         |                      |
| <a href="#">tutorials/notebooks/Analysis_pipeline</a>         | 2024-03-14 11:08 | force  | 11.26        |                      |
| <a href="#">tutorials/notebooks/Cluster_LocData</a>           | 2024-03-14 11:08 | force  | 8.39         |                      |
| <a href="#">tutorials/notebooks/Example Datasets</a>          | 2024-03-14 11:08 | force  | 26.22        |                      |
| <a href="#">tutorials/notebooks/File_management</a>           | 2024-03-14 11:08 | force  | 4.29         |                      |
| <a href="#">tutorials/notebooks/Filter_LocData</a>            | 2024-03-14 11:08 | force  | 5.67         |                      |
| <a href="#">tutorials/notebooks/Load_LocData</a>              | 2024-03-14 11:08 | force  | 4.09         |                      |
| <a href="#">tutorials/notebooks/LocData</a>                   | 2024-03-14 11:08 | force  | 5.12         |                      |
| <a href="#">tutorials/notebooks/LocData_hulls</a>             | 2024-03-14 11:09 | force  | 7.66         |                      |
| <a href="#">tutorials/notebooks/Logging</a>                   | 2024-03-14 11:09 | force  | 8.77         |                      |
| <a href="#">tutorials/notebooks/Metadata</a>                  | 2024-03-14 11:09 | force  | 4.07         |                      |
| <a href="#">tutorials/notebooks/Multiprocessing_with_ray</a>  | 2024-03-14 11:09 | force  | 20.56        |                      |
| <a href="#">tutorials/notebooks/Regions</a>                   | 2024-03-14 11:09 | force  | 5.85         |                      |
| <a href="#">tutorials/notebooks/Render_LocData</a>            | 2024-03-14 11:09 | force  | 9.02         |                      |
| <a href="#">tutorials/notebooks/Simulate_LocData</a>          | 2024-03-14 11:09 | force  | 6.94         |                      |
| <b>2.27. Notebook execution table</b>                         | 11:09            |        |              | <b>245</b>           |
| <a href="#">tutorials/notebooks/Track_LocData</a>             | 2024-03-14 11:10 | force  | 7.52         | (CellExecutionError) |
| <a href="#">tutorials/notebooks/Transform_LocData</a>         | 2024-03-14       | force  | 6.69         |                      |

You can use the provided Dockerfiles to set up a docker image and run the project within a container.

### 3.1 Prepare for using Docker

- 1) Install docker (Docker Desktop)
- 2) Switch to Linux containers
- 3) Make shared drives on your host system accessible to mount local directories:  
Docker → Settings → Shared Drives

### 3.2 Dockerfiles

We provide dockerfiles for testing and deployment.

- 1) Run tests in a Python 3 environment based on conda or a PyPI environment on Debian Linux.
- 2) Run the project in a Python 3 conda environment with Jupyter lab for interactive work (Debian Linux).

### 3.3 Build a docker image

Download the source code in a project directory. Make sure the .dockerignore file is present.

Enter the project directory and run the following command to build the docker image from one of the Dockerfiles:

```
docker build -t <ImageName> -f <docker/choose directory/Dockerfile> .
```

## 3.4 Start a container from the image

### 3.4.1 Run project tests:

Run a container to just run the project tests and close afterwards:

```
docker run --rm <ImageName>
```

### 3.4.2 Run project in an interactive environment:

Open a bash shell for interactive work within a container:

```
docker run -it <ImageName> bash
```

Open the shell with a host directory mounted as volume:

```
docker run -it -v <host directory>:/home/asterix/shared <ImageName> bash
```

To make sure a gui output (e.g. from napari) is directed to an X server (that must be installed on your host) add the DISPLAY environment variable:

```
docker run -it -v <host directory>:/home/asterix/shared -e DISPLAY=<IP-  
↪address>:0.0 <ImageName> bash
```

### 3.4.3 Use Jupyter notebooks:

Start a container providing browser access to jupyter lab with a host directory mounted as volume:

```
docker run -p 8888:8888 -e JUPYTER_ENABLE_LAB=yes -v <host directory>:/home/  
↪asterix/shared <ImageName>
```

For gui (e.g. napari) interaction add the DISPLAY environment variable.

## 3.5 Clean up

After closing a jupyter lab you might have to delete the container before running the image again (e.g. if the option `-rm` was omitted):

```
docker rm -f <container>
```

or close all containers:

```
docker rm -f $(docker ps -q)
```

To clean up your system:

```
docker system prune
```



## PACKAGE DESIGN

### 4.1 Data structures

Locan provides `LocData` as the one data class to deal with the following data structures:

- 1) localization data
- 2) selections of localization data
- 3) collections of selections

#### 4.1.1 Comments on `LocData`:

##### 1) **Properties:**

Localization data consists of a list of individual localizations with spatial coordinates and several other properties (detailed list in *Properties*) like

- coordinates
- intensity
- frame
- ...

There is no fundamental difference between 2D and 3D localization data.

Multiple sets of localization data like two-color localizations will be represented by independent `LocData` objects.

The unit for length throughout the project is given by the input unit for localization coordinates, unless specified in the metadata. In most cases the unit will be nanometre.

##### 2) **Dataset:**

A `LocData` object is typically instantiated from an experimental or simulated localization dataset. The list of localizations is represented in a `pandas.DataFrame` under the attribute `LocData.data`. In addition `LocData` has some single-number-properties describing the dataset as a whole. Such properties could be (detailed list in *Properties*):

- total number of localizations
- centroid
- total emission strength
- area/volume (for all possible hulls)

- ...

### 3) **Selection:**

A `LocData` object can also be instantiated from a selection of localizations. Selections carry a reference to another `LocData` object and a list of selected indices to not copy the original dataset with all localization properties for each selection. This way a large set of selections can be created with minimal memory requirements.

### 4) **Collection:**

A `LocData` object can also represent a list of selections as e.g. generated by clustering the localization data. The `LocData.data` of a collection consists of the list of all properties from the collected `LocData` (e.g. selections).

### 5) **Hulls:**

For the spatial coordinates in each `LocData` object there are various hulls that might be of interest:

1. minimal bounding box
2. convex hull
3. alpha complex or alpha shape
4. minimal oriented bounding box

These hulls are provided from stand-alone methods or through `LocData` attributes. Hull objects expose attributes such as `LocData.region_measure` (among others) that are also added as `LocData` properties.

### 6) **Additional properties:**

Other `LocData` properties might be provided by independent methods.

## 4.2 Metadata

Each instance of the `LocData` class carries metadata that give details about the data history and allows the addition of further comments.

Each instance of an analysis class carries metadata that includes input parameters for reprocessing and allows the addition of further comments.

Metadata is saved together with data or analysis results in order to provide serialized human- and machine-readable information and serve information exchange between methods.

All metadata is structured as (nested) key:value pairs and implemented through Google's protobuf format.

### 4.2.1 Structure of metadata for `LocData`

A standard set of metadata elements is implemented into the metadata classes that are accessible by the `meta` attribute. The current structure for metadata of the `LocData` class is defined in `locan/data/metadata.proto` and can be accessed through `locan/data/metadata_pb2`. It includes the following keys:

- **identifier: str**  
Some short name.

- **comment: str**  
A user comment.
- **production\_date: str (ISO standard)**  
Date and time when the data was initially generated by experiment or simulation.
- **creation\_date: str (ISO standard)**  
Date and time when the current dataset was created from original data.
- **modification\_date: str (ISO standard)**  
Date and time when the data was last modified.
- **source: enum**  
Describes where the data is from. Must be one of the following:  
[unknown\_source, design, experiment, simulation, import (from some other program)]
- **state: enum**  
Indicator if data is original (as recorded) or has been modified.  
[unknown\_state, raw, modified]
- **history: Operation**  
Operation that was applied to the original data containing a function name and parameter for the applied method.
- **ancestor\_identifiers: str**  
Identifier of locdata from which this locdata object is derived.
- **units: Unit**  
Units for localization properties containing *property* and *unit* keys.
- **element\_count: int**  
Number of elements in locdata.
- **frame\_count: int**  
Number of frames in locdata.
- **file\_type: enum**  
Name of the file or program that produced the data (a fitter or simulation program).  
[unknown\_file\_Type, custom, rapidstorm, elyra, thunderstorm, asdf, nanoimager]
- **file\_path: str**  
Path and name of data file.
- **experimental\_setup: map(str, str)**  
Information about the setup on which the data was generated in form of variable key-value pairs.
- **experimental\_sample: map(str, str)**  
Information about the sample that was imaged in form of variable key-value pairs.
- **map: map(str, str)**  
Miscellaneous information in form of variable key-value pairs.

### 4.2.2 Structure of metadata for analysis classes

The current structure for metadata of Analysis classes is defined in *locan/analysis/metadata\_analysis.proto* and can be accessed through *locan/analysis/metadata\_analysis\_pb2*. It includes the following keys:

- **identifier: str**  
Some short name.
- **comment: str**  
A user comment.
- **creation\_date: str (ISO standard)**  
Date and time when the results were created.
- **modification\_date: str (ISO standard)**  
Date and time when the results were last recreated.
- **method: Analysis\_routine**  
A dictionary with ‘name’ and ‘parameter’ describing the applied analysis procedure. This data can be used to programmatically rerun the procedure.
- **map: map(str, str)**  
Miscellaneous information in form of variable key-value pairs.

## 4.3 Properties

LocData, the data class for localization data, carries certain properties that describe individual (localization properties) or averaged features of the underlying localizations or groups thereof (LocData properties). In the following we provide names (or keys) for those properties.

The datatype for all keys is *string*. We stick to the following conventions:

- be explicit
- start with lower case letters
- use underscore
- do not use CamelCase or blanks
- use reverse notation in the sense that coordinate identifiers or identifiers of statistical functions are added in the end (*position\_x\_mean\_mean*)

A list of well defined property keys used throughout locan is given by the constant: *locan/constants/PropertyKey*. An up-to-date description can be inspected by *locan/constants/PropertyKey.index.value.description*.

### 4.3.1 Localization properties:

Each localization has properties that can usually be identified in the various input (file) formats. We will use the following keys (where *c* stands for the coordinate *x*, *y* or *z*):

- ***index***  
localization index
- ***position\_c***  
coordinate for the *c*-position
- ***frame***  
frame number in which the localization occurs
- ***intensity***  
intensity or emission strength as estimated by the fitter
- ***local\_background***  
background in the neighborhood of localization as estimated by the fitter
- ***local\_background\_sigma***  
variation of local background in terms of standard deviation
- ***signal\_noise\_ratio***  
ratio between mean intensity (i.e. intensity for a single localization) and the standard deviation of *local\_background* (i.e. *local\_background\_sigma* for a single localization)
- ***signal\_background\_ratio***  
ratio between mean intensity (i.e. *intensity* for a single localization) and the *local\_background*
- ***chi\_square***  
chi-square value of the fitting procedure as estimated by the fitter
- ***psf\_sigma\_c***  
sigma of the fitted Gauss-function in *c*-dimension as estimated by the fitter
- ***psf\_sigma***  
sigma of the fitted Gauss-function - being isotropic or representing the root-mean-square of *psf\_sigma\_c* for all dimensions
- ***psf\_width\_c***  
full-width-half-max of the fitted Gauss-function in *c*-dimension as estimated by the fitter
- ***psf\_width***  
full-width-half-max of the fitted Gauss-function - being isotropic or representing the root-mean-square of *psf\_width\_c* for all dimensions
- ***uncertainty\_c***  
localization error in *c*-dimension estimated by a fitter or representing a value proportional to *psf\_sigma\_c* / sqrt(intensity)
- ***uncertainty***  
localization error for all dimensions or representing a value proportional to *psf\_sigma* / sqrt(intensity) or representing the root-mean-square of *uncertainty\_c* for all dimensions.
- ***channel***  
identifier for various channels
- ***two\_kernel\_improvement***  
a rapidSTORM parameter describing the improvement from two kernel fitting

- *frames\_number*  
number of frames that contribute to a merged localization
- *frames\_missing*  
number of frames that occurred between two successive localizations

Additional localization properties are computed by various analysis procedures including (among others):

- *cluster\_label*  
identifier for a localization cluster
- *nn\_distance*  
nearest-neighbor distance
- *nn\_distance\_k*  
k-nearest-neighbor distance (k can be any integer)
- *colocalization\_cbc*  
coordinate-based colocalization value

### 4.3.2 LocData properties:

A group of localizations makes up a LocData entity for which further properties are defined.

The coordinates for a localization group are defined by their centroids.

In general, we will use the following keys for statistics of localization properties:

- *property\_stats*  
where property is the localization property key and stats is one of the following:
  - count (number of elements)
  - min (minimum of all elements)
  - max (maximum of all elements)
  - sum (sum of all elements)
  - mean (the mean of all elements)
  - std (standard deviation of all elements)
  - sem (standard error of the mean of all elements)

For example:

- *intensity\_sum*  
total intensity of all localizations in the group

Some properties are derived from a hull of all element positions. We provide four hulls:

1. bounding box
2. convex hull
3. alpha shape
4. oriented bounding box

From each hull a region measure (e.g. the area in 2D) and a subregion measure (e.g. the circumference in 2D) is computed.

We will use the following keys for additional properties (where *c* stands for the coordinate *x*, *y* or *z* and *h* stands for the corresponding hull *bb*, *ch*, *as*, *obb*):

- **centroid**  
tuple with mean of all localization coordinates
- **localization\_count**  
number of localizations within a group
- **region\_measure\_h**  
area/volume (for all possible hulls)
- **subregion\_measure\_h**  
circumference/surface (for all possible hulls)
- **localization\_density\_h**  
density of localizations (for all possible hulls)
- **boundary\_localizations\_h**  
absolute number of localizations on boundary (for all possible hulls)
- **boundary\_localizations\_ratio\_h**  
ratio between number of localizations on hull boundary and within hull (for all possible hulls)
- **max\_distance**  
maximum distance between any two localizations
- **inertia\_moments**  
inertia moments of all localizations
- **orientation\_obb**  
angle between the x-axis and the long axis of the oriented bounding box
- **orientation\_im**  
angle between inertia moment principal component vectors
- **circularity\_obb**  
elongation estimated from oriented bounding box
- **circularity\_im**  
excentricity estimated from inertia moments

## 4.4 Methods for data analysis

Locan will provide methods to work on `LocData` objects and carry out standard analysis procedures. Some of these functions are merely wrapper functions for well established analysis routines in third-party packages.

The methods in this context can either be stand-alone functions with a well-defined input and output and absolutely no side-effects.

In general however, results consist of numeric data, statistical properties (e.g. a histogram of numeric results) and/or fit parameter from comparing the numeric results with theoretical expectations, and annotated plots. In this case the method will be part of a more complex analysis class.

Locan therefore provides specific analysis classes that follow a common structure. Any `Analysis` object is instantiated with a set of parameters that define the precise analysis procedure. The computation is then performed by calling `Analysis.compute()` with a parameter specifying one or more `LocData` objects

on which to perform the analysis. The main result is typically provided under the attribute `Analysis.results` and accompanied by a flexible set of further attributes. Common methods often include `plot`, `hist`, and `report` functions. Metadata is provided under the `Analysis.meta` attribute.

## 4.5 Aim

We aim at providing a python package with data structures and methods for analyzing single-molecule localization data. Locan is a Python package providing functionality to load, manipulate and analyze localization data.

The package provides:

- a class structure for localization data with appropriate meta data
- rendering functions to visualize localization data
- methods for carrying out established analysis routines
- an interface to save analysis results and run batch processes.

Locan provides standard routines and interfaces for setting up reproducible analysis pipelines and batch processes. At the same time it allows flexibility on all programmatic levels either in Python scripts, Jupyter notebooks or dashboard apps. We provide original algorithms and include existing solutions by providing appropriate wrapper functions.

## 4.6 Outline

Locan provides a standardized class structure to hold and deal with *data structures* such as localization data and analysis results.

*Metadata* will be part of each data class and is either added by user input or generated during manipulation of data classes.

`LocData`, the data class for localization data, carries certain properties that describe individual or averaged features of the underlying localizations or groups thereof. We suggest a canonical set of *Properties*.

*Methods* will either create or manipulate these data structures or perform some analysis routine and provide the results in an appropriate form.



## COMMAND-LINE INTERFACE

Locan is designed as a Python library. Full functionality of the package is accessible through Python scripts or Jupyter notebooks.

A command-line interface is provided for selected routines. It is accessible from any terminal by the base command `locan` and provides a number of compound commands. For a complete description of available commands see documentation of the [\*locan.scripts\*](#) module or look up command-line help using `locan -h`.

## COLORMAPS

For visual inspection and presentation SMLM data is typically rendered in 2D or 3D images. Various binning algorithms are used to create a representation of localisation densities. Intensity values are represented by color according to a selected colormap. Various colormaps can be chosen that accentuate certain data structures.

For SMLM images we aim at using colormaps that SMLM users are used to but that perform well with respect to human perception. Therefore, we recommend using colormaps that are optimized for accurate perception as for instance provided by the [colorcet](#) library.

We recommend default use for various types of colormaps: All 2D one-channel plot functions use the [fire](#) colormap as default if *colorcet* is installed. Otherwise we use the matplotlib colormap [viridis](#) as default.

For categorical data we use [glasbey\\_dark](#) from *colorcet* as default colormap or alternatively *tab20* from *matplotlib*. For diverging data we use [coolwarm](#) from *colorcet* as default colormap or alternatively *coolwarm* from *matplotlib*. For rainbow/jet-like data we use [turbo](#) as default colormap.

For details on how to use colormaps see API documentation for [locan.visualize.colormap](#).

## USER INTERFACE

Locan is mostly designed to work without a graphical user interface (gui). However, a few methods make use of a gui or interact with third-party libraries that provide a gui.

Locan and various other libraries make use of *qt* as a gui backend if it is installed together with appropriate python bindings. Different python bindings exist to interact with *qt* including *pyside2* and *pyqt5*.

Locan makes use of *pyqt* to choose a binding. To force locan and other libraries to use a specific binding you can set the python environment variable *QT\_API* before importing any QT-dependent package:

```
import os
os.environ["QT_API"] = "pyside2"
```

For more details see also the documentation on the *locan.gui* module.

## API REFERENCE

Locan consists of the following modules:

|                      |   |
|----------------------|---|
| <i>analysis</i>      | Standard analysis procedures.                           |
| <i>constants</i>     | Constants used throughout the project.                  |
| <i>configuration</i> | Configuration variables used throughout the project.    |
| <i>data</i>          | Localization data.                                      |
| <i>datasets</i>      | Utility functions to deal with exemplary datasets.      |
| <i>dependencies</i>  | Module to deal with required and optional dependencies. |
| <i>gui</i>           | User interfaces.  |
| <i>locan_io</i>      | File input/output functions.                            |
| <i>rois</i>          | Region of interest.                                     |
| <i>scripts</i>       | Command-line utility scripts                            |
| <i>simulation</i>    | Synthetic data  |
| <i>tests</i>         | Test the locan package.                                 |
| <i>utils</i>         | Utility functions.                                      |
| <i>visualize</i>     | Visualize localization data.                            |

### 8.1 locan.analysis

Standard analysis procedures.

This module contains classes and functions for carrying out standardized analysis procedures on localization data. All functions typically take `LocData` objects as input and provide an analysis class with derived analysis results and standard functions for presentation.

### 8.1.1 Submodules:

|   |  |
|---|--|
| <i>accumulation_analysis</i>              | Localization-density variation analysis to characterize localization cluster.  |
| <i>blinking</i>                           | Compute on- and off-periods from localization frames.                          |
| <i>cbc</i>                                | Coordinate-based colocalization.   |
| <i>convex_hull_expectation</i>            | Analyze geometrical properties of the convex hull of localization coordinates. |
| <i>drift</i>                              | Drift analysis for localization coordinates.                                   |
| <i>grouped_property_expectation</i>       | Analyze grouped property expectations.   |
| <i>localization_precision</i>             | Compute localization precision from successive nearby localizations.           |
| <i>localization_property</i>              | Analyze localization property.   |
| <i>localization_property_2d</i>           | Analyze the distribution of a localization property in 2D.                     |
| <i>localization_property_correlations</i> | Analyze cross dependencies between localization properties.                    |
| <i>localizations_per_frame</i>            | Compute localizations per frame.   |
| <i>nearest_neighbor</i>                   | Nearest-neighbor distance distribution analysis                                |
| <i>pipeline</i>                           | Building an analysis pipeline.   |
| <i>position_variance_expectation</i>      | Analyze variance of localization coordinates.                                  |
| <i>ripley</i>                             | Compute Ripley's k function.   |
| <i>subpixel_bias</i>                      | Compute subpixel bias in localization data.                                    |
| <i>uncertainty</i>                        | Compute localization uncertainty.  |

### locan.analysis.accumulation\_analysis

Localization-density variation analysis to characterize localization cluster.

The existence of clusters is tested by analyzing variations in cluster area and localization density within clusters. The analysis routine follows the ideas in<sup>1</sup> and<sup>2</sup>.

---

**Note:** The analysis procedure is in an exploratory state and has not been fully developed and tested.

---



---

<sup>1</sup> Baumgart F., Arnold AM., Leskovaar K., Staszek K., Fölser M., Weghuber J., Stockinger H., Schütz GJ., Varying label density allows artifact-free analysis of membrane-protein nanoclusters. Nat Methods. 2016 Aug;13(8):661-4. doi: 10.1038/nmeth.3897

<sup>2</sup> Spahn C., Herrmannsdörfer F., Kuner T., Heilemann M. Temporal accumulation analysis provides simplified artifact-free analysis of membrane-protein nanoclusters. Nat Methods. 2016 Nov 29;13(12):963-964. doi: 10.1038/nmeth.406

## References

## Classes

|   |   |
|---|---|
| <code>AccumulationClusterCheck</code> ([meta, ...]) | Check for the presence of clusters in localization data by analyzing variations in cluster area and localization density within clusters. |
|---|---|

### locan.analysis.accumulation\_analysis.AccumulationClusterCheck

```
class locan.analysis.accumulation_analysis.AccumulationClusterCheck(meta=None,
                                                                    re-
                                                                    gion_measure='bb',
                                                                    algo-
                                                                    rithm=<function
                                                                    clus-
                                                                    ter_hdbscan>,
                                                                    algo_parameter=None,
                                                                    hull='bb',
                                                                    n_loc=10, di-
                                                                    vide='random',
                                                                    n_extrapolate=5)
```

Bases: `_Analysis`

Check for the presence of clusters in localization data by analyzing variations in cluster area and localization density within clusters.

#### Parameters

- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata` | `None`) – Metadata about the current analysis routine.
- **region\_measure** (`float` | `Literal["bb", "ch"]`) –  
Region measure (area or volume) for the support of locdata. String can be any of standard hull identifier.
- **algorithm** (`Callable[...]`, `tuple[LocData, LocData]`) – Clustering algorithm.
- **algo\_parameter** (`dict`) – Dictionary with kwargs for *algorithm*.
- **hull** (`Literal["bb", "ch"]`) – Hull computation that is used to compute cluster region measures (area or volume). The identifier string can be one of the defined hulls.
- **n\_loc** (`int` | `Sequence[int]`) – If *n\_loc* is an int, it defines the number of localization subsets into which the total number of localizations are distributed. If *n\_loc* is a sequence, it defines the number of localizations used for each localization subset.
- **divide** (`Literal["random", "sequential"]`) – Identifier to choose how to partition the localization data. For *random* localizations are selected ran-

domly. For *sequential* localizations are selected as chunks of increasing size always starting from the first element.

- **n\_extrapolate** (*int*) – The number of rho values taken to extrapolate rho\_zero.

### Variables

- **count** (*int*) – A counter for counting instantiations.
- **parameter** (*dict[str, Any]*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Meta-data about the current analysis routine.
- **results** (*pandas.DataFrame*) – Data frame with localization density, relative area coverage by the clusters (eta), average density of localizations within apparent clusters (rho), and rho normalized to the extrapolated value of rho for localization\_density=0 (rho\_zero). If the extrapolation of rho yields a negative value rho\_zero is set to 1.

### Methods

|   |  |
|---|--|
| <code>__init__([meta, region_measure, algorithm, ...])</code> |  |
| <code>compute(locdata)</code>                                 | Run the computation.   |
| <code>plot([ax])</code>                                       | Provide plot of results as <code>matplotlib.axes.Axes</code> object. |
| <code>report(*args, **kwargs)</code>                          | Show a report about analysis results.                                |

### Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*)

Run the computation.

#### Parameters

**locdata** (*LocData*) – Localization data that might be clustered.

#### Return type

Self

**count:** `int = 0`

A counter for counting Analysis class instantiations (class attribute).

**plot**(*ax=None, \*\*kwargs*)

Provide plot of results as `matplotlib.axes.Axes` object.

#### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.plot()`.

**Returns**

Axes object with the plot.

**Return type**

`matplotlib.axes.Axes`

## locan.analysis.blinking

Compute on- and off-periods from localization frames.

Assuming that the provided localizations are acquired from the same label, we analyze the times of recording as provided by the *frame* property.

## Classes

|  |   |
|--|---|
| <i>BlinkStatistics</i> ([meta, memory, ...]) | Estimate on and off times from the frame values provided. |
|--|---|

### locan.analysis.blinking.BlinkStatistics

```
class locan.analysis.blinking.BlinkStatistics(meta=None, memory=0,  
                                             remove_heading_off_periods=True)
```

Bases: `_Analysis`

Estimate on and off times from the frame values provided.

On and off-periods and the first frame of each period are determined from the sorted frame values. A series of frame values that constantly increase by one is considered an on-period. Each series of missing frame values between two given frame values is considered an off-period.

A log warning is provided if a frame number occurs multiple times.

Missing localizations within an on-period can be taken into account by increasing the *memory* parameter. There is no way to correct for false positive localizations.

**Parameters**

- **memory** (*int*) – The maximum number of intermittent frames without any localization that are still considered to belong to the same on-period.
- **remove\_heading\_off\_periods** (*bool*) – Flag to indicate if off-periods at the beginning of the series are excluded.
- **meta** (*metadata\_analysis\_pb2.AMetadata* / *None*) – Metadata about the current analysis routine.

**Variables**

- **count** (*int*) – A counter for counting instantiations.



- **parameter** (*dict[str, Any]*) – A dictionary with all settings for the current computation.
- **meta** (*metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **results** (*dict[str, npt.NDArray[np.int64 | np.float64] | list[int | float]] | None*) – ‘on\_periods’ and ‘off\_periods’ in units of frame numbers. ‘on\_periods\_frame’ and ‘off\_periods\_frame’ with the first frame in each on/off-period. ‘on\_periods\_indices’ are groups of indices to the input frames or more precise `np.unique(frames)`
- **distribution\_statistics** (*dict[str, Any]*) – Distribution parameters derived from MLE fitting of results.

## Methods

|  |  |
|--|--|
| <code>__init__([meta, memory, ...])</code>               |  |
| <code>compute(locdata)</code>                            | Run the computation.   |
| <code>fit_distributions([distribution, ...])</code>      | Fit probability density functions to the distributions of on- and off-periods in the results using MLE ( <code>scipy.stats</code> ). |
| <code>hist([data_identifier, ax, bins, log, fit])</code> | Provide histogram as <code>matplotlib.axes.Axes</code> object showing <code>hist(results)</code> .                                   |
| <code>report(*args, **kwargs)</code>                     | Show a report about analysis results.  |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

### `compute(locdata)`

Run the computation.

#### Parameters

**locdata** (Union[*LocData*, *\_SupportsArray*[*dtype*[*Any*]], *\_NestedSequence*[*\_SupportsArray*[*dtype*[*Any*]]], *bool*, *int*, *float*, *complex*, *str*, *bytes*, *\_NestedSequence*[Union[*bool*, *int*, *float*, *complex*, *str*, *bytes*]]]) – Localization data or just the frame values of given localizations.

#### Return type

Self

**count:** `int = 0`

A counter for counting Analysis class instantiations (class attribute).

**fit\_distributions** (*distribution=<scipy.stats.\_continuous\_distns.expon\_gen object>*, *data\_identifier=('on\_periods', 'off\_periods')*, *with\_constraints=True*, *\*\*kwargs*)

Fit probability density functions to the distributions of on- and off-periods in the results using MLE (`scipy.stats`).

If `with_constraints` is true we put the following constraints on the fit procedure: If distribution is `expon` then `floc=np.min(self.analysis_class.results[self.loc_property].values)`.

#### Parameters

- **distribution** (`str` | `rv_continuous`) – Distribution model to fit.
- **data\_identifier** (`str` | `Iterable[str]`) – String to identify the data in *results* for which to fit an appropriate distribution, here ‘on\_periods’ or ‘off\_periods’. For True all are fitted.
- **with\_constraints** (`bool`) – Flag to use predefined constraints on fit parameters.
- **kwargs** (`Any`) – Other parameters are passed to the `scipy.stat.distribution.fit()` function.

#### Return type

None

**hist**(`data_identifier='on_periods'`, `ax=None`, `bins='auto'`, `log=True`, `fit=True`, `**kwargs`)

Provide histogram as `matplotlib.axes.Axes` object showing `hist(results)`.

#### Parameters

- **data\_identifier** (`str`) – ‘on\_periods’ or ‘off\_periods’.
- **ax** (`Optional[Axes]`) – The axes on which to show the image
- **bins** (`int` | `Sequence[int | float]` | `str`) – Bin specifications (passed to `matplotlib.hist()`).
- **log** (`bool`) – Flag for plotting on a log scale.
- **fit** (`Optional[bool]`) – Flag indicating if distribution fit is shown. The fit will only be computed if `distribution_statistics` is None.
- **kwargs** (`Any`) – Other parameters passed to `matplotlib.pyplot.hist()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

## locan.analysis.cbc

Coordinate-based colocalization.

Colocalization is estimated by computing a colocalization index for each localization using the so-called coordinate-based colocalization algorithm<sup>1</sup>.

---

<sup>1</sup> Malkusch S, Endesfelder U, Mondry J, Gelléri M, Verveer PJ, Heilemann M., Coordinate-based colocalization analysis of single-molecule localization microscopy data. *Histochem Cell Biol.* 2012, 137(1):1-10. doi: 10.1007/s00418-011-0880-5

## References

## Classes

|  |  |
|--|--|
| <code>CoordinateBasedColocalization</code> ([meta, ...]) | Compute a colocalization index for each localization by coordinate-based colocalization (CBC). |
|--|--|

### `locan.analysis.cbc.CoordinateBasedColocalization`

**class** `locan.analysis.cbc.CoordinateBasedColocalization`(*meta=None, radius=100, n\_steps=10*)

Bases: `_Analysis`

Compute a colocalization index for each localization by coordinate-based colocalization (CBC).

The colocalization index is calculated for each localization in *locdata* by finding nearest neighbors in *locdata* or *other\_locdata* within *radius*. A normalized number of nearest neighbors at a certain radius is computed for *n\_steps* equally-sized steps of increasing radii ranging from 0 to *radius*. The Spearman rank correlation coefficient is computed for these values and weighted by `Exp[-nearestNeighborDistance/distanceMax]`.

#### Parameters

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **radius** (*int* / *float*) – The maximum radius up to which nearest neighbors are determined
- **n\_steps** (*int*) – The number of bins from which Spearman correlation is computed.

#### Variables

- **count** (*int*) – A counter for counting instantiations.
- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **results** (*pandas.DataFrame*) – Coordinate-based colocalization coefficients for each input point.

## Methods

|  |  |
|--|--|
| <code>__init__([meta, radius, n_steps])</code> |  |
| <code>compute(locdata[, other_locdata])</code> | Run the computation.   |
| <code>hist([ax, bins, density])</code>         | Provide histogram as <code>matplotlib.axes.Axes</code> object showing <code>hist(results)</code> . |
| <code>report(*args, **kwargs)</code>           | Show a report about analysis results.  |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*, *other\_locdata=None*)

Run the computation.

### Parameters

- **locdata** (*LocData*) – Localization data for which CBC values are computed.
- **other\_locdata** (Optional[*LocData*]) – Localization data to be colocalized. If None other\_locdata is set to locdata.

### Return type

Self

**count:** `int = 0`

A counter for counting Analysis class instantiations (class attribute).

**hist**(*ax=None*, *bins=(-1, -0.3333, 0.3333, 1)*, *density=True*, *\*\*kwargs*)

Provide histogram as `matplotlib.axes.Axes` object showing `hist(results)`.

### Parameters

- **ax** (Optional[*Axes*]) – The axes on which to show the image
- **bins** (Union[int, Sequence[int | float], Literal['auto']]) – Bin specification as used in `matplotlib.hist()`.
- **density** (bool) – Flag for normalization as used in `matplotlib.hist()`. True returns probability density function; None returns counts.
- **kwargs** (Any) – Other parameters passed to `matplotlib.hist()`.

### Returns

`Axes` object with the plot.

### Return type

`matplotlib.axes.Axes`

## locan.analysis.convex\_hull\_expectation

Analyze geometrical properties of the convex hull of localization coordinates.

Localization coordinates in localization clusters come with a certain variance. In resolution-limited clusters the variance is determined by the localization precision. Accordingly, the geometrical properties of convex hulls also vary and can be analyzed as function of localization counts to help characterize localization clusters<sup>1</sup>.

## References

## Classes

|   |   |
|---|---|
| <code>ConvexHullExpectation</code> ([meta, ...])              | Analyze geometrical properties of the convex hull of localization coordinates in relation to expected values. |
| <code>ConvexHullExpectationBatch</code> ([meta, ...])         | Analyze geometrical properties of the convex hull of localization coordinates in relation to expected values. |
| <code>ConvexHullExpectationResults</code> ([values, grouped]) |   |
| <code>ConvexHullExpectationValues</code> (n_points, ...)      |   |
| <code>ConvexHullProperty</code> (value[, names, module, ...]) |   |

## locan.analysis.convex\_hull\_expectation.ConvexHullExpectation

```
class locan.analysis.convex_hull_expectation.ConvexHullExpectation(meta=None,
                                                                    con-
                                                                    vex_hull_property='region_measure',
                                                                    ex-
                                                                    pected_variance=None)
```

Bases: `_Analysis`

Analyze geometrical properties of the convex hull of localization coordinates in relation to expected values.

### Parameters

- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata` | `None`)  
– Metadata about the current analysis routine.
- **convex\_hull\_property** (`Literal["region_measure_ch", "subregion_measure_ch"]`) – One of ‘region\_measure\_ch’ (i.e. area or volume) or ‘subregion\_measure\_ch’ (i.e. circumference or surface.)

<sup>1</sup> Ebert V, Eiring P, Helmerich DA, Seifert R, Sauer M, Doose S. Convex hull as diagnostic tool in single-molecule localization microscopy. *Bioinformatics* 38(24), 2022, 5421-5429, doi: 10.1093/bioinformatics/btac700.

- **expected\_variance** (*float* | *Iterable[float]* | *None*) – The expected variance for all or each localization property. The expected variance equals the squared localization precision for localization position coordinates.

### Variables

- **count** (*int*) – A counter for counting instantiations (class attribute).
- **parameter** (*dict[str, Any]*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **results** (*ConvexHullExpectationResults* | *None*) – Computed results.
- **distribution\_statistics** (*dict[str, Any]* | *None*) – Distribution parameters derived from MLE fitting of results.

### Methods

|  |  |
|--|--|
| <code>__init__([meta, convex_hull_property, ...])</code> |  |
| <code>compute(locdata)</code>                            | Run the computation.   |
| <code>hist([ax, bins, n_bins, bin_size, ...])</code>     | Provide plot as <code>matplotlib.axes.Axes</code> object showing the 2-dimensional histogram of <code>convex_hull_property</code> and localization counts. |
| <code>plot([ax])</code>                                  | Provide plot as <code>matplotlib.axes.Axes</code> object showing the <code>convex_hull_property</code> as function of localization counts.                 |
| <code>report(*args, **kwargs)</code>                     | Show a report about analysis results.  |

### Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

#### **compute**(*locdata*)

Run the computation.

#### Parameters

**locdata** (*LocData*) – Localization data.

#### Return type

Self

**hist**(*ax=None, bins=None, n\_bins=None, bin\_size=None, bin\_edges=None, bin\_range=None, log=True, fit=False, \*\*kwargs*)

Provide plot as `matplotlib.axes.Axes` object showing the 2-dimensional histogram of `convex_hull_property` and localization counts.

### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **bins** (UnionType[Bins, Axis, AxesTuple, None]) – The bin specification as defined in Bins
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (UnionType[tuple[float, float], Sequence[float], Sequence[Sequence[float]], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2).
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **log** (bool) – Flag for plotting on a log scale.
- **fit** (bool) – Flag indicating if distribution fit is shown. The fit will only be computed if *distribution\_statistics* is None.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.pcolormesh()`.

### Returns

Axes object with the plot.

### Return type

`matplotlib.axes.Axes`

**plot**(*ax=None*, *\*\*kwargs*)

Provide plot as `matplotlib.axes.Axes` object showing the `convex_hull_property` as function of localization counts.

### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.plot()`.

### Returns

Axes object with the plot.

### Return type

`matplotlib.axes.Axes`

## locan.analysis.convex\_hull\_expectation.ConvexHullExpectationBatch

```
class locan.analysis.convex_hull_expectation.ConvexHullExpectationBatch(meta=None,  
                                                                           convex_hull_property='region_measure_ch',  
                                                                           expected_variance=None)
```

Bases: `_Analysis`

Analyze geometrical properties of the convex hull of localization coordinates in relation to expected values.

See also:

[`ConvexHullExpectation`](#)

### Parameters

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **convex\_hull\_property** (*Literal['region\_measure\_ch', 'subregion\_measure\_ch']*) – One of 'region\_measure\_ch' (i.e. area or volume) or 'subregion\_measure\_ch' (i.e. circumference or surface.)
- **expected\_variance** (*float | Iterable[float] | None*) – The expected variance for all or each localization property. The expected variance equals the squared localization precision for localization position coordinates.

### Variables

- **count** (*int*) – A counter for counting instantiations (class attribute).
- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **results** ([`ConvexHullExpectationResults`](#)) – Computed results.
- **distribution\_statistics** (*Distribution\_stats, None*) – Distribution parameters derived from MLE fitting of results.



## Methods

|  |                                       |
|--|---------------------------------------|
| <code>__init__([meta, convex_hull_property, ...])</code> |                                       |
| <code>compute(locdatas)</code>                           | Run the computation.                  |
| <code>from_batch(batch[, dimension])</code>              | <b>rtype</b><br>Self                  |
| <code>hist([ax, bins, n_bins, bin_size, ...])</code>     | <b>rtype</b><br>Axes                  |
| <code>plot([ax])</code>                                  | <b>rtype</b><br>Axes                  |
| <code>report(*args, **kwargs)</code>                     | Show a report about analysis results. |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdatas*)

Run the computation.

### Parameters

**locdatas** (*Iterable*[[LocData](#)]) – Localization data.

### Return type

Self

**from\_batch**(*batch*, *dimension=None*)

### Return type

Self

**hist**(*ax=None*, *bins=None*, *n\_bins=None*, *bin\_size=None*, *bin\_edges=None*, *bin\_range=None*, *log=True*, *fit=False*, *\*\*kwargs*)

### Return type

Axes

**plot**(*ax=None*, *\*\*kwargs*)

### Return type

Axes

## locan.analysis.convex\_hull\_expectation.ConvexHullExpectationResults

```
class locan.analysis.convex_hull_expectation.ConvexHullExpectationResults(values=<factory>,
                                                                           grouped=<factory>)
```

Bases: object

### Methods

|  |
|--|
| <code>__init__([values, grouped])</code> |
|--|

### Attributes

|                |
|----------------|
| <i>values</i>  |
| <i>grouped</i> |

**grouped:** DataFrame

**values:** DataFrame

## locan.analysis.convex\_hull\_expectation.ConvexHullExpectationValues

```
class locan.analysis.convex_hull_expectation.ConvexHullExpectationValues(n_points,
                                                                           expectation,
                                                                           std_pos,
                                                                           std_neg)
```

Bases: NamedTuple

### Methods

|  |  |
|--|--|
| <code>__init__()</code>                  |  |
| <code>count(value, /)</code>             | Return number of occurrences of value. |
| <code>index(value[, start, stop])</code> | Return first index of value.           |

## Attributes

|                          |                          |
|--------------------------|--------------------------|
| <code>expectation</code> | Alias for field number 1 |
| <code>n_points</code>    | Alias for field number 0 |
| <code>std_neg</code>     | Alias for field number 3 |
| <code>std_pos</code>     | Alias for field number 2 |

**expectation:** `ndarray[Any, dtype[int64 | float64]]`

Alias for field number 1

**n\_points:** `list[int] | ndarray[Any, dtype[int64]]`

Alias for field number 0

**std\_neg:** `ndarray[Any, dtype[float64]]`

Alias for field number 3

**std\_pos:** `ndarray[Any, dtype[float64]]`

Alias for field number 2

## locan.analysis.convex\_hull\_expectation.ConvexHullProperty

```
class locan.analysis.convex_hull_expectation.ConvexHullProperty(value,
                                                                names=None, *,
                                                                module=None,
                                                                qualname=None,
                                                                type=None,
                                                                start=1,
                                                                boundary=None)
```

Bases: Enum

## Attributes

|                                   |
|-----------------------------------|
| <code>REGION_MEASURE_2D</code>    |
| <code>SUBREGION_MEASURE_2D</code> |
| <code>REGION_MEASURE_3D</code>    |
| <code>SUBREGION_MEASURE_3D</code> |

**REGION\_MEASURE\_2D** = 1

**REGION\_MEASURE\_3D** = 3

**SUBREGION\_MEASURE\_2D** = 2

**SUBREGION\_MEASURE\_3D** = 4

## Functions

```
compute_convex_hull_region_measure_2d(n_points, sigma)
```

Compute the expected convex hull area for *n\_points* data points that are normal distributed in 2-dimensional space with standard deviation *sigma*.

### `locan.analysis.convex_hull_expectation.compute_convex_hull_region_measure_2d`

`locan.analysis.convex_hull_expectation.compute_convex_hull_region_measure_2d(n_points, sigma=1)`

Compute the expected convex hull area for *n\_points* data points that are normal distributed in 2-dimensional space with standard deviation *sigma*.

#### Parameters

- ***n\_points*** (int) – Number of points of convex hull
- ***sigma*** (float) – Standard deviation of normal-distributed point coordinates

#### Returns

Expectation value for convex hull area

#### Return type

float

### `locan.analysis.drift`

Drift analysis for localization coordinates.

This module provides functions for estimating spatial drift in localization data.

Software based drift correction using image correlation has been described in several publications . Methods employed for drift estimation comprise single molecule localization analysis (an iterative closest point (icp) algorithm as implemented in the open3d library<sup>1, 2</sup>) or image cross-correlation analysis<sup>3, 4, 5</sup>.

<sup>1</sup> Qian-Yi Zhou, Jaesik Park, Vladlen Koltun, Open3D: A Modern Library for 3D Data Processing, arXiv 2018, 1801.09847

<sup>2</sup> Rusinkiewicz and M. Levoy, Efficient variants of the ICP algorithm, In 3-D Digital Imaging and Modeling, 2001.

<sup>3</sup> C. Geisler, Drift estimation for single marker switching based imaging schemes, Optics Express. 2012, 20(7):7274-89.

<sup>4</sup> Yina Wang et al., Localization events-based sample drift correction for localization microscopy with redundant cross-correlation algorithm, Optics Express 2014, 22(13):15982-91.

<sup>5</sup> Michael J. Mlodzianoski et al., Sample drift correction in 3D fluorescence photoactivation localization microscopy, Opt Express. 2011 Aug 1;19(16):15009-19.

## Examples

Please use the following procedure to estimate and correct for spatial drift:

```
from lmfit import LinearModel
drift = Drift(chunk_size=1000, target='first').\
    compute(locdata).\
    fit_transformations(slice_data=slice(0, -1),\
                        matrix_models=None,\
                        offset_models=(LinearModel(), LinearModel())).\
    apply_correction()
locdata_corrected = drift.locdata_corrected
```

## References

## Classes

|   |   |
|---|---|
| <code>Drift([meta, chunks, chunk_size, n_chunks, ...])</code> | Estimate drift from localization coordinates by registering points in successive time-chunks of localization data using an iterative closest point algorithm (icp) or image cross-correlation algorithm (cc). |
| <code>DriftComponent([type])</code>                           | Class carrying model functions to describe drift over time (in unit of frames).   |
| <code>DriftModel(*args, **kwargs)</code>                      |   |

## locan.analysis.drift.Drift

```
class locan.analysis.drift.Drift(meta=None, chunks=None, chunk_size=None,
                                n_chunks=None, target='first', method='icp',
                                kwargs_chunk=None, kwargs_register=None)
```

Bases: `_Analysis`

Estimate drift from localization coordinates by registering points in successive time-chunks of localization data using an iterative closest point algorithm (icp) or image cross-correlation algorithm (cc).

### Parameters

- **locdata** (`LocData`) – Localization data representing the source on which to perform the manipulation.
- **chunks** (`Sequence[tuple[int, ...]] | None`) – Localization chunks as defined by a list of index-tuples. One of `chunks`, `chunk_size` or `n_chunks` must be different from `None`.
- **chunk\_size** (`int | None`) – Number of consecutive localizations to form a single chunk of data. One of `chunks`, `chunk_size` or `n_chunks` must be different from `None`.

- **n\_chunks** (*int* / *None*) – Number of chunks. One of *chunks*, *chunk\_size* or *n\_chunks* must be different from *None*.
- **target** (*Literal["first", "previous"]*) – The chunk on which all other chunks are aligned. One of ‘first’, ‘previous’.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Meta-data about the current analysis routine.
- **method** (*Literal["cc", "icp"]*) – The method used for computation. One of iterative closest point algorithm ‘icp’ or image cross-correlation algorithm ‘cc’.
- **bin\_size** (*tuple*) – Only for method=‘cc’: Size per image pixel
- **kwargs\_chunk** (*dict[str, Any]* / *None*) – Other parameter passed to *LocData.from\_chunks()*.
- **kwargs\_icp** (*dict[str, Any]* / *None*) – Other parameter passed to *\_register\_icp\_open3d()*.
- **kwargs\_cc** (*dict[str, Any]* / *None*) – Other parameter passed to *register\_cc()*.

#### Variables

- **count** (*int*) – A counter for counting instantiations.
- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Meta-data about the current analysis routine.
- **locdata** (*LocData* / *None*) – Localization data representing the source on which to perform the manipulation.
- **collection** (*LocData* / *None*) – Collection of locdata chunks
- **transformations** (*list[Transformation]* / *None*) – Transformations for locdata chunks
- **transformation\_models** (*dict[str, list[DriftModel / DriftComponent]* / *None]*) – The fitted model objects.
- **locdata\_corrected** (*LocData* / *None*) – Localization data with drift-corrected coordinates.

## Methods

|  |   |
|--|---|
| <code>__init__([meta, chunks, chunk_size, ...])</code> |   |
| <code>apply_correction([locdata, from_model])</code>   | Correct drift by applying the estimated transformations to locdata.                     |
| <code>compute(locdata)</code>                          | Run the computation.  |
| <code>fit_transformation([slice_data, ...])</code>     | Fit drift model to selected component of the estimated transformations.                 |
| <code>fit_transformations([slice_data, ...])</code>    | Fit model parameter to all estimated transformation components.                         |
| <code>plot([ax, transformation_component, ...])</code> | Plot the transformation components as function of average frame for each locdata chunk. |
| <code>report(*args, **kwargs)</code>                   | Show a report about analysis results.   |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**`apply_correction(locdata=None, from_model=True)`**

Correct drift by applying the estimated transformations to locdata.

### Parameters

- **locdata** (Optional[[LocData](#)]) – Localization data to apply correction on. If None correction is applied to self.locdata.
- **from\_model** (bool) – If *True* compute transformation matrix from fitted transformation models and apply interpolated transformations. If *False* use the estimated transformation matrix for each data chunk.

### Return type

Self

**`compute(locdata)`**

Run the computation.

### Parameters

**locdata** ([LocData](#)) – Localization data representing the source on which to perform the manipulation.

### Return type

Self

**`count: int = 0`**

A counter for counting Analysis class instantiations (class attribute).

**`fit_transformation(slice_data=None, transformation_component='matrix', element=0, drift_model='linear', verbose=False)`**

Fit drift model to selected component of the estimated transformations.

**Parameters**

- **slice\_data** (Optional[slice]) – Reduce data to a selection on the localization chunks.
- **transformation\_component** (Literal['matrix', 'offset']) – One of 'matrix' or 'offset'
- **element** (int) – The element of flattened transformation matrix or offset
- **drift\_model** (UnionType[*DriftComponent*, str, None]) – A drift model as defined by a *DriftComponent* instance or the parameter *type* as defined in *DriftComponent*. For None no change will occur. To reset transformation\_models set the transformation\_component to None: e.g. self.transformation\_components = None or self.transformation\_components['matrix'] = None.
- **verbose** (bool) – Print the fitted model curves using lmfit.

**Return type**

Self

**fit\_transformations**(*slice\_data=None, matrix\_models=None, offset\_models=None, verbose=False*)

Fit model parameter to all estimated transformation components.

**Parameters**

- **slice\_data** (Optional[slice]) – Reduce data to a selection on the localization chunks.
- **matrix\_models** (Optional[list[*DriftComponent*]]) – Models to use for fitting each matrix component. Length of list must be equal to the square of the transformations dimension (4 or 9). If None, no matrix transformation will be carried out when calling *apply\_correction()*.
- **offset\_models** (Optional[list[*DriftComponent*]]) – Models to use for fitting each offset component. Length of list must be equal to the transformations dimension (2 or 3). If None, no offset transformation will be carried out when calling *apply\_correction()*.
- **verbose** (bool) – Print the fitted model curves.

**Return type**

Self

**plot**(*ax=None, transformation\_component='matrix', element=None, window=1, \*\*kwargs*)

Plot the transformation components as function of average frame for each locdata chunk.

**Parameters**

- **ax** (Optional[Axes]) – The axes on which to show the image
- **transformation\_component** (Literal['matrix', 'offset']) – One of 'matrix' or 'offset'
- **element** (Optional[int]) – The element of flattened transformation matrix or offset to be plotted; if None all plots are shown.



- **window** (int) – Window for running average that is applied before plotting. Not implemented yet.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.plot()`.

**Returns**

Axes object with the plot.

**Return type**

`matplotlib.axes.Axes`

**locan.analysis.drift.DriftComponent**

**class** `locan.analysis.drift.DriftComponent`(*type=None, \*\*kwargs*)

Bases: `object`

Class carrying model functions to describe drift over time (in unit of frames). `DriftComponent` provides a transformation to apply a drift correction.

Standard models for constant, linear or polynomial drift correction are taken from `lmfit.models`. For fitting splines we use the `scipy.interpolate.splrep()`.

**Parameters**

**type** (`Literal["none", "zero", "one", "constant", "linear", "polynomial", "spline"] | lmfit.models.Model | None`) – Model class or indicator for setting up the corresponding model class.

**Variables**

- **type** – String indicator for model.
- **model** (`lmfit.models.Model | None`) – The model definition (return value of `scipy.interpolate.splrep()`)
- **model\_result** (`lmfit.model.ModelResult, collection of model results.`) – The results collected from fitting the model to specified data.

**Methods**

|                                   |  |
|-----------------------------------|--|
| <code>__init__([type])</code>     |  |
| <code>eval(x)</code>              | Compute a transformation for time <i>x</i> from the drift model.   |
| <code>fit(x, y[, verbose])</code> | Fit model to the given data and create <i>self.model_results</i> . |

**eval(x)**

Compute a transformation for time *x* from the drift model.

**Parameters**

**x** (`Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]`) – frame values

### Return type

npt.NDArray[np.float64].

**fit**(*x*, *y*, *verbose=False*, *\*\*kwargs*)

Fit model to the given data and create *self.model\_results*.

### Parameters

- **x** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – x data
- **y** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – y values
- **verbose** (bool) – show plot
- **kwargs** (Any) – Other parameters passed to `lmfit.model.fit()` or to `scipy.interpolate.splrep()` Use the parameter *s* to set the amount of smoothing.

### Return type

Self

## locan.analysis.drift.DriftModel

**class** locan.analysis.drift.DriftModel(*\*args*, *\*\*kwargs*)

Bases: Protocol

### Methods

|  |                     |
|--|---------------------|
| <code>__init__(*args, **kwargs)</code> |                     |
| <code>eval(*args, **kwargs)</code>     | <b>rtype</b><br>Any |
| <code>fit(*args, **kwargs)</code>      | <b>rtype</b><br>Any |
| <code>plot(*args, **kwargs)</code>     | <b>rtype</b><br>Any |

`eval(*args, **kwargs)`

**Return type**

Any

**fit**(\*args, \*\*kwargs)

**Return type**

Any

**plot**(\*args, \*\*kwargs)

**Return type**

Any

## locan.analysis.grouped\_property\_expectation

Analyze grouped property expectations.

As an example, assume a collection of locdatas representing localization clusters. Some property of locdata *loc\_property* (i.e. *cluster property*; e.g. *intensity*) might depend on another property *other\_loc\_property* (e.g. number of localizations in cluster). For inspection, all values of *loc\_property* are grouped by *other\_loc\_property* and mean with standard deviations are displayed.

**See also:**

`position_variance_expectation`, `convex_hull_expectation`

## Classes

|  |  |
|--|--|
| <code>GroupedPropertyExpectation</code> ([meta, ...])        | Analyze variation of localization property in relation to other localization property that is grouped. |
| <code>GroupedPropertyExpectationResults</code> ([value ...]) |  |

## locan.analysis.grouped\_property\_expectation.GroupedPropertyExpectation

```
class locan.analysis.grouped_property_expectation.GroupedPropertyExpectation(meta=None,
loc_property=None,
other_loc_property=None,
expec=None,
meta=None)

```

Bases: `_Analysis`

Analyze variation of localization property in relation to other localization property that is grouped.

### Parameters

- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata`) – Meta-data about the current analysis routine.
- **loc\_property** (`str` / `None`) – The localization property to analyze.

- **other\_loc\_property** (*str* | *None*) – The localization property to group on.
- **expectation** (*int* | *float* | *Mapping* | *pd.Series[Any]* | *None*) – The expected value for all or each other localization property.

### Variables

- **count** (*int*) – A counter for counting instantiations (class attribute).
- **parameter** (*dict[str, Any]*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Meta-data about the current analysis routine.
- **results** (*GroupedPropertyExpectationResults*) – Computed results.
- **distribution\_statistics** (*Distribution\_stats*, *None*) – Distribution parameters derived from MLE fitting of results.grouped.

### Methods

|  |  |
|--|--|
| <code>__init__([meta, loc_property, ...])</code>     |  |
| <code>compute(locdata)</code>                        | Run the computation.   |
| <code>hist([ax, bins, n_bins, bin_size, ...])</code> | Provide plot as <code>matplotlib.axes.Axes</code> object showing the 2-dimensional histogram.                      |
| <code>plot([ax])</code>                              | Provide plot as <code>matplotlib.axes.Axes</code> object showing the variances as function of localization counts. |
| <code>report(*args, **kwargs)</code>                 | Show a report about analysis results.  |

### Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

#### **compute**(*locdata*)

Run the computation.

##### Parameters

**locdata** (*LocData*) – Localization data.

##### Return type

Self

**hist**(*ax=None*, *bins=None*, *n\_bins=None*, *bin\_size=None*, *bin\_edges=None*, *bin\_range=None*, *log=True*, *fit=False*, *\*\*kwargs*)

Provide plot as `matplotlib.axes.Axes` object showing the 2-dimensional histogram.

##### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **bins** (UnionType[Bins, Axis, AxesTuple, None]) – The bin specification as defined in Bins
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (UnionType[tuple[float, float], Sequence[float], Sequence[Sequence[float]], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2).
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **log** (bool) – Flag for plotting on a log scale.
- **fit** (bool) – Flag indicating if distribution fit is shown.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.pcolormesh()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

**plot**(*ax=None*, *\*\*kwargs*)

Provide plot as `matplotlib.axes.Axes` object showing the variances as function of localization counts.

#### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.plot()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

**locan.analysis.grouped\_property\_expectation.GroupedPropertyExpectationResults**

```
class locan.analysis.grouped_property_expectation.GroupedPropertyExpectationResults(values=<
                                                    grouped=
```

Bases: object

**Methods**

```
__init__([values, grouped])
```

**Attributes**

*values*

*grouped*

**grouped:** DataFrame

**values:** DataFrame

**locan.analysis.localization\_precision**

Compute localization precision from successive nearby localizations.

Localization precision is estimated from spatial variations of all localizations that appear in successive frames within a specified search radius<sup>1</sup>.

Localization pair distance distributions are fitted according to the probability density functions in<sup>2</sup>.

The estimated sigmas describe the standard deviation for pair distances. Localization precision is often defined as the standard deviation for localization distances from the center position. With that definition, the localization precision is equal to  $\sigma / \sqrt{2}$ .

<sup>1</sup> Endesfelder, Ulrike, et al., A simple method to estimate the average localization precision of a single-molecule localization microscopy experiment. *Histochemistry and Cell Biology* 141.6 (2014): 629-638.

<sup>2</sup> L. Stirling Churchman, Henrik Flyvbjerg, James A. Spudich, A Non-Gaussian Distribution Quantifies Distances Measured with Fluorescence Localization Techniques. *Biophysical Journal* 90 (2), 2006, 668-671, doi.org/10.1529/biophysj.105.065599.

## References

## Classes

|   |  |
|---|--|
| <code>LocalizationPrecision([meta, radius])</code>            | Compute the localization precision from consecutive nearby localizations.                                      |
| <code>PairwiseDistance1d([momtype, a, b, xtol, ...])</code>   | A random variable describing the distribution of pair distances for two normal distributed point clouds in 1D. |
| <code>PairwiseDistance1dIdenticalSigmaZeroMu</code>           | A random variable describing the distribution of pair distances for two normal distributed point clouds in 1D. |
| <code>PairwiseDistance2d([momtype, a, b, xtol, ...])</code>   | A random variable describing the distribution of pair distances for two normal distributed point clouds in 2D. |
| <code>PairwiseDistance2dIdenticalSigma([momtype, ...])</code> | A random variable describing the distribution of pair distances for two normal distributed point clouds in 2D. |
| <code>PairwiseDistance2dIdenticalSigmaZeroMu</code>           | A random variable describing the distribution of pair distances for two normal distributed point clouds in 2D. |
| <code>PairwiseDistance3d([momtype, a, b, xtol, ...])</code>   | A random variable describing the distribution of pair distances for two normal distributed point clouds in 3D. |
| <code>PairwiseDistance3dIdenticalSigmaZeroMu</code>           | A random variable describing the distribution of pair distances for two normal distributed point clouds in 3D. |

### locan.analysis.localization\_precision.LocalizationPrecision

```
class locan.analysis.localization_precision.LocalizationPrecision(meta=None,
                                                                radius=50)
```

Bases: `_Analysis`

Compute the localization precision from consecutive nearby localizations.

#### Parameters

- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata` | `None`) – Metadata about the current analysis routine.
- **radius** (`int` | `float`) – Search radius for nearest-neighbor searches.

#### Variables

- **count** (`int`) – A counter for counting instantiations (class attribute).
- **parameter** (`dict`) – A dictionary with all settings for the current computation.
- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata`) – Metadata about the current analysis routine.

- **results** (*pandas.DataFrame*) – Computed results.
- **distribution\_statistics** (*Distribution\_fits* | *None*) – Distribution parameters derived from MLE fitting of results.

## Methods

|  |   |
|--|---|
| <code>__init__([meta, radius])</code>            |   |
| <code>compute(locdata)</code>                    | Run the computation.  |
| <code>fit_distributions([loc_property])</code>   | Fit probability density functions to the distributions of <i>loc_property</i> values in the results using MLE ( <i>scipy.stats</i> ). |
| <code>hist([ax, loc_property, bins, fit])</code> | Provide histogram as <i>matplotlib.axes.Axes</i> object showing the distributions of results.   |
| <code>plot([ax, loc_property, window])</code>    | Provide plot as <i>matplotlib.axes.Axes</i> object showing the running average of results over window size.                           |
| <code>report(*args, **kwargs)</code>             | Show a report about analysis results.   |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*)

Run the computation.

### Parameters

**locdata** (*LocData*) – Localization data.

### Return type

Self

**fit\_distributions**(*loc\_property=None, \*\*kwargs*)

Fit probability density functions to the distributions of *loc\_property* values in the results using MLE (*scipy.stats*).

### Parameters

- **loc\_property** (*str*) – The LocData property for which to fit an appropriate distribution; if None all plots are shown.
- **kwargs** (*Any*) – Other parameters passed to the *distribution.fit()* method.

### Return type

None

**hist**(*ax=None, loc\_property='position\_distance', bins='auto', fit=True, \*\*kwargs*)

Provide histogram as *matplotlib.axes.Axes* object showing the distributions of results.



### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **loc\_property** (str) – The property for which to plot localization precision.
- **bins** (int | Sequence[int | float] | str) – Bin specifications (passed to `matplotlib.hist()`).
- **fit** (bool) – Flag indicating if distributions fit are shown.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.his()`.

### Returns

Axes object with the plot.

### Return type

`matplotlib.axes.Axes`

**plot**(*ax=None, loc\_property=None, window=1, \*\*kwargs*)

Provide plot as `matplotlib.axes.Axes` object showing the running average of results over window size.

### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **loc\_property** (UnionType[str, list[str], None]) – The property for which to plot localization precision; if None all plots are shown.
- **window** (int) – Window for running average that is applied before plotting.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.plot()`.

### Returns

Axes object with the plot.

### Return type

`matplotlib.axes.Axes`

## locan.analysis.localization\_precision.PairwiseDistance1d

```
class locan.analysis.localization_precision.PairwiseDistance1d(momtype=1,
                                                             a=None, b=None,
                                                             xtol=1e-14,
                                                             badvalue=None,
                                                             name=None,
                                                             longname=None,
                                                             shapes=None,
                                                             seed=None)
```

Bases: `rv_continuous`

A random variable describing the distribution of pair distances for two normal distributed point clouds in 1D.

The continuous distribution class inherits from `scipy.stats.rv_continuous` a set of methods and is defined by overriding the `_pdf` method.

For theoretical background see<sup>1</sup>.

### Parameters

- **x** (*npt.ArrayLike*) – distance
- **mu** (*npt.ArrayLike*) – Distance between the point cloud center positions.
- **sigma\_1** (*npt.ArrayLike*) – Standard deviation for one point cloud.
- **sigma\_2** (*npt.ArrayLike*) – Standard deviation for the other point cloud.

### References

---

<sup>1</sup> L. Stirling Churchman, Henrik Flyvbjerg, James A. Spudich, A Non-Gaussian Distribution Quantifies Distances Measured with Fluorescence Localization Techniques. Biophysical Journal 90 (2), 2006, 668-671, doi.org/10.1529/biophysj.105.065599.

## Methods

|   |   |
|---|---|
| <code>__init__([momtype, a, b, xtol, badvalue, ...])</code> |   |
| <code>cdf(x, *args, **kwargs)</code>                        | Cumulative distribution function of the given RV.   |
| <code>entropy(*args, **kwargs)</code>                       | Differential entropy of the RV.   |
| <code>expect([func, args, loc, scale, lb, ub, ...])</code>  | Calculate expected value of a function with respect to the distribution by numerical integration. |
| <code>fit(data, *args, **kwargs)</code>                     | Return estimates of shape (if applicable), location, and scale parameters from data.              |
| <code>fit_loc_scale(data, *args)</code>                     | Estimate loc and scale parameters from data using 1st and 2nd moments.                            |
| <code>freeze(*args, **kwargs)</code>                        | Freeze the distribution for the given arguments.  |
| <code>interval(confidence, *args, **kwargs)</code>          | Confidence interval with equal areas around the median.   |
| <code>isf(q, *args, **kwargs)</code>                        | Inverse survival function (inverse of <i>sf</i> ) at <i>q</i> of the given RV.                    |
| <code>logcdf(x, *args, **kwargs)</code>                     | Log of the cumulative distribution function at <i>x</i> of the given RV.                          |
| <code>logpdf(x, *args, **kwargs)</code>                     | Log of the probability density function at <i>x</i> of the given RV.                              |
| <code>logsf(x, *args, **kwargs)</code>                      | Log of the survival function of the given RV.   |
| <code>mean(*args, **kwargs)</code>                          | Mean of the distribution.   |
| <code>median(*args, **kwargs)</code>                        | Median of the distribution.   |
| <code>moment(order, *args, **kwargs)</code>                 | non-central moment of distribution of specified order.  |
| <code>nnlf(theta, x)</code>                                 | Negative loglikelihood function.  |
| <code>pdf(x, *args, **kwargs)</code>                        | Probability density function at <i>x</i> of the given RV.   |
| <code>ppf(q, *args, **kwargs)</code>                        | Percent point function (inverse of <i>cdf</i> ) at <i>q</i> of the given RV.                      |
| <code>rvs(*args, **kwargs)</code>                           | Random variates of given type.  |
| <code>sf(x, *args, **kwargs)</code>                         | Survival function ( $1 - cdf$ ) at <i>x</i> of the given RV.                                      |
| <code>stats(*args, **kwargs)</code>                         | Some statistics of the given RV.  |
| <code>std(*args, **kwargs)</code>                           | Standard deviation of the distribution.   |
| <code>support(*args, **kwargs)</code>                       | Support of the distribution.  |
| <code>var(*args, **kwargs)</code>                           | Variance of the distribution.   |

## Attributes

|                           |   |
|---------------------------|---|
| <code>random_state</code> | Get or set the generator object for generating random variates. |
|---------------------------|---|

### `locan.analysis.localization_precision.PairwiseDistance1dIdenticalSigmaZeroMu`

```
class locan.analysis.localization_precision.PairwiseDistance1dIdenticalSigmaZeroMu(momtype=
    a=None,
    b=None,
    xtol=1e-
    14,
    bad-
    value=None,
    name=None,
    long-
    name=None,
    shapes=None,
    seed=None)
```

Bases: `rv_continuous`

A random variable describing the distribution of pair distances for two normal distributed point clouds in 1D.

The continuous distribution class inherits from `scipy.stats.rv_continuous` a set of methods and is defined by overriding the `_pdf` method.

For theoretical background see<sup>1</sup>.

#### Parameters

- **`x`** (`npt.ArrayLike`) – distance
- **`sigma`** (`npt.ArrayLike`) – Standard deviation for point clouds

<sup>1</sup> L. Stirling Churchman, Henrik Flyvbjerg, James A. Spudich, A Non-Gaussian Distribution Quantifies Distances Measured with Fluorescence Localization Techniques. Biophysical Journal 90 (2), 2006, 668-671, doi.org/10.1529/biophysj.105.065599.

## References

## Methods

|   |   |
|---|---|
| <code>__init__([momtype, a, b, xtol, badvalue, ...])</code> |   |
| <code>cdf(x, *args, **kwargs)</code>                        | Cumulative distribution function of the given RV.   |
| <code>entropy(*args, **kwargs)</code>                       | Differential entropy of the RV.   |
| <code>expect([func, args, loc, scale, lb, ub, ...])</code>  | Calculate expected value of a function with respect to the distribution by numerical integration. |
| <code>fit(data, *args, **kwargs)</code>                     | Return estimates of shape (if applicable), location, and scale parameters from data.              |
| <code>fit_loc_scale(data, *args)</code>                     | Estimate loc and scale parameters from data using 1st and 2nd moments.                            |
| <code>freeze(*args, **kwargs)</code>                        | Freeze the distribution for the given arguments.  |
| <code>interval(confidence, *args, **kwargs)</code>          | Confidence interval with equal areas around the median.   |
| <code>isf(q, *args, **kwargs)</code>                        | Inverse survival function (inverse of <i>sf</i> ) at q of the given RV.                           |
| <code>logcdf(x, *args, **kwargs)</code>                     | Log of the cumulative distribution function at x of the given RV.                                 |
| <code>logpdf(x, *args, **kwargs)</code>                     | Log of the probability density function at x of the given RV.                                     |
| <code>logsf(x, *args, **kwargs)</code>                      | Log of the survival function of the given RV.   |
| <code>mean(*args, **kwargs)</code>                          | Mean of the distribution.   |
| <code>median(*args, **kwargs)</code>                        | Median of the distribution.   |
| <code>moment(order, *args, **kwargs)</code>                 | non-central moment of distribution of specified order.  |
| <code>nnlf(theta, x)</code>                                 | Negative loglikelihood function.  |
| <code>pdf(x, *args, **kwargs)</code>                        | Probability density function at x of the given RV.  |
| <code>ppf(q, *args, **kwargs)</code>                        | Percent point function (inverse of <i>cdf</i> ) at q of the given RV.                             |
| <code>rvs(*args, **kwargs)</code>                           | Random variates of given type.  |
| <code>sf(x, *args, **kwargs)</code>                         | Survival function ( $1 - cdf$ ) at x of the given RV.   |
| <code>stats(*args, **kwargs)</code>                         | Some statistics of the given RV.  |
| <code>std(*args, **kwargs)</code>                           | Standard deviation of the distribution.   |
| <code>support(*args, **kwargs)</code>                       | Support of the distribution.  |
| <code>var(*args, **kwargs)</code>                           | Variance of the distribution.   |

## Attributes

|                           |   |
|---------------------------|---|
| <code>random_state</code> | Get or set the generator object for generating random variates. |
|---------------------------|---|

### `locan.analysis.localization_precision.PairwiseDistance2d`

```
class locan.analysis.localization_precision.PairwiseDistance2d(momtype=1,  
                                                             a=None, b=None,  
                                                             xtol=1e-14,  
                                                             badvalue=None,  
                                                             name=None,  
                                                             longname=None,  
                                                             shapes=None,  
                                                             seed=None)
```

Bases: `rv_continuous`

A random variable describing the distribution of pair distances for two normal distributed point clouds in 2D.

The continuous distribution class inherits from `scipy.stats.rv_continuous` a set of methods and is defined by overriding the `_pdf` method.

For theoretical background see<sup>1</sup>.

#### Parameters

- **`x`** (*npt.ArrayLike*) – distance
- **`mu`** (*npt.ArrayLike*) – Distance between the point cloud center positions.
- **`sigma_1`** (*npt.ArrayLike*) – Standard deviation for one point cloud.
- **`sigma_2`** (*npt.ArrayLike*) – Standard deviation for the other point cloud.

---

<sup>1</sup> L. Stirling Churchman, Henrik Flyvbjerg, James A. Spudich, A Non-Gaussian Distribution Quantifies Distances Measured with Fluorescence Localization Techniques. Biophysical Journal 90 (2), 2006, 668-671, doi.org/10.1529/biophysj.105.065599.

## References

## Methods

|   |   |
|---|---|
| <code>__init__([momtype, a, b, xtol, badvalue, ...])</code> |   |
| <code>cdf(x, *args, **kwargs)</code>                        | Cumulative distribution function of the given RV.   |
| <code>entropy(*args, **kwargs)</code>                       | Differential entropy of the RV.   |
| <code>expect([func, args, loc, scale, lb, ub, ...])</code>  | Calculate expected value of a function with respect to the distribution by numerical integration. |
| <code>fit(data, *args, **kwargs)</code>                     | Return estimates of shape (if applicable), location, and scale parameters from data.              |
| <code>fit_loc_scale(data, *args)</code>                     | Estimate loc and scale parameters from data using 1st and 2nd moments.                            |
| <code>freeze(*args, **kwargs)</code>                        | Freeze the distribution for the given arguments.  |
| <code>interval(confidence, *args, **kwargs)</code>          | Confidence interval with equal areas around the median.   |
| <code>isf(q, *args, **kwargs)</code>                        | Inverse survival function (inverse of <i>sf</i> ) at <i>q</i> of the given RV.                    |
| <code>logcdf(x, *args, **kwargs)</code>                     | Log of the cumulative distribution function at <i>x</i> of the given RV.                          |
| <code>logpdf(x, *args, **kwargs)</code>                     | Log of the probability density function at <i>x</i> of the given RV.                              |
| <code>logsf(x, *args, **kwargs)</code>                      | Log of the survival function of the given RV.   |
| <code>mean(*args, **kwargs)</code>                          | Mean of the distribution.   |
| <code>median(*args, **kwargs)</code>                        | Median of the distribution.   |
| <code>moment(order, *args, **kwargs)</code>                 | non-central moment of distribution of specified order.  |
| <code>nnlf(theta, x)</code>                                 | Negative loglikelihood function.  |
| <code>pdf(x, *args, **kwargs)</code>                        | Probability density function at <i>x</i> of the given RV.   |
| <code>ppf(q, *args, **kwargs)</code>                        | Percent point function (inverse of <i>cdf</i> ) at <i>q</i> of the given RV.                      |
| <code>rvs(*args, **kwargs)</code>                           | Random variates of given type.  |
| <code>sf(x, *args, **kwargs)</code>                         | Survival function ( $1 - cdf$ ) at <i>x</i> of the given RV.                                      |
| <code>stats(*args, **kwargs)</code>                         | Some statistics of the given RV.  |
| <code>std(*args, **kwargs)</code>                           | Standard deviation of the distribution.   |
| <code>support(*args, **kwargs)</code>                       | Support of the distribution.  |
| <code>var(*args, **kwargs)</code>                           | Variance of the distribution.   |

## Attributes

|                           |   |
|---------------------------|---|
| <code>random_state</code> | Get or set the generator object for generating random variates. |
|---------------------------|---|

### `locan.analysis.localization_precision.PairwiseDistance2dIdenticalSigma`

```
class locan.analysis.localization_precision.PairwiseDistance2dIdenticalSigma(momtype=1,  
                                                                           a=None,  
                                                                           b=None,  
                                                                           xtol=1e-  
                                                                           14,  
                                                                           bad-  
                                                                           value=None,  
                                                                           name=None,  
                                                                           long-  
                                                                           name=None,  
                                                                           shapes=None,  
                                                                           seed=None)
```

Bases: `rv_continuous`

A random variable describing the distribution of pair distances for two normal distributed point clouds in 2D.

The continuous distribution class inherits from `scipy.stats.rv_continuous` a set of methods and is defined by overriding the `_pdf` method.

For theoretical background see<sup>1</sup>.

#### Parameters

- **x** (*npt.ArrayLike*) – distance
- **mu** (*npt.ArrayLike*) – Distance between the point cloud center positions.
- **sigma** (*npt.ArrayLike*) – Standard deviation for both point clouds.

## References

---

<sup>1</sup> L. Stirling Churchman, Henrik Flyvbjerg, James A. Spudich, A Non-Gaussian Distribution Quantifies Distances Measured with Fluorescence Localization Techniques. Biophysical Journal 90 (2), 2006, 668-671, doi.org/10.1529/biophysj.105.065599.



## Methods

|   |   |
|---|---|
| <code>__init__([momtype, a, b, xtol, badvalue, ...])</code> |   |
| <code>cdf(x, *args, **kwargs)</code>                        | Cumulative distribution function of the given RV.   |
| <code>entropy(*args, **kwargs)</code>                       | Differential entropy of the RV.   |
| <code>expect([func, args, loc, scale, lb, ub, ...])</code>  | Calculate expected value of a function with respect to the distribution by numerical integration. |
| <code>fit(data, *args, **kwargs)</code>                     | Return estimates of shape (if applicable), location, and scale parameters from data.              |
| <code>fit_loc_scale(data, *args)</code>                     | Estimate loc and scale parameters from data using 1st and 2nd moments.                            |
| <code>freeze(*args, **kwargs)</code>                        | Freeze the distribution for the given arguments.  |
| <code>interval(confidence, *args, **kwargs)</code>          | Confidence interval with equal areas around the median.   |
| <code>isf(q, *args, **kwargs)</code>                        | Inverse survival function (inverse of <i>sf</i> ) at <i>q</i> of the given RV.                    |
| <code>logcdf(x, *args, **kwargs)</code>                     | Log of the cumulative distribution function at <i>x</i> of the given RV.                          |
| <code>logpdf(x, *args, **kwargs)</code>                     | Log of the probability density function at <i>x</i> of the given RV.                              |
| <code>logsf(x, *args, **kwargs)</code>                      | Log of the survival function of the given RV.   |
| <code>mean(*args, **kwargs)</code>                          | Mean of the distribution.   |
| <code>median(*args, **kwargs)</code>                        | Median of the distribution.   |
| <code>moment(order, *args, **kwargs)</code>                 | non-central moment of distribution of specified order.  |
| <code>nnlf(theta, x)</code>                                 | Negative loglikelihood function.  |
| <code>pdf(x, *args, **kwargs)</code>                        | Probability density function at <i>x</i> of the given RV.   |
| <code>ppf(q, *args, **kwargs)</code>                        | Percent point function (inverse of <i>cdf</i> ) at <i>q</i> of the given RV.                      |
| <code>rvs(*args, **kwargs)</code>                           | Random variates of given type.  |
| <code>sf(x, *args, **kwargs)</code>                         | Survival function ( $1 - cdf$ ) at <i>x</i> of the given RV.                                      |
| <code>stats(*args, **kwargs)</code>                         | Some statistics of the given RV.  |
| <code>std(*args, **kwargs)</code>                           | Standard deviation of the distribution.   |
| <code>support(*args, **kwargs)</code>                       | Support of the distribution.  |
| <code>var(*args, **kwargs)</code>                           | Variance of the distribution.   |

## Attributes

|                           |   |
|---------------------------|---|
| <code>random_state</code> | Get or set the generator object for generating random variates. |
|---------------------------|---|

### `locan.analysis.localization_precision.PairwiseDistance2dIdenticalSigmaZeroMu`

```
class locan.analysis.localization_precision.PairwiseDistance2dIdenticalSigmaZeroMu(momtype=
    a=None,
    b=None,
    xtol=1e-
    14,
    bad-
    value=None,
    name=None,
    long-
    name=None,
    shapes=None,
    seed=None)
```

Bases: `rv_continuous`

A random variable describing the distribution of pair distances for two normal distributed point clouds in 2D.

The continuous distribution class inherits from `scipy.stats.rv_continuous` a set of methods and is defined by overriding the `_pdf` method.

For theoretical background see<sup>1</sup>.

#### Parameters

- **`x`** (`npt.ArrayLike`) – distance
- **`sigma`** (`npt.ArrayLike`) – Standard deviation for point clouds

<sup>1</sup> L. Stirling Churchman, Henrik Flyvbjerg, James A. Spudich, A Non-Gaussian Distribution Quantifies Distances Measured with Fluorescence Localization Techniques. Biophysical Journal 90 (2), 2006, 668-671, doi.org/10.1529/biophysj.105.065599.

## References

## Methods

|   |   |
|---|---|
| <code>__init__([momtype, a, b, xtol, badvalue, ...])</code> |   |
| <code>cdf(x, *args, **kwargs)</code>                        | Cumulative distribution function of the given RV.   |
| <code>entropy(*args, **kwargs)</code>                       | Differential entropy of the RV.   |
| <code>expect([func, args, loc, scale, lb, ub, ...])</code>  | Calculate expected value of a function with respect to the distribution by numerical integration. |
| <code>fit(data, *args, **kwargs)</code>                     | Return estimates of shape (if applicable), location, and scale parameters from data.              |
| <code>fit_loc_scale(data, *args)</code>                     | Estimate loc and scale parameters from data using 1st and 2nd moments.                            |
| <code>freeze(*args, **kwargs)</code>                        | Freeze the distribution for the given arguments.  |
| <code>interval(confidence, *args, **kwargs)</code>          | Confidence interval with equal areas around the median.   |
| <code>isf(q, *args, **kwargs)</code>                        | Inverse survival function (inverse of <i>sf</i> ) at q of the given RV.                           |
| <code>logcdf(x, *args, **kwargs)</code>                     | Log of the cumulative distribution function at x of the given RV.                                 |
| <code>logpdf(x, *args, **kwargs)</code>                     | Log of the probability density function at x of the given RV.                                     |
| <code>logsf(x, *args, **kwargs)</code>                      | Log of the survival function of the given RV.   |
| <code>mean(*args, **kwargs)</code>                          | Mean of the distribution.   |
| <code>median(*args, **kwargs)</code>                        | Median of the distribution.   |
| <code>moment(order, *args, **kwargs)</code>                 | non-central moment of distribution of specified order.  |
| <code>nnlf(theta, x)</code>                                 | Negative loglikelihood function.  |
| <code>pdf(x, *args, **kwargs)</code>                        | Probability density function at x of the given RV.  |
| <code>ppf(q, *args, **kwargs)</code>                        | Percent point function (inverse of <i>cdf</i> ) at q of the given RV.                             |
| <code>rvs(*args, **kwargs)</code>                           | Random variates of given type.  |
| <code>sf(x, *args, **kwargs)</code>                         | Survival function ( $1 - cdf$ ) at x of the given RV.   |
| <code>stats(*args, **kwargs)</code>                         | Some statistics of the given RV.  |
| <code>std(*args, **kwargs)</code>                           | Standard deviation of the distribution.   |
| <code>support(*args, **kwargs)</code>                       | Support of the distribution.  |
| <code>var(*args, **kwargs)</code>                           | Variance of the distribution.   |

## Attributes

|                           |   |
|---------------------------|---|
| <code>random_state</code> | Get or set the generator object for generating random variates. |
|---------------------------|---|

### `locan.analysis.localization_precision.PairwiseDistance3d`

```
class locan.analysis.localization_precision.PairwiseDistance3d(momtype=1,  
                                                             a=None, b=None,  
                                                             xtol=1e-14,  
                                                             badvalue=None,  
                                                             name=None,  
                                                             longname=None,  
                                                             shapes=None,  
                                                             seed=None)
```

Bases: `rv_continuous`

A random variable describing the distribution of pair distances for two normal distributed point clouds in 3D.

The continuous distribution class inherits from `scipy.stats.rv_continuous` a set of methods and is defined by overriding the `_pdf` method.

For theoretical background see<sup>1</sup>.

#### Parameters

- **`x`** (*npt.ArrayLike*) – distance
- **`mu`** (*npt.ArrayLike*) – Distance between the point cloud center positions.
- **`sigma_1`** (*npt.ArrayLike*) – Standard deviation for one point cloud.
- **`sigma_2`** (*npt.ArrayLike*) – Standard deviation for the other point cloud.

---

<sup>1</sup> L. Stirling Churchman, Henrik Flyvbjerg, James A. Spudich, A Non-Gaussian Distribution Quantifies Distances Measured with Fluorescence Localization Techniques. Biophysical Journal 90 (2), 2006, 668-671, doi.org/10.1529/biophysj.105.065599.

## References

## Methods

|   |   |
|---|---|
| <code>__init__([momtype, a, b, xtol, badvalue, ...])</code> |   |
| <code>cdf(x, *args, **kwargs)</code>                        | Cumulative distribution function of the given RV.   |
| <code>entropy(*args, **kwargs)</code>                       | Differential entropy of the RV.   |
| <code>expect([func, args, loc, scale, lb, ub, ...])</code>  | Calculate expected value of a function with respect to the distribution by numerical integration. |
| <code>fit(data, *args, **kwargs)</code>                     | Return estimates of shape (if applicable), location, and scale parameters from data.              |
| <code>fit_loc_scale(data, *args)</code>                     | Estimate loc and scale parameters from data using 1st and 2nd moments.                            |
| <code>freeze(*args, **kwargs)</code>                        | Freeze the distribution for the given arguments.  |
| <code>interval(confidence, *args, **kwargs)</code>          | Confidence interval with equal areas around the median.   |
| <code>isf(q, *args, **kwargs)</code>                        | Inverse survival function (inverse of <i>sf</i> ) at q of the given RV.                           |
| <code>logcdf(x, *args, **kwargs)</code>                     | Log of the cumulative distribution function at x of the given RV.                                 |
| <code>logpdf(x, *args, **kwargs)</code>                     | Log of the probability density function at x of the given RV.                                     |
| <code>logsf(x, *args, **kwargs)</code>                      | Log of the survival function of the given RV.   |
| <code>mean(*args, **kwargs)</code>                          | Mean of the distribution.   |
| <code>median(*args, **kwargs)</code>                        | Median of the distribution.   |
| <code>moment(order, *args, **kwargs)</code>                 | non-central moment of distribution of specified order.  |
| <code>nnlf(theta, x)</code>                                 | Negative loglikelihood function.  |
| <code>pdf(x, *args, **kwargs)</code>                        | Probability density function at x of the given RV.  |
| <code>ppf(q, *args, **kwargs)</code>                        | Percent point function (inverse of <i>cdf</i> ) at q of the given RV.                             |
| <code>rvs(*args, **kwargs)</code>                           | Random variates of given type.  |
| <code>sf(x, *args, **kwargs)</code>                         | Survival function ( $1 - cdf$ ) at x of the given RV.   |
| <code>stats(*args, **kwargs)</code>                         | Some statistics of the given RV.  |
| <code>std(*args, **kwargs)</code>                           | Standard deviation of the distribution.   |
| <code>support(*args, **kwargs)</code>                       | Support of the distribution.  |
| <code>var(*args, **kwargs)</code>                           | Variance of the distribution.   |

## Attributes

|                           |   |
|---------------------------|---|
| <code>random_state</code> | Get or set the generator object for generating random variates. |
|---------------------------|---|

### `locan.analysis.localization_precision.PairwiseDistance3dIdenticalSigmaZeroMu`

```
class locan.analysis.localization_precision.PairwiseDistance3dIdenticalSigmaZeroMu(momtype=
    a=None,
    b=None,
    xtol=1e-
    14,
    bad-
    value=None,
    name=None,
    long-
    name=None,
    shapes=None,
    seed=None)
```

Bases: `rv_continuous`

A random variable describing the distribution of pair distances for two normal distributed point clouds in 3D.

The continuous distribution class inherits from `scipy.stats.rv_continuous` a set of methods and is defined by overriding the `_pdf` method.

For theoretical background see<sup>1</sup>.

#### Parameters

- **`x`** (`npt.ArrayLike`) – distance
- **`sigma`** (`npt.ArrayLike`) – Standard deviation for point clouds

<sup>1</sup> L. Stirling Churchman, Henrik Flyvbjerg, James A. Spudich, A Non-Gaussian Distribution Quantifies Distances Measured with Fluorescence Localization Techniques. Biophysical Journal 90 (2), 2006, 668-671, doi.org/10.1529/biophysj.105.065599.

## References

## Methods

|   |   |
|---|---|
| <code>__init__([momtype, a, b, xtol, badvalue, ...])</code> |   |
| <code>cdf(x, *args, **kwargs)</code>                        | Cumulative distribution function of the given RV.   |
| <code>entropy(*args, **kwargs)</code>                       | Differential entropy of the RV.   |
| <code>expect([func, args, loc, scale, lb, ub, ...])</code>  | Calculate expected value of a function with respect to the distribution by numerical integration. |
| <code>fit(data, *args, **kwargs)</code>                     | Return estimates of shape (if applicable), location, and scale parameters from data.              |
| <code>fit_loc_scale(data, *args)</code>                     | Estimate loc and scale parameters from data using 1st and 2nd moments.                            |
| <code>freeze(*args, **kwargs)</code>                        | Freeze the distribution for the given arguments.  |
| <code>interval(confidence, *args, **kwargs)</code>          | Confidence interval with equal areas around the median.   |
| <code>isf(q, *args, **kwargs)</code>                        | Inverse survival function (inverse of <i>sf</i> ) at <i>q</i> of the given RV.                    |
| <code>logcdf(x, *args, **kwargs)</code>                     | Log of the cumulative distribution function at <i>x</i> of the given RV.                          |
| <code>logpdf(x, *args, **kwargs)</code>                     | Log of the probability density function at <i>x</i> of the given RV.                              |
| <code>logsf(x, *args, **kwargs)</code>                      | Log of the survival function of the given RV.   |
| <code>mean(*args, **kwargs)</code>                          | Mean of the distribution.   |
| <code>median(*args, **kwargs)</code>                        | Median of the distribution.   |
| <code>moment(order, *args, **kwargs)</code>                 | non-central moment of distribution of specified order.  |
| <code>nnlf(theta, x)</code>                                 | Negative loglikelihood function.  |
| <code>pdf(x, *args, **kwargs)</code>                        | Probability density function at <i>x</i> of the given RV.   |
| <code>ppf(q, *args, **kwargs)</code>                        | Percent point function (inverse of <i>cdf</i> ) at <i>q</i> of the given RV.                      |
| <code>rvs(*args, **kwargs)</code>                           | Random variates of given type.  |
| <code>sf(x, *args, **kwargs)</code>                         | Survival function ( $1 - cdf$ ) at <i>x</i> of the given RV.                                      |
| <code>stats(*args, **kwargs)</code>                         | Some statistics of the given RV.  |
| <code>std(*args, **kwargs)</code>                           | Standard deviation of the distribution.   |
| <code>support(*args, **kwargs)</code>                       | Support of the distribution.  |
| <code>var(*args, **kwargs)</code>                           | Variance of the distribution.   |

## Attributes

|                           |   |
|---------------------------|---|
| <code>random_state</code> | Get or set the generator object for generating random variates. |
|---------------------------|---|

## `locan.analysis.localization_property`

Analyze localization property.

Localizations come with a range of properties including position coordinates, emission strength, local background etc.. Most properties represent random variables that were drawn from an unknown probability distribution. It is often useful to analyze the properties from all localizations within a selection and estimate the corresponding probability distribution.

## Classes

|  |  |
|--|--|
| <code>LocalizationProperty([meta, loc_property, index])</code> | Analyze localization property with respect to probability density or variation over a specified index. |
|--|--|

## `locan.analysis.localization_property.LocalizationProperty`

```
class locan.analysis.localization_property.LocalizationProperty(meta=None,
                                                             loc_property='intensity',
                                                             index=None)
```

Bases: `_Analysis`

Analyze localization property with respect to probability density or variation over a specified index.

### Parameters

- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata`) – Meta-data about the current analysis routine.
- **loc\_property** (`str`) – The property to analyze.
- **index** (`str` / `None`) – The property name that should serve as index (i.e. x-axis in x-y-plot)

### Variables

- **count** (`int`) – A counter for counting instantiations (class attribute).
- **parameter** (`dict[str, Any]`) – A dictionary with all settings for the current computation.
- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata`) – Meta-data about the current analysis routine.
- **results** (`pandas.DataFrame`) – Computed results.



- **distribution\_statistics** (*Distribution\_stats* | *None*) – Distribution parameters derived from MLE fitting of results.

## Methods

|   |   |
|---|---|
| <code>__init__([meta, loc_property, index])</code>  |   |
| <code>compute(locdata)</code>                       | Run the computation.  |
| <code>fit_distributions([distribution, ...])</code> | Fit probability density functions to the distributions of <i>loc_property</i> values in the results using MLE ( <i>scipy.stats</i> ). |
| <code>hist([ax, bins, log, fit])</code>             | Provide histogram as <i>matplotlib.axes.Axes</i> object showing <i>hist(results)</i> .  |
| <code>plot([ax, window])</code>                     | Provide plot as <i>matplotlib.axes.Axes</i> object showing the running average of results over window size.                           |
| <code>report(*args, **kwargs)</code>                | Show a report about analysis results.   |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*)

Run the computation.

### Parameters

**locdata** (*LocData*) – Localization data.

### Return type

Self

**fit\_distributions**(*distribution=<scipy.stats.\_continuous\_distns.expon\_gen object>*,  
*with\_constraints=True*, *\*\*kwargs*)

Fit probability density functions to the distributions of *loc\_property* values in the results using MLE (*scipy.stats*).

If *with\_constraints* is true we put the following constraints on the fit procedure: If *distribution* is *expon* then *floc=np.min(self.analysis\_class.results[self.loc\_property].values)*.

### Parameters

- **distribution** (*str* | *rv\_continuous*) – Distribution model to fit.
- **with\_constraints** (*bool*) – Flag to use predefined constraints on fit parameters.
- **kwargs** (*Any*) – Other parameters are passed to *scipy.stat.distribution.fit()*.

### Return type

None

**hist**(*ax=None, bins='auto', log=True, fit=True, \*\*kwargs*)

Provide histogram as `matplotlib.axes.Axes` object showing `hist(results)`. Nan entries are ignored.

#### Parameters

- **ax** (Optional[`Axes`]) – The axes on which to show the image
- **bins** (int | Sequence[int | float] | str) – Bin specifications (passed to `matplotlib.hist()`).
- **log** (bool) – Flag for plotting on a log scale.
- **fit** (Optional[bool]) – Flag indicating if distribution fit is shown. The fit will only be computed if *distribution\_statistics* is None.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.hist()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

**plot**(*ax=None, window=1, \*\*kwargs*)

Provide plot as `matplotlib.axes.Axes` object showing the running average of results over window size.

#### Parameters

- **ax** (`matplotlib.axes.Axes`) – The axes on which to show the image
- **window** (int) – Window for running average that is applied before plotting.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.plot()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

## locan.analysis.localization\_property\_2d

Analyze the distribution of a localization property in 2D.

Analyze the distribution of a localization property as function of two other localization properties in 2D. E.g. looking at how the local background is distributed over localization coordinates helps to characterize the illumination profile in SMLM experiments.

## Classes

|   |   |
|---|---|
| <i>FitImageResults</i> (image, bins, label, model_result) |   |
| <i>LocalizationProperty2d</i> ([meta, ...])               | Analyze 2d distribution of histogram for two localization properties. |

### locan.analysis.localization\_property\_2d.FitImageResults

**class** locan.analysis.localization\_property\_2d.**FitImageResults**(*image, bins, label, model\_result*)

Bases: NamedTuple

#### Methods

|  |  |
|--|--|
| <code>__init__()</code>                  |  |
| <code>count(value, /)</code>             | Return number of occurrences of value. |
| <code>index(value[, start, stop])</code> | Return first index of value.           |

#### Attributes

|                     |                          |
|---------------------|--------------------------|
| <i>bins</i>         | Alias for field number 1 |
| <i>image</i>        | Alias for field number 0 |
| <i>label</i>        | Alias for field number 2 |
| <i>model_result</i> | Alias for field number 3 |

**bins:** *Bins*

Alias for field number 1

**image:** `ndarray[Any, dtype[Any]]`

Alias for field number 0

**label:** `list[str]`

Alias for field number 2

**model\_result:** `ModelResult`

Alias for field number 3

**locan.analysis.localization\_property\_2d.LocalizationProperty2d**

```
class locan.analysis.localization_property_2d.LocalizationProperty2d(meta=None,
                                                                    loc_properties=None,
                                                                    other_property='local_backg
                                                                    bins=None,
                                                                    n_bins=None,
                                                                    bin_size=100,
                                                                    bin_edges=None,
                                                                    bin_range=None,
                                                                    rescale=None)
```

Bases: `_Analysis`

Analyze 2d distribution of histogram for two localization properties.

Fit a two dimensional Gauss distribution.

**Parameters**

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Meta-data about the current analysis routine.
- **loc\_properties** (*Sequence[str] | None*) – Localization properties to be grouped into bins. If *None*, the *coordinate\_values* of *locdata* are used.
- **other\_property** (*str | None*) – Localization property (columns in *locdata.data*) that is averaged in each pixel. If *None*, the localization counts are shown.
- **bins** (*Bins | boost\_histogram.axis.Axis | boost\_histogram.axis.AxesTuple | None*) – The bin specification as defined in *Bins*
- **bin\_edges** (*Sequence[float] | Sequence[Sequence[float]] | None*) – Bin edges for all or each dimension with shape (dimension, *n\_bin\_edges*).
- **bin\_range** (*tuple[float, float] | Sequence[float] | Sequence[Sequence[float]] | str | None*) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If *None* (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (*int | Sequence[int] | None*) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (*float | Sequence[float] | Sequence[Sequence[float]] | None*) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, *n\_bins*). 5 would describe *bin\_size* of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.

- **rescale** (*int* | *str* | *locan.Trafo* | *Callable[... , Any]* | *bool* | *None*) – Transformation as defined in *locan.Trafo* or by transformation function. For *None* or *False* no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For ‘equal’ intensity values are rescaled by histogram equalization.

### Variables

- **count** (*int*) – A counter for counting instantiations (class attribute).
- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **results** (*lmfit.model.ModelResult*) – Computed fit results.

### Methods

|  |  |
|--|--|
| <code>__init__([meta, loc_properties, ...])</code> |  |
| <code>compute(locdata)</code>                      | Run the computation.   |
| <code>plot([ax])</code>                            | Provide histogram as <code>matplotlib.axes.Axes</code> object showing <code>plot(results)</code> . |
| <code>plot_deviation_from_mean([ax])</code>        | Provide histogram as <code>matplotlib.axes.Axes</code> object showing <code>plot(results)</code> . |
| <code>plot_deviation_from_median([ax])</code>      | Provide histogram as <code>matplotlib.axes.Axes</code> object showing <code>plot(results)</code> . |
| <code>plot_residuals([ax])</code>                  | Provide histogram as <code>matplotlib.axes.Axes</code> object showing <code>plot(results)</code> . |
| <code>report()</code>                              | Show a report about analysis results.  |

### Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*)

Run the computation.

#### Parameters

**locdata** (*LocData*) – Localization data.

#### Return type

Self

**plot**(*ax=None*, *\*\*kwargs*)

Provide histogram as `matplotlib.axes.Axes` object showing `plot(results)`.

#### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.contour()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

**plot\_deviation\_from\_mean**(*ax=None*)

Provide histogram as `matplotlib.axes.Axes` object showing `plot(results)`.

#### Parameters

**ax** (Optional[Axes]) – The axes on which to show the image

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

**plot\_deviation\_from\_median**(*ax=None*)

Provide histogram as `matplotlib.axes.Axes` object showing `plot(results)`.

#### Parameters

**ax** (Optional[Axes]) – The axes on which to show the image

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

**plot\_residuals**(*ax=None, \*\*kwargs*)

Provide histogram as `matplotlib.axes.Axes` object showing `plot(results)`.

#### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.contour()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

**report**()

Show a report about analysis results.

#### Return type

`None`

## locan.analysis.localization\_property\_correlations

Analyze cross dependencies between localization properties.

Analyze cross dependencies as indicated by the correlation coefficients between any two localization properties.

### Classes

|  |   |
|--|---|
| <code>LocalizationPropertyCorrelations([meta, ...])</code> | Compute and analyze correlation coefficients between any two localization |
|--|---|

## locan.analysis.localization\_property\_correlations.LocalizationPropertyCorrelations

**class** locan.analysis.localization\_property\_correlations.LocalizationPropertyCorrelations(*meta*, *loc*)

Bases: `_Analysis`

**Compute and analyze correlation coefficients between any two localization properties.**

#### Parameters

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **loc\_properties** (*list[str] | None*) – Localization properties to be analyzed. If None all are used.

#### Variables

- **count** (*int*) – A counter for counting instantiations (class attribute).
- **parameter** (*dict[str, Any]*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata | None*) – Metadata about the current analysis routine.
- **results** (*pandas.DataFrame | None*) – The correlation coefficients..

### Methods

|   |  |
|---|--|
| <code>__init__([meta, loc_properties])</code> |  |
| <code>compute(locdata)</code>                 | Run the computation.   |
| <code>plot([ax, cbar, colorbar_kws])</code>   | Provide heatmap of all correlation values as <code>matplotlib.axes.Axes</code> object. |
| <code>report()</code>                         | Show a report about analysis results.  |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*)

Run the computation.

**Parameters**

**locdata** (*LocData*) – Localization data.

**Return type**

Self

**plot**(*ax=None, cbar=True, colorbar\_kws=None, \*\*kwargs*)

Provide heatmap of all correlation values as `matplotlib.axes.Axes` object.

**Parameters**

- **ax** (Optional[*Axes*]) – The axes on which to show the image
- **cbar** (bool) – If true draw a colorbar.
- **colorbar\_kws** (Optional[dict[str, Any]]) – Keyword arguments for `matplotlib.pyplot.colorbar()`.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.imshow()`.

**Returns**

Axes object with the plot.

**Return type**

`matplotlib.axes.Axes`

**report**()

Show a report about analysis results.

**Return type**

None

## `locan.analysis.localizations_per_frame`

Compute localizations per frame.



## Classes

|   |                                  |
|---|----------------------------------|
| <code>LocalizationsPerFrame([meta, norm, ...])</code> | Compute localizations per frame. |
|---|----------------------------------|

### locan.analysis.localizations\_per\_frame.LocalizationsPerFrame

```
class locan.analysis.localizations_per_frame.LocalizationsPerFrame(meta=None,
                                                                    norm=None,
                                                                    time_delta='integration_time',
                                                                    resample=None,
                                                                    **kwargs)
```

Bases: `_Analysis`

Compute localizations per frame.

#### Parameters

- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata`) – Meta-data about the current analysis routine.
- **norm** (`int | float | str | None`) – Normalization factor that can be None, a number, or another property in `locdata`.
- **time\_delta** (`int | float | str | pd.Timedelta | None`) – Time per frame in milliseconds. String must specify the unit like ‘10ms’. For ‘integration\_time’ the time is taken from `locdata.meta.experiment.setups[0].optical_units[0].detection.camera.integration_time`
- **resample** (`DateOffset | Timedelta | str`) – Parameter for `pandas.Series.resample()`: The offset string or object representing target conversion.
- **kwargs** (Any) – Other parameters passed to `pandas.Series.resample()`.

#### Variables

- **count** (`int`) – A counter for counting instantiations.
- **parameter** (`dict`) – A dictionary with all settings for the current computation.
- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata`) – Meta-data about the current analysis routine.
- **results** (`pandas.Series`) – Computed results.
- **distribution\_statistics** (`Distribution_fits object | None`) – Distribution parameters derived from MLE fitting of results.

## Methods

|   |   |
|---|---|
| <code>__init__([meta, norm, time_delta, resample])</code> |   |
| <code>compute(locdata)</code>                             | Run the computation.  |
| <code>fit_distributions(**kwargs)</code>                  | Fit probability density functions to the distributions of `loc_property` values in the results using MLE (scipy.stats). |
| <code>hist([ax, fit, bins])</code>                        | Provide histogram as <code>matplotlib.axes.Axes</code> object showing <code>hist(results)</code> .                      |
| <code>plot([ax, window, cumulative, normalize])</code>    | Provide plot as <code>matplotlib.axes.Axes</code> object showing the running average of results over window size.       |
| <code>report(*args, **kwargs)</code>                      | Show a report about analysis results.   |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*)

Run the computation.

### Parameters

**locdata** (Union[*LocData*, \_SupportsArray[*dtype*[Any]], \_NestedSequence[\_SupportsArray[*dtype*[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Points in time: either localization data that contains a column *frame* or an array with time points.

### Return type

Self

**count:** int = 0

A counter for counting Analysis class instantiations (class attribute).

**fit\_distributions**(\*\**kwargs*)

Fit probability density functions to the distributions of `loc\_property` values in the results using MLE (scipy.stats).

### Parameters

**loc\_property** (*str*) – The LocData property for which to fit an appropriate distribution; if None all plots are shown.

### Return type

None

**hist**(*ax=None, fit=True, bins='auto', \*\*kwargs*)

Provide histogram as `matplotlib.axes.Axes` object showing `hist(results)`.

### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **bins** (int | Sequence[int | float] | str) – Bin specifications (passed to `matplotlib.hist`).
- **fit** (bool) – Flag indicating if distributions fit are shown.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.hist()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

**plot**(*ax=None, window=1, cumulative=False, normalize=False, \*\*kwargs*)

Provide plot as `matplotlib.axes.Axes` object showing the running average of results over window size.

#### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **window** (int) – Window for running average that is applied before plotting.
- **cumulative** (bool) – Plot the cumulated results if true.
- **normalize** (bool) – Normalize cumulative plot to the last value
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.plot()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

## locan.analysis.nearest\_neighbor

Nearest-neighbor distance distribution analysis

Nearest-neighbor distance distributions provide information about deviations from a spatial homogeneous Poisson process (i.e. complete spatial randomness, CSR). Point-event distances are given by the distance between a random point (not being an event) and the nearest event. The point-event distance distribution is estimated from a number of random sample points and plotted in comparison to the analytical function for equal localization density.

For a homogeneous 2D Poisson process with intensity  $\rho$  (expected number of points per unit area) the distance from a randomly chosen event to the nearest other event (nearest-neighbor distance) is distributed according to the following probability density (pdf) or cumulative density function (cdf)<sup>1</sup>:

$$\begin{aligned} pdf(w) &= 2\rho\pi w \exp(-\rho\pi w^2) \\ cdf(w) &= 1 - \exp(-\rho\pi w^2) \end{aligned}$$

The same distribution holds for point-event distances if events are distributed as a homogeneous Poisson process with intensity  $\rho$ .

---

<sup>1</sup> Philip M. Dixon, Nearest Neighbor Methods, Department of Statistics, Iowa State University, 20 December 2001

## References

## Classes

|  |  |
|--|--|
| <code>NNDistances_csr_2d</code> ([momtype, a, b, xtol, ...]) | Continuous distribution function for nearest-neighbor distances of points distributed in 2D under complete spatial randomness. |
| <code>NNDistances_csr_3d</code> ([momtype, a, b, xtol, ...]) | Continuous distribution function for nearest-neighbor distances of points distributed in 3D under complete spatial randomness. |
| <code>NearestNeighborDistances</code> ([meta, k])            | Compute the k-nearest-neighbor distances within data or between data and other_data.   |

### locan.analysis.nearest\_neighbor.NNDistances\_csr\_2d

```
class locan.analysis.nearest_neighbor.NNDistances_csr_2d(momtype=1, a=None,
                                                         b=None, xtol=1e-14,
                                                         badvalue=None,
                                                         name=None,
                                                         longname=None,
                                                         shapes=None, seed=None)
```

Bases: `rv_continuous`

Continuous distribution function for nearest-neighbor distances of points distributed in 2D under complete spatial randomness.

#### Parameters

**density** (*float*) – Shape parameter *density*, being the density of points.

## Methods

|   |   |
|---|---|
| <code>__init__([momtype, a, b, xtol, badvalue, ...])</code> |   |
| <code>cdf(x, *args, **kwargs)</code>                        | Cumulative distribution function of the given RV.   |
| <code>entropy(*args, **kwargs)</code>                       | Differential entropy of the RV.   |
| <code>expect([func, args, loc, scale, lb, ub, ...])</code>  | Calculate expected value of a function with respect to the distribution by numerical integration. |
| <code>fit(data, *args, **kwargs)</code>                     | Return estimates of shape (if applicable), location, and scale parameters from data.              |
| <code>fit_loc_scale(data, *args)</code>                     | Estimate loc and scale parameters from data using 1st and 2nd moments.                            |
| <code>freeze(*args, **kwargs)</code>                        | Freeze the distribution for the given arguments.  |
| <code>interval(confidence, *args, **kwargs)</code>          | Confidence interval with equal areas around the median.   |
| <code>isf(q, *args, **kwargs)</code>                        | Inverse survival function (inverse of <i>sf</i> ) at <i>q</i> of the given RV.                    |
| <code>logcdf(x, *args, **kwargs)</code>                     | Log of the cumulative distribution function at <i>x</i> of the given RV.                          |
| <code>logpdf(x, *args, **kwargs)</code>                     | Log of the probability density function at <i>x</i> of the given RV.                              |
| <code>logsf(x, *args, **kwargs)</code>                      | Log of the survival function of the given RV.   |
| <code>mean(*args, **kwargs)</code>                          | Mean of the distribution.   |
| <code>median(*args, **kwargs)</code>                        | Median of the distribution.   |
| <code>moment(order, *args, **kwargs)</code>                 | non-central moment of distribution of specified order.  |
| <code>nnlf(theta, x)</code>                                 | Negative loglikelihood function.  |
| <code>pdf(x, *args, **kwargs)</code>                        | Probability density function at <i>x</i> of the given RV.   |
| <code>ppf(q, *args, **kwargs)</code>                        | Percent point function (inverse of <i>cdf</i> ) at <i>q</i> of the given RV.                      |
| <code>rvs(*args, **kwargs)</code>                           | Random variates of given type.  |
| <code>sf(x, *args, **kwargs)</code>                         | Survival function ( $1 - cdf$ ) at <i>x</i> of the given RV.                                      |
| <code>stats(*args, **kwargs)</code>                         | Some statistics of the given RV.  |
| <code>std(*args, **kwargs)</code>                           | Standard deviation of the distribution.   |
| <code>support(*args, **kwargs)</code>                       | Support of the distribution.  |
| <code>var(*args, **kwargs)</code>                           | Variance of the distribution.   |

## Attributes

|                           |   |
|---------------------------|---|
| <code>random_state</code> | Get or set the generator object for generating random variates. |
|---------------------------|---|

### `locan.analysis.nearest_neighbor.NNDistances_csr_3d`

```
class locan.analysis.nearest_neighbor.NNDistances_csr_3d(momtype=1, a=None,  
                                                         b=None, xtol=1e-14,  
                                                         badvalue=None,  
                                                         name=None,  
                                                         longname=None,  
                                                         shapes=None, seed=None)
```

Bases: `rv_continuous`

Continuous distribution function for nearest-neighbor distances of points distributed in 3D under complete spatial randomness.

#### Parameters

- **`x`** (*float*) – distance
- **`density`** (*float*) – Shape parameter *density*, being the density of points.

## Methods

|   |   |
|---|---|
| <code>__init__([momtype, a, b, xtol, badvalue, ...])</code> |   |
| <code>cdf(x, *args, **kwargs)</code>                        | Cumulative distribution function of the given RV.   |
| <code>entropy(*args, **kwargs)</code>                       | Differential entropy of the RV.   |
| <code>expect([func, args, loc, scale, lb, ub, ...])</code>  | Calculate expected value of a function with respect to the distribution by numerical integration. |
| <code>fit(data, *args, **kwargs)</code>                     | Return estimates of shape (if applicable), location, and scale parameters from data.              |
| <code>fit_loc_scale(data, *args)</code>                     | Estimate loc and scale parameters from data using 1st and 2nd moments.                            |
| <code>freeze(*args, **kwargs)</code>                        | Freeze the distribution for the given arguments.  |
| <code>interval(confidence, *args, **kwargs)</code>          | Confidence interval with equal areas around the median.   |
| <code>isf(q, *args, **kwargs)</code>                        | Inverse survival function (inverse of <i>sf</i> ) at <i>q</i> of the given RV.                    |
| <code>logcdf(x, *args, **kwargs)</code>                     | Log of the cumulative distribution function at <i>x</i> of the given RV.                          |
| <code>logpdf(x, *args, **kwargs)</code>                     | Log of the probability density function at <i>x</i> of the given RV.                              |
| <code>logsf(x, *args, **kwargs)</code>                      | Log of the survival function of the given RV.   |
| <code>mean(*args, **kwargs)</code>                          | Mean of the distribution.   |
| <code>median(*args, **kwargs)</code>                        | Median of the distribution.   |
| <code>moment(order, *args, **kwargs)</code>                 | non-central moment of distribution of specified order.  |
| <code>nnlf(theta, x)</code>                                 | Negative loglikelihood function.  |
| <code>pdf(x, *args, **kwargs)</code>                        | Probability density function at <i>x</i> of the given RV.   |
| <code>ppf(q, *args, **kwargs)</code>                        | Percent point function (inverse of <i>cdf</i> ) at <i>q</i> of the given RV.                      |
| <code>rvs(*args, **kwargs)</code>                           | Random variates of given type.  |
| <code>sf(x, *args, **kwargs)</code>                         | Survival function ( $1 - cdf$ ) at <i>x</i> of the given RV.                                      |
| <code>stats(*args, **kwargs)</code>                         | Some statistics of the given RV.  |
| <code>std(*args, **kwargs)</code>                           | Standard deviation of the distribution.   |
| <code>support(*args, **kwargs)</code>                       | Support of the distribution.  |
| <code>var(*args, **kwargs)</code>                           | Variance of the distribution.   |

## Attributes

|                           |   |
|---------------------------|---|
| <code>random_state</code> | Get or set the generator object for generating random variates. |
|---------------------------|---|

## `locan.analysis.nearest_neighbor.NearestNeighborDistances`

**class** `locan.analysis.nearest_neighbor.NearestNeighborDistances`(*meta=None, k=1*)

Bases: `_Analysis`

Compute the k-nearest-neighbor distances within data or between data and other\_data.

The algorithm relies on `sklearn.neighbors.NearestNeighbors`.

### Parameters

- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata`) – Meta-data about the current analysis routine.
- **k** (`int`) – Compute the kth nearest neighbor.

### Variables

- **count** (`int`) – A counter for counting instantiations.
- **parameter** (`dict`) – A dictionary with all settings for the current computation.
- **meta** (`locan.analysis.metadata_analysis_pb2.AMetadata`) – Meta-data about the current analysis routine.
- **results** (`pandas.DataFrame`) – Computed results.
- **distribution\_statistics** (`Distribution_stats` / `None`) – Distribution parameters derived from MLE fitting of results.

## Methods

|  |   |
|--|---|
| <code>__init__</code> ([ <i>meta, k</i> ])                       |   |
| <code>compute</code> ( <i>locdata</i> [, <i>other_locdata</i> ]) | Run the computation.  |
| <code>fit_distributions</code> ([ <i>with_constraints</i> ])     | Fit probability density functions to the distributions of <i>loc_property</i> values in the results using MLE ( <code>scipy.stats</code> ). |
| <code>hist</code> ([ <i>ax, bins, density, fit</i> ])            | Provide histogram as <code>matplotlib.axes.Axes</code> object showing <code>hist(results)</code> .  |
| <code>report</code> (*args, **kwargs)                            | Show a report about analysis results.   |



## Attributes

|              |   |
|--------------|---|
| <i>count</i> | A counter for counting Analysis class instantiations (class attribute). |
|--------------|---|

**compute**(*locdata*, *other\_locdata=None*)

Run the computation.

### Parameters

- **locdata** (*LocData*) – Localization data.
- **other\_locdata** (Optional[*LocData*]) – Other localization data from which nearest neighbors are taken.

### Return type

Self

**count:** `int = 0`

A counter for counting Analysis class instantiations (class attribute).

**fit\_distributions**(*with\_constraints=True*)

Fit probability density functions to the distributions of *loc\_property* values in the results using MLE (scipy.stats).

If *with\_constraints* is true we put the following constraints on the fit procedure: If distribution is *expon* then `floc=np.min(self.analysis_class.results[self.loc_property].values)`.

### Parameters

**with\_constraints** (bool) – Flag to use predefined constraints on fit parameters.

### Return type

None

**hist**(*ax=None*, *bins='auto'*, *density=True*, *fit=False*, *\*\*kwargs*)

Provide histogram as `matplotlib.axes.Axes` object showing `hist(results)`.

### Parameters

- **ax** (Optional[*Axes*]) – The axes on which to show the image.
- **bins** (Union[int, list[int | float], Literal['auto']]) – Bin specification as used in `matplotlib.hist()`
- **density** (bool) – Flag for normalization as used in `matplotlib.hist`. True returns probability density function; None returns counts.
- **fit** (bool) – Flag indicating to fit pdf of nearest-neighbor distances under complete spatial randomness.
- **kwargs** (Any) – Other parameters passed to `matplotlib.plot()`.

### Returns

Axes object with the plot.

### Return type

`matplotlib.axes.Axes`

## Functions

|   |   |
|---|---|
| <code>pdf_nnDistances_csr_2D(x, density)</code> | Probability density function for nearest-neighbor distances of points distributed in 2D with complete spatial randomness. |
| <code>pdf_nnDistances_csr_3D(x, density)</code> | Probability density function for nearest-neighbor distances of points distributed in 3D with complete spatial randomness. |

### `locan.analysis.nearest_neighbor.pdf_nnDistances_csr_2D`

`locan.analysis.nearest_neighbor.pdf_nnDistances_csr_2D(x, density)`

Probability density function for nearest-neighbor distances of points distributed in 2D with complete spatial randomness.

#### Parameters

- **x** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – distance
- **density** (float) – density of points

#### Returns

Probability density function pdf(x).

#### Return type

float

### `locan.analysis.nearest_neighbor.pdf_nnDistances_csr_3D`

`locan.analysis.nearest_neighbor.pdf_nnDistances_csr_3D(x, density)`

Probability density function for nearest-neighbor distances of points distributed in 3D with complete spatial randomness.

#### Parameters

- **x** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – distance
- **density** (float) – density of points

#### Returns

Probability density function pdf(x).

#### Return type

`npt.NDArray[np.float64]`

## locan.analysis.pipeline

Building an analysis pipeline.

Pipeline refers to sequential analysis steps that are applied to a single LocData object. An analysis pipeline here includes true piped analysis, where a preliminary result serves as input to the next analysis step, but also workflows that provide different results in parallel.

A batch process is a procedure for running a pipeline over multiple LocData objects while collecting and combining results.

This module provides a class *Pipeline* to combine the analysis procedure, parameters and results in a single pickleable object.

## Classes

|                                       |   |
|---------------------------------------|---|
| <i>Pipeline</i> (computation[, meta]) | The base class for a specialized analysis pipeline to be used on LocData objects. |
|---------------------------------------|---|

## locan.analysis.pipeline.Pipeline

**class** locan.analysis.pipeline.Pipeline(*computation*, *meta*=None, *\*\*kwargs*)

Bases: *\_Analysis*

The base class for a specialized analysis pipeline to be used on LocData objects.

The custom analysis routine has to be added by implementing the method *computation(self, \*\*kwargs)*. Keyword arguments must include the locdata reference and optional parameters.

Results are provided as customized attributes. We suggest abbreviated standard names for the most common procedures such as:

- *lp* - Localization Precision
- *lprop* - Localization Property
- *lpf* - Localizations per Frame
- *rhf* - Ripley H function
- *clust* - locdata with clustered elements

### Parameters

- **computation** (*Callable[...]*, *Any*) – A function *computation(self, \*\*kwargs)* specifying the analysis procedure.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Meta-data about the current analysis routine.
- **kwargs** (*Any*) – Locdata reference and optional parameters passed to *computation(self, \*\*kwargs)*.

### Variables

- **count** (*int*) – A counter for counting instantiations (class attribute).

- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata | None*) – Metadata about the current analysis routine.
- **computation** (*Callable[..., Any]*) – A function *computation(self, \*\*kwargs)* specifying the analysis procedure.
- **kwargs** – All parameters including the locdata reference that are passed to *computation(self, \*\*kwargs)*.

---

**Note:** The class variable *Pipeline.count* is only incremented in a single process. In multiprocessing *Pipeline.count* and *Pipeline.meta.identifier* (which is set using *count*) cannot be used to identify distinct Pipeline objects.

---



---

**Note:** For the Pipeline object to be pickleable attention has to be paid to the *computation()* method. With multiprocessing it will have to be re-injected for each Pipeline object by *pipeline.computation = computation* after computation and before pickling.

---

## Methods

|  |  |
|--|--|
| <code>__init__(computation[, meta])</code> |  |
| <code>computation_as_string()</code>       | Return the analysis procedure (i.e. the <i>computation()</i> method) as string.            |
| <code>compute()</code>                     | Run the analysis procedure.  |
| <code>report(*args, **kwargs)</code>       | Show a report about analysis results.  |
| <code>save_computation(path)</code>        | Save the analysis procedure (i.e. the <i>computation()</i> method) as human readable text. |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

### `computation_as_string()`

Return the analysis procedure (i.e. the *computation()* method) as string.

#### Return type

`str`

### `compute()`

Run the analysis procedure. All parameters must be given upon Pipeline instantiation.

**Return type**

Any

**save\_computation**(*path*)

Save the analysis procedure (i.e. the `computation()` method) as human readable text.

**Parameters**

**path** (*str* / *os.PathLike*[*Any*]) – Path and file name for saving the text file.

**Return type**

None

## Functions

---

|   |                                    |
|---|------------------------------------|
| <code>computation_test</code> ( <i>self</i> [, <i>locdata</i> , <i>parameter</i> ]) | A pipeline definition for testing. |
|---|------------------------------------|

---

### `locan.analysis.pipeline.computation_test`

`locan.analysis.pipeline.computation_test`(*self*, *locdata*=None, *parameter*='test')

A pipeline definition for testing.

**Return type**

TypeVar(*T\_Pipeline*, bound= Pipeline)

### `locan.analysis.position_variance_expectation`

Analyze variance of localization coordinates.

Localization coordinates in localization clusters come with a certain variance. In resolution-limited clusters the variance is determined by the localization precision.

A close look at the variance of localization coordinates as function of localization counts helps to characterize localization clusters<sup>1</sup>.

## References

## Classes

---

|   |  |
|---|--|
| <code>PositionVarianceExpectation</code> ([ <i>meta</i> , ...])         | Analyze variation of localization properties in relation to expected variations. |
| <code>PositionVarianceExpectationResults</code> ([ <i>valu</i><br>...]) |  |

---

<sup>1</sup> Ebert V, Eiring P, Helmerich DA, Seifert R, Sauer M, Doose S. Convex hull as diagnostic tool in single-molecule localization microscopy. *Bioinformatics* 38(24), 2022, 5421-5429, doi: 10.1093/bioinformatics/btac700.

**locan.analysis.position\_variance\_expectation.PositionVarianceExpectation**

```
class locan.analysis.position_variance_expectation.PositionVarianceExpectation(meta=None,
                                                                              loc_property='p
                                                                              ex-
                                                                              pec-
                                                                              ta-
                                                                              tion=None,
                                                                              bi-
                                                                              ased=True)
```

Bases: `_Analysis`

Analyze variation of localization properties in relation to expected variations.

**Parameters**

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Meta-data about the current analysis routine.
- **loc\_property** (*str*) – The localization property to analyze.
- **expectation** (*int | float | Mapping[str, Any] | pd.Series[Any] | None*) – The expected variance for all or each localization property. The expected variance equals the squared localization precision for localization position coordinates.
- **biased** (*bool*) – Flag to use biased or unbiased (Bessel-corrected) variance

**Variables**

- **count** (*int*) – A counter for counting instantiations (class attribute).
- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Meta-data about the current analysis routine.
- **results** (*PositionVarianceExpectationResults*) – Computed results.
- **distribution\_statistics** (*Distribution\_stats, None*) – Distribution parameters derived from MLE fitting of results.

**Methods**

|   |  |
|---|--|
| <code>__init__([meta, loc_property, expectation, ...])</code> |  |
| <code>compute(locdata)</code>                                 | Run the computation.   |
| <code>hist([ax, bins, n_bins, bin_size, ...])</code>          | Provide plot as <code>matplotlib.axes.Axes</code> object showing the 2-dimensional histogram of variances and localization counts. |
| <code>plot([ax])</code>                                       | Provide plot as <code>matplotlib.axes.Axes</code> object showing the variances as function of localization counts.                 |
| <code>report(*args, **kwargs)</code>                          | Show a report about analysis results.  |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*)

Run the computation.

### Parameters

**locdata** (*LocData*) – Localization data.

### Return type

Self

**hist**(*ax=None, bins=None, n\_bins=None, bin\_size=None, bin\_edges=None, bin\_range=None, log=True, fit=False, \*\*kwargs*)

Provide plot as `matplotlib.axes.Axes` object showing the 2-dimensional histogram of variances and localization counts.

### Parameters

- **ax** (Optional[*Axes*]) – The axes on which to show the image
- **bins** (UnionType[*Bins*, *Axis*, *AxesTuple*, None]) – The bin specification as defined in *Bins*
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, *n\_bin\_edges*).
- **bin\_range** (UnionType[tuple[float, float], Sequence[float], Sequence[Sequence[float]], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2).
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, *n\_bins*). 5 would describe *bin\_size* of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **log** (bool) – Flag for plotting on a log scale.
- **fit** (bool) – Flag indicating if distribution fit is shown. The fit will only be computed if *distribution\_statistics* is None.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.pcolormesh()`.

### Returns

Axes object with the plot.

### Return type

matplotlib.axes.Axes

**plot**(*ax=None*, *\*\*kwargs*)

Provide plot as `matplotlib.axes.Axes` object showing the variances as function of localization counts.

### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.plot()`.

### Returns

Axes object with the plot.

### Return type

matplotlib.axes.Axes

## locan.analysis.position\_variance\_expectation.PositionVarianceExpectationResults

**class** locan.analysis.position\_variance\_expectation.PositionVarianceExpectationResults(*values=*  
*grouped*)

Bases: object

### Methods

```
__init__([values, grouped])
```

### Attributes

*values*

*grouped*

**grouped:** DataFrame

**values:** DataFrame



## locan.analysis.ripley

Compute Ripley's k function.

Spatial clustering of localization data is characterized by Ripley's k or related l and h functions<sup>1</sup>.

Ripley's k function is computed for 2D and 3D data for a series of radii as described in<sup>2</sup> in order to provide evidence for deviations from a spatially homogeneous Poisson process (i.e. complete spatial randomness, CSR). Ripley's k function is estimated by summing all points or over test points being a random subset of all points:

$$k(r) = \frac{1}{\lambda(n-1)} \sum_{i=1}^n N_{p_i}(r)$$

here  $p_i$  is the  $i^{th}$  point of n test points,  $N_{p_i}$  is the number of points within the region of radius r around  $p_i$ , and  $\lambda$  is the density of all points.

We follow the definition of l and h functions in<sup>2</sup>. Ripley's l function is:

$$l(r) = \sqrt{k(r)/\pi} \quad \text{in 2D}$$
$$l(r) = \sqrt[3]{\frac{3}{4\pi}k(r)} \quad \text{in 3D}$$

And Ripley's h function is:

$$h(r) = l(r) - r$$

## References

### Classes

|  |        |     |  |
|--|--------|-----|--|
| <code>RipleysHFunction</code> ([meta, region_measure]) | radii, | re- | Compute Ripley's H function for two- or three-dimensional data at the given radii. |
| <code>RipleysKFunction</code> ([meta, region_measure]) | radii, | re- | Compute Ripley's K function for two- or three-dimensional data at the given radii. |
| <code>RipleysLFunction</code> ([meta, region_measure]) | radii, | re- | Compute Ripley's L function for two- or three-dimensional data at the given radii. |

## locan.analysis.ripley.RipleysHFunction

```
class locan.analysis.ripley.RipleysHFunction(meta=None, radii=None,
                                             region_measure='bb')
```

Bases: `_Analysis`

Compute Ripley's H function for two- or three-dimensional data at the given radii.

### Parameters

<sup>1</sup> B.D. Ripley, Modelling spatial patterns. Journal of the Royal Statistical Society, 1977, 172–212.

<sup>2</sup> Kiskowski, M. A., Hancock, J. F., and Kenworthy, A. K., On the use of Ripley's K-function and its derivatives to analyze domain size. Biophysical journal, 2009, 97, 1095–1103.

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata* | *None*) – Metadata about the current analysis routine.
- **radii** (*npt.ArrayLike* | *None*) – Radii at which to compute Ripley’s k function.
- **region\_measure** (*float* | *Literal['bb']*) – Region measure (area or volume) for point region. For ‘bb’ the region measure of the bounding\_box is used.

### Variables

- **count** (*int*) – A counter for counting instantiations.
- **parameter** (*dict[str, Any]*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **results** (*pd.DataFrame* | *None*) – Data frame with radii as provided and Ripley’s H function.
- **Ripley\_h\_maximum** (*pd.DataFrame* | *None*) – Data frame with radius and Ripley’s H value for the radius at which the H function has its maximum.

### Methods

|  |                                       |
|--|---------------------------------------|
| <code>__init__([meta, radii, region_measure])</code> |                                       |
| <code>compute(locdata[, other_locdata])</code>       | Run the computation.                  |
| <code>plot([ax])</code>                              |                                       |
|  | <b>rtype</b><br>Axes                  |
| <code>report(*args, **kwargs)</code>                 | Show a report about analysis results. |

### Attributes

|                               |   |
|-------------------------------|---|
| <code>Ripley_h_maximum</code> |   |
| <code>count</code>            | A counter for counting Analysis class instantiations (class attribute). |

**property Ripley\_h\_maximum:** *DataFrame* | *None*

**compute**(*locdata*, *other\_locdata=None*)

Run the computation.

### Parameters

- **locdata** (*LocData*) – Localization data with 2D or 3D coordinates on which to estimate Ripley’s H function.
- **other\_locdata** (Optional[*LocData*]) – Other localization data from which to estimate Ripley’s H function (e.g. subset of points). For None other\_points is set to points (default).

**Return type**

Self

**count:** int = 0

A counter for counting Analysis class instantiations (class attribute).

**plot**(*ax=None, \*\*kwargs*)

**Return type**

Axes

## locan.analysis.ripley.RipleysKFunction

**class** locan.analysis.ripley.RipleysKFunction(*meta=None, radii=None, region\_measure='bb'*)

Bases: *\_Analysis*

Compute Ripley’s K function for two- or three-dimensional data at the given radii.

**Parameters**

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata* | None) – Metadata about the current analysis routine.
- **radii** (*Iterable[float]* | None) – Radii at which to compute Ripley’s k function.
- **region\_measure** (*float* | *Literal['bb']*) – Region measure (area or volume) for point region. For ‘bb’ the region measure of the bounding\_box is used.

**Variables**

- **count** (*int*) – A counter for counting instantiations.
- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **results** (*pandas.DataFrame*) – Data frame with radii as provided and Ripley’s K function.

## Methods

|  |                                       |
|--|---------------------------------------|
| <code>__init__([meta, radii, region_measure])</code> |                                       |
| <code>compute(locdata[, other_locdata])</code>       | Run the computation.                  |
| <code>plot([ax])</code>                              | <b>rtype</b><br>Axes                  |
| <code>report(*args, **kwargs)</code>                 | Show a report about analysis results. |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*, *other\_locdata=None*)

Run the computation.

### Parameters

- **locdata** (*LocData*) – Localization data with 2D or 3D coordinates on which to estimate Ripley’s K function.
- **other\_locdata** (Optional[*LocData*]) – Other localization data from which to estimate Ripley’s K function (e.g. subset of points). For None other\_points is set to points (default).

### Return type

Self

**count:** `int = 0`

A counter for counting Analysis class instantiations (class attribute).

**plot**(*ax=None*, *\*\*kwargs*)

### Return type

Axes

## locan.analysis.ripley.RipleysLFunction

**class** locan.analysis.ripley.RipleysLFunction(*meta=None*, *radii=None*, *region\_measure='bb'*)

Bases: `_Analysis`

Compute Ripley’s L function for two- or three-dimensional data at the given radii.

### Parameters

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata* | None) – Metadata about the current analysis routine.

- **radii** (*npt.ArrayLike* / *None*) – Radii at which to compute Ripley’s k function.
- **region\_measure** (*float* / *Literal["bb"]*) – Region measure (area or volume) for point region. For ‘bb’ the region measure of the bounding\_box is used.

#### Variables

- **count** (*int*) – A counter for counting instantiations.
- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Meta-data about the current analysis routine.
- **results** (*pandas.DataFrame*) – Data frame with radii as provided and Ripley’s L function.

#### Methods

|  |                                       |
|--|---------------------------------------|
| <code>__init__([meta, radii, region_measure])</code> |                                       |
| <code>compute(locdata[, other_locdata])</code>       | Run the computation.                  |
| <code>plot([ax])</code>                              | <b>rtype</b><br>Axes                  |
| <code>report(*args, **kwargs)</code>                 | Show a report about analysis results. |

#### Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*, *other\_locdata=None*)

Run the computation.

##### Parameters

- **locdata** (*LocData*) – Localization data with 2D or 3D coordinates on which to estimate Ripley’s L function.
- **other\_locdata** (*Optional[LocData]*) – Other localization data from which to estimate Ripley’s L function (e.g. subset of points). For None other\_points is set to points (default).

##### Return type

Self

**count:** `int = 0`

A counter for counting Analysis class instantiations (class attribute).

**plot**(*ax=None, \*\*kwargs*)

**Return type**

Axes

## Functions

|                               |   |
|-------------------------------|---|
| <code>plot(self[, ax])</code> | Provide plot of results as matplotlib.axes.Axes object. |
|-------------------------------|---|

### locan.analysis.ripley.plot

locan.analysis.ripley.**plot**(*self, ax=None, \*\*kwargs*)

Provide plot of results as matplotlib.axes.Axes object.

**Parameters**

- **ax** (Optional[Axes]) – The axes on which to show the image
- **kwargs** (Any) – Other parameters passed to matplotlib.pyplot.plot().

**Returns**

Axes object with the plot.

**Return type**

matplotlib.axes.Axes

### locan.analysis.subpixel\_bias

Compute subpixel bias in localization data.

Subpixel bias in localization coordinates may arise depending on the localization algorithm<sup>1</sup>.

## References

### Classes

|   |  |
|---|--|
| <code>SubpixelBias([meta, pixel_size])</code> | Check for subpixel bias by computing the modulo of localization coordinates for each localization's spatial coordinate in locdata. |
|---|--|

---

<sup>1</sup> Gould, T. J., Verkhusha, V. V. & Hess, S. T., Imaging biological structures with fluorescence photoactivation localization microscopy. Nat. Protoc. 4 (2009), 291–308.

## locan.analysis.subpixel\_bias.SubpixelBias

**class** locan.analysis.subpixel\_bias.SubpixelBias(*meta=None, pixel\_size=None*)

Bases: `_Analysis`

Check for subpixel bias by computing the modulo of localization coordinates for each localization's spatial coordinate in locdata.

### Parameters

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata* | *None*) – Metadata about the current analysis routine.
- **pixel\_size** (*int* | *float* | *Sequence[int | float]*) – Camera pixel size in coordinate units.

### Variables

- **count** (*int*) – A counter for counting instantiations.
- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **results** (*pandas.DataFrame*) – The number of localizations per frame or the number of localizations per frame normalized to `region_measure(hull)`.

## Methods

|   |  |
|---|--|
| <code>__init__([meta, pixel_size])</code> |  |
| <code>compute(locdata)</code>             | Run the computation.   |
| <code>hist([ax, bins, log])</code>        | Provide histogram as <code>matplotlib.axes.Axes</code> object showing <code>hist(results)</code> . |
| <code>report(*args, **kwargs)</code>      | Show a report about analysis results.  |

## Attributes

|                    |   |
|--------------------|---|
| <code>count</code> | A counter for counting Analysis class instantiations (class attribute). |
|--------------------|---|

**compute**(*locdata*)

Run the computation.

### Parameters

**locdata** (*LocData*) – Localization data.

### Return type

Self

**count:** `int = 0`

A counter for counting Analysis class instantiations (class attribute).

**hist**(*ax=None, bins='auto', log=True, \*\*kwargs*)

Provide histogram as `matplotlib.axes.Axes` object showing `hist(results)`. Nan entries are ignored.

#### Parameters

- **ax** (Optional[Axes]) – The axes on which to show the image
- **bins** (int | Sequence[int | float] | str) – Bin specifications (passed to `matplotlib.hist()`).
- **log** (bool) – Flag for plotting on a log scale.
- **kwargs** (Any) – Other parameters passed to `matplotlib.pyplot.hist()`.

#### Returns

Axes object with the plot.

#### Return type

`matplotlib.axes.Axes`

## locan.analysis.uncertainty

Compute localization uncertainty.

A theoretical estimate for localization uncertainty is given by the Cramer-Rao-lower-bound for the localization precision. The localization precision depends on a number of experimental factors including camera and photophysical characteristics<sup>12</sup>. We provide functions to compute localization uncertainty from available localization properties as predicted by the Cramer-Rao-lower-bound for the localization precision.

## References

## Classes

|  |  |
|--|--|
| <i>LocalizationUncertainty</i> ([meta, model]) | Compute the Cramer Rao lower bound for localization uncertainty for each localization's spatial coordinate in locdata. |
|--|--|

<sup>1</sup> K.I. Mortensen, L. S. Churchman, J. A. Spudich, H. Flyvbjerg, Nat. Methods 7 (2010): 377–384.

<sup>2</sup> Rieger B., Stallinga S., The lateral and axial localization uncertainty in super-resolution light microscopy. Chemphyschem 17;15(4), 2014:664-70. doi: 10.1002/cphc.201300711



## locan.analysis.uncertainty.LocalizationUncertainty

```
class locan.analysis.uncertainty.LocalizationUncertainty(meta=None, model=1,
                                                         **kwargs)
```

Bases: `_Analysis`

Compute the Cramer Rao lower bound for localization uncertainty for each localization's spatial coordinate in locdata.

Uncertainty is computed according to one of the following model functions:

- 1)  $1 / \sqrt{\text{intensity}}$
- 2)  $\text{psf\_sigma} / \sqrt{\text{intensity}}$
- 3)  $f(\text{psf\_sigma}, \text{intensity}, \text{pixel\_size}, \text{background})$

Localization properties have to be available for all or each spatial dimension (like *psf\_sigma* or *psf\_sigma\_x*). The localization property *intensity* must have the unit *photons*. The unit of *pixel\_size* must be the same as that of position coordinates.

### Parameters

- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata* | *None*) – Metadata about the current analysis routine.
- **model** (*int*) – Model function for theoretical localization uncertainty.
- **kwargs** (*Any*) – kwargs for the chosen model. If none are given the localization properties are taken from locdata.

### Variables

- **count** (*int*) – A counter for counting instantiations.
- **parameter** (*dict*) – A dictionary with all settings for the current computation.
- **meta** (*locan.analysis.metadata\_analysis\_pb2.AMetadata*) – Metadata about the current analysis routine.
- **results** (*pandas.DataFrame*) – Uncertainty for each localization and in each dimension.

### Methods

|                                      |                                       |
|--------------------------------------|---------------------------------------|
| <code>__init__([meta, model])</code> |                                       |
| <code>compute(locdata)</code>        | Run the computation.                  |
| <code>report(*args, **kwargs)</code> | Show a report about analysis results. |

## Attributes

|              |   |
|--------------|---|
| <i>count</i> | A counter for counting Analysis class instantiations (class attribute). |
|--------------|---|

**compute**(*locdata*)

Run the computation.

### Parameters

**locdata** (*LocData*) – Localization data.

### Return type

Self

**count:** `int = 0`

A counter for counting Analysis class instantiations (class attribute).

## Functions

|  |   |
|--|---|
| <i>localization_precision_model_1</i> (intensity)      | Localization precision as function of intensity (I) according to:   |
| <i>localization_precision_model_2</i> (intensity, ...) | Localization precision as function of intensity (I), PSF width (sigma_PSF) according to:                                  |
| <i>localization_precision_model_3</i> (intensity, ...) | Localization precision as function of intensity (I), PSF size (sigma_{PSF}), pixel size (a), background (b) according to: |

### locan.analysis.uncertainty.localization\_precision\_model\_1

**locan.analysis.uncertainty.localization\_precision\_model\_1**(*intensity*)

Localization precision as function of intensity (I) according to:

$$\sigma_{loc} = \frac{1}{\sqrt{I}}$$

### Parameters

**intensity** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Intensity values

### Return type

npt.NDArray[np.float64]

## locan.analysis.uncertainty.localization\_precision\_model\_2

`locan.analysis.uncertainty.localization_precision_model_2(intensity, psf_sigma)`

Localization precision as function of intensity (I), PSF width (sigma\_PSF) according to:

$$\sigma_{loc} = \frac{\sigma_{PSF}}{\sqrt{I}}$$

### Parameters

- **intensity** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Intensity values
- **psf\_sigma** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The PSF size

### Return type

`npt.NDArray[np.float64]`

## locan.analysis.uncertainty.localization\_precision\_model\_3

`locan.analysis.uncertainty.localization_precision_model_3(intensity, psf_sigma, pixel_size, local_background)`

Localization precision as function of intensity (I), PSF size (sigma\_{PSF}), pixel size (a), background (b) according to:

$$\begin{aligned}\sigma_a^2 &= \sigma_{PSF}^2 + a^2/12 \\ \tau &= 2 * \pi * b * \sigma_a^2 / (I * a^2) \\ \sigma_{loc}^2 &= \sigma_a^2 / I * (1 + 4 * \tau + \sqrt{2 * \tau / (1 + 4 * \tau)})\end{aligned}$$

### Parameters

- **intensity** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Intensity values
- **psf\_sigma** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The PSF size
- **pixel\_size** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Size of camera pixel
- **local\_background** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,

complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The local background

#### Return type

npt.NDArray[np.float64]

## 8.2 locan.constants

Constants used throughout the project.

|                                |  |
|--------------------------------|--|
| <code>ROOT_DIR</code>          | Path subclass for non-Windows systems.                                     |
| <code>PROPERTY_KEYS</code>     | Keys for the most common LocData properties.                               |
| <code>DECODE_KEYS</code>       | Mapping column names in DECODE files to <i>LocData</i> property keys       |
| <code>ELYRA_KEYS</code>        | Mapping column names in Zeiss Elyra files to <i>LocData</i> property keys  |
| <code>NANOIMAGER_KEYS</code>   | Mapping column names in Nanoimager files to <i>LocData</i> property keys   |
| <code>RAPIDSTORM_KEYS</code>   | Mapping column names in RapidSTORM files to <i>LocData</i> property keys   |
| <code>SMAP_KEYS</code>         | Mapping column names in SMAP files to <i>LocData</i> property keys         |
| <code>SMLM_KEYS</code>         | Mapping column names in SMLM files to <i>LocData</i> property keys         |
| <code>THUNDERSTORM_KEYS</code> | Mapping column names in Thunderstorm files to <i>LocData</i> property keys |

### 8.2.1 locan.constants.ROOT\_DIR

`locan.constants.ROOT_DIR: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/locan/envs/latest/lib/python3.11/site-packages/locan')`

Path subclass for non-Windows systems.

On a POSIX system, instantiating a Path should return this object.

### 8.2.2 locan.constants.PROPERTY\_KEYS

```
locan.constants.PROPERTY_KEYS = {'channel': 'integer', 'chi_square':
'float', 'cluster_label': 'integer', 'frame': 'integer', 'frames_missing':
'integer', 'frames_number': 'integer', 'index': 'integer', 'intensity':
'float', 'local_background': 'float', 'local_background_sigma': 'float',
'original_index': 'integer', 'plane': 'integer', 'position_x': 'float',
'position_y': 'float', 'position_z': 'float', 'psf_amplitude': 'float',
'psf_half_width': 'float', 'psf_half_width_x': 'float', 'psf_half_width_y':
'float', 'psf_half_width_z': 'float', 'psf_sigma': 'float', 'psf_sigma_x':
'float', 'psf_sigma_y': 'float', 'psf_sigma_z': 'float', 'psf_width':
'float', 'psf_width_x': 'float', 'psf_width_y': 'float', 'psf_width_z':
'float', 'signal_background_ratio': 'float', 'signal_noise_ratio': 'float',
'slice_z': 'float', 'time': 'float', 'two_kernel_improvement': 'float',
'uncertainty': 'float', 'uncertainty_x': 'float', 'uncertainty_y': 'float',
'uncertainty_z': 'float'}
```

Keys for the most common LocData properties. Values suggest a type for conversion. If ‘integer’, ‘signed’, ‘unsigned’, ‘float’ `pandas.to_numeric()` can be applied. Otherwise `pandas.astype()` can be applied.

### 8.2.3 locan.constants.DECODE\_KEYS

```
locan.constants.DECODE_KEYS: Final[dict[str, str]] = {'bg':
'local_background', 'frame_ix': 'frame', 'id': 'original_index', 'phot':
'intensity', 'x': 'position_x', 'y': 'position_y', 'z': 'position_z'}
```

Mapping column names in DECODE files to *LocData* property keys

### 8.2.4 locan.constants.ELYRA\_KEYS

```
locan.constants.ELYRA_KEYS: Final[dict[str, str]] = {'Background variance':
'local_background_sigma', 'Channel': 'channel', 'Chi square': 'chi_square',
'First Frame': 'frame', 'Frames Missing': 'frames_missing', 'Index':
'original_index', 'Number Frames': 'frames_number', 'Number Photons':
'intensity', 'PSF half width [nm]': 'psf_half_width', 'PSF width [nm]':
'psf_width', 'Position X [nm]': 'position_x', 'Position Y [nm]':
'position_y', 'Position Z [nm]': 'position_z', 'Precision [nm]':
'uncertainty', 'Z Slice': 'slice_z'}
```

Mapping column names in Zeiss Elyra files to *LocData* property keys

### 8.2.5 locan.constants.NANOIMAGER\_KEYS

```
locan.constants.NANOIMAGER_KEYS: Final[dict[str, str]] = {'Background':
'local_background', 'Channel': 'channel', 'Frame': 'frame', 'Photons':
'intensity', 'X (nm)': 'position_x', 'Y (nm)': 'position_y', 'Z (nm)':
'position_z'}
```

Mapping column names in Nanoimager files to *LocData* property keys

### 8.2.6 locan.constants.RAPIDSTORM\_KEYS

```
locan.constants.RAPIDSTORM_KEYS: Final[dict[str, str]] = {'Amplitude-0-0':  
'intensity', 'FitResidues-0-0': 'chi_square', 'ImageNumber-0-0': 'frame',  
'LocalBackground-0-0': 'local_background', 'Position-0-0': 'position_x',  
'Position-0-0-uncertainty': 'uncertainty_x', 'Position-1-0': 'position_y',  
'Position-1-0-uncertainty': 'uncertainty_y', 'Position-2-0': 'position_z',  
'Position-2-0-uncertainty': 'uncertainty_z', 'TwoKernelImprovement-0-0':  
'two_kernel_improvement'}
```

Mapping column names in RapidSTORM files to *LocData* property keys

### 8.2.7 locan.constants.SMAP\_KEYS

```
locan.constants.SMAP_KEYS: Final[dict[str, str]] = {'bg': 'local_background',  
'channel': 'channel', 'frame': 'frame', 'phot': 'intensity', 'xnm':  
'position_x', 'xmerr': 'uncertainty_x', 'ynm': 'position_y', 'ymerr':  
'uncertainty_y', 'znm': 'position_z', 'zmerr': 'uncertainty_z'}
```

Mapping column names in SMAP files to *LocData* property keys

### 8.2.8 locan.constants.SMLM\_KEYS

```
locan.constants.SMLM_KEYS: Final[dict[str, str]] = {'Amplitude_0_0':  
'intensity', 'FitResidues_0_0': 'chi_square', 'LocalBackground_0_0':  
'local_background', 'background': 'local_background', 'frame': 'frame',  
'id': 'original_index', 'intensity': 'intensity', 'uncertainty [nm]':  
'uncertainty', 'uncertainty_xy [nm]': 'uncertainty_x', 'uncertainty_z [nm]':  
'uncertainty_z', 'x': 'position_x', 'x_position': 'position_x', 'y':  
'position_y', 'y_position': 'position_y', 'z': 'position_z', 'z_position':  
'position_z'}
```

Mapping column names in SMLM files to *LocData* property keys

### 8.2.9 locan.constants.THUNDERSTORM\_KEYS

```
locan.constants.THUNDERSTORM_KEYS: Final[dict[str, str]] = {'bkgstd [photon]':  
'local_background_sigma', 'chi2': 'chi_square', 'detections':  
'frames_number', 'frame': 'frame', 'id': 'original_index', 'intensity  
[photon]': 'intensity', 'offset [photon]': 'local_background', 'sigma [nm]':  
'psf_sigma', 'sigma1 [nm]': 'psf_sigma_x', 'sigma2 [nm]': 'psf_sigma_y',  
'uncertainty [nm]': 'uncertainty', 'uncertainty_xy [nm]': 'uncertainty_x',  
'uncertainty_z [nm]': 'uncertainty_z', 'x [nm]': 'position_x', 'y [nm]':  
'position_y', 'z [nm]': 'position_z'}
```

Mapping column names in Thunderstorm files to *LocData* property keys

## Classes

|  |   |
|--|---|
| <code>FileType(value[, names, module, qualname, ...])</code> | File types for localization files.  |
| <code>HullType(value[, names, module, qualname, ...])</code> | Hull definitions that are supported for <i>LocData</i> objects.   |
| <code>PropertyDescription(name, type[, unit_SI, ...])</code> | A property of a localization or group of localizations representing column names in <i>LocData.data</i> and <i>LocData.properties</i> . |
| <code>PropertyKey(value[, names, module, ...])</code>        | Property descriptions for standard properties used in <i>locan.LocData</i> and throughout <i>locan</i> .                                |
| <code>RenderEngine(value[, names, module, ...])</code>       | Engine to be used for rendering and displaying localization data as 2d or 3d images.  |

### 8.2.10 locan.constants.FileType

```
class locan.constants.FileType(value, names=None, *, module=None, qualname=None,
                               type=None, start=1, boundary=None)
```

Bases: Enum

File types for localization files.

The listed file types are supported with input/output functions in `io.io_locdata()`. The types correspond to the metadata keys for *LocData* objects. That is, they are equal to the file types in the protobuf message *locan.data.metadata\_pb2.Metadata*.

## Attributes

|                          |
|--------------------------|
| <i>UNKNOWN_FILE_TYPE</i> |
| <i>CUSTOM</i>            |
| <i>RAPIDSTORM</i>        |
| <i>ELYRA</i>             |
| <i>THUNDERSTORM</i>      |
| <i>ASDF</i>              |
| <i>NANOIMAGER</i>        |
| <i>RAPIDSTORMTRACK</i>   |
| <i>SMLM</i>              |
| <i>DECODE</i>            |
| <i>SMAP</i>              |

**ASDF = 5**

**CUSTOM = 1**

**DECODE = 9**

**ELYRA = 3**

**NANOIMAGER = 6**

**RAPIDSTORM = 2**

**RAPIDSTORMTRACK = 7**

**SMAP = 10**

**SMLM = 8**

**THUNDERSTORM = 4**

**UNKNOWN\_FILE\_TYPE = 0**



### 8.2.11 locan.constants.HullType

```
class locan.constants.HullType(value, names=None, *, module=None, qualname=None,
                               type=None, start=1, boundary=None)
```

Bases: Enum

Hull definitions that are supported for *LocData* objects.

#### Attributes

|                              |
|------------------------------|
| <i>BOUNDING_BOX</i>          |
| <i>CONVEX_HULL</i>           |
| <i>ORIENTED_BOUNDING_BOX</i> |
| <i>ALPHA_SHAPE</i>           |

**ALPHA\_SHAPE** = 'alpha\_shape'

**BOUNDING\_BOX** = 'bounding\_box'

**CONVEX\_HULL** = 'convex\_hull'

**ORIENTED\_BOUNDING\_BOX** = 'oriented\_bounding\_box'

### 8.2.12 locan.constants.PropertyDescription

```
class locan.constants.PropertyDescription(name, type, unit_SI=None, unit=None,
                                          description="")
```

Bases: object

A property of a localization or group of localizations representing column names in *LocData.data* and *LocData.properties*.

#### Variables

- **name** – property name
- **type** – property type
- **unit\_SI** – SI unit that is appropriate for property
- **unit** – The actual unit currently used
- **description** – Explanation what the property represents

## Methods

```
__init__(name, type[, unit_SI, unit, ...])
```

## Attributes

```
description
```

```
unit
```

```
unit_SI
```

```
name
```

```
type
```

```
description: str = ''
```

```
name: str
```

```
type: str
```

```
unit: Optional[str] = None
```

```
unit_SI: Optional[str] = None
```

### 8.2.13 locan.constants.PropertyKey

```
class locan.constants.PropertyKey(value, names=None, *, module=None, qualname=None,  
                                   type=None, start=1, boundary=None)
```

Bases: Enum

Property descriptions for standard properties used in *locan.LocData* and throughout locan.

## Methods

|                                 |  |
|---------------------------------|--|
| <i>coordinate_properties()</i>  | Property descriptions for properties representing spatial coordinates                  |
| <i>coordinate_keys()</i>        | Property keys for properties representing spatial coordinates                          |
| <i>uncertainty_properties()</i> | Property descriptions for properties representing spatial coordinate uncertainties     |
| <i>uncertainty_keys()</i>       | Property keys for properties representing spatial coordinate uncertainties.            |
| <i>intensity_properties()</i>   | Property descriptions for properties representing photon intensities.                  |
| <i>intensity_keys()</i>         | Property keys for properties representing photon intensities.                          |
| <i>summary()</i>                | A formatted string representation of PropertyKey showing all elements with attributes. |

## Attributes

|                                |
|--------------------------------|
| <i>index</i>                   |
| <i>original_index</i>          |
| <i>position_x</i>              |
| <i>position_y</i>              |
| <i>position_z</i>              |
| <i>frame</i>                   |
| <i>frames_number</i>           |
| <i>frames_missing</i>          |
| <i>time</i>                    |
| <i>intensity</i>               |
| <i>local_background</i>        |
| <i>local_background_sigma</i>  |
| <i>signal_noise_ratio</i>      |
| <i>signal_background_ratio</i> |

continues on next page

Table 1 – continued from previous page

|                               |
|-------------------------------|
| <i>chi_square</i>             |
| <i>two_kernel_improvement</i> |
| <i>psf_amplitude</i>          |
| <i>psf_width</i>              |
| <i>psf_width_x</i>            |
| <i>psf_width_y</i>            |
| <i>psf_width_z</i>            |
| <i>psf_half_width</i>         |
| <i>psf_half_width_x</i>       |
| <i>psf_half_width_y</i>       |
| <i>psf_half_width_z</i>       |
| <i>psf_sigma</i>              |
| <i>psf_sigma_x</i>            |
| <i>psf_sigma_y</i>            |
| <i>psf_sigma_z</i>            |
| <i>uncertainty</i>            |
| <i>uncertainty_x</i>          |
| <i>uncertainty_y</i>          |
| <i>uncertainty_z</i>          |
| <i>channel</i>                |
| <i>slice_z</i>                |
| <i>plane</i>                  |
| <i>cluster_label</i>          |

```
channel = PropertyDescription(name='channel', type='integer',
unit_SI=None, unit=None, description='identifier for an imaging channel')
```

```
chi_square = PropertyDescription(name='chi_square', type='float',
unit_SI=None, unit=None, description='chi-square value of the fitting
procedure as estimated by the fitter')
```

```
cluster_label = PropertyDescription(name='cluster_label', type='integer',
unit_SI=None, unit=None, description='identifier for a localization
cluster')
```

```
classmethod coordinate_keys()
```

Property keys for properties representing spatial coordinates

**Return type**

list[str]

```
classmethod coordinate_properties()
```

Property descriptions for properties representing spatial coordinates

**Return type**

list[[PropertyKey](#)]

```
frame = PropertyDescription(name='frame', type='integer', unit_SI=None,
unit=None, description='frame number in which the localization occurs')
```

```
frames_missing = PropertyDescription(name='frames_missing',
type='integer', unit_SI=None, unit=None, description='number of frames
that occurred between two successive localizations')
```

```
frames_number = PropertyDescription(name='frames_number', type='integer',
unit_SI=None, unit=None, description='number of frames that contribute to
a merged localization')
```

```
index = PropertyDescription(name='index', type='integer', unit_SI=None,
unit=None, description='a positive integer identifying the localization')
```

```
intensity = PropertyDescription(name='intensity', type='float',
unit_SI=None, unit=None, description='intensity or emission strength as
estimated by the fitter')
```

```
classmethod intensity_keys()
```

Property keys for properties representing photon intensities.

**Return type**

list[str]

```
classmethod intensity_properties()
```

Property descriptions for properties representing photon intensities.

**Return type**

list[[PropertyKey](#)]

```
local_background = PropertyDescription(name='local_background',
type='float', unit_SI=None, unit=None, description='background in the
neighborhood of localization as estimated by the fitter')
```

```
local_background_sigma =
PropertyDescription(name='local_background_sigma', type='float',
unit_SI=None, unit=None, description='variation of local background in
terms of standard deviation')

original_index = PropertyDescription(name='original_index',
type='integer', unit_SI=None, unit=None, description='')

plane = PropertyDescription(name='plane', type='integer', unit_SI=None,
unit=None, description='identifier for an imaging plane')

position_x = PropertyDescription(name='position_x', type='float',
unit_SI='m', unit=None, description='spatial coordinate.')

position_y = PropertyDescription(name='position_y', type='float',
unit_SI='m', unit=None, description='spatial coordinate.')

position_z = PropertyDescription(name='position_z', type='float',
unit_SI='m', unit=None, description='spatial coordinate.')

psf_amplitude = PropertyDescription(name='psf_amplitude', type='float',
unit_SI=None, unit=None, description='')

psf_half_width = PropertyDescription(name='psf_half_width', type='float',
unit_SI=None, unit=None, description='')

psf_half_width_x = PropertyDescription(name='psf_half_width_x',
type='float', unit_SI=None, unit=None, description='')

psf_half_width_y = PropertyDescription(name='psf_half_width_y',
type='float', unit_SI=None, unit=None, description='')

psf_half_width_z = PropertyDescription(name='psf_half_width_z',
type='float', unit_SI=None, unit=None, description='')

psf_sigma = PropertyDescription(name='psf_sigma', type='float',
unit_SI=None, unit=None, description='sigma of the fitted Gauss-function -
being isotropic or representing the root-mean-square of psf_sigma_c for
all dimensions')

psf_sigma_x = PropertyDescription(name='psf_sigma_x', type='float',
unit_SI=None, unit=None, description='sigma of the fitted Gauss-function
in x-dimension as estimated by the fitter')

psf_sigma_y = PropertyDescription(name='psf_sigma_y', type='float',
unit_SI=None, unit=None, description='sigma of the fitted Gauss-function
in y-dimension as estimated by the fitter')

psf_sigma_z = PropertyDescription(name='psf_sigma_z', type='float',
unit_SI=None, unit=None, description='sigma of the fitted Gauss-function
in z-dimension as estimated by the fitter')
```

```
psf_width = PropertyDescription(name='psf_width', type='float',
unit_SI=None, unit=None, description='full-width-half-max of the fitted
Gauss-function - being isotropic or representing the root-mean-square of
psf_width_c for all dimensions')
```

```
psf_width_x = PropertyDescription(name='psf_width_x', type='float',
unit_SI=None, unit=None, description='full-width-half-max of the fitted
Gauss-function in x-dimension as estimated by the fitter')
```

```
psf_width_y = PropertyDescription(name='psf_width_y', type='float',
unit_SI=None, unit=None, description='full-width-half-max of the fitted
Gauss-function in y-dimension as estimated by the fitter')
```

```
psf_width_z = PropertyDescription(name='psf_width_z', type='float',
unit_SI=None, unit=None, description='full-width-half-max of the fitted
Gauss-function in z-dimension as estimated by the fitter')
```

```
signal_background_ratio =
PropertyDescription(name='signal_background_ratio', type='float',
unit_SI=None, unit=None, description='ratio between mean intensity (i.e.
intensity for a single localization) and the local_background')
```

```
signal_noise_ratio = PropertyDescription(name='signal_noise_ratio',
type='float', unit_SI=None, unit=None, description='ratio between mean
intensity (i.e. intensity for a single localization) and the standard
deviation of local_background (i.e. local_background_sigma for a single
localization)')
```

```
slice_z = PropertyDescription(name='slice_z', type='float', unit_SI=None,
unit=None, description='identifier for an imaging slice')
```

```
classmethod summary()
```

A formatted string representation of PropertyKey showing all elements with attributes.

Return type  
str

```
time = PropertyDescription(name='time', type='float', unit_SI='s',
unit=None, description='')
```

```
two_kernel_improvement =
PropertyDescription(name='two_kernel_improvement', type='float',
unit_SI=None, unit=None, description='a rapidSTORM parameter describing
the improvement from two kernel fitting')
```

```
uncertainty = PropertyDescription(name='uncertainty', type='float',
unit_SI=None, unit=None, description='localization error for all
dimensions or representing a value proportional to psf_sigma /
sqrt(intensity) or representing the root-mean-square of uncertainty_c for
all dimensions.')
```

```
classmethod uncertainty_keys()
```

Property keys for properties representing spatial coordinate uncertainties.

**Return type**

list[str]

**classmethod uncertainty\_properties()**

Property descriptions for properties representing spatial coordinate uncertainties

**Return type**list[*PropertyKey*]

```
uncertainty_x = PropertyDescription(name='uncertainty_x', type='float',
unit_SI=None, unit=None, description='localization error in x-dimension
estimated by a fitter or representing a value proportional to psf_sigma_x
/ sqrt(intensity)')
```

```
uncertainty_y = PropertyDescription(name='uncertainty_y', type='float',
unit_SI=None, unit=None, description='localization error in y-dimension
estimated by a fitter or representing a value proportional to psf_sigma_y
/ sqrt(intensity)')
```

```
uncertainty_z = PropertyDescription(name='uncertainty_z', type='float',
unit_SI=None, unit=None, description='localization error in z-dimension
estimated by a fitter or representing a value proportional to psf_sigma_z
/ sqrt(intensity)')
```

## 8.2.14 locan.constants.RenderEngine

```
class locan.constants.RenderEngine(value, names=None, *, module=None, qualname=None,
type=None, start=1, boundary=None)
```

Bases: Enum

Engine to be used for rendering and displaying localization data as 2d or 3d images.

Each engine represents a library to be used as backend for rendering and plotting.

**Attributes**

|                            |                     |
|----------------------------|---------------------|
| <i>MPL</i>                 | matplotlib          |
| <i>MPL_SCATTER_DENSITY</i> | mpl-scatter-density |
| <i>NAPARI</i>              | napari              |

**MPL = 0**

matplotlib

**MPL\_SCATTER\_DENSITY = 1**

mpl-scatter-density

**NAPARI = 2**

napari



## 8.3 locan.configuration

Configuration variables used throughout the project.

|                                |   |
|--------------------------------|---|
| <code>DATASETS_DIR</code>      | Standard directory for example datasets.  |
| <code>RENDER_ENGINE</code>     | Render engine.  |
| <code>N_JOBS</code>            | The number of cores that are used in parallel for some algorithms.                |
| <code>COLORMAP_DEFAULTS</code> | Mapping a type of colormap to its default colormap that is used throughout locan. |
| <code>TQDM_LEAVE</code>        | Leave tqdm progress bars after finishing the iteration.                           |
| <code>TQDM_DISABLE</code>      | Disable tqdm progress bars.   |

### 8.3.1 locan.configuration.DATASETS\_DIR

```
locan.configuration.DATASETS_DIR = PosixPath('/home/docs/LocanDatasets')
```

Standard directory for example datasets.

### 8.3.2 locan.configuration.RENDER\_ENGINE

```
locan.configuration.RENDER_ENGINE = RenderEngine.MPL
```

Render engine.

### 8.3.3 locan.configuration.N\_JOBS

```
locan.configuration.N_JOBS: int = 1
```

The number of cores that are used in parallel for some algorithms. Following the scikit convention: `n_jobs` is the number of parallel jobs to run. If -1, then the number of jobs is set to the number of CPU cores.

### 8.3.4 locan.configuration.COLORMAP\_DEFAULTS

```
locan.configuration.COLORMAP_DEFAULTS: Mapping[str, str] = {'CATEGORICAL':
'cet_glasbey_dark', 'CONTINUOUS': 'cet_fire', 'CONTINUOUS_GRAY': 'cet_gray',
'CONTINUOUS_GRAY_REVERSE': 'cet_gray_r', 'CONTINUOUS_REVERSE': 'cet_fire_r',
'DIVERGING': 'cet_coolwarm', 'TURBO': 'turbo'}
```

Mapping a type of colormap to its default colormap that is used throughout locan. See details in locan documentation on colormaps.

### 8.3.5 locan.configuration.TQDM\_LEAVE

`locan.configuration.TQDM_LEAVE: bool = True`

Leave tqdm progress bars after finishing the iteration. Flag to leave tqdm progress bars.

### 8.3.6 locan.configuration.TQDM\_DISABLE

`locan.configuration.TQDM_DISABLE: bool = False`

Disable tqdm progress bars. Flag to disable all tqdm progress bars.

## 8.4 locan.data

Localization data.

This module contains classes and functions to deal with single-molecule localization data. All functions provide or modify LocData objects.

### 8.4.1 Submodules:

|                       |  |
|-----------------------|--|
| <i>locdata</i>        | A class to carry localization data.                          |
| <i>properties</i>     | Compute additional properties for locdata objects.           |
| <i>hulls</i>          | Hull objects of localization data.                           |
| <i>region</i>         | Regions as support for localization data.                    |
| <i>region_utils</i>   | Utility functions for working with regions.                  |
| <i>register</i>       | Register localization data.                                  |
| <i>aggregate</i>      | Aggregate localization data.                                 |
| <i>filter</i>         | Filter localization data.                                    |
| <i>transform</i>      | Transform localization data.                                 |
| <i>cluster</i>        | This module provides functions for clustering localizations. |
| <i>tracking</i>       | Track localizations.   |
| <i>metadata_utils</i> | Deal with metadata in LocData objects.                       |
| <i>validation</i>     | Validate localization data.                                  |

### locan.data.locdata

A class to carry localization data.

## Classes

|  |  |
|--|--|
| <code>LocData([references, dataframe, indices, meta])</code> | This class carries localization data, aggregated properties and meta data. |
|--|--|

### locan.data.locdata.LocData

```
class locan.data.locdata.LocData(references=None, dataframe=None, indices=None, meta=None)
```

Bases: object

This class carries localization data, aggregated properties and meta data.

Data consist of individual elements being either localizations or other *LocData* objects. Both, localizations and *Locdata* objects have properties. Properties come from the original data or are added by analysis procedures.

#### Parameters

- **references** – A *LocData* reference or an array with references to *LocData* objects referring to the selected localizations in dataset.
- **dataframe** – Dataframe with localization data.
- **indices** – Indices for dataframe in references that makes up the data. *indices* refers to index label, not position.
- **meta** – Metadata about the current dataset and its history.

#### Variables

- **references** (*LocData* | *list[LocData]* | *None*) – A *LocData* reference or an array with references to *LocData* objects referring to the selected localizations in dataframe.
- **dataframe** (*pandas.DataFrame*) – Dataframe with localization data.
- **indices** (*slice* | *list[int]* | *None*) – Indices for dataframe in references that makes up the data.
- **meta** (*locan.data.metadata\_pb2.Metadata*) – Metadata about the current dataset and its history.
- **properties** (*dict[str, Any]*) – List of properties generated from data.
- **coordinate\_keys** (*list[str]*) – The available coordinate properties.
- **uncertainty\_keys** (*list[str]*) – The available uncertainty properties.
- **dimension** (*int*) – Number of coordinates available for each localization (i.e. size of *coordinate\_keys*).

## Methods

|   |   |
|---|---|
| <code>__init__([references, dataframe, indices, meta])</code> |   |
| <code>concat(locdatas[, meta])</code>                         | Concatenate LocData objects.  |
| <code>from_chunks(locdata[, chunks, chunk_size, ...])</code>  | Divide locdata in chunks of localization elements.  |
| <code>from_collection(locdatas[, meta])</code>                | Create new LocData object by collecting LocData objects.  |
| <code>from_coordinates([coordinates, ...])</code>             | Create new LocData object from a sequence of localization coordinates.  |
| <code>from_dataframe([dataframe, meta])</code>                | Create new LocData object from DataFrame with localization data.  |
| <code>from_selection(locdata[, indices, meta])</code>         | Create new LocData object from selected elements in another <i>LocData</i> .  |
| <code>print_meta()</code>                                     | Print Locdata.metadata.   |
| <code>print_summary()</code>                                  | Print a summary containing the most common metadata keys.   |
| <code>projection(coordinate_labels)</code>                    | Reduce dimensions by projecting all localization coordinates onto selected coordinates.                               |
| <code>reduce([reset_index])</code>                            | Clean up references.  |
| <code>reset([reset_index])</code>                             | Reset hulls and properties.   |
| <code>update(dataframe[, reset_index, meta])</code>           | Update the dataframe attribute in place.  |
| <code>update_alpha_shape(alpha)</code>                        | Compute the alpha shape for specific <i>alpha</i> and update <i>self.alpha_shape</i> .                                |
| <code>update_alpha_shape_in_references(alpha)</code>          | Compute the alpha shape for each element in <i>locdata.references</i> and update <i>locdata.dataframe</i> .           |
| <code>update_convex_hulls_in_references()</code>              | Compute the convex hull for each element in <i>locdata.references</i> and update <i>locdata.dataframe</i> .           |
| <code>update_inertia_moments_in_references</code>             | Compute inertia_moments for each element in <i>locdata.references</i> and update <i>locdata.dataframe</i> .           |
| <code>update_oriented_bounding_box_in_refe</code>             | Compute the oriented bounding box for each element in <i>locdata.references</i> and update <i>locdata.dataframe</i> . |
| <code>update_properties_in_references([prop</code>            | Add properties for each element in <i>self.references</i> and update <i>self.dataframe</i> .                          |

## Attributes

|                              |  |
|------------------------------|--|
| <i>alpha_shape</i>           | Return an object representing the alpha-shape of all localizations.                                |
| <i>bounding_box</i>          | Return an object representing the axis-aligned minimal bounding box.                               |
| <i>centroid</i>              | Return coordinate values of the centroid (being the property values for all coordinate labels).    |
| <i>convex_hull</i>           | Return an object representing the convex hull of all localizations.                                |
| <i>coordinate_keys</i>       |  |
| <i>coordinates</i>           | Return all coordinate values.  |
| <i>count</i>                 | A counter for counting LocData instantiations (class attribute).                                   |
| <i>data</i>                  | Return all elements either copied from the reference or referencing the current dataframe.         |
| <i>inertia_moments</i>       | Inertia moments are returned as computed by <code>locan.data.properties.inertia_moments()</code> . |
| <i>oriented_bounding_box</i> | Return an object representing the oriented minimal bounding box.                                   |
| <i>region</i>                | Return the region that supports all localizations.   |
| <i>uncertainty_keys</i>      |  |

**property `alpha_shape`:** *AlphaShape* | None

Return an object representing the alpha-shape of all localizations.

### Type

Hull object

**property `bounding_box`:** *BoundingBox* | None

Return an object representing the axis-aligned minimal bounding box.

### Type

Hull object

**property `centroid`:** `ndarray[Any, dtype[float64]]`

Return coordinate values of the centroid (being the property values for all coordinate labels).

### Type

`npt.NDArray[np.float64]`

**classmethod `concat`** (*locdatas*, *meta=None*)

Concatenate LocData objects.

### Parameters

- **`locdatas`** (`Iterable[LocData]`) – Locdata objects to concatenate.

- **meta** (UnionType[Metadata, dict[str, Any], str, bytes, PathLike[Any], BinaryIO, None]) – Metadata about the current dataset and its history.

#### Returns

A new LocData instance with dataframe representing the concatenated data.

#### Return type

*LocData*

**property convex\_hull:** *ConvexHull* | None

Return an object representing the convex hull of all localizations.

#### Type

Hull object

**property coordinate\_keys:** list[str]

**property coordinates:** ndarray[Any, dtype[float64]]

Return all coordinate values.

#### Type

npt.NDArray[float]

**count = 0**

A counter for counting LocData instantiations (class attribute).

#### Type

int

**property data:** DataFrame

Return all elements either copied from the reference or referencing the current dataframe.

#### Type

pandas.DataFrame

**classmethod from\_chunks**(*locdata*, *chunks=None*, *chunk\_size=None*, *n\_chunks=None*, *order='successive'*, *drop=False*, *meta=None*)

Divide locdata in chunks of localization elements.

#### Parameters

- **locdata** (*LocData*) – Locdata to divide.
- **chunks** (Optional[Sequence[tuple[int, ...]]]) – Localization chunks as defined by a list of index-tuples. One of *chunks*, *chunk\_size* or *n\_chunks* must be different from None.
- **chunk\_size** (Optional[int]) – Number of consecutive localizations to form a single chunk of data. One of *chunks*, *chunk\_size* or *n\_chunks* must be different from None.
- **n\_chunks** (Optional[int]) – Number of chunks. One of *chunks*, *chunk\_size* or *n\_chunks* must be different from None.
- **order** (Literal['successive', 'alternating']) – The order in which to select localizations. One of ‘successive’ or ‘alternating’.
- **drop** (bool) – If True the last chunk will be eliminated if it has fewer localizations than the other chunks.

- **meta** (UnionType[Metadata, dict[str, Any], str, bytes, PathLike[Any], BinaryIO, None]) – Metadata about the current dataset and its history.

#### Returns

A new LocData instance with references and dataframe elements representing the individual chunks.

#### Return type

*LocData*

**classmethod from\_collection**(*locdatas*, *meta=None*)

Create new LocData object by collecting LocData objects.

#### Parameters

- **locdatas** (Iterable[*LocData*]) – Locdata objects to collect.
- **meta** (UnionType[Metadata, dict[str, Any], str, bytes, PathLike[Any], BinaryIO, None]) – Metadata about the current dataset and its history.

#### Returns

A new LocData instance with dataframe representing the concatenated data.

#### Return type

*LocData*

**classmethod from\_coordinates**(*coordinates=None*, *coordinate\_labels=None*, *meta=None*)

Create new LocData object from a sequence of localization coordinates.

#### Parameters

- **coordinates** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], None]) – Sequence of tuples with localization coordinates with shape (n\_loclizations, dimension)
- **coordinate\_labels** (Optional[Sequence[str]]) – The available coordinate properties.
- **meta** (UnionType[Metadata, dict[str, Any], str, bytes, PathLike[Any], BinaryIO, None]) – Metadata about the current dataset and its history.

#### Returns

A new LocData instance with dataframe representing the oncatenated data.

#### Return type

*LocData*

**classmethod from\_dataframe**(*dataframe=None*, *meta=None*)

Create new LocData object from DataFrame with localization data.

#### Parameters

- **dataframe** (Optional[DataFrame]) – Localization data.

- **meta** (UnionType[Metadata, dict[str, Any], str, bytes, PathLike[Any], BinaryIO, None]) – Metadata about the current dataset and its history.

#### Returns

A new LocData instance with dataframe representing the concatenated data.

#### Return type

*LocData*

**classmethod** `from_selection(locdata, indices=None, meta=None)`

Create new LocData object from selected elements in another *LocData*.

#### Parameters

- **locdata** – Locdata object from which to select elements.
- **indices** – Index labels for elements in locdata that make up the new data. Note that contrary to usual python slices, both the start and the stop are included (see pandas documentation). *Indices* refer to index value not position in list.
- **meta** – Metadata about the current dataset and its history.

#### Returns

A new LocData instance with dataframe representing the selected data.

#### Return type

*LocData*

---

**Note:** No error is raised if indices do not exist in locdata.

---

**property** `inertia_moments: InertiaMoments | None`

Inertia moments are returned as computed by `locan.data.properties.inertia_moments()`.

**property** `oriented_bounding_box: OrientedBoundingBox | None`

Return an object representing the oriented minimal bounding box.

#### Type

Hull object

**print\_meta()**

Print Locdata.metadata. :rtype: None

#### See also:

*locan.data.metadata\_utils.metadata\_to\_formatted\_string()*

**print\_summary()**

Print a summary containing the most common metadata keys.

#### Return type

None

**projection**(*coordinate\_labels*)

Reduce dimensions by projecting all localization coordinates onto selected coordinates.



#### Parameters

**coordinate\_labels** (str | list[str]) – The coordinate labels to project onto.

#### Return type

*LocData*

#### Returns

LocData

**reduce**(*reset\_index=False*)

Clean up references.

This includes to update *Locdata.dataframe* and set *LocData.references* and *LocData.indices* to None.

#### Parameters

**reset\_index** (bool) – Flag indicating if the index is reset to integer values. If True the previous index values are discarded.

#### Returns

The modified object

#### Return type

Self

**property region:** *Region* | None

Return the region that supports all localizations.

#### Type

RoiRegion object

**reset**(*reset\_index=False*)

Reset hulls and properties. This is needed after the dataframe attribute has been modified in place.

---

**Note:** Should be used with care because metadata is not updated accordingly. The region property is not changed. Better to just re-instantiate with *LocData.from\_dataframe()* or use *locdata.update()*.

---

#### Parameters

**reset\_index** (bool) – Flag indicating if the index is reset to integer values. If True the previous index values are discarded.

#### Returns

The modified object

#### Return type

Self

**property uncertainty\_keys:** list[str]

**update**(*dataframe, reset\_index=False, meta=None*)

Update the dataframe attribute in place.

Use this function rather than setting `locdata.dataframe` directly in order to automatically update the attributes for dimension, hulls, properties, and metadata.

#### Parameters

- **dataframe** (Optional[DataFrame]) – Dataframe with localization data.
- **reset\_index** (bool) – Flag indicating if the index is reset to integer values. If True the previous index values are discarded.
- **meta** (`locan.data.metadata_pb2.Metadata` | `dict` | `str` | `bytes` | `os.PathLike` | `BinaryIO` | `None`) – Metadata about the current dataset and its history.

#### Returns

The modified object

#### Return type

Self

#### `update_alpha_shape(alpha)`

Compute the alpha shape for specific *alpha* and update *self.alpha\_shape*.

#### Parameters

**alpha** (float) – Alpha parameter specifying a unique alpha complex.

#### Returns

The modified object

#### Return type

Self

#### `update_alpha_shape_in_references(alpha)`

Compute the alpha shape for each element in *locdata.references* and update *locdata.dataframe*.

#### Parameters

**alpha** (float) – Alpha parameter specifying a unique alpha complex.

#### Returns

The modified object

#### Return type

Self

#### `update_convex_hulls_in_references()`

Compute the convex hull for each element in *locdata.references* and update *locdata.dataframe*.

#### Returns

The modified object

#### Return type

Self

#### `update_inertia_moments_in_references()`

Compute *inertia\_moments* for each element in *locdata.references* and update *locdata.dataframe*.

#### Returns

The modified object

**Return type**

Self

**update\_oriented\_bounding\_box\_in\_references()**

Compute the oriented bounding box for each element in `locdata.references` and update `locdata.dataframe`.

**Returns**

The modified object

**Return type**

Self

**update\_properties\_in\_references(*properties=None*)**

Add properties for each element in `self.references` and update `self.dataframe`.

**Parameters**

**properties** – new property values for each reference or function to compute property for `LocData` object.

**Return type**

Self

**locan.data.properties**

Compute additional properties for `locdata` objects.

These functions take `locdata` as input, and compute one or more new properties.

**Submodules:**

|  |   |
|--|---|
| <i>misc</i>  | Functions to compute <code>locdata</code> properties.         |
| <i>statistics</i> ( <code>locdata[, statistic_keys]</code> ) | Compute selected statistical parameter for localization data. |

**locan.data.properties.misc**

Functions to compute `locdata` properties.

**Classes**

|   |
|---|
| <i>InertiaMoments</i> ( <code>eigenvalues, eigenvectors, ...</code> ) |
|---|

## locan.data.properties.misc.InertiaMoments

```
class locan.data.properties.misc.InertiaMoments(eigenvalues, eigenvectors,
                                                variance_explained, orientation,
                                                eccentricity)
```

Bases: `NamedTuple`

### Methods

|  |  |
|--|--|
| <code>__init__()</code>                  |  |
| <code>count(value, /)</code>             | Return number of occurrences of value. |
| <code>index(value[, start, stop])</code> | Return first index of value.           |

### Attributes

|  |                          |
|--|--------------------------|
| <code><i>eccentricity</i></code>       | Alias for field number 4 |
| <code><i>eigenvalues</i></code>        | Alias for field number 0 |
| <code><i>eigenvectors</i></code>       | Alias for field number 1 |
| <code><i>orientation</i></code>        | Alias for field number 3 |
| <code><i>variance_explained</i></code> | Alias for field number 2 |

**eccentricity:** `float`

Alias for field number 4

**eigenvalues:** `ndarray[Any, dtype[Any]]`

Alias for field number 0

**eigenvectors:** `ndarray[Any, dtype[Any]]`

Alias for field number 1

**orientation:** `float`

Alias for field number 3

**variance\_explained:** `list[float]`

Alias for field number 2

## Functions

|  |   |
|--|---|
| <code>distance_to_region</code> (locdata, region)          | Determine the distance to the nearest point within <i>region</i> for all localizations.       |
| <code>distance_to_region_boundary</code> (locdata, region) | Determine the distance to the nearest region boundary for all localizations.                  |
| <code>inertia_moments</code> (points)                      | Return inertia moments (or principal components) and related properties for the given points. |
| <code>max_distance</code> (locdata)                        | Return maximum of all distances between any two localizations in locdata.                     |

### locan.data.properties.misc.distance\_to\_region

`locan.data.properties.misc.distance_to_region`(locdata, region)

Determine the distance to the nearest point within *region* for all localizations. Returns zero if localization is within the region.

#### Parameters

- **locdata** (*LocData*) – Localizations for which distances are determined.
- **region** (*Region*) – Region from which the closest point is selected.

#### Returns

Distance for each localization.

#### Return type

`npt.NDArray[np.float64]`

### locan.data.properties.misc.distance\_to\_region\_boundary

`locan.data.properties.misc.distance_to_region_boundary`(locdata, region)

Determine the distance to the nearest region boundary for all localizations. Returns a positive value regardless of whether the point is within or outside the region.

#### Parameters

- **locdata** (*LocData*) – Localizations for which distances are determined.
- **region** (*Region*) – Region from which the closest point is selected.

#### Returns

Distance for each localization.

#### Return type

`npt.NDArray[np.float64]`

## locan.data.properties.misc.inertia\_moments

`locan.data.properties.misc.inertia_moments(points)`

Return inertia moments (or principal components) and related properties for the given points. Inertia moments are represented by eigenvalues (and corresponding eigenvectors) of the covariance matrix. `Variance_explained` represents the eigenvalues normalized to the sum of all eigenvalues. For 2-dimensional data, orientation is the angle (in degrees) between the principal axis with the largest variance and the x-axis. Also, for 2-dimensional data, eccentricity is computed as  $e = \sqrt{1 - M_{\min}/M_{\max}}$ .

### Parameters

**points** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Coordinates of input points with shape (npoints, ndim).

### Returns

A tuple with eigenvalues, eigenvectors, `variance_explained`, orientation, eccentricity

### Return type

*InertiaMoments*

---

**Note:** The data is not standardized.

---

## locan.data.properties.misc.max\_distance

`locan.data.properties.misc.max_distance(locdata)`

Return maximum of all distances between any two localizations in `locdata`.

### Parameters

**locdata** (*LocData*) – Localization data

### Returns

A dict with key `max_distance` and the corresponding value being the maximum distance.

### Return type

dict[str, float]

## locan.data.properties.statistics

`locan.data.properties.statistics(locdata, statistic_keys=('count', 'min', 'max', 'mean', 'median', 'std', 'sem'))`

Compute selected statistical parameter for localization data.

### Parameters

- **locdata** – Localization data
- **statistic\_keys** – Pandas statistic functions. Default: ('count', 'min', 'max', 'mean', 'median', 'std', 'sem')

**Returns**

A dict with descriptive statistics.

**Return type**

dict[str, Any]

**locan.data.hulls**

Hull objects of localization data.

**Submodules:**

|                    |                                    |
|--------------------|------------------------------------|
| <i>hull</i>        | Hull objects of localization data. |
| <i>alpha_shape</i> | Alpha shape                        |

**locan.data.hulls.hull**

Hull objects of localization data.

This module computes specific hulls for the bounding box, convex hull and oriented bounding box and related properties for LocData objects.

**Classes**

|                                      |   |
|--------------------------------------|---|
| <i>BoundingBox</i> (points)          | Class with bounding box computed using numpy operations.                                      |
| <i>ConvexHull</i> (points[, method]) | Class with convex hull of localization data.  |
| <i>OrientedBoundingBox</i> (points)  | Class with oriented bounding box computed using the shapely minimum_rotated_rectangle method. |

**locan.data.hulls.hull.BoundingBox**

**class** locan.data.hulls.hull.**BoundingBox**(*points*)

Bases: object

Class with bounding box computed using numpy operations.

**Parameters**

**points** (*npt.ArrayLike*) – Coordinates of input points with shape (npoints, ndim).

**Variables**

- **hull** (*npt.NDArray[np.float64]*) – Array of point coordinates of shape (2, ndim) that represent [[min\_coordinates], [max\_coordinates]].
- **dimension** (*int*) – Spatial dimension of hull

- **vertices** (*npt.NDArray[np.float64]*) – Coordinates of points that make up the hull. Array of shape (ndim, 2).
- **width** (*npt.NDArray[np.float64]*) – Array with differences between max and min for each coordinate.
- **region\_measure** (*float*) – Hull measure, i.e. area or volume
- **subregion\_measure** (*float*) – Measure of the sub-dimensional region, i.e. circumference or surface
- **region** (*RoiRegion*) – Convert the hull to a RoiRegion object.

## Methods

```
__init__(points)
```

## Attributes

*region*

*vertices*

property region: *Region*

property vertices: ndarray[Any, dtype[float64]]

## locan.data.hulls.hull.ConvexHull

**class** locan.data.hulls.hull.ConvexHull(*points, method='scipy'*)

Bases: object

Class with convex hull of localization data.

### Parameters

- **points** (*npt.ArrayLike*) – Coordinates of input points. Array with shape (npoints, ndim).
- **method** (*Literal['scipy', 'shapely']*) – Specific class to compute the convex hull and attributes. One of 'scipy', 'shapely'.

### Variables

- **method** (*Literal['scipy', 'shapely']*) – Specific class to compute the convex hull and attributes. One of 'scipy', 'shapely'.
- **hull** (*Hull*) – Polygon object from the .convex\_hull method
- **dimension** (*int*) – Spatial dimension of hull



- **vertices** (*npt.NDArray[np.float64]*) – Coordinates of points that make up the hull. Array of shape (ndim, 2).
- **vertex\_indices** (*npt.NDArray[np.int64]*) – indices identifying a polygon of all points that make up the hull
- **points\_on\_boundary** (*int*) – The absolute number of points on the hull
- **points\_on\_boundary\_rel** (*int*) – The number of points on the hull relative to all input points
- **region\_measure** (*float*) – hull measure, i.e. area or volume
- **subregion\_measure** (*float*) – measure of the sub-dimensional region, i.e. circumference or surface
- **region** ([Region](#)) – Convert the hull to a Region object.

## Methods

```
__init__(points[, method])
```

## locan.data.hulls.hull.OrientedBoundingBox

**class** locan.data.hulls.hull.OrientedBoundingBox(*points*)

Bases: object

Class with oriented bounding box computed using the shapely minimum\_rotated\_rectangle method.

### Parameters

**points** (*npt.ArrayLike*) – Coordinates of input points with shape (npoints, ndim).

### Variables

- **hull** ([Polygon](#)) – Polygon object from the minimum\_rotated\_rectangle method
- **dimension** (*int*) – Spatial dimension of hull
- **vertices** (*npt.NDArray[np.float64]*) – Coordinates of points that make up the hull. Array of shape (ndim, 2).
- **width** (*npt.NDArray[np.float64]*) – Array with lengths of box edges.
- **region\_measure** (*float*) – hull measure, i.e. area or volume
- **subregion\_measure** (*float*) – measure of the sub-dimensional region, i.e. circumference or surface
- **region** ([Region](#)) – Convert the hull to a Region object.
- **angle** (*float*) – Orientation defined as angle (in degrees) between the vector from first to last point and x-axis.

## Methods

|                               |
|-------------------------------|
| <code>__init__(points)</code> |
|-------------------------------|

## Attributes

|                     |
|---------------------|
| <code>region</code> |
|---------------------|

|                       |
|-----------------------|
| <code>vertices</code> |
|-----------------------|

**property region:** *Rectangle*

**property vertices:** `ndarray[Any, dtype[float64]]`

## locan.data.hulls.alpha\_shape

### Alpha shape

This module provides methods for computing the alpha complex and specific alpha shapes together with related properties for LocData objects.

Alpha shape is a hull object that defines a group of localizations bordering a concave hull (which does not have to be connected and might have holes)<sup>1</sup>. It depends on a single parameter *alpha* (here defined to be a distance with a unit equal to the coordinate units of the given localization data). For *alpha* approaching infinity the alpha shape is equal to the convex hull.

In this context we call an alpha-complex the subgroup of simplexes of a Delaunay triangulation that is computed according to Edelsbrunner algorithm. All localizations that lie on the boundary of the alpha-complex make up the alpha shape.

Internally we also work with an alpha-independent representation of the alpha-complex that allows efficient computation of alpha shapes for arbitrary alpha values.

Simplices are classified as exterior (not part of the alpha complex), interior (not part of the boundary), regular (part of the boundary but not singular), and singular (part of the boundary but all simplices of higher dimension they are incident to are exterior).

---

<sup>1</sup> H. Edelsbrunner and E. P. Mücke, Three-dimensional alpha shapes. ACM Trans. Graph. 13(1):43-72, 1994.

## References

## Classes

|  |   |
|--|---|
| <code>AlphaComplex(points[, delaunay])</code>                | Class for an alpha-independent representation of the alpha complex of the given points. |
| <code>AlphaShape(alpha[, points, alpha_complex, ...])</code> | Class for the alpha shape of points for a specific alpha value.                         |

### locan.data.hulls.alpha\_shape.AlphaComplex

**class** locan.data.hulls.alpha\_shape.**AlphaComplex**(points, delaunay=None)

Bases: object

Class for an alpha-independent representation of the alpha complex of the given points.

Here the alpha complex is the simplicial subcomplex of the Delaunay triangulation together with the intervals defining simplex membership for an alpha complex for a specific *alpha*.

#### Parameters

- **points** (*npt.ArrayLike*) – Coordinates of input points with shape (npoints, ndim).
- **delaunay** (*scipy.spatial.Delaunay* | *None*) – Object with attribute *simplices* specifying a list of indices in the array of points that define the simplexes in the Delaunay triangulation. Also, an attribute *neighbor* is required that specifies indices of neighboring simplices. If *None*, *scipy.stat.Delaunay(points)* is computed.

#### Variables

- **lines** (*list[tuple[tuple[int, ...], float, float, float]]*) – 1-simplices (lines) that represent a simplicial subcomplex of the Delaunay triangulation with intervals. Array with shape (n\_lines, 2).
- **triangles** (*list[tuple[int, float, float, float]]*) – 2-simplices (triangles) that represent a simplicial subcomplex of the Delaunay triangulation with intervals. Array with shape (n\_triangles, 3).
- **tetrahedrons** (*list[tuple]*) – 3-simplices (tetrahedrons) that represent a simplicial subcomplex of the Delaunay triangulation with intervals. Array with shape (n\_tetrahedrons, 4).
- **dimension** (*int*) – Spatial dimension of the hull.
- **delaunay\_triangulation** (*scipy.spatial.Delaunay*) – Object with attribute *simplices* specifying a list of indices in the array of points that define the simplexes in the Delaunay triangulation. Also, an attribute *neighbor* is required that specifies indices of neighboring simplices.

## Methods

|   |   |
|---|---|
| <code>__init__(points[, delaunay])</code>               |   |
| <code>alpha_shape(alpha)</code>                         | Return the unique alpha shape for <i>alpha</i> .  |
| <code>alphas()</code>                                   | Return alpha values at which the corresponding alpha shape changes.   |
| <code>get_alpha_complex_lines(alpha[, type])</code>     | Simplicial subcomplex (lines) of the Delaunay triangulation for the specific alpha complex for the given <i>alpha</i> .                           |
| <code>get_alpha_complex_triangles(alpha[, type])</code> | Simplicial subcomplex (triangles) of the Delaunay triangulation for the specific alpha complex for the given <i>alpha</i> .                       |
| <code>graph_from_lines(alpha[, type])</code>            | Return networkx Graph object with nodes and edges from selected lines.  |
| <code>graph_from_triangles(alpha[, type])</code>        | Return networkx Graph object with nodes and edges from selected triangles.  |
| <code>optimal_alpha()</code>                            | Find the minimum alpha value for which all points belong to the alpha shape, in other words, no edges of the Delaunay triangulation are exterior. |

### `alpha_shape(alpha)`

Return the unique alpha shape for *alpha*.

#### Parameters

**alpha** (float) – Alpha parameter specifying a unique alpha complex.

#### Return type

*AlphaShape*

### `alphas()`

Return alpha values at which the corresponding alpha shape changes.

#### Return type

`npt.NDArray[np.float64]`

### `get_alpha_complex_lines(alpha, type='all')`

Simplicial subcomplex (lines) of the Delaunay triangulation for the specific alpha complex for the given *alpha*.

#### Parameters

- **alpha** (float) – Alpha parameter specifying a unique alpha complex.
- **type** (`Literal['all', 'regular', 'singular', 'interior', 'exterior']`) – Type of alpha complex edges to be included in the graph. One of ‘all’, ‘regular’, ‘singular’, ‘interior’, ‘exterior’.

#### Returns

The indices to specific points in *self.points*.

#### Return type

`list[list[int]]`

**get\_alpha\_complex\_triangles**(*alpha*, *type*='all')

Simplicial subcomplex (triangles) of the Delaunay triangulation for the specific alpha complex for the given *alpha*.

**Parameters**

- **alpha** (float) – Alpha parameter specifying a unique alpha complex.
- **type** (Literal['all', 'regular', 'singular', 'interior', 'exterior']) – Type of alpha complex edges to be included in the graph. One of 'all', 'regular', 'singular', 'interior', 'exterior'.

**Returns**

The indices to specific d-simplices in *self.delaunay\_triangulation.simplices*.

**Return type**

list[int]

**graph\_from\_lines**(*alpha*, *type*='all')

Return networkx Graph object with nodes and edges from selected lines.

**Parameters**

- **alpha** (float) – Alpha parameter specifying a unique alpha complex.
- **type** (Literal['all', 'regular', 'singular', 'interior', 'exterior']) – Type of alpha complex edges to be included in the graph. One of 'all', 'regular', 'singular', 'interior', 'exterior'.

**Return type**

networkx.Graph

**graph\_from\_triangles**(*alpha*, *type*='all')

Return networkx Graph object with nodes and edges from selected triangles.

**Parameters**

- **alpha** (float) – Alpha parameter specifying a unique alpha complex.
- **type** (Literal['all', 'regular', 'singular', 'interior', 'exterior']) – Type of alpha complex edges to be included in the graph. One of 'all', 'regular', 'singular', 'interior', 'exterior'.

**Return type**

networkx.Graph

**optimal\_alpha**()

Find the minimum alpha value for which all points belong to the alpha shape, in other words, no edges of the Delaunay triangulation are exterior.

**Return type**

float | None

## locan.data.hulls.alpha\_shape.AlphaShape

```
class locan.data.hulls.alpha_shape.AlphaShape(alpha, points=None,
                                              alpha_complex=None, delaunay=None)
```

Bases: object

Class for the alpha shape of points for a specific alpha value.

Here the alpha complex is the simplicial subcomplex of the Delaunay triangulation for a given alpha value. The alpha shape is the union of all simplexes of the alpha complex, specified by the boundary points of the alpha complex.

In order to update an existing AlphaShape object to a different *alpha* reset AlphaShape.alpha.

### Parameters

- **alpha** (*float*) – Alpha parameter specifying a unique alpha complex.
- **points** (*npt.ArrayLike* | *None*) – Coordinates of input points with shape (npoints, ndim). Either *points* or *alpha\_complex* have to be specified but not both.
- **alpha\_complex** (*AlphaComplex* | *None*) – The unfiltered alpha complex with computed interval values.
- **delaunay** (*scipy.spatial.Delaunay* | *None*) – Object with attribute *simplices* specifying a list of indices in the array of points that define the simplexes in the Delaunay triangulation. Also, an attribute *neighbor* is required that specifies indices of neighboring simplices. If *None*, *scipy.stat.Delaunay(points)* is computed.

### Variables

- **alpha\_complex** (*AlphaComplex*) – The unfiltered alpha complex with computed interval values.
- **alpha\_shape** (*npt.NDArray*) – The list of k-simplices (edges) from the alpha complex that make up the alpha shape. Or: Simplicial subcomplex of the Delaunay triangulation with regular simplices from the alpha complex.
- **region** (*Region*) – Region object.
- **connected\_components** (*list[Region]*) – Connected components, i.e. a list of the individual unconnected polygons that together make up the alpha shape.
- **dimension** (*int*) – Spatial dimension of the hull as determined from the dimension of *points*
- **vertices** (*npt.NDArray*) – Coordinates of points that make up the hull with shape (n\_points, 2). (regular alpha\_shape line-simplices).
- **vertex\_indices** (*list[int]*) – Indices identifying a polygon of all points that make up the hull (regular alpha\_shape line-simplices).
- **vertices\_alpha\_shape** (*npt.NDArray*) – Coordinates of points with shape (n\_points, 2) that make up the interior and boundary of the hull (regular, singular and interior alpha\_shape line-simplices).

- **vertex\_alpha\_shape\_indices** (*list[int]*) – Indices to all points that make up the interior and boundary of the hull. (regular, singular and interior alpha\_shape line-simplices).
- **vertices\_connected\_components\_indices** (*list[list[int]]*) – Indices to the points for each connected component of the alpha shape.
- **n\_points\_on\_boundary** (*float*) – The number of points on the hull (regular and singular alpha\_shape simplices).
- **n\_points\_on\_boundary\_rel** (*float*) – The number of points on the hull (regular and singular alpha\_shape simplices) relative to all alpha\_shape points.
- **n\_points\_alpha\_shape** (*int*) – Absolute number of points that are part of the alpha\_shape (regular, singular and interior alpha\_shape simplices).
- **n\_points\_alpha\_shape\_rel** (*int*) – Absolute number of points that are part of the alpha\_shape relative to all input points (regular, singular and interior alpha\_shape simplices).
- **region\_measure** (*float*) – Hull measure, i.e. area or volume.
- **subregion\_measure** (*float*) – Measure of the sub-dimensional region, i.e. circumference or surface.

## Methods

```
__init__(alpha[, points, alpha_complex, ...])
```

## Attributes

*alpha*

*alpha\_shape*

*connected\_components*

*region*

*vertex\_alpha\_shape\_indices*

*vertex\_indices*

*vertices*

*vertices\_alpha\_shape*

*vertices\_connected\_components\_indice*

```
property alpha: float
property alpha_shape: list[list[int]]
property connected_components: list[Region]
property region: Region
property vertex_alpha_shape_indices: list[int]
property vertex_indices: list[int]
property vertices: ndarray[Any, dtype[float64]]
property vertices_alpha_shape: ndarray[Any, dtype[float64]]
property vertices_connected_components_indices: list[list[int]]
```

### **locan.data.region**

Regions as support for localization data.

This module provides classes to define geometric regions for localization data. All region classes inherit from the abstract base class *Region*.



## Classes

|  |   |
|--|---|
| <a href="#"><i>AxisOrientedCuboid</i></a> ([corner, length, width, ...]) | Region class to define an axis-oriented cuboid.   |
| <a href="#"><i>AxisOrientedHypercuboid</i></a> ([corner, lengths])       | Region class to define an axis-oriented n-dimensional hypercuboid.  |
| <a href="#"><i>Cuboid</i></a> ([corner, length, width, height, ...])     | Region class to define a cuboid.  |
| <a href="#"><i>Ellipse</i></a> ([center, width, height, angle])          | Region class to define an ellipse.  |
| <a href="#"><i>EmptyRegion</i></a> ()                                    | Region class to define an empty region that has no dimension.   |
| <a href="#"><i>Interval</i></a> ([lower_bound, upper_bound])             | Region class to define an interval.   |
| <a href="#"><i>MultiPolygon</i></a> (polygons)                           | Region class to define a region that represents the union of multiple polygons.   |
| <a href="#"><i>Polygon</i></a> ([points, holes])                         | Region class to define a polygon.   |
| <a href="#"><i>Rectangle</i></a> ([corner, width, height, angle])        | Region class to define a rectangle.   |
| <a href="#"><i>Region</i></a> ()   | Abstract Region class to define the interface for Region-derived classes that specify geometric objects to represent regions of interest. |
| <a href="#"><i>Region1D</i></a> ()                                       | Abstract Region class to define the interface for 1-dimensional Region classes.   |
| <a href="#"><i>Region2D</i></a> ()                                       | Abstract Region class to define the interface for 2-dimensional Region classes.   |
| <a href="#"><i>Region3D</i></a> ()                                       | Abstract Region class to define the interface for 3-dimensional Region classes.   |
| <a href="#"><i>RegionND</i></a> ()                                       | Abstract Region class to define the interface for n-dimensional Region classes.   |
| <a href="#"><i>RoiRegion</i></a> (region_type, region_specs)             | Deprecated Region object to specify regions of interest.  |

### locan.data.region.AxisOrientedCuboid

**class** locan.data.region.**AxisOrientedCuboid**(corner=(0, 0, 0), length=1, width=1, height=1)

Bases: [\*Region3D\*](#)

Region class to define an axis-oriented cuboid.

This is a 3-dimensional convex region with rectangular faces and edges that are parallel to coordinate axes. Extension in x-, y-, z-coordinates correspond to length, width, height.

#### Parameters

- **corner** (*npt.ArrayLike*) – A point that defines the lower left corner with shape (3,)
- **length** (*float*) – The length of a vector describing the edge in x-direction.
- **width** (*float*) – The length of a vector describing the edge in y-direction.
- **height** (*float*) – The length of a vector describing the edge in z-direction.

## Methods

|  |  |
|--|--|
| <code>__init__([corner, length, width, height])</code> |  |
| <code>as_artist([origin])</code>                       | Matplotlib patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ).                     |
| <code>buffer(distance, **kwargs)</code>                | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>                          | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>                 | Constructor for instantiating Region from list of (min, max) bounds.                                   |
| <code>intersection(other)</code>                       | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>plot([ax])</code>                                | Provide plot of region as <code>matplotlib.axes.Axes</code> object.                                    |
| <code>symmetric_difference(other)</code>               | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                              | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                                |  |
|--------------------------------|--|
| <code>bounding_box</code>      | A region describing the minimum axis-aligned bounding box that encloses the original region. |
| <code>bounds</code>            | Region bounds <code>min_x, min_y, ..., max_x, max_y, .</code>                                |
| <code>centroid</code>          | Point coordinates for region centroid.   |
| <code>corner</code>            | A point that defines the lower left corner.  |
| <code>dimension</code>         | The region dimension.  |
| <code>extent</code>            | The extent ( <code>max_x - min_x</code> ), ( <code>max_y - min_y</code> ), <code>.</code>    |
| <code>height</code>            | The length of a vector describing the edge in z-direction.                                   |
| <code>intervals</code>         | Provide bounds in a tuple (min, max) arrangement.  |
| <code>length</code>            | The length of a vector describing the edge in x-direction.                                   |
| <code>max_distance</code>      | The maximum distance between any two points within the region.                               |
| <code>points</code>            | Point coordinates.   |
| <code>region_measure</code>    | Region measure, i.e. area (for 2d) or volume (for 3d).                                       |
| <code>subregion_measure</code> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).      |
| <code>width</code>             | The length of a vector describing the edge in y-direction.                                   |

**as\_artist**(*origin*=(0, 0), *\*\*kwargs*)

Matplotlib patch object for this region (e.g. *matplotlib.patches.Ellipse*).

**Parameters**

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The (x, y, z) pixel position of the origin of the displayed image. Default is (0, 0, 0).
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

**Returns**

Matplotlib patch for the specified region.

**Return type**

*matplotlib.patches.Patch*

**property bounding\_box:** *Self*

A region describing the minimum axis-aligned bounding box that encloses the original region.

**Return type**

*Region*

**property bounds:** *ndarray*[Any, *dtype*[float64]]

Region bounds min\_x, min\_y, ..., max\_x, max\_y, ... for each dimension.

**Returns**

of shape (2 \* dimension,)

**Return type**

*npt.NDArray*[*np.float64*] | None

**buffer**(*distance*, *\*\*kwargs*)

Extend the region perpendicular by a *distance*.

**Parameters**

**distance** (float) – Distance by which the region is extended.

**Returns**

The extended region.

**Return type**

*Region*

**property centroid:** *ndarray*[Any, *dtype*[float64]]

Point coordinates for region centroid.

**Returns**

of shape (dimension,)

**Return type**

*npt.NDArray*[*np.float64*] | None

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

#### Parameters

**points** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]]]) – Coordinates of points that are tested for being  
inside the specified region.

#### Returns

Array with indices for all points in original point array that are within the region.

#### Return type

npt.NDArray[np.int64]

**property corner:** ndarray[Any, dtype[float64]]

A point that defines the lower left corner.

#### Returns

with shape (2,)

#### Return type

npt.NDArray[np.float64]

**property extent:** ndarray[Any, dtype[float64]]

The extent (max\_x - min\_x), (max\_y - min\_y), ... for each dimension.

#### Returns

of shape (dimension,)

#### Return type

npt.NDArray[np.float64] | None

**classmethod from\_intervals**(*intervals*)

Constructor for instantiating Region from list of (min, max) bounds.

#### Parameters

**intervals** (*npt.ArrayLike*) – The region bounds for each dimension of  
shape (3, 2)

#### Return type

*AxisOrientedCuboid*

**property height:** float

The length of a vector describing the edge in z-direction.

#### Return type

float

**property intervals:** ndarray[Any, dtype[float64]]

Provide bounds in a tuple (min, max) arrangement.

#### Returns

((min\_x, max\_x), ...) of shape(dimension, 2)

#### Return type

tuple[tuple[float, float], ...]

**property length:** float

The length of a vector describing the edge in x-direction.

**Return type**

float

**property max\_distance: float**

The maximum distance between any two points within the region.

**Return type**

float

**property points: ndarray[Any, dtype[float64]]**

Point coordinates.

**Returns**

of shape (n\_points, dimension)

**Return type**

npt.NDArray[np.float64] | list[npt.NDArray[np.float64]]

**property region\_measure: float**

Region measure, i.e. area (for 2d) or volume (for 3d).

**Return type**

float

**property subregion\_measure: float**

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

**property width: float**

The length of a vector describing the edge in y-direction.

**Return type**

float

## locan.data.region.AxisOrientedHypercuboid

**class** locan.data.region.**AxisOrientedHypercuboid**(*corner=(0, 0, 0), lengths=(1, 1, 1)*)

Bases: [RegionND](#)

Region class to define an axis-oriented n-dimensional hypercuboid.

This is a n-dimensional convex region with rectangular faces and edges that are parallel to coordinate axes. Extension in x-, y-, z-coordinates correspond to length, width, height.

**Parameters**

- **corner** (*npt.ArrayLike*) – A point with shape (dimension,) that defines the lower left corner.
- **lengths** (*npt.ArrayLike*) – Array of shape(dimension,) of length values for the 1-dimensional edge vectors.

## Methods

|  |  |
|--|--|
| <code>__init__([corner, lengths])</code> |  |
| <code>buffer(distance, **kwargs)</code>  | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>            | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>   | Constructor for instantiating Region from list of (min, max) bounds.                                   |
| <code>intersection(other)</code>         | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>symmetric_difference(other)</code> | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                                |  |
|--------------------------------|--|
| <code>bounding_box</code>      | A region describing the minimum axis-aligned bounding box that encloses the original region. |
| <code>bounds</code>            | Region bounds min_x, min_y, ..., max_x, max_y, .   |
| <code>centroid</code>          | Point coordinates for region centroid.   |
| <code>corner</code>            | A point that defines the lower left corner.  |
| <code>dimension</code>         | The region dimension.  |
| <code>extent</code>            | The extent (max_x - min_x), (max_y - min_y), .   |
| <code>intervals</code>         | Provide bounds in a tuple (min, max) arrangement.  |
| <code>lengths</code>           | Array of length values for the 1-dimensional edge vectors.                                   |
| <code>max_distance</code>      | The maximum distance between any two points within the region.                               |
| <code>points</code>            | Point coordinates.   |
| <code>region_measure</code>    | Region measure, i.e. area (for 2d) or volume (for 3d).                                       |
| <code>subregion_measure</code> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).      |

### property `bounding_box`: Self

A region describing the minimum axis-aligned bounding box that encloses the original region.

#### Return type

*Region*

### property `bounds`: ndarray[Any, dtype[float64]]

Region bounds min\_x, min\_y, ..., max\_x, max\_y, ... for each dimension.

**Returns**

of shape (2 \* dimension,)

**Return type**

npt.NDArray[np.float64] | None

**buffer**(*distance*, *\*\*kwargs*)

Extend the region perpendicular by a *distance*.

**Parameters**

**distance** (float) – Distance by which the region is extended.

**Returns**

The extended region.

**Return type**

*Region*

**property centroid:** ndarray[Any, dtype[float64]]

Point coordinates for region centroid.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

**Parameters**

**points** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]]]) – Coordinates of points that are tested for being  
inside the specified region.

**Returns**

Array with indices for all points in original point array that are within the region.

**Return type**

npt.NDArray[np.int64]

**property corner:** ndarray[Any, dtype[float64]]

A point that defines the lower left corner.

**Returns**

with shape (dimension,)

**Return type**

npt.NDArray[np.float64]

**property dimension:** int

The region dimension.

**Return type**

int | None

**property extent:** ndarray[Any, dtype[float64]]

The extent (max\_x - min\_x), (max\_y - min\_y), ... for each dimension.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**classmethod from\_intervals**(*intervals*)

Constructor for instantiating Region from list of (min, max) bounds.

**Parameters**

**intervals** (*npt.ArrayLike*) – The region bounds for each dimension of shape (dimension, 2)

**Return type**

*AxisOrientedHypercuboid*

**property intervals:** ndarray[Any, dtype[float64]]

Provide bounds in a tuple (min, max) arrangement.

**Returns**

((min\_x, max\_x), ...) of shape(dimension, 2).

**Return type**

tuple[tuple[float, float], ...]

**property lengths:** ndarray[Any, dtype[float64]]

Array of length values for the 1-dimensional edge vectors.

**Returns**

of shape(dimension,)

**Return type**

npt.NDArray[np.float64]

**property max\_distance:** float

The maximum distance between any two points within the region.

**Return type**

float

**property points:** ndarray[Any, dtype[float64]]

Point coordinates.

**Returns**

of shape (n\_points, dimension)

**Return type**

npt.NDArray[np.float64] | list[npt.NDArray[np.float64]]

**property region\_measure:** float

Region measure, i.e. area (for 2d) or volume (for 3d).

**Return type**

float



**property subregion\_measure:** float

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

### **locan.data.region.Cuboid**

**class** locan.data.region.**Cuboid**(*corner=(0, 0, 0), length=1, width=1, height=1, alpha=0, beta=0, gamma=0*)

Bases: [Region3D](#)

Region class to define a cuboid.

This is a 3-dimensional convex region with rectangular faces. Extension in x-, y-, z-coordinates correspond to length, width, height. Corresponding Euler angles are defined by alpha, beta, gamma.

#### **Parameters**

- **corner** (*npt.ArrayLike*) – A point that defines the lower left corner with shape (2,).
- **length** (*float*) – The length of a vector describing the edge in x-direction.
- **width** (*float*) – The length of a vector describing the edge in y-direction.
- **alpha** (*float*) – The first Euler angle (in degrees) by which the cuboid is rotated.
- **beta** (*float*) – The second Euler angle (in degrees) by which the cuboid is rotated.
- **gamma** (*float*) – The third Euler angle (in degrees) by which the cuboid is rotated.

## Methods

|   |  |
|---|--|
| <code>__init__([corner, length, width, height, ...])</code> |  |
| <code>as_artist([origin])</code>                            | Matplotlib patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ).                     |
| <code>buffer(distance, **kwargs)</code>                     | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>                               | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>                      | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>intersection(other)</code>                            | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>plot([ax])</code>                                     | Provide plot of region as <code>matplotlib.axes.Axes</code> object.                                    |
| <code>symmetric_difference(other)</code>                    | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                                   | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                          |  |
|--------------------------|--|
| <i>alpha</i>             | The first Euler angle (in degrees) by which the cuboid is rotated.                           |
| <i>beta</i>              | The second Euler angle (in degrees) by which the cuboid is rotated.                          |
| <i>bounding_box</i>      | A region describing the minimum axis-aligned bounding box that encloses the original region. |
| <i>bounds</i>            | Region bounds min_x, min_y, ..., max_x, max_y, .   |
| <i>centroid</i>          | Point coordinates for region centroid.   |
| <i>corner</i>            | A point that defines the lower left corner.  |
| <i>dimension</i>         | The region dimension.  |
| <i>extent</i>            | The extent (max_x - min_x), (max_y - min_y), .   |
| <i>gamma</i>             | The third Euler angle (in degrees) by which the cuboid is rotated.                           |
| <i>height</i>            | The length of a vector describing the edge in z-direction.                                   |
| <i>length</i>            | The length of a vector describing the edge in x-direction.                                   |
| <i>max_distance</i>      | The maximum distance between any two points within the region.                               |
| <i>points</i>            | Point coordinates.   |
| <i>region_measure</i>    | Region measure, i.e. area (for 2d) or volume (for 3d).                                       |
| <i>subregion_measure</i> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).      |
| <i>width</i>             | The length of a vector describing the edge in y-direction.                                   |

**property alpha:** float

The first Euler angle (in degrees) by which the cuboid is rotated.

**Return type**

float

**as\_artist**(*origin*=(0, 0), *\*\*kwargs*)

Matplotlib patch object for this region (e.g. *matplotlib.patches.Ellipse*).

**Parameters**

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The (x, y, z) pixel position of the origin of the displayed image. Default is (0, 0, 0).
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

**Returns**

Matplotlib patch for the specified region.

**Return type**

matplotlib.patches.Patch

**property beta:** float

The second Euler angle (in degrees) by which the cuboid is rotated.

**Return type**

float

**property bounding\_box:** Self

A region describing the minimum axis-aligned bounding box that encloses the original region.

**Return type**

*Region*

**property bounds:** ndarray[Any, dtype[float64]]

Region bounds min\_x, min\_y, ..., max\_x, max\_y, ... for each dimension.

**Returns**

of shape (2 \* dimension,)

**Return type**

npt.NDArray[np.float64] | None

**buffer**(*distance*, *\*\*kwargs*)

Extend the region perpendicular by a *distance*.

**Parameters**

**distance** (float) – Distance by which the region is extended.

**Returns**

The extended region.

**Return type**

*Region*

**property centroid:** ndarray[Any, dtype[float64]]

Point coordinates for region centroid.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

**Parameters**

**points** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]]]) – Coordinates of points that are tested for being  
inside the specified region.

**Returns**

Array with indices for all points in original point array that are within the region.

**Return type**

npt.NDArray[np.int64]

**property corner:** ndarray[Any, dtype[float64]]

A point that defines the lower left corner.

**Returns**

with shape (2,)

**Return type**

npt.NDArray[np.float64]

**property extent:** ndarray[Any, dtype[float64]]

The extent (max\_x - min\_x), (max\_y - min\_y), ... for each dimension.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**property gamma:** float

The third Euler angle (in degrees) by which the cuboid is rotated.

**Return type**

float

**property height:** float

The length of a vector describing the edge in z-direction.

**Return type**

float

**property length:** float

The length of a vector describing the edge in x-direction.

**Return type**

float

**property max\_distance:** float

The maximum distance between any two points within the region.

**Return type**

float

**property points:** ndarray[Any, dtype[float64]]

Point coordinates.

**Returns**

of shape (n\_points, dimension)

**Return type**

npt.NDArray[np.float64] | list[npt.NDArray[np.float64]]

**property region\_measure:** float

Region measure, i.e. area (for 2d) or volume (for 3d).

**Return type**

float

**property subregion\_measure: float**

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

**property width: float**

The length of a vector describing the edge in y-direction.

**Return type**

float

### **locan.data.region.Ellipse**

**class** locan.data.region.**Ellipse**(center=(0, 0), width=1, height=1, angle=0)

Bases: [\*Region2D\*](#)

Region class to define an ellipse.

#### **Parameters**

- **center** (*npt.ArrayLike*) – A point that defines the center of the ellipse with shape (2,).
- **width** (*float*) – The length of a vector describing the principal axis in x-direction (before rotation).
- **height** (*float*) – The length of a vector describing the principal axis in y-direction (before rotation).
- **angle** (*float*) – The angle (in degrees) by which the ellipse is rotated counterclockwise around the center point.

## Methods

|   |  |
|---|--|
| <code>__init__([center, width, height, angle])</code> |  |
| <code>as_artist([origin])</code>                      | Matplotlib 2D patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ).                  |
| <code>buffer(distance, **kwargs)</code>               | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>                         | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>                | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>from_shapely(shapely_object)</code>             | Constructor for instantiating Region from <i>shapely</i> object.                                       |
| <code>intersection(other)</code>                      | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>plot([ax])</code>                               | Provide plot of region as <code>matplotlib.axes.Axes</code> object.                                    |
| <code>symmetric_difference(other)</code>              | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                             | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                           |  |
|---------------------------|--|
| <i>angle</i>              | The angle (in degrees) by which the ellipse is rotated counterclockwise around the center point.         |
| <code>bounding_box</code> | A region describing the minimum axis-aligned bounding box that encloses the original region.             |
| <code>bounds</code>       | Region bounds <code>min_x</code> , <code>min_y</code> , ..., <code>max_x</code> , <code>max_y</code> , . |
| <i>center</i>             | A point that defines the center of the ellipse.  |
| <i>centroid</i>           | Point coordinates for region centroid.   |
| <code>dimension</code>    | The region dimension.  |
| <code>extent</code>       | The extent ( <code>max_x - min_x</code> ), ( <code>max_y - min_y</code> ), .                             |
| <i>height</i>             | The length of a vector describing the principal axis in y-direction (before rotation).                   |
| <i>max_distance</i>       | The maximum distance between any two points within the region.   |
| <i>points</i>             | Point coordinates.   |
| <i>region_measure</i>     | Region measure, i.e. area (for 2d) or volume (for 3d).   |
| <i>region_specs</i>       | Legacy interface to serve legacy <code>RoiRegion</code> .  |
| <i>shapely_object</i>     | Geometric object as defined in <i>shapely</i> .  |
| <i>subregion_measure</i>  | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).                  |
| <i>width</i>              | The length of a vector describing the principal axis in x-direction (before rotation).                   |

### property `angle`: float

The angle (in degrees) by which the ellipse is rotated counterclockwise around the center point.

#### Return type

float

### `as_artist`(*origin*=(0, 0), *\*\*kwargs*)

Matplotlib 2D patch object for this region (e.g. *matplotlib.patches.Ellipse*).

#### Parameters

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The (x, y) pixel position of the origin of the displayed image. Default is (0, 0).
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

#### Returns

Matplotlib patch for the specified region.

#### Return type

*matplotlib.patches.Patch*



**property center:** ndarray[Any, dtype[float64]]

A point that defines the center of the ellipse.

**Returns**

with shape (2,)

**Return type**

npt.NDArray[np.float64]

**property centroid:** ndarray[Any, dtype[float64]]

Point coordinates for region centroid.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

**Parameters**

**points** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]]]) – Coordinates of points that are tested for being  
inside the specified region.

**Returns**

Array with indices for all points in original point array that are within the region.

**Return type**

npt.NDArray[np.int64]

**property height:** float

The length of a vector describing the principal axis in y-direction (before rotation).

**Return type**

float

**property max\_distance:** float

The maximum distance between any two points within the region.

**Return type**

float

**property points:** ndarray[Any, dtype[float64]]

Point coordinates.

**Returns**

of shape (n\_points, dimension)

**Return type**

npt.NDArray[np.float64] | list[npt.NDArray[np.float64]]

**property region\_measure:** float

Region measure, i.e. area (for 2d) or volume (for 3d).

**Return type**

float

**property region\_specs**

Legacy interface to serve legacy RoiRegion.

**Warning:** Do not use - will be deprecated.

**Return type**

dict

**property shapely\_object: Polygon**

Geometric object as defined in *shapely*.

**Return type**

Shapely object

**property subregion\_measure: float**

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

**property width: float**

The length of a vector describing the principal axis in x-direction (before rotation).

**Return type**

float

## locan.data.region.EmptyRegion

**class** locan.data.region.**EmptyRegion**

Bases: [Region](#)

Region class to define an empty region that has no dimension.

## Methods

|   |  |
|---|--|
| <code>__init__()</code>                   |  |
| <code>as_artist(**kwargs)</code>          | <b>rtype</b><br>None   |
| <code>buffer(distance, **kwargs)</code>   | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>             | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>    | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>from_shapely(shapely_object)</code> | <b>rtype</b><br><i>EmptyRegion</i>   |
| <code>intersection(other)</code>          | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>symmetric_difference(other)</code>  | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                 | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                                |  |
|--------------------------------|--|
| <code>bounding_box</code>      | A region describing the minimum axis-aligned bounding box that encloses the original region. |
| <code>bounds</code>            | Region bounds min_x, min_y, ..., max_x, max_y, .   |
| <code>centroid</code>          | Point coordinates for region centroid.   |
| <code>dimension</code>         | The region dimension.  |
| <code>extent</code>            | The extent (max_x - min_x), (max_y - min_y), .   |
| <code>max_distance</code>      | The maximum distance between any two points within the region.                               |
| <code>points</code>            | Point coordinates.   |
| <code>region_measure</code>    | Region measure, i.e. area (for 2d) or volume (for 3d).                                       |
| <code>subregion_measure</code> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).      |

`as_artist(**kwargs)`

**Return type**  
None

**property bounding\_box:** *EmptyRegion*

A region describing the minimum axis-aligned bounding box that encloses the original region.

**Return type**

*Region*

**property bounds:** **None**

Region bounds min\_x, min\_y, ..., max\_x, max\_y, ... for each dimension.

**Returns**

of shape (2 \* dimension,)

**Return type**

npt.NDArray[np.float64] | None

**buffer**(*distance*, *\*\*kwargs*)

Extend the region perpendicular by a *distance*.

**Parameters**

**distance** (float) – Distance by which the region is extended.

**Returns**

The extended region.

**Return type**

*Region*

**property centroid:** **None**

Point coordinates for region centroid.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

**Parameters**

**points** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]]]) – Coordinates of points that are tested for being  
inside the specified region.

**Returns**

Array with indices for all points in original point array that are within the region.

**Return type**

npt.NDArray[np.int64]

**property dimension:** **None**

The region dimension.

**Return type**

int | None

**property extent: None**

The extent (max\_x - min\_x), (max\_y - min\_y), ... for each dimension.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**classmethod from\_shapely(*shapely\_object*)****Return type**

*EmptyRegion*

**intersection(*other*)**

Returns a region representing the intersection of this region with *other*.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

**property max\_distance: int**

The maximum distance between any two points within the region.

**Return type**

float

**property points: ndarray[Any, dtype[float64]]**

Point coordinates.

**Returns**

of shape (n\_points, dimension)

**Return type**

npt.NDArray[np.float64] | list[npt.NDArray[np.float64]]

**property region\_measure: int**

Region measure, i.e. area (for 2d) or volume (for 3d).

**Return type**

float

**property subregion\_measure: int**

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

**symmetric\_difference(*other*)**

Returns the union of the two regions minus any areas contained in the intersection of the two regions.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

**union**(*other*)

Returns a region representing the union of this region with *other*.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

## **locan.data.region.Interval**

**class** locan.data.region.**Interval**(*lower\_bound=0, upper\_bound=1*)

Bases: *Region1D*

Region class to define an interval.

**Parameters**

- **lower\_bound** (float) – The lower bound of the interval.
- **upper\_bound** (float) – The upper bound of the interval.

## **Methods**

|   |  |
|---|--|
| <code>__init__([lower_bound, upper_bound])</code> |  |
| <code>as_artist([origin])</code>                  | Matplotlib 2D patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ).                  |
| <code>buffer(distance, **kwargs)</code>           | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>                     | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>            | Constructor for instantiating Region from list of (min, max) bounds.                                   |
| <code>intersection(other)</code>                  | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>symmetric_difference(other)</code>          | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                         | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                          |  |
|--------------------------|--|
| <i>bounding_box</i>      | A region describing the minimum axis-aligned bounding box that encloses the original region. |
| <i>bounds</i>            | Region bounds min_x, min_y, ..., max_x, max_y, .   |
| <i>centroid</i>          | Point coordinates for region centroid.   |
| <i>dimension</i>         | The region dimension.  |
| <i>extent</i>            | The extent (max_x - min_x), (max_y - min_y), .   |
| <i>intervals</i>         | Provide bounds in a tuple (min, max) arrangement.  |
| <i>lower_bound</i>       | The lower boundary.  |
| <i>max_distance</i>      | The maximum distance between any two points within the region.                               |
| <i>points</i>            | Point coordinates.   |
| <i>region_measure</i>    | Region measure, i.e. area (for 2d) or volume (for 3d).                                       |
| <i>region_specs</i>      | Legacy interface to serve legacy RoiRegion.  |
| <i>subregion_measure</i> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).      |
| <i>upper_bound</i>       | The upper boundary.  |

**as\_artist**(*origin*=(0, 0), *\*\*kwargs*)

Matplotlib 2D patch object for this region (e.g. *matplotlib.patches.Ellipse*).

### Parameters

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The (x, y) pixel position of the origin of the displayed image. Default is (0, 0).
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

### Returns

Matplotlib patch for the specified region.

### Return type

*matplotlib.patches.Patch*

**property bounding\_box:** *Self*

A region describing the minimum axis-aligned bounding box that encloses the original region.

### Return type

*Region*

**property bounds:** *ndarray[Any, dtype[float64]]*

Region bounds min\_x, min\_y, ..., max\_x, max\_y, ... for each dimension.

### Returns

of shape (2 \* dimension,)

**Return type**

npt.NDArray[np.float64] | None

**buffer**(*distance*, *\*\*kwargs*)

Extend the region perpendicular by a *distance*.

**Parameters**

**distance** (float) – Distance by which the region is extended.

**Returns**

The extended region.

**Return type**

*Region*

**property centroid:** ndarray[Any, dtype[float64]]

Point coordinates for region centroid.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

**Parameters**

**points** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]]]) – Coordinates of points that are tested for being  
inside the specified region.

**Returns**

Array with indices for all points in original point array that are within the region.

**Return type**

npt.NDArray[np.int64]

**property extent:** ndarray[Any, dtype[float64]]

The extent (max\_x - min\_x), (max\_y - min\_y), ... for each dimension.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**classmethod from\_intervals**(*intervals*)

Constructor for instantiating Region from list of (min, max) bounds. Takes array-like intervals instead of interval to be consistent with *Rectangle.from\_intervals*.

**Parameters**

**intervals** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,



complex, str, bytes]])) – The region bounds for each dimension of shape (2,)

**Return type**

*Interval*

**property intervals:** ndarray[Any, dtype[float64]]

Provide bounds in a tuple (min, max) arrangement.

**Returns**

((min\_x, max\_x), ...) of shape(dimension, 2).

**Return type**

tuple[tuple[float, float], ...]

**property lower\_bound:** float

The lower boundary.

**Return type**

float

**property max\_distance:** float

The maximum distance between any two points within the region.

**Return type**

float

**property points:** ndarray[Any, dtype[float64]]

Point coordinates.

**Returns**

of shape (n\_points, dimension)

**Return type**

npt.NDArray[np.float64] | list[npt.NDArray[np.float64]]

**property region\_measure:** float

Region measure, i.e. area (for 2d) or volume (for 3d).

**Return type**

float

**property region\_specs:** tuple[float, float]

Legacy interface to serve legacy RoiRegion.

**Warning:** Do not use - will be deprecated.

**Return type**

tuple[float, float]

**property subregion\_measure:** int

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

**property upper\_bound:** float

The upper boundary.

**Return type**

float

## locan.data.region.MultiPolygon

**class** locan.data.region.**MultiPolygon**(*polygons*)

Bases: [Region2D](#)

Region class to define a region that represents the union of multiple polygons.

**Parameters**

**polygons** (*list* [[Polygon](#)]) – Polygons that define the individual polygons.

## Methods

|  |  |
|--|--|
| <code>__init__(polygons)</code>          |  |
| <code>as_artist([origin])</code>         | Matplotlib 2D patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ).                  |
| <code>buffer(distance, **kwargs)</code>  | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>            | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>   | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>from_shapely(multipolygon)</code>  | Constructor for instantiating Region from <i>shapely</i> object.                                       |
| <code>intersection(other)</code>         | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>plot([ax])</code>                  | Provide plot of region as <code>matplotlib.axes.Axes</code> object.                                    |
| <code>symmetric_difference(other)</code> | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                                |  |
|--------------------------------|--|
| <code>bounding_box</code>      | A region describing the minimum axis-aligned bounding box that encloses the original region.             |
| <code>bounds</code>            | Region bounds <code>min_x</code> , <code>min_y</code> , ..., <code>max_x</code> , <code>max_y</code> , . |
| <code>centroid</code>          | Point coordinates for region centroid.   |
| <code>dimension</code>         | The region dimension.  |
| <code>extent</code>            | The extent ( <code>max_x - min_x</code> ), ( <code>max_y - min_y</code> ), .                             |
| <code>holes</code>             | Points defining holes.   |
| <code>max_distance</code>      | The maximum distance between any two points within the region.   |
| <code>points</code>            | Exterior polygon points.   |
| <code>polygons</code>          | All polygons that make up the MultiPolygon   |
| <code>region_measure</code>    | Region measure, i.e. area (for 2d) or volume (for 3d).   |
| <code>region_specs</code>      | Legacy interface to serve legacy RoiRegion.  |
| <code>shapely_object</code>    | Geometric object as defined in <i>shapely</i> .  |
| <code>subregion_measure</code> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).                  |

**as\_artist**(*origin*=(0, 0), *\*\*kwargs*)

Matplotlib 2D patch object for this region (e.g. *matplotlib.patches.Ellipse*).

### Parameters

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The (x, y) pixel position of the origin of the displayed image. Default is (0, 0).
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

### Returns

Matplotlib patch for the specified region.

### Return type

*matplotlib.patches.Patch*

**property centroid:** `ndarray[Any, dtype[float64]]`

Point coordinates for region centroid.

### Returns

of shape (dimension,)

### Return type

`npt.NDArray[np.float64]` | None

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

#### Parameters

**points** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]]]) – Coordinates of points that are tested for being  
inside the specified region.

#### Returns

Array with indices for all points in original point array that are within the region.

#### Return type

npt.NDArray[np.int64]

**classmethod from\_shapely**(*multipolygon*)

Constructor for instantiating Region from *shapely* object.

#### Parameters

**shapely\_object** – Geometric object to be converted into Region

#### Return type

*Polygon* | *MultiPolygon* | *EmptyRegion*

**property holes:** list[list[ndarray[Any, dtype[float64]]] | None]

Points defining holes.

#### Returns

list of polygon holes

#### Return type

list[list[npt.NDArray[np.float64]] | None]

**property max\_distance:** float

The maximum distance between any two points within the region.

#### Return type

float

**property points:** list[ndarray[Any, dtype[float64]]]

Exterior polygon points.

#### Returns

n\_polygons of shape(n\_points, dimension)

#### Return type

list[npt.NDArray[np.float64]]

**property polygons:** list[*Polygon*]

All polygons that make up the MultiPolygon

#### Return type

list[*Polygon*]

**property region\_measure:** float

Region measure, i.e. area (for 2d) or volume (for 3d).

#### Return type

float

**property region\_specs**

Legacy interface to serve legacy RoiRegion.

|  |
|--|
| <b>Warning:</b> Do not use - will be deprecated. |
|--|

**Return type**

dict

**property shapely\_object: MultiPolygon**

Geometric object as defined in *shapely*.

**Return type**

Shapely object

**property subregion\_measure: float**

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

**locan.data.region.Polygon**

**class** locan.data.region.Polygon(*points=((0, 0), (0, 1), (1, 1), (1, 0)), holes=None*)

Bases: [Region2D](#)

Region class to define a polygon.

The polygon is constructed from a list of points that can be closed (i.e. the first and last point are identical) or not (in this case the list of points will be closed implicitly).

**Parameters**

- **points** (*npt.ArrayLike*) – Points with shape (n\_points, 2) that define the exterior boundary of a polygon.
- **holes** (*list[npt.ArrayLike] | None*) – Points with shape (n\_holes, n\_points, 2) that define holes within the polygon.

## Methods

|  |  |
|--|--|
| <code>__init__([points, holes])</code>   |  |
| <code>as_artist([origin])</code>         | Matplotlib 2D patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ).                  |
| <code>buffer(distance, **kwargs)</code>  | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>            | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>   | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>from_shapely(polygon)</code>       | Constructor for instantiating Region from <i>shapely</i> object.                                       |
| <code>intersection(other)</code>         | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>plot([ax])</code>                  | Provide plot of region as <i>matplotlib.axes.Axes</i> object.  |
| <code>symmetric_difference(other)</code> | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                                |  |
|--------------------------------|--|
| <code>bounding_box</code>      | A region describing the minimum axis-aligned bounding box that encloses the original region. |
| <code>bounds</code>            | Region bounds <code>min_x, min_y, ..., max_x, max_y, .</code>                                |
| <code>centroid</code>          | Point coordinates for region centroid.   |
| <code>dimension</code>         | The region dimension.  |
| <code>extent</code>            | The extent ( <code>max_x - min_x</code> ), ( <code>max_y - min_y</code> ), <code>.</code>    |
| <code>holes</code>             | Holes where each hole is specified by polygon points.  |
| <code>max_distance</code>      | The maximum distance between any two points within the region.                               |
| <code>points</code>            | Exterior polygon points.   |
| <code>region_measure</code>    | Region measure, i.e. area (for 2d) or volume (for 3d).                                       |
| <code>region_specs</code>      | Legacy interface to serve legacy <i>RoiRegion</i> .  |
| <code>shapely_object</code>    | Geometric object as defined in <i>shapely</i> .  |
| <code>subregion_measure</code> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).      |

`as_artist(origin=(0, 0), **kwargs)`

Matplotlib 2D patch object for this region (e.g. *matplotlib.patches.Ellipse*).

**Parameters**

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The (x, y) pixel position of the origin of the displayed image. Default is (0, 0).
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

**Returns**

Matplotlib patch for the specified region.

**Return type**

matplotlib.patches.Patch

**property centroid:** ndarray[Any, dtype[float64]]

Point coordinates for region centroid.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

**Parameters**

**points** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Coordinates of points that are tested for being inside the specified region.

**Returns**

Array with indices for all points in original point array that are within the region.

**Return type**

npt.NDArray[np.int64]

**classmethod from\_shapely**(*polygon*)

Constructor for instantiating Region from *shapely* object.

**Parameters**

**shapely\_object** – Geometric object to be converted into Region

**Return type**

*Polygon* | *MultiPolygon* | *EmptyRegion*

**property holes:** list[ndarray[Any, dtype[float64]]] | None

Holes where each hole is specified by polygon points.

**Returns**

n\_holes of shape(n\_points, dimension)

**Return type**

list[npt.NDArray[np.float64]] | None

**property max\_distance: float**

The maximum distance between any two points within the region.

**Return type**

float

**property points: ndarray[Any, dtype[float64]]**

Exterior polygon points.

**Returns**

of shape(n\_points, dimension)

**Return type**

npt.NDArray[np.float64]

**property region\_measure: float**

Region measure, i.e. area (for 2d) or volume (for 3d).

**Return type**

float

**property region\_specs**

Legacy interface to serve legacy RoiRegion.

|  |
|--|
| <b>Warning:</b> Do not use - will be deprecated. |
|--|

**Return type**

dict[str, Any]

**property shapely\_object: Polygon**

Geometric object as defined in *shapely*.

**Return type**

Shapely object

**property subregion\_measure: float**

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

## locan.data.region.Rectangle

**class** locan.data.region.Rectangle(*corner=(0, 0), width=1, height=1, angle=0*)

Bases: [Region2D](#)

Region class to define a rectangle.

**Parameters**

- **corner** (*npt.ArrayLike*) – A point that defines the lower left corner with shape (2,).
- **width** (*float*) – The length of a vector describing the edge in x-direction.



- **height** (*float*) – The length of a vector describing the edge in y-direction.
- **angle** (*float*) – The angle (in degrees) by which the rectangle is rotated counterclockwise around the corner point.

## Methods

|   |  |
|---|--|
| <code>__init__([corner, width, height, angle])</code> |  |
| <code>as_artist([origin])</code>                      | Matplotlib 2D patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ).                  |
| <code>buffer(distance, **kwargs)</code>               | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>                         | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>                | Constructor for instantiating Region from list of (min, max) bounds.                                   |
| <code>from_shapely(shapely_object)</code>             | Constructor for instantiating Region from <i>shapely</i> object.                                       |
| <code>intersection(other)</code>                      | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>plot([ax])</code>                               | Provide plot of region as <i>matplotlib.axes.Axes</i> object.  |
| <code>symmetric_difference(other)</code>              | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                             | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                           |  |
|---------------------------|--|
| <i>angle</i>              | The angle (in degrees) by which the rectangle is rotated counterclockwise around the corner point.       |
| <code>bounding_box</code> | A region describing the minimum axis-aligned bounding box that encloses the original region.             |
| <code>bounds</code>       | Region bounds <code>min_x</code> , <code>min_y</code> , ..., <code>max_x</code> , <code>max_y</code> , . |
| <i>centroid</i>           | Point coordinates for region centroid.   |
| <i>corner</i>             | A point that defines the lower left corner.  |
| <code>dimension</code>    | The region dimension.  |
| <code>extent</code>       | The extent ( <code>max_x - min_x</code> ), ( <code>max_y - min_y</code> ), .                             |
| <i>height</i>             | The length of a vector describing the edge in y-direction.   |
| <i>intervals</i>          | Provide bounds in a tuple (min, max) arrangement.  |
| <i>max_distance</i>       | The maximum distance between any two points within the region.   |
| <i>points</i>             | Point coordinates.   |
| <i>region_measure</i>     | Region measure, i.e. area (for 2d) or volume (for 3d).   |
| <i>region_specs</i>       | Legacy interface to serve legacy RoiRegion.  |
| <i>shapely_object</i>     | Geometric object as defined in <i>shapely</i> .  |
| <i>subregion_measure</i>  | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).                  |
| <i>width</i>              | The length of a vector describing the edge in x-direction.   |

**property angle:** float

The angle (in degrees) by which the rectangle is rotated counterclockwise around the corner point.

**Return type**

float

**as\_artist**(*origin*=(0, 0), *\*\*kwargs*)

Matplotlib 2D patch object for this region (e.g. *matplotlib.patches.Ellipse*).

**Parameters**

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The (x, y) pixel position of the origin of the displayed image. Default is (0, 0).
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

**Returns**

Matplotlib patch for the specified region.

**Return type**

matplotlib.patches.Patch

**property centroid:** ndarray[Any, dtype[float64]]

Point coordinates for region centroid.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

**Parameters**

**points** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]]]) – Coordinates of points that are tested for being  
inside the specified region.

**Returns**

Array with indices for all points in original point array that are within the region.

**Return type**

npt.NDArray[np.int64]

**property corner:** ndarray[Any, dtype[float64]]

A point that defines the lower left corner.

**Returns**

with shape (2,)

**Return type**

npt.NDArray[np.float64]

**classmethod from\_intervals**(*intervals*)

Constructor for instantiating Region from list of (min, max) bounds.

**Parameters**

**intervals** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]]]) – The region bounds for each dimension of shape  
(2, 2)

**Return type**

cls

**property height:** float

The length of a vector describing the edge in y-direction.

**Return type**

float

**property intervals:** ndarray[Any, dtype[float64]]

Provide bounds in a tuple (min, max) arrangement.

**Returns**

((min\_x, max\_x), ...) of shape(dimension, 2)

**Return type**

npt.NDArray[np.float64]

**property max\_distance:** float

The maximum distance between any two points within the region.

**Return type**

float

**property points:** ndarray[Any, dtype[float64]]

Point coordinates.

**Returns**

of shape (n\_points, dimension)

**Return type**

npt.NDArray[np.float64] | list[npt.NDArray[np.float64]]

**property region\_measure:** float

Region measure, i.e. area (for 2d) or volume (for 3d).

**Return type**

float

**property region\_specs**

Legacy interface to serve legacy RoiRegion.

**Warning:** Do not use - will be deprecated.

**Return type**

dict

**property shapely\_object:** Polygon

Geometric object as defined in *shapely*.

**Return type**

Shapely object

**property subregion\_measure:** float

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

**property width:** float

The length of a vector describing the edge in x-direction.

**Return type**

float

## locan.data.region.Region

**class** locan.data.region.Region

Bases: ABC

Abstract Region class to define the interface for Region-derived classes that specify geometric objects to represent regions of interest.

### Methods

|  |  |
|--|--|
| <code>__init__()</code>                  |  |
| <code>buffer(distance, **kwargs)</code>  | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>            | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>   | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>intersection(other)</code>         | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>symmetric_difference(other)</code> | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                | Returns a region representing the union of this region with <i>other</i> .                             |

### Attributes

|                                |  |
|--------------------------------|--|
| <code>bounding_box</code>      | A region describing the minimum axis-aligned bounding box that encloses the original region. |
| <code>bounds</code>            | Region bounds min_x, min_y, ..., max_x, max_y, .   |
| <code>centroid</code>          | Point coordinates for region centroid.   |
| <code>dimension</code>         | The region dimension.  |
| <code>extent</code>            | The extent (max_x - min_x), (max_y - min_y), .   |
| <code>max_distance</code>      | The maximum distance between any two points within the region.                               |
| <code>points</code>            | Point coordinates.   |
| <code>region_measure</code>    | Region measure, i.e. area (for 2d) or volume (for 3d).                                       |
| <code>subregion_measure</code> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).      |

**abstract property bounding\_box:** *Region*

A region describing the minimum axis-aligned bounding box that encloses the original region.

**Return type***Region***abstract property bounds:** `ndarray[Any, dtype[float64]] | None`Region bounds `min_x`, `min_y`, ..., `max_x`, `max_y`, ... for each dimension.**Returns**of shape  $(2 * \text{dimension},)$ **Return type**`npt.NDArray[np.float64] | None`**abstract buffer**(*distance*, *\*\*kwargs*)Extend the region perpendicular by a *distance*.**Parameters****distance** (float) – Distance by which the region is extended.**Returns**

The extended region.

**Return type***Region***abstract property centroid:** `ndarray[Any, dtype[float64]] | None`

Point coordinates for region centroid.

**Returns**of shape  $(\text{dimension},)$ **Return type**`npt.NDArray[np.float64] | None`**abstract contains**(*points*)

Return list of indices for all points that are inside the region of interest.

**Parameters****points** (`Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]]]`) – Coordinates of points that are tested for being inside the specified region.**Returns**

Array with indices for all points in original point array that are within the region.

**Return type**`npt.NDArray[np.int64]`**abstract property dimension:** `int | None`

The region dimension.

**Return type**`int | None`**abstract property extent:** `ndarray[Any, dtype[float64]] | None`The extent  $(\text{max}_x - \text{min}_x)$ ,  $(\text{max}_y - \text{min}_y)$ , ... for each dimension.

**Returns**

of shape (dimension,)

**Return type**

npt.NDArray[np.float64] | None

**static from\_intervals(intervals)**

Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds. Takes array-like intervals instead of interval to be consistent with *Rectangle.from\_intervals*.

**Parameters**

**intervals** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region bounds for each dimension of shape (dimension, 2).

**Return type**

*Interval* | *Rectangle* | *AxisOrientedCuboid* | *AxisOrientedHyperCuboid*

**abstract intersection(other)**

Returns a region representing the intersection of this region with other.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

**abstract property max\_distance: float**

The maximum distance between any two points within the region.

**Return type**

float

**abstract property points: ndarray[Any, dtype[float64]] | list[ndarray[Any, dtype[float64]]]**

Point coordinates.

**Returns**

of shape (n\_points, dimension)

**Return type**

npt.NDArray[np.float64] | list[npt.NDArray[np.float64]]

**abstract property region\_measure: float**

Region measure, i.e. area (for 2d) or volume (for 3d).

**Return type**

float

**abstract property subregion\_measure: float**

Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).

**Return type**

float

**abstract** `symmetric_difference(other)`

Returns the union of the two regions minus any areas contained in the intersection of the two regions.

**Parameters**

**other** (`Region`) – Other region

**Return type**

`Region`

**abstract** `union(other)`

Returns a region representing the union of this region with `other`.

**Parameters**

**other** (`Region`) – Other region

**Return type**

`Region`

## `locan.data.region.Region1D`

**class** `locan.data.region.Region1D`

Bases: `Region`

Abstract Region class to define the interface for 1-dimensional Region classes.

## Methods

|   |  |
|---|--|
| <code>__init__()</code>                         |  |
| <code>as_artist([origin])</code>                | Matplotlib 2D patch object for this region (e.g. <code>matplotlib.patches.Ellipse</code> ).            |
| <code>buffer(distance, **kwargs)</code>         | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>                   | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>          | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>intersection(<i>other</i>)</code>         | Returns a region representing the intersection of this region with <code>other</code> .                |
| <code>symmetric_difference(<i>other</i>)</code> | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(<i>other</i>)</code>                | Returns a region representing the union of this region with <code>other</code> .                       |



## Attributes

|                                |  |
|--------------------------------|--|
| <code>bounding_box</code>      | A region describing the minimum axis-aligned bounding box that encloses the original region.             |
| <code>bounds</code>            | Region bounds <code>min_x</code> , <code>min_y</code> , ..., <code>max_x</code> , <code>max_y</code> , . |
| <code>centroid</code>          | Point coordinates for region centroid.   |
| <code>dimension</code>         | The region dimension.  |
| <code>extent</code>            | The extent ( <code>max_x - min_x</code> ), ( <code>max_y - min_y</code> ), .                             |
| <code>max_distance</code>      | The maximum distance between any two points within the region.   |
| <code>points</code>            | Point coordinates.   |
| <code>region_measure</code>    | Region measure, i.e. area (for 2d) or volume (for 3d).   |
| <code>subregion_measure</code> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).                  |

**abstract** `as_artist(origin=(0, 0), **kwargs)`

Matplotlib 2D patch object for this region (e.g. `matplotlib.patches.Ellipse`).

### Parameters

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The (x, y) pixel position of the origin of the displayed image. Default is (0, 0).
- **kwargs** (Any) – Other parameters passed to the `matplotlib.patches` object.

### Returns

Matplotlib patch for the specified region.

### Return type

`matplotlib.patches.Patch`

**property** `dimension: int`

The region dimension.

### Return type

`int | None`

**intersection**(*other*)

Returns a region representing the intersection of this region with *other*.

### Parameters

**other** (*Region*) – Other region

### Return type

*Region*

**symmetric\_difference**(*other*)

Returns the union of the two regions minus any areas contained in the intersection of the two regions.

**Parameters****other** ([Region](#)) – Other region**Return type**[Region](#)**union**(*other*)Returns a region representing the union of this region with *other*.**Parameters****other** ([Region](#)) – Other region**Return type**[Region](#)**locan.data.region.Region2D****class** `locan.data.region.Region2D`Bases: [Region](#)

Abstract Region class to define the interface for 2-dimensional Region classes.

**Methods**

|   |  |
|---|--|
| <code>__init__()</code>                   |  |
| <code>as_artist([origin])</code>          | Matplotlib 2D patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ).                  |
| <code>buffer(distance, **kwargs)</code>   | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>             | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>    | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>from_shapely(shapely_object)</code> | Constructor for instantiating Region from <i>shapely</i> object.                                       |
| <code>intersection(other)</code>          | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>plot([ax])</code>                   | Provide plot of region as <code>matplotlib.axes.Axes</code> object.                                    |
| <code>symmetric_difference(other)</code>  | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                 | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                          |  |
|--------------------------|--|
| <i>bounding_box</i>      | A region describing the minimum axis-aligned bounding box that encloses the original region. |
| <i>bounds</i>            | Region bounds min_x, min_y, ..., max_x, max_y, .   |
| <i>centroid</i>          | Point coordinates for region centroid.   |
| <i>dimension</i>         | The region dimension.  |
| <i>extent</i>            | The extent (max_x - min_x), (max_y - min_y), .   |
| <i>max_distance</i>      | The maximum distance between any two points within the region.                               |
| <i>points</i>            | Point coordinates.   |
| <i>region_measure</i>    | Region measure, i.e. area (for 2d) or volume (for 3d).                                       |
| <i>shapely_object</i>    | Geometric object as defined in <i>shapely</i> .  |
| <i>subregion_measure</i> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).      |

**abstract as\_artist**(*origin*=(0, 0), *\*\*kwargs*)

Matplotlib 2D patch object for this region (e.g. *matplotlib.patches.Ellipse*).

### Parameters

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The (x, y) pixel position of the origin of the displayed image. Default is (0, 0).
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

### Returns

Matplotlib patch for the specified region.

### Return type

*matplotlib.patches.Patch*

**property bounding\_box:** *Rectangle*

A region describing the minimum axis-aligned bounding box that encloses the original region.

### Return type

*Region*

**property bounds:** *ndarray[Any, dtype[float64]]*

Region bounds min\_x, min\_y, ..., max\_x, max\_y, ... for each dimension.

### Returns

of shape (2 \* dimension,)

### Return type

*npt.NDArray[np.float64]* | None

**buffer**(*distance*, *\*\*kwargs*)

Extend the region perpendicular by a *distance*.

**Parameters**

- **distance** (float) – Distance by which the region is extended.
- **kwargs** (Any) – Other parameters passed to `shapely.geometry.buffer()`.

**Returns**

The extended region.

**Return type**

*Polygon* | *MultiPolygon* | *EmptyRegion*

**property dimension: int**

The region dimension.

**Return type**

int | None

**property extent: ndarray[Any, dtype[float64]]**

The extent (max\_x - min\_x), (max\_y - min\_y), ... for each dimension.

**Returns**

of shape (dimension,)

**Return type**

`npt.NDArray[np.float64]` | None

**static from\_shapely**(*shapely\_object*)

Constructor for instantiating Region from *shapely* object.

**Parameters**

**shapely\_object** (*Polygon* | *MultiPolygon*) – Geometric object to be converted into Region

**Return type**

*Polygon* | *MultiPolygon* | *EmptyRegion*

**intersection**(*other*)

Returns a region representing the intersection of this region with *other*.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

**plot**(*ax=None*, *\*\*kwargs*)

Provide plot of region as `matplotlib.axes.Axes` object.

**Parameters**

- **ax** (Optional[*Axes*]) – The axes on which to show the image
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

**Returns**

Axes object with the plot.

**Return type**

matplotlib.axes.Axes

**abstract property shapely\_object:** Any

Geometric object as defined in *shapely*.

**Return type**

Shapely object

**symmetric\_difference**(*other*)

Returns the union of the two regions minus any areas contained in the intersection of the two regions.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

**union**(*other*)

Returns a region representing the union of this region with **other**.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

## **locan.data.region.Region3D**

**class** locan.data.region.Region3D

Bases: *Region*

Abstract Region class to define the interface for 3-dimensional Region classes.

## Methods

|  |  |
|--|--|
| <code>__init__()</code>                  |  |
| <code>as_artist([origin])</code>         | Matplotlib patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ).                     |
| <code>buffer(distance, **kwargs)</code>  | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>            | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>   | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>intersection(other)</code>         | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>plot([ax])</code>                  | Provide plot of region as <code>matplotlib.axes.Axes</code> object.                                    |
| <code>symmetric_difference(other)</code> | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                | Returns a region representing the union of this region with <i>other</i> .                             |

## Attributes

|                                |  |
|--------------------------------|--|
| <code>bounding_box</code>      | A region describing the minimum axis-aligned bounding box that encloses the original region.             |
| <code>bounds</code>            | Region bounds <code>min_x</code> , <code>min_y</code> , ..., <code>max_x</code> , <code>max_y</code> , . |
| <code>centroid</code>          | Point coordinates for region centroid.   |
| <code>dimension</code>         | The region dimension.  |
| <code>extent</code>            | The extent ( <code>max_x - min_x</code> ), ( <code>max_y - min_y</code> ), .                             |
| <code>max_distance</code>      | The maximum distance between any two points within the region.   |
| <code>points</code>            | Point coordinates.   |
| <code>region_measure</code>    | Region measure, i.e. area (for 2d) or volume (for 3d).   |
| <code>subregion_measure</code> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).                  |

**abstract** `as_artist(origin=(0, 0, 0), **kwargs)`

Matplotlib patch object for this region (e.g. *matplotlib.patches.Ellipse*).

### Parameters

- **origin** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float,

`complex, str, bytes]]])` – The (x, y, z) pixel position of the origin of the displayed image. Default is (0, 0, 0).

- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

**Returns**

Matplotlib patch for the specified region.

**Return type**

`matplotlib.patches.Patch`

**property dimension: int**

The region dimension.

**Return type**

`int | None`

**intersection(*other*)**

Returns a region representing the intersection of this region with *other*.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

**plot(*ax=None, \*\*kwargs*)**

Provide plot of region as `matplotlib.axes.Axes` object.

**Parameters**

- **ax** (Optional[*Axes*]) – The axes on which to show the image
- **kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

**Returns**

*Axes* object with the plot.

**Return type**

`matplotlib.axes.Axes`

**symmetric\_difference(*other*)**

Returns the union of the two regions minus any areas contained in the intersection of the two regions.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

**union(*other*)**

Returns a region representing the union of this region with *other*.

**Parameters**

**other** (*Region*) – Other region

**Return type**

*Region*

**locan.data.region.RegionND****class** locan.data.region.RegionNDBases: [Region](#)

Abstract Region class to define the interface for n-dimensional Region classes.

**Methods**

|  |  |
|--|--|
| <code>__init__()</code>                  |  |
| <code>buffer(distance, **kwargs)</code>  | Extend the region perpendicular by a <i>distance</i> .   |
| <code>contains(points)</code>            | Return list of indices for all points that are inside the region of interest.                          |
| <code>from_intervals(intervals)</code>   | Constructor for instantiating axis-oriented, box-like Region from list of (min, max) bounds.           |
| <code>intersection(other)</code>         | Returns a region representing the intersection of this region with <i>other</i> .                      |
| <code>symmetric_difference(other)</code> | Returns the union of the two regions minus any areas contained in the intersection of the two regions. |
| <code>union(other)</code>                | Returns a region representing the union of this region with <i>other</i> .                             |

**Attributes**

|                                |  |
|--------------------------------|--|
| <code>bounding_box</code>      | A region describing the minimum axis-aligned bounding box that encloses the original region. |
| <code>bounds</code>            | Region bounds min_x, min_y, ..., max_x, max_y, .   |
| <code>centroid</code>          | Point coordinates for region centroid.   |
| <code>dimension</code>         | The region dimension.  |
| <code>extent</code>            | The extent (max_x - min_x), (max_y - min_y), .   |
| <code>max_distance</code>      | The maximum distance between any two points within the region.                               |
| <code>points</code>            | Point coordinates.   |
| <code>region_measure</code>    | Region measure, i.e. area (for 2d) or volume (for 3d).                                       |
| <code>subregion_measure</code> | Measure of the sub-dimensional region, i.e. circumference (for 2d) or surface (for 3d).      |

**intersection**(*other*)Returns a region representing the intersection of this region with *other*.**Parameters****other** ([Region](#)) – Other region



**Return type***Region***symmetric\_difference**(*other*)

Returns the union of the two regions minus any areas contained in the intersection of the two regions.

**Parameters**

**other** (*Region*) – Other region

**Return type***Region***union**(*other*)

Returns a region representing the union of this region with *other*.

**Parameters**

**other** (*Region*) – Other region

**Return type***Region***locan.data.region.RoiRegion**

**class** locan.data.region.**RoiRegion**(*region\_type*, *region\_specs*)

Bases: object

Deprecated Region object to specify regions of interest.

A region that defines a region of interest with methods for getting a printable representation (that can also be saved in a yaml file), for returning a matplotlib patch that can be shown in a graph, for finding points within the region.

**Warning:** This class is to be deprecated and should only be used to deal with legacy `_roi.yaml` files. Use `Region` classes instead.

**Parameters**

- **region\_type** (*str*) – A string indicating the roi shape. In 1D it can be *interval*. In 2D it can be either *rectangle*, *ellipse*, or closed *polygon*. In 2D it can also be *shapelyPolygon* or *shapelyMultiPolygon*. In 3D it can be either *cuboid* or *ellipsoid* or *polyhedron* (not implemented yet).
- **region\_specs** (*tuple*) – 1D rois are defined by the following tuple: \* interval: (start, stop) 2D rois are defined by the following tuples: \* rectangle: ((corner\_x, corner\_y), width, height, angle) with angle in degree \* ellipse: ((center\_x, center\_y), width, height, angle) with angle in degree \* polygon: ((point1\_x, point1\_y), (point2\_x, point2\_y), ..., (point1\_x, point1\_y)) \* shapelyPolygon: ((point\_tuples), ((hole\_tuples), ...)) \* shapelyMultiPolygon: (shapelyPolygon\_specs\_1, shapelyPolygon\_specs\_2, ...) 3D rois are defined by the following tuples: \* cuboid: ((corner\_x, corner\_y, corner\_z), length, width, height, angle\_1, angle\_2, angle\_3) \* ellipsoid: ((center\_x, center\_y,

center\_z), length, width, height, angle\_1, angle\_2, angle\_3) \* polyhedron:  
(...)

## Variables

- **region\_type** (*str*) – Type of region
- **region\_specs** (*tuple*) – Specifications for region
- **\_region** (*RoiRegion*) – RoiRegion instance for the specified region type.
- **polygon** (*tuple[npt.ArrayLike, ...]*) – Array of points for a closed polygon approximating the region of interest in clockwise orientation. The first and last point must be identical.
- **dimension** (*int*) – Spatial dimension of region
- **centroid** (*tuple[float, ...]*) – Centroid coordinates
- **max\_distance** (*npt.NDArray[np.float64]*) – Maximum distance between any two points in the region
- **region\_measure** (*float*) – Hull measure, i.e. area or volume
- **subregion\_measure** (*float*) – Measure of the sub-dimensional region, i.e. circumference or surface.

## Methods

|   |  |
|---|--|
| <code>__init__(region_type, region_specs)</code>    |  |
| <code>as_artist(**kwargs)</code>                    | Matplotlib patch object for this region (e.g. <i>matplotlib.patches.Ellipse</i> ). |
| <code>contains(points)</code>                       | Return list of indices for all points that are inside the region of interest.      |
| <code>from_shapely(region_type, shapely_obj)</code> |  |
| <code>to_shapely()</code>                           | Convert region to a polygon and return as shapely object.                          |

## Attributes

|                     |
|---------------------|
| <code>region</code> |
|---------------------|

**as\_artist**(*\*\*kwargs*)

Matplotlib patch object for this region (e.g. *matplotlib.patches.Ellipse*).

### Parameters

**kwargs** (Any) – Other parameters passed to the *matplotlib.patches* object.

### Returns

Matplotlib patch for the specified region.

**Return type**

matplotlib.patches.Patch

**contains**(*points*)

Return list of indices for all points that are inside the region of interest.

**Parameters**

**points** (*npt.ArrayLike*) – 2D or 3D coordinates of oints that are tested for being inside the specified region.

**Returns**

Array with indices for all points in original point array that are within the region.

**Return type**

npt.NDArray[np.int64]

**classmethod** **from\_shapely**(*region\_type, shapely\_obj*)

**property** **region**

**to\_shapely**()

Convert region to a polygon and return as shapely object.

**Return type**

shapely.Polygon

## locan.data.region\_utils

Utility functions for working with regions.

**See also:**

*locan.data.filter.select\_by\_region()*, *locan.data.properties.misc.distance\_to\_region()*, *locan.data.properties.misc.distance\_to\_region\_boundary()*

## Functions

|   |   |
|---|---|
| <i>expand_region</i> (region[, distance, support])      | Expand a region by <i>distance</i> .  |
| <i>regions_union</i> (regions)                          | Return the union of <i>regions</i> .  |
| <i>surrounding_region</i> (region[, distance, support]) | Define surrounding region by extending a region and returning the extended region excluding the input region. |

## locan.data.region\_utils.expand\_region

locan.data.region\_utils.**expand\_region**(*region, distance=100, support=None, \*\*kwargs*)

Expand a region by *distance*. If region contains a list of regions, the unification of all expanded regions is returned.

**Parameters**

- **region** (*Region*) – Original region(s)

- **distance** (int | float) – Distance by which the region is expanded orthogonal to its boundary.
- **support** (Optional[[Region](#)]) – A region defining the maximum outer boundary.
- **kwargs** (Any) – Other parameters passed to `shapely.geometry.buffer()` for `Region2D` objects.

**Return type**[Region](#)**locan.data.region\_utils.regions\_union**`locan.data.region_utils.regions_union(regions)`

Return the union of *regions*.

**Parameters**

**regions** (list[[Region](#)]) – Original region(s)

**Return type**[Region](#)**locan.data.region\_utils.surrounding\_region**`locan.data.region_utils.surrounding_region(region, distance=100, support=None, **kwargs)`

Define surrounding region by extending a region and returning the extended region excluding the input region. If region contains a list of regions, the unification of all extended regions is returned.

**Parameters**

- **region** ([Region](#)) – Original region(s)
- **distance** (int | float) – Distance by which the region is extended orthogonal to its boundary.
- **support** (Optional[[Region](#)]) – A region defining the maximum outer boundary.
- **kwargs** (Any) – Other parameters passed to `shapely.geometry.buffer()` for `Region2D` objects.

**Return type**[Region](#)

## locan.data.register

Register localization data.

This module registers localization data and provides transformation parameters to put other localization data in registry.

Parts of this code is adapted from <https://github.com/jungmannlab/picasso>. (MIT license, Copyright (c) 2016 Jungmann Lab, MPI of Biochemistry)

## Classes

|  |
|--|
| <i>Transformation</i> (matrix, offset) |
|--|

## locan.data.register.Transformation

**class** locan.data.register.**Transformation**(*matrix*, *offset*)

Bases: NamedTuple

## Methods

|  |  |
|--|--|
| <code>__init__()</code>                  |  |
| <code>count(value, /)</code>             | Return number of occurrences of value. |
| <code>index(value[, start, stop])</code> | Return first index of value.           |

## Attributes

|               |                          |
|---------------|--------------------------|
| <i>matrix</i> | Alias for field number 0 |
| <i>offset</i> | Alias for field number 1 |

**matrix:** ndarray[Any, dtype[float64]]

Alias for field number 0

**offset:** ndarray[Any, dtype[float64]]

Alias for field number 1

## Functions

|   |   |
|---|---|
| <code>register_cc(locdata, other_locdata[, bins, ...])</code> | Register <i>points</i> or coordinates in <i>locdata</i> by a cross-correlation algorithm.                       |
| <code>register_icp(locdata, other_locdata[, ...])</code>      | Register <i>points</i> or coordinates in <i>locdata</i> by an "Iterative Closest Point" algorithm using open3d. |

### locan.data.register.register\_cc

`locan.data.register.register_cc(locdata, other_locdata, bins=None, n_bins=None, bin_size=None, bin_edges=None, bin_range=None, **kwargs)`

Register *points* or coordinates in *locdata* by a cross-correlation algorithm.

#### Parameters

- **locdata** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], *LocData*)) – Localization data representing the source on which to perform the manipulation.
- **other\_locdata** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], *LocData*)) – Localization data representing the target.
- **bins** (UnionType[*Bins*, Axis, AxesTuple, None]) – Specific class specifying the bins.
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If None (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2,

5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.

- **kwargs** (Any) – Other parameters passed to `skimage.phase_cross_correlation()`.

#### Returns

Matrix and offset representing the optimized transformation.

#### Return type

*Transformation*

### `locan.data.register.register_icp`

```
locan.data.register.register_icp(locdata, other_locdata, matrix=None, offset=None,
                                pre_translation=None,
                                max_correspondence_distance=1000,
                                max_iteration=10000, verbose=True)
```

Register *points* or coordinates in *locdata* by an “Iterative Closest Point” algorithm using `open3d`.

#### Parameters

- **locdata** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], *LocData*]) – Localization data representing the source on which to perform the manipulation.
- **other\_locdata** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], *LocData*) – Localization data representing the target.
- **matrix** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], None]) – Transformation matrix with shape (d, d) used as initial value. If None the unit matrix is used.
- **offset** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], None]) – Translation vector with shape (d,) used as initial value. If None a vector of zeros is used.
- **pre\_translation** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], None]) – Values for translation of coordinates before registration.
- **max\_correspondence\_distance** (float) – Threshold distance for the icp algorithm. Parameter is passed to `open3d` algorithm.

- **max\_iteration** (int) – Maximum number of iterations. Parameter is passed to open3d algorithm.
- **verbose** (bool) – Flag indicating if transformation results are printed out.

**Returns**

Matrix and offset representing the optimized transformation.

**Return type**

*Transformation*

**locan.data.aggregate**

Aggregate localization data.

This module provides functions to bin LocData objects to form a histogram or image.

Specify bins through one of the parameters (*bins*, *bin\_edges*, *n\_bins*, *bin\_size*, *bin\_range*, *labels*) as further outlined in the documentation for [Bins](#).

**Classes**

|  |   |
|--|---|
| <i>Bins</i> ([bins, n_bins, bin_size, bin_edges, ...]) | Bin definitions to be used in histogram and render functions. |
|--|---|

**locan.data.aggregate.Bins**

**class** locan.data.aggregate.**Bins**(bins=None, n\_bins=None, bin\_size=None, bin\_edges=None, bin\_range=None, labels=None, extend\_range=None)

Bases: object

Bin definitions to be used in histogram and render functions. Bin edges are continuous, contiguous and monotonic. Bins can be instantiated from specifications for *bins* or *bin\_edges* or for one of *n\_bins* or *bin\_size* in combination with *bin\_range*. One and only one of (*bins*, *bin\_edges*, *n\_bins*, *bin\_size*) must be different from None in any instantiating function. To pass bin specifications to other functions use an instance of *Bins* or *bin\_edges*.

**Parameters**

- **bins** (*Bins* | *boost\_histogram.axis.Axis* | *boost\_histogram.axis.AxesTuple* | None) – Specific class specifying the bins.
- **bin\_edges** (*Sequence[float]* | *Sequence[Sequence[float]]* | None) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (*tuple[float, float]* | *Sequence[float]* | *Sequence[Sequence[float]]*) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2).
- **n\_bins** (*int* | *Sequence[int]* | None) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.



- **bin\_size** (*float* | *Sequence[*float*]* | *Sequence[*Sequence[*float*]*]* | *None*) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **labels** (*list[*str*]* | *None*) – Names for each bin axis with shape (dimension,)
- **extend\_range** (*bool* | *None*) – If for equally-sized bins the final bin\_edge is different from the maximum bin\_range, the last bin\_edge will be smaller than the maximum bin\_range but all bins are equally-sized (*None*); the last bin\_edge will be equal to the maximum bin\_range but bins are not equally-sized (*False*); the last bin\_edge will be larger than the maximum bin\_range but all bins are equally-sized (*True*).

If for variable-sized bins the final bin\_edge is different from the maximum bin\_range, the last bin\_edge will be smaller than the maximum bin\_range (*None*); the last bin\_edge will be equal to the maximum bin\_range (*False*); the last bin\_edge will be larger than the maximum bin\_range but taken from the input sequence (*True*).

#### Variables

- **dimension** (*int*) – The number of dimensions for which bins are provided.
- **bin\_range** (*tuple[tuple[*float*, *float*], ...]*) – Minimum and maximum edge for each dimension with shape (dimension, 2).
- **bin\_edges** (*tuple[npt.NDArray[np.float64], ...]*) – Array(s) with bin edges for each dimension with shape (dimension,)
- **n\_bins** (*tuple[int, ...]*) – Number of bins for each dimension with shape (dimension,)
- **bin\_size** (*tuple[*float*, ...]* | *tuple[npt.NDArray[np.float64], ...]*) – Size of bins for each dimension with shape (dimension,) or with shape (dimension, n\_bins).
- **bin\_centers** (*tuple[npt.NDArray[np.float64], ...]*) – Array(s) with bin centers for all or each dimension with shape (dimension,).
- **labels** (*list[*str*]* | *None*) – Names for each bin axis.
- **boost\_histogram\_axes** (*boost\_histogram.axis.AxesTuple*) – Axis definitions for boost-histogram

## Methods

|  |   |
|--|---|
| <code>__init__([bins, n_bins, bin_size, ...])</code> |   |
| <code>equalize_bin_size()</code>                     | Return a new instance of <i>Bins</i> with <code>bin_size</code> set equal to the first <code>bin_size</code> element in each dimension and <code>extend_range=None</code> . |

## Attributes

|                                   |   |
|-----------------------------------|---|
| <code>bin_centers</code>          |   |
| <code>bin_edges</code>            |   |
| <code>bin_range</code>            |   |
| <code>bin_size</code>             |   |
| <code>boost_histogram_axes</code> | Axis definitions for boost-histogram                      |
| <code>dimension</code>            |   |
| <code>is_equally_sized</code>     | True for each dimension if all bins are of the same size. |
| <code>labels</code>               |   |
| <code>n_bins</code>               |   |

**property** `bin_centers`: `tuple[ndarray[Any, dtype[float64]], ...]`

**property** `bin_edges`: `tuple[ndarray[Any, dtype[float64]], ...]`

**property** `bin_range`: `tuple[tuple[float, float], ...]`

**property** `bin_size`: `tuple[float, ...] | tuple[ndarray[Any, dtype[float64]], ...]`

**property** `boost_histogram_axes`: `AxesTuple`

Axis definitions for boost-histogram

**property** `dimension`: `int`

**equalize\_bin\_size()**

Return a new instance of *Bins* with `bin_size` set equal to the first `bin_size` element in each dimension and `extend_range=None`.

**Return type**

*Bins*

**property is\_equally\_sized:** `tuple[bool, ...]`

True for each dimension if all bins are of the same size.

**property labels:** `list[str] | None`

**property n\_bins:** `tuple[int, ...]`

## Functions

|  |   |
|--|---|
| <code>histogram(locdata[, loc_properties, ...])</code> | Make histogram of <code>loc_properties</code> (columns in <code>locdata.data</code> ) by binning all localizations or averaging other_property within each bin. |
| <code>is_array_like(anything)</code>                   | Return true if <i>anything</i> can be turned into a <code>numpy.ndarray</code> without creating elements of type object.  |

### locan.data.aggregate.histogram

`locan.data.aggregate.histogram(locdata, loc_properties=None, other_property=None, bins=None, n_bins=None, bin_size=None, bin_edges=None, bin_range=None)`

Make histogram of `loc_properties` (columns in `locdata.data`) by binning all localizations or averaging other\_property within each bin.

#### Parameters

- **locdata** (`LocData`) – Localization data.
- **loc\_properties** (`UnionType[str, Iterable[str], None]`) – Localization properties to be grouped into bins. If `None` The `coordinate_values` of `locdata` are used.
- **other\_property** (`Optional[str]`) – Localization property that is averaged in each pixel. If `None` localization counts are shown.
- **bins** (`UnionType[Bins, Axis, AxesTuple, None]`) – The bin specification as defined in [Bins](#)
- **bin\_edges** (`UnionType[Sequence[float], Sequence[Sequence[float]], None]`) – Bin edges for all or each dimension with shape (dimension, `n_bin_edges`).
- **bin\_range** (`Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]`) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If `None` (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (`UnionType[int, Sequence[int], None]`) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.

- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.

#### Returns

namedtuple('Histogram', "data bins labels")

#### Return type

(npt.NDArray[np.int64 | np.float64], *Bins*, list[str])

### locan.data.aggregate.is\_array\_like

locan.data.aggregate.is\_array\_like(*anything*)

Return true if *anything* can be turned into a numpy.ndarray without creating elements of type object.

Catches numpy.VisibleDeprecationWarning or ValueError when setting an array element with a sequence.

#### Parameters

**anything** (Any) – Anything to be classified as being array-like or not.

#### Return type

bool

### locan.data.filter

Filter localization data.

This module provides functions for filtering LocData objects. The functions take LocData as input and compute new LocData objects.

### Classes

|   |   |
|---|---|
| <i>Selector</i> (loc_property, activate, ...) | Define selection interval for a single localization property. |
|---|---|

## locan.data.filter.Selector

**class** locan.data.filter.Selector(*loc\_property*, *activate*, *lower\_bound*, *upper\_bound*)

Bases: object

Define selection interval for a single localization property.

### Parameters

- **loc\_property** (*str*) – Localization property
- **activate** (*bool*) – Indicator to apply the selection or not
- **lower\_bound** (*int* | *float*) – min fo selection interval
- **upper\_bound** (*int* | *float*) – max of selection interval

### Variables

- **loc\_property** (*str*) – Localization property
- **activate** (*bool*) – Indicator to apply the selection or not
- **lower\_bound** (*int* | *float*) – min fo selection interval
- **upper\_bound** (*int* | *float*) – max of selection interval
- **interval** ([Interval](#)) – Class with interval specifications
- **condition** (*str*) – specification turned into condition string

### Methods

|  |
|--|
| <code>__init__(loc_property, activate, ...)</code> |
|--|

### Attributes

|                               |
|-------------------------------|
| <code><i>condition</i></code> |
|-------------------------------|

|                                 |
|---------------------------------|
| <code><i>lower_bound</i></code> |
|---------------------------------|

|                                 |
|---------------------------------|
| <code><i>upper_bound</i></code> |
|---------------------------------|

**property condition:** `str`

**property lower\_bound:** `int` | `float`

**property upper\_bound:** `int` | `float`

## Functions

|  |  |
|--|--|
| <code>exclude_sparse_points(locdata[, ...])</code>             | Exclude localizations by thresholding a local density.   |
| <code>filter_condition(selectors)</code>                       | Get a condition string from selection specifications.  |
| <code>localizations_in_cluster_regions(locdata ...)</code>     | Identify localizations from <i>locdata</i> within the regions of all <i>collection</i> elements.               |
| <code>random_subset(locdata, n_points[, replace, seed])</code> | Take a random subset of localizations.   |
| <code>select_by_condition(locdata, condition)</code>           | Select by specifying conditions on data properties.  |
| <code>select_by_image_mask(locdata, mask)</code>               | Select by masking using a binary image(e.g. generated by thresholding a transmitted-light microscopy image).   |
| <code>select_by_region(locdata, region[, ...])</code>          | Select localizations from <i>locdata</i> that are within <i>region</i> and return a new <i>LocData</i> object. |

### locan.data.filter.exclude\_sparse\_points

`locan.data.filter.exclude_sparse_points(locdata, other_locdata=None, radius=50, min_samples=5)`

Exclude localizations by thresholding a local density.

A subset of localizations, that exhibit a small local density of localizations from *locdata* or alternatively from *other\_locdata*, is identified as noise and excluded. Noise is identified by using a nearest-neighbor search (`sklearn.neighbors.NearestNeighbors`) to find all localizations within a circle (sphere) of the given *radius*. If the number of localizations is below the threshold value *min\_samples*, the localization is considered to be noise.

The method identifies the same noise points as done by the clustering algorithm DBSCAN<sup>1</sup>.

#### Parameters

- **locdata** (*LocData*) – Specifying the localization data from which to exclude localization data.
- **other\_locdata** (Optional[*LocData*]) – Specifying the localization data on which to compute local density.
- **radius** (float) – Radius of a circle or sphere in which neighbors are identified (equivalent to epsilon in DBSCAN).
- **min\_samples** (int) – The minimum number of samples in the neighborhood that need to be found for each localization to not be identified as noise (equivalent to minPoints in DBSCAN).

#### Returns

All localizations except those identified as sparse (noise) points.

<sup>1</sup> Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, A density-based algorithm for discovering clusters in large spatial databases with noise. In: Evangelos Simoudis, Jiawei Han, Usama M. Fayyad (Hrsg.): Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press, 1996, S. 226-231, ISBN 1-57735-004-9.

**Return type***LocData***References****locan.data.filter.filter\_condition**`locan.data.filter.filter_condition(selectors)`

Get a condition string from selection specifications.

**Parameters**

**selectors** (Iterable[*Selector*]) – Specifications for loc\_property selections

**Return type**

str

**locan.data.filter.localizations\_in\_cluster\_regions**`locan.data.filter.localizations_in_cluster_regions(locdata, collection,  
hull_type=HullType.CONVEX_HULL)`

Identify localizations from *locdata* within the regions of all *collection* elements.

**Parameters**

- **locdata** (*LocData*) – Localization data that is tested for being inside the region
- **collection** (*LocData* | list[*LocData*]) – A set of Locdata objects collected in a collection or list.
- **hull\_type** (*HullType* | str) – The hull type for each LocData object that is used to define the region.

**Returns**

A collection of LocData objects with all elements of locdata contained by the region.

**Return type***LocData***locan.data.filter.random\_subset**`locan.data.filter.random_subset(locdata, n_points, replace=True, seed=None)`

Take a random subset of localizations.

**Parameters**

- **locdata** (*LocData*) – Specifying the localization data from which to select localization data.
- **n\_points** (int) – Number of localizations to randomly choose from locdata.
- **replace** (bool) – Indicate if sampling is with or without replacement

- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – Random number generation seed

**Returns**

A new instance of LocData carrying the subset of localizations.

**Return type**

*LocData*

### **locan.data.filter.select\_by\_condition**

`locan.data.filter.select_by_condition(locdata, condition)`

Select by specifying conditions on data properties.

**Parameters**

- **locdata** (*LocData*) – Specifying the localization data from which to select.
- **condition** (str) – Conditions as input in select method. More precise: query specifications to be used with pandas query.

**Returns**

A new instance of LocData referring to the specified dataset.

**Return type**

*LocData*

### **locan.data.filter.select\_by\_image\_mask**

`locan.data.filter.select_by_image_mask(locdata, mask)`

Select by masking using a binary image(e.g. generated by thresholding a transmitted-light microscopy image).

**Parameters**

- **locdata** (*LocData*) – specifying the localization data from which to select.
- **mask** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – binary image.

**Returns**

a new instance of Selection referring to the specified dataset.

**Return type**

*LocData*



## locan.data.filter.select\_by\_region

`locan.data.filter.select_by_region(locdata, region, loc_properties=None, reduce=True)`

Select localizations from *locdata* that are within *region* and return a new LocData object. Selection is with respect to *loc\_properties* or to localization coordinates that correspond to region dimension.

### Parameters

- **locdata** (*LocData*) – Localization data that is tested for being inside the region.
- **region** (*Region*) – Tested region
- **loc\_properties** (Optional[list[str]]) – Localization properties to be tested.
- **reduce** (bool) – Return the reduced LocData object or keep references alive.

### Returns

A new instance of LocData with all localizations within region of interest.

### Return type

*LocData*

---

**Note:** Points on boundary of regions are considered outside if region is a shapely object, but inside if region is a matplotlib (which is the case for RoiRegion) object.

---

## locan.data.transform

Transform localization data.

This module provides functions that take locdata as input, transform the localization data, and return a new LocData object.

### Submodules:

|                                 |  |
|---------------------------------|--|
| <i>bunwarpj</i>                 | Transform localization data with a BUnwarpJ transformation matrix. |
| <i>spatial_transformation</i>   | Transform localization data.                                       |
| <i>intensity_transformation</i> | Transform localization intensities.                                |

## locan.data.transform.bunwarpj

Transform localization data with a BUnwarpy transformation matrix.

This module provides functions to transform coordinates in LocData objects by applying a B-spline transformation as defined with the ImageJ/Fiji plugin [Bunwarpy](#)<sup>1,2</sup>.

## References

## Functions

|  |  |
|--|--|
| <code>bunwarpy</code> (locdata, matrix_path, pixel_size[, flip]) | Transform coordinates by applying a B-spline transformation as represented by a raw transformation matrix from Bunwarpy. |
|--|--|

## locan.data.transform.bunwarpy.bunwarpy

`locan.data.transform.bunwarpy.bunwarpy`(locdata, matrix\_path, pixel\_size, flip=False)

Transform coordinates by applying a B-spline transformation as represented by a raw transformation matrix from Bunwarpy.

### Parameters

- **locdata** (*LocData*) – specifying the localization data on which to perform the manipulation.
- **matrix\_path** (str | PathLike[Any]) – Path to file with a raw matrix from Bunwarpy.
- **pixel\_size** (tuple[float, float]) – Pixel sizes used to determine transition matrix in ImageJ
- **flip** (bool) – Flip locdata along x-axis before transformation

### Returns

New localization data with transformed coordinates.

### Return type

*LocData*

---

<sup>1</sup> I. Arganda-Carreras, C. O. S. Sorzano, R. Marabini, J.-M. Carazo, C. Ortiz-de Solorzano, and J. Kybic, “Consistent and Elastic Registration of Histological Sections using Vector-Spline Regularization”, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, volume 4241/2006, CVAMIA: Computer Vision Approaches to Medical Image Analysis, pages 85-95, 2006.

<sup>2</sup> C.Ó. Sánchez Sorzano, P. Thévenaz, M. Unser, “Elastic Registration of Biological Images Using Vector-Spline Regularization”, IEEE Transactions on Biomedical Engineering, vol. 52, no. 4, pages 652-663, 2005.

## locan.data.transform.spatial\_transformation

Transform localization data.

This module takes localization data and applies transformation procedures on coordinates or other properties.

### Functions

|  |  |
|--|--|
| <code>overlay</code> (locdatas[, centers, orientations])       | Translate locdatas to their common center and rotate according to their orientation.                                   |
| <code>standardize</code> (locdata[, loc_properties, ...])      | Transform locdata properties by centering to the mean and property-wise scaling to unit standard deviation (variance). |
| <code>transform_affine</code> (locdata[, matrix, offset, ...]) | Transform <i>points</i> or coordinates in <i>locdata</i> by an affine transformation.                                  |

## locan.data.transform.spatial\_transformation.overlay

`locan.data.transform.spatial_transformation.overlay`(*locdatas*, *centers*='centroid',  
*orientations*=None)

Translate locdatas to their common center and rotate according to their orientation.

#### Parameters

- **locdatas** (Iterable[*LocData*]) – Localization data to overlay.
- **centers** (UnionType[Iterable[Any], str, None]) – centers to which locdatas are translated. Must have the same length as locdatas. One of *centroid*, *ch*, 'bb', 'obb', or 'region'. If None, no translation is applied.
- **orientations** (UnionType[Sequence[UnionType[int, float, str, None]], str, None]) – Orientation value to use in degree. Must have the same length as locdatas. If str, it must be one of *orientation\_im*, *orientation\_obb*. If None, no rotation is applied.

#### Returns

Collection with transformed locdatas.

#### Return type

*LocData*

## locan.data.transform.spatial\_transformation.standardize

```
locan.data.transform.spatial_transformation.standardize(locdata,  
                                                         loc_properties=None,  
                                                         with_mean=True,  
                                                         with_std=True)
```

Transform locdata properties by centering to the mean and property-wise scaling to unit standard deviation (variance).

---

**Note:** This function makes use of :func:sklearn.preprocessing.scale and thus works with a biased estimator for the standard deviation.

---

### Parameters

- **locdata** (*LocData*) – Localization data to be standardized.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be standardized. If None The coordinate\_values of locdata are used.
- **with\_mean** (bool) – If True center to the mean.
- **with\_std** (bool) – If True scale to unit standard deviation (variance).

### Returns

New localization data with standardized properties.

### Return type

*LocData*

## locan.data.transform.spatial\_transformation.transform\_affine

```
locan.data.transform.spatial_transformation.transform_affine(locdata, matrix=None,  
                                                             offset=None,  
                                                             pre_translation=None,  
                                                             method='numpy')
```

Transform *points* or coordinates in *locdata* by an affine transformation.

### Parameters

- **locdata** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]], *LocData*]) – Localization data on which to perform  
the manipulation.
- **matrix** (Union[\_SupportsArray[dtype[Any]],  
\_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float,  
complex, str, bytes, \_NestedSequence[Union[bool, int, float,  
complex, str, bytes]], None]) – Transformation matrix. If None the unit  
matrix is used. Array with shape (ndim, ndim). If None the unit matrix is  
used.

- **offset** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], None]) – Translation vector. Array with shape (ndim,). If None a vector of zeros is used.
- **pre\_translation** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], None]) – Translation vector for coordinates applied before affine transformation. Array with shape (ndim,). The reverse translation is applied after the affine transformation.
- **method** (Literal['numpy', 'open3d']) – The method (i.e. library or algorithm) used for computation. One of ‘numpy’, ‘open3d’.

**Returns**

New localization data with transformed coordinates.

**Return type**

npt.NDArray[np.float64] | [LocData](#)

**locan.data.transform.intensity\_transformation**

Transform localization intensities.

Localization intensities can be given in counts, electrons or photons. This module provides often used transformation functions.

**Functions**

|   |   |
|---|---|
| <code>transform_counts_to_photons</code> (locdata[, ...]) | Convert camera analog-to-digital converter (ADC) counts into photo electrons. |
|---|---|

**locan.data.transform.intensity\_transformation.transform\_counts\_to\_photons**

`locan.data.transform.intensity_transformation.transform_counts_to_photons`(locdata, loc\_properties=None, meta-data=None)

Convert camera analog-to-digital converter (ADC) counts into photo electrons. Quantum efficiency at the detected wavelength and collection efficiency of the optical system are not taken into account here.

**Parameters**

- **locdata** ([LocData](#)) – Localization data on which to perform the manipulation.
- **loc\_properties** (UnionType[str, list[str], None]) – Localization properties to be converted. If None the *intensity* values of locdata are used.

- **metadata** (Optional[Camera]) – Camera metadata with attribute offset, gain, electrons\_per\_count. If None, locdata.meta.experiment.setups[0].optical\_units[0].detection.camera is used.

#### Returns

Localization data with converted intensity values.

#### Return type

*LocData*

### locan.data.cluster

This module provides functions for clustering localizations.

The functions take LocData as input and compute new LocData objects representing collections of clustered localizations.

#### Submodules:

|                   |  |
|-------------------|--|
| <i>clustering</i> | Methods for clustering localization data in LocData objects. |
| <i>utils</i>      | Utility methods for clustering locdata.                      |

### locan.data.cluster.clustering

Methods for clustering localization data in LocData objects.

#### Functions

|   |   |
|---|---|
| <i>cluster_by_bin</i> (locdata[, loc_properties, ...])    | Cluster localizations in locdata by binning all localizations with regard to <i>loc_properties</i> and collecting all localizations per bin as cluster. |
| <i>cluster_dbscan</i> (locdata[, eps, min_samples, ...])  | Cluster localizations in locdata using the dbscan clustering algorithm as implemented in sklearn.   |
| <i>cluster_hdbscan</i> (locdata[, min_cluster_size, ...]) | Cluster localizations in locdata using the hdbscan clustering algorithm.  |

**locan.data.cluster.clustering.cluster\_by\_bin**

```
locan.data.cluster.clustering.cluster_by_bin(locdata, loc_properties=None,
                                             min_samples=1, bins=None, n_bins=None,
                                             bin_size=None, bin_edges=None,
                                             bin_range=None, return_counts=False)
```

Cluster localizations in locdata by binning all localizations with regard to *loc\_properties* and collecting all localizations per bin as cluster.

**Parameters**

- **locdata** (*LocData*) – Localization data.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If None The coordinate\_values of locdata are used.
- **min\_samples** (int) – The minimum number of samples per bin to be considered as cluster.
- **bins** (UnionType[Bins, Axis, AxesTuple, None]) – The bin specification as defined in Bins
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Array of bin edges with shape (n\_bin\_edges,) or (dimension, n\_bin\_edges) for all or each dimension.
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins in units of locdata coordinate units for all or each dimension. 5 would describe bin\_size of 5 for all bins in all dimensions. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. To specify arbitrary sequence of *bin\_sizes* use *bin\_edges* instead.
- **bin\_range** (tuple[float, ...] | tuple[tuple[float, float], ...] | Literal['zero'] | None) – The data bin\_range to be taken into consideration for all or each dimension. ((min\_x, max\_x), (min\_y, max\_y), ...) bin\_range for each coordinate; for None (min, max) bin\_range are determined from data; for 'zero' (0, max) bin\_range with max determined from data.
- **return\_counts** (bool) – If true, n\_elements per bin are returned.

**Returns**

Tuple with bins, bin\_indices, collection of all generated selections (i.e. localization clusters), and counts per bin.

**Return type**

tuple[Bins | None, npt.NDArray[np.int64], *LocData*, npt.NDArray[np.int64] | None]

## locan.data.cluster.clustering.cluster\_dbscan

`locan.data.cluster.clustering.cluster_dbscan(locdata, eps=20, min_samples=5, loc_properties=None, **kwargs)`

Cluster localizations in locdata using the dbscan clustering algorithm as implemented in sklearn.

### Parameters

- **locdata** (*LocData*) – specifying the localization data on which to perform the manipulation.
- **eps** (float) – The maximum distance between two samples for them to be considered as in the same neighborhood.
- **min\_samples** (int) – The number of samples in a neighborhood for a point to be considered as a core point. This includes the point itself.
- **loc\_properties** (Optional[list[str]]) – The LocData properties to be used for clustering. If None, *locdata.coordinates* will be used.
- **kwargs** (Any) – Other parameters passed to *sklearn.cluster.DBSCAN*.

### Returns

A tuple with noise and cluster. The first LocData object is a selection of all localizations that are defined as noise, in other words all localizations that are not part of any cluster. The second LocData object is a LocData instance assembling all generated selections (i.e. localization cluster).

### Return type

tuple[*LocData*, *LocData*]

## locan.data.cluster.clustering.cluster\_hdbscan

`locan.data.cluster.clustering.cluster_hdbscan(locdata, min_cluster_size=5, loc_properties=None, allow_single_cluster=False, **kwargs)`

Cluster localizations in locdata using the hdbscan clustering algorithm.

### Parameters

- **locdata** (*LocData*) – Localization data on which to perform the manipulation.
- **loc\_properties** (Optional[list[str]]) – The LocData properties to be used for clustering. If None, *locdata.coordinates* will be used.
- **min\_cluster\_size** (int) – Minimum cluster size in HDBSCAN algorithm (default: 5)
- **allow\_single\_cluster** (bool) – If True, return single cluster (default: False)
- **kwargs** (Any) – Other parameters passed to *hdbscan.HDBSCAN*.

### Returns

A tuple with noise and cluster. The first LocData object is a selection of all localizations that are defined as noise, in other words all localizations that are not part



of any cluster. The second `LocData` object is a `LocData` instance assembling all generated selections (i.e. localization cluster).

#### Return type

tuple[*LocData*, *LocData*]

### locan.data.cluster.utils

Utility methods for clustering locdata.

#### Functions

|  |  |
|--|--|
| <i>serial_clustering</i> (locdata, algorithm, ...) | Run and analyse a series of clustering processes to identify optimal parameters. |
|--|--|

### locan.data.cluster.utils.serial\_clustering

`locan.data.cluster.utils.serial_clustering`(*locdata*, *algorithm*, *parameter\_lists*,  
\*\**kwargs*)

Run and analyse a series of clustering processes to identify optimal parameters.

#### Parameters

- **locdata** (*LocData*) – Localization data.
- **algorithm** (Callable[... , Any]) – The locan clustering algorithm to use on locdata.
- **parameter\_lists** (dict[str, Any]) – A dictionary with all parameter lists that are to be iterated. The keys should be identical to parameter names of the used algorithm.
- **kwargs** (Any) – Optional keyword arguments that are passed to the algorithm.

#### Returns

The first element is a `LocData` object with a selection of all localizations that are defined as noise. If noise is false this element will be None. The second element is a new `LocData` instance assembling all generated selections (i.e. localization cluster).

#### Return type

tuple[*LocData*, ...]

## locan.data.tracking

Track localizations.

This module provides functions for tracking localizations (i.e. clustering localization data in time). The functions take LocData as input and compute new LocData objects. It makes use of the trackpy package.

## Functions

|   |   |
|---|---|
| <code>link_locdata</code> (locdata[, search_range, memory]) | Track localizations, i.e. cluster localizations in time when nearby in successive frames. |
| <code>track</code> (locdata[, search_range, memory])        | Cluster (in time) localizations in LocData that are nearby in successive frames.          |

## locan.data.tracking.link\_locdata

`locan.data.tracking.link_locdata`(locdata, search\_range=40, memory=0, \*\*kwargs)

Track localizations, i.e. cluster localizations in time when nearby in successive frames. This function applies the trackpy linking method to LocData objects.

### Parameters

- **locdata** – Localization data on which to perform the manipulation.
- **search\_range** – The maximum distance features can move between frames, optionally per dimension
- **memory** – The maximum number of frames during which a feature can vanish, then reappear nearby, and be considered the same particle.
- **kwargs** – Other parameters passed to trackpy.link\_df().

### Returns

A series named 'Track' referring to the track number.

### Return type

pandas.Series[Any]

---

**Note:** In order to switch off the printout from `trackpy.link()` and increase performance use `trackpy.quiet()` to silence the logging outputs.

---

## locan.data.tracking.track

`locan.data.tracking.track(locdata, search_range=40, memory=0, **kwargs)`

Cluster (in time) localizations in LocData that are nearby in successive frames. Clustered localizations are identified by the trackpy linking method.

The new locdata object carries properties with the same name as the original *locata*. They are computed as sum for *intensity*, as the first value for *frame*, and as mean for all other properties.

### Parameters

- **locdata** – Localization data on which to perform the manipulation.
- **search\_range** – The maximum distance features can move between frames, optionally per dimension
- **memory** – The maximum number of frames during which a feature can vanish, then reappear nearby, and be considered the same particle.
- **kwargs** – Other parameters passed to trackpy.link\_df.

### Returns

A new LocData instance assembling all generated selections (i.e. localization cluster). A series named ‘Track’ referring to the track number.

### Return type

tuple[Locdata, pandas.Series[Any]]

---

**Note:** In order to switch off the printout from `trackpy.link()` and increase performance use `trackpy.quiet()` to silence the logging outputs.

---

## locan.data.metadata\_utils

Deal with metadata in LocData objects.

Functions to modify metadata in LocData objects.

## Functions

|  |   |
|--|---|
| <code>load_metadata_from_toml(path_or_file_like)</code>      | Turn toml file into protobuf message instances.                 |
| <code>merge_metadata([metadata, other_metadata])</code>      | Merge <i>other_metadata</i> into Locdata.meta.                  |
| <code>message_scheme(message)</code>                         | Provide message scheme with defaults including nested messages. |
| <code>metadata_from_toml_string(toml_string)</code>          | Turn toml string into protobuf message instances.               |
| <code>metadata_to_formatted_string(message, **kwargs)</code> | Get formatted string from Locdata.metadata.                     |

### `locan.data.metadata_utils.load_metadata_from_toml`

`locan.data.metadata_utils.load_metadata_from_toml(path_or_file_like)`

Turn toml file into protobuf message instances.

---

**Note:** Parses Timestamp elements from string '2022-05-14T06:58:00Z'. Parses Duration elements from int in nanoseconds.

---

#### Parameters

**path\_or\_file\_like** (UnionType[str, bytes, PathLike[Any], BinaryIO, None]) – File path or file-like for a TOML file.

#### Returns

Message instances with name as declared in toml file.

#### Return type

dict[str, google.protobuf.message.Message] | None

### `locan.data.metadata_utils.merge_metadata`

`locan.data.metadata_utils.merge_metadata(metadata=None, other_metadata=None)`

Merge *other\_metadata* into Locdata.meta.

#### Parameters

- **metadata** (Optional[Metadata]) – Original LocData metadata before modification
- **other\_metadata** (UnionType[Metadata, None, dict[str, Any], str, bytes, PathLike[Any], BinaryIO]) – Metadata to be merged.

#### Returns

Merged metadata

#### Return type

locan.data.metadata\_pb2.Metadata

### `locan.data.metadata_utils.message_scheme`

`locan.data.metadata_utils.message_scheme(message)`

Provide message scheme with defaults including nested messages.

#### Parameters

**message** (Message) – Protobuf message

#### Returns

A nested dictionary with all message fields including default values.

#### Return type

dict[str, Any]

### `locan.data.metadata_utils.metadata_from_toml_string`

`locan.data.metadata_utils.metadata_from_toml_string(toml_string)`

Turn toml string into protobuf message instances.

---

**Note:** Parses Timestamp elements from string '2022-05-14T06:58:00Z'. Parses Duration elements from int in nanoseconds.

---

#### Parameters

**toml\_string** (Optional[str]) – TOML string with metadata.

#### Returns

Message instances with name as declared in toml file.

#### Return type

dict[str, google.protobuf.message.Message] | None

### `locan.data.metadata_utils.metadata_to_formatted_string`

`locan.data.metadata_utils.metadata_to_formatted_string(message, **kwargs)`

Get formatted string from Locdata.metadata.

#### Parameters

- **message** (Message) – Protobuf message like `locan.data.metadata_pb2.Metadata`
- **kwargs** (Any) – Other kwargs that are passed to `google.protobuf.text_format.MessageToString()`.

#### Returns

Formatted metadata string.

#### Return type

str

### `locan.data.validation`

Validate localization data.

This module provides functions to validate LocData properties.

## Classes

---

*Collection*(\*args, \*\*kwargs)

---

### locan.data.validation.Collection

```
class locan.data.validation.Collection(*args, **kwargs)
```

Bases: Protocol

#### Methods

---

```
__init__(*args, **kwargs)
```

---

#### Attributes

---

*data*

---

*references*

---

*coordinate\_keys*

---

**coordinate\_keys:** list[str]

**data:** DataFrame

**references:** Iterable[*LocData*]

## 8.5 locan.datasets

Utility functions to deal with exemplary datasets.

The data is located in a separate repository <https://github.com/super-resolution/LocanDatasets>.

When calling a function the datasets are expected to reside in a directory specified by the *locan.constants.DATASETS\_DIR* variable. If the directory does not exist the exemplary files are downloaded from GitHub.

## Functions

|                                     |  |
|-------------------------------------|--|
| <code>load_npc(**kwargs)</code>     | Locdata representing nuclear pore complexes. |
| <code>load_tubulin(**kwargs)</code> | Locdata representing microtubules.           |

### 8.5.1 locan.datasets.load\_npc

`locan.datasets.load_npc(**kwargs)`

Locdata representing nuclear pore complexes.

The data was generated by dSTORM<sup>1</sup>. It shows the gp210 protein of the nuclear pore complex labeled with AlexaFluor647.

#### References

##### Parameters

**kwargs** (Any) – Parameters passed to `locan.load_asdf_file()`.

##### Return type

*LocData*

### 8.5.2 locan.datasets.load\_tubulin

`locan.datasets.load_tubulin(**kwargs)`

Locdata representing microtubules.

The data was generated by dSTORM<sup>1</sup>. It shows alpha-tubulin as part of microtubules within COS-7 cells. Tubulin was targeted by primary IgG-antibodies labeled with AlexaFluor647 (2.1 degree of labeling) and recorded over 75\_000 frames.

#### References

##### Parameters

**kwargs** (Any) – Parameters passed to `locan.load_rapidSTORM_file()`.

##### Return type

*LocData*

<sup>1</sup> Löschberger A, van de Linde S, Dabauvalle MC, Rieger B, Heilemann M, Krohne G, Sauer M., Super-resolution imaging visualizes the eightfold symmetry of gp210 proteins around the nuclear pore complex and resolves the central channel with nanometer resolution. J Cell Sci. 2012, 125:570-5, doi: 10.1242/jcs.098822.

<sup>1</sup> Dominic A. Helmerich, Gerti Beliu, and Markus Sauer, Multiple-Labeled Antibodies Behave Like Single Emitters in Photoswitching Buffer ACS Nano 2020, 14, 10, 12629–12641, DOI: 10.1021/acsnano.0c06099

## 8.6 locan.dependencies

Module to deal with required and optional dependencies.

Optional dependencies are defined in pyproject.toml.

In any module that requires an optional dependency the import should be conditioned:

```
if HAS_DEPENDENCY["package"]: import package
```

Any function that makes use of the optional dependency should be decorated with

```
@needs_package("package")
```

CONSTANTS

|                         |   |
|-------------------------|---|
| <i>INSTALL_REQUIRES</i> | List of required dependencies (PyPi package names)                            |
| <i>EXTRAS_REQUIRE</i>   | List of optional dependencies (PyPi package names)                            |
| <i>IMPORT_NAMES</i>     | A dictionary mapping PyPi package names to import names if they are different |
| <i>HAS_DEPENDENCY</i>   | A dictionary indicating if dependency is available.                           |

### 8.6.1 locan.dependencies.INSTALL\_REQUIRES

```
locan.dependencies.INSTALL_REQUIRES: set[str] = {'asdf', 'boost-histogram',
'cython', 'fast-histogram', 'lazy-loader', 'lmfit', 'matplotlib', 'networkx',
'numpy', 'pandas', 'protobuf', 'qtpy', 'ruamel', 'scikit-image',
'scikit-learn', 'scipy', 'shapely', 'tiffiff', 'tomli', 'tqdm',
'typing-extensions'}
```

List of required dependencies (PyPi package names)

### 8.6.2 locan.dependencies.EXTRAS\_REQUIRE

```
locan.dependencies.EXTRAS_REQUIRE: set[str] = {'PyQt5', 'PySide2', 'asv',
'black', 'build', 'colorcet', 'coverage', 'cupy', 'furo', 'h5py', 'hdbscan',
'ipython', 'locan', 'mpl-scatter-density', 'mypy', 'myst-nb', 'napari',
'open3d', 'open3d-cpu', 'pandas-stubs', 'pre-commit', 'pytest', 'pytest-qt',
'ray', 'requests', 'ruff', 'sphinx', 'sphinx-autodoc-typehints',
'sphinx-copybutton', 'trackpy', 'twine', 'types-protobuf', 'types-requests',
'types-tqdm'}
```

List of optional dependencies (PyPi package names)



### 8.6.3 locan.dependencies.IMPORT\_NAMES

```
locan.dependencies.IMPORT_NAMES = {'boost-histogram': 'boost_histogram',
'fast-histogram': 'fast_histogram', 'mpl-scatter-density':
'mpl_scatter_density', 'protobuf': 'google.protobuf', 'pytest-qt':
'pytestqt', 'ruamel': 'ruamel.yaml', 'scikit-image': 'skimage',
'scikit-learn': 'sklearn'}
```

A dictionary mapping PyPi package names to import names if they are different

### 8.6.4 locan.dependencies.HAS\_DEPENDENCY

```
locan.dependencies.HAS_DEPENDENCY = {'PyQt5': True, 'PySide2': False,
'asdf': True, 'asv': False, 'black': False, 'boost_histogram': True,
'build': True, 'colorcet': True, 'coverage': False, 'cupy': False,
'cython': True, 'fast_histogram': True, 'furo': True, 'google.protobuf':
True, 'h5py': True, 'hdbscan': True, 'ipython': False, 'lazy-loader':
False, 'lmfit': True, 'locan': True, 'matplotlib': True,
'mpl_scatter_density': True, 'mypy': False, 'myst-nb': False, 'napari':
True, 'networkx': True, 'numpy': True, 'open3d': True, 'open3d-cpu':
False, 'pandas': True, 'pandas-stubs': False, 'pre-commit': False,
'pytest': True, 'pytestqt': False, 'qt': True, 'qtpy': True, 'ray': True,
'requests': True, 'ruamel.yaml': True, 'ruff': False, 'scipy': True,
'shapely': True, 'skimage': True, 'sklearn': True, 'sphinx': True,
'sphinx-autodoc-typehints': False, 'sphinx-copybutton': False, 'tiffiff':
True, 'tomli': False, 'tqdm': True, 'trackpy': True, 'twine': False,
'types-protobuf': False, 'types-requests': False, 'types-tqdm': False,
'typing-extensions': False}
```

A dictionary indicating if dependency is available.

## Classes

|   |
|---|
| <b>QtBindings</b> (value[, names, module, qualname, Python bindings to interact with Qt.<br>...]) |
|---|

### 8.6.5 locan.dependencies.QtBindings

```
class locan.dependencies.QtBindings(value, names=None, *, module=None,
qualname=None, type=None, start=1,
boundary=None)
```

Bases: Enum

Python bindings to interact with Qt.

## Attributes

*NONE*

*PYSIDE2*

*PYQT5*

*PYSIDE6*

*PYQT6*

**NONE** = ''

**PYQT5** = 'pyqt5'

**PYQT6** = 'pyqt6'

**PYSIDE2** = 'pyside2'

**PYSIDE6** = 'pyside6'

## Functions

|   |   |
|---|---|
| <i>needs_package</i> (package[, import_names, ...]) | Function that returns a decorator to check for optional dependency. |
|---|---|

### 8.6.6 locan.dependencies.needs\_package

`locan.dependencies.needs_package(package, import_names=None, has_dependency=None)`

Function that returns a decorator to check for optional dependency.

#### Parameters

- **package** (str) – Package or dependency name that needs to be imported.
- **import\_names** (Optional[dict[str, str]]) – Mapping of package names onto import names.
- **has\_dependency** (Optional[dict[str, bool]]) – Dictionary with bool indicator if package (import name) is available.

#### Returns

A decorator that raises ImportError if package is not available.

#### Return type

callable

## 8.7 locan.gui

User interfaces.

This module provides functions and classes for using graphical user interfaces (GUI).

Functions provide a GUI based on QT if the QT backend and appropriate python bindings are available. Supported bindings are found in the enum class *locan.QtBindings*.

The configuration variable *locan.QT\_BINDING* declares which python binding to use if several are installed. If an environment variable *QT\_API* is defined it takes precedence over *locan.QT\_BINDING*.

If neither *locan.QT\_BINDING* nor *QT\_API* is defined, qtpy will choose the binding.

### 8.7.1 Submodules:

|           |   |
|-----------|---|
| <i>io</i> | Functions for user interaction with paths and file names. |
|-----------|---|

#### locan.gui.io

Functions for user interaction with paths and file names.

---

**Note:** These functions start a QT app and might interfere with previously started apps.

---

#### Functions

|   |  |
|---|--|
| <i>file_dialog</i> ([directory, message, filter])       | Select file names in a ui dialog.                |
| <i>set_file_path_dialog</i> ([directory, message, ...]) | Set file path (path/name.suffix) in a ui dialog. |

#### locan.gui.io.file\_dialog

`locan.gui.io.file_dialog(directory=None, message='Select a file...', filter='Text files (*.txt)'; All files (*'))`

Select file names in a ui dialog.

##### Parameters

- **directory** (Optional[str]) – directory path to start dialog in. If None the current directory is used.
- **message** (str) – Hint what to do
- **filter** (str) – filter for file type

##### Returns

list with file names or empty list

**Return type**

str | list[str]

**locan.gui.io.set\_file\_path\_dialog**

```
locan.gui.io.set_file_path_dialog(directory=None, message='Set a file path...', filter='All files (*)')
```

Set file path (path/name.suffix) in a ui dialog.

**Parameters**

- **directory** (Optional[str]) – directory path to start dialog in. If None the current directory is used.
- **message** (str) – Hint what to do
- **filter** (str) – filter for file type

**Returns**

new file path

**Return type**

str

## 8.8 locan.locan\_io

File input/output functions.

This module provides functions for file input and output of data related to single-molecule localization microscopy.

### 8.8.1 Submodules:

|                  |  |
|------------------|--|
| <i>files</i>     | File manager                             |
| <i>locdata</i>   | File input/output for localization data. |
| <i>utilities</i> | Utility functions for file input/output. |

**locan.locan\_io.files**

File manager

Identify, match, and group files to be batch-processed. The class Files is a wrapper for a pandas.DataFrame with selected methods to identify, match, and group file paths.

## Classes

|   |   |
|---|---|
| <code>Files(df, directory, exists, column)</code> | Wrapper for a <code>pandas.DataFrame</code> with selected methods to identify, match, and group file paths. |
|---|---|

### `locan.locan_io.files.Files`

```
class locan.locan_io.files.Files(df=None, directory=None, exists=True,
                                  column='file_path')
```

Bases: `object`

Wrapper for a `pandas.DataFrame` with selected methods to identify, match, and group file paths.

---

**Note:** Iteration and indexing is implemented in a way that integer indexing or iterating over the `Files` instance returns a single row (as `Series` or `namedtuple`). Slice indexing returns a new `Files` instance with selected rows.

---

#### Parameters

- **df** (`pd.DataFrame` | `dict[str, str]` | `None`) – file names
- **directory** (`str` | `os.PathLike[Any]` | `None`) – base directory
- **exists** (`bool`) – raise `FileExistsError` if file in *df* does not exist
- **column** (`str`) – key/column in *df* from which to take a file list

#### Variables

- **df** (`pd.DataFrame`) – dataframe carrying file paths
- **directory** (`Path`) – base directory

## Methods

|   |   |
|---|---|
| <code>__init__([df, directory, exists, column])</code>        |   |
| <code>add_glob([pattern, regex, column])</code>               | Search for file paths using glob and/or regex pattern in base directory and provide files in new <i>column</i> .  |
| <code>concatenate([files, directory, exists])</code>          | Concatenate the file lists from multiple File instances and set the base directory without further action.  |
| <code>exclude([stoplist, column, column_stoplist])</code>     | Exclude files in <i>self.df.column</i> according to stoplist.   |
| <code>from_glob([directory, pattern, regex, column])</code>   | Instantiate <i>Files</i> from a search with glob and/or regex patterns.   |
| <code>from_path([files, directory, column])</code>            | Instantiate <i>Files</i> from a collection of file paths.   |
| <code>group_identifiers()</code>                              | Get categories defined in <i>self.df.group</i> .  |
| <code>grouped()</code>  | Get groupby instance based on <i>group_identifiers</i> .  |
| <code>match_file_upstream([column, pattern, ...])</code>      | Find a matching file by applying <i>locan.find_file_upstream()</i> on each file in <i>self.df[column]</i> .   |
| <code>match_files(files[, column, other_column])</code>       | Add files in new column.  |
| <code>print_summary()</code>                                  | Print summary of <i>Files</i> .   |
| <code>set_group_identifier([name, pattern, glob, ...])</code> | Set <i>group_identifier name</i> for files in <i>column</i> as identified by string pattern and/or glob pattern and/or regex and keep them in column "group". |

**add\_glob**(*pattern*='\*.txt', *regex*=None, *column*='other\_file\_path')

Search for file paths using glob and/or regex pattern in base directory and provide files in new *column*.

A logging.warning is given if the number of found files and those in *self.df* are different.

### Parameters

- **pattern** (Optional[str]) – glob pattern passed to *Path.glob()*
- **regex** (Optional[str]) – regex pattern passed to *re.search()* and applied in addition to glob pattern
- **column** (str) – Name of column in *Files.df* carrying these files

### Return type

Self

**classmethod concatenate**(*files*=None, *directory*=None, *exists*=True)

Concatenate the file lists from multiple File instances and set the base directory without further action.

### Parameters

- **files** (Optional[Iterable[Files]]) – sequence with File instances

- **directory** (UnionType[str, PathLike[Any], None]) – new base directory
- **exists** (bool) – raise *FileExistsError* if file in files does not exist

**Return type**

*Files*

**exclude**(stoplist=None, column='file\_path', column\_stoplist='file\_path')

Exclude files in *self.df.column* according to stoplist.

**Parameters**

- **stoplist** (UnionType[*Files*, Iterable[bool | str | PathLike[Any]], None]) – Files to be excluded
- **column** (str) – key/column in *df* from which to exclude files
- **column\_stoplist** (str) – key/column in *stoplist* from which to take files

**Return type**

Self

**classmethod from\_glob**(directory=None, pattern='\*.txt', regex=None, column='file\_path')

Instantiate *Files* from a search with glob and/or regex patterns.

**Parameters**

- **pattern** (str) – glob pattern passed to *Path.glob()*
- **regex** (Optional[str]) – regex pattern passed to *re.search()* and applied in addition to glob pattern
- **directory** (UnionType[str, PathLike[Any], None]) – new base directory in which to search
- **column** (str) – Name of column in *Files.df* carrying these files

**Return type**

*Files*

**classmethod from\_path**(files=None, directory=None, column='file\_path')

Instantiate *Files* from a collection of file paths.

**Parameters**

- **files** (UnionType[Sequence[str | PathLike[Any]], str, PathLike[Any], None]) – sequence with File instances
- **directory** (UnionType[str, PathLike[Any], None]) – new base directory
- **column** (str) – Name of column in *Files.df* carrying these files

**Return type**

*Files*

**group\_identifiers**()

Get categories defined in *self.df.group*.

**Return type**

categories

**grouped()**

Get groupby instance based on group\_identifiers.

**Return type**

pandas.core.groupby.DataFrameGroupBy

**match\_file\_upstream**(*column='file\_path', pattern='\*.toml', regex=None, directory=None, other\_column='metadata'*)

Find a matching file by applying `locan.find_file_upstream()` on each file in `self.df[column]`.

**Parameters**

- **column** (str) – Name of column in *Files.df* carrying files to match
- **pattern** (Optional[str]) – glob pattern passed to `Path.glob()`
- **regex** (Optional[str]) – regex pattern passed to `re.search()` and applied in addition to glob pattern
- **directory** (UnionType[str, PathLike[Any], None]) – top directory in which to search
- **other\_column** (str) – Name of new column carrying files

**Return type**

Self

**match\_files**(*files, column='file\_path', other\_column='other\_file\_path'*)

Add files in new column.

A logging.warning is given if the number of files and those in `self.df` are different.

**Parameters**

- **files** – New file list
- **column** – Name of column in *Files.df* carrying files to match
- **other\_column** – Name of new column carrying files

**Return type**

Self

**print\_summary()**

Print summary of Files.

**Return type**

None

**set\_group\_identifier**(*name=None, pattern=None, glob=None, regex=None, column='file\_path'*)

Set group\_identifier *name* for files in *column* as identified by string pattern and/or glob pattern and/or regex and keep them in column “group”.

**Parameters**

- **name** (Optional[str]) – new group\_identifier
- **pattern** (Optional[str]) – string pattern
- **glob** (Optional[str]) – glob pattern passed to `Path.match()`



- **regex** (Optional[str]) – regex pattern
- **column** (str) – Name of column in *Files.df* carrying files to match

**Return type**

Self

**locan.locan\_io.locdata**

File input/output for localization data.

There are functions for reading the following file structures (with an indicator string in parenthesis - see also *locan.constants.FileType*):

- custom text file (CUSTOM)
- rapidSTORM file format (RAPIDSTORM)<sup>1</sup>
- Elyra file format (ELYRA)
- Thunderstorm file format (THUNDERSTORM)<sup>2</sup>
- asdf file format (ASDF)<sup>3</sup>
- Nanoimager file format (NANOIMAGER)
- rapidSTORM track file format (RAPIDSTORMTRACK)<sup>Page 464, 1</sup>
- smlm file format (SMLM)<sup>4, 5</sup>
- decode file (DECODE)<sup>6</sup>
- smap file format (SMAP)<sup>7</sup>

**References**

---

<sup>1</sup> Wolter S, Löschberger A, Holm T, Aufmkolk S, Dabauvalle MC, van de Linde S, Sauer M., rapidSTORM: accurate, fast open-source software for localization microscopy. Nat Methods 9(11):1040-1, 2012, doi: 10.1038/nmeth.2224

<sup>2</sup> M. Ovesný, P. Křížek, J. Borkovec, Z. Švindrych, G. M. Hagen. ThunderSTORM: a comprehensive ImageJ plugin for PALM and STORM data analysis and super-resolution imaging. Bioinformatics 30(16):2389-2390, 2014.

<sup>3</sup> Greenfield, P., Droettboom, M., & Bray, E., ASDF: A new data format for astronomy. Astronomy and Computing, 12: 240-251, 2015, doi:10.1016/j.ascom.2015.06.004

<sup>4</sup> Ouyang et al., Deep learning massively accelerates super-resolution localization microscopy. Nat. Biotechnol. 2018, doi:10.1038/nbt.4106.

<sup>5</sup> <https://github.com/imodpasteur/smlm-file-format>

<sup>6</sup> Artur Speiser, Lucas-Raphael Müller, Philipp Hoess, Ulf Matti, Christopher J. Obara, Wesley R. Legant, Anna Kreshuk, Jakob H. Macke, Jonas Ries, and Srinivas C. Turaga, Deep learning enables fast and dense single-molecule localization with high accuracy. Nat Methods 18: 1082–1090, 2021, doi:10.1038/s41592-021-01236-x

<sup>7</sup> Ries, J., SMAP: a modular super-resolution microscopy analysis platform for SMLM data. Nat Methods 17: 870–872, 2020, doi.org/10.1038/s41592-020-0938-1

**Submodules:**

|                        |  |
|------------------------|--|
| <i>io_locdata</i>      | File input/output for localization data.                       |
| <i>utilities</i>       | Utility functions for file input/output of localization data.  |
| <i>rapidstorm_io</i>   | File input/output for localization data in rapid-STORM files.  |
| <i>thunderstorm_io</i> | File input/output for localization data in Thunderstorm files. |
| <i>elyra_io</i>        | File input/output for localization data in Elyra files.        |
| <i>nanoimager_io</i>   | File input/output for localization data Nanoimager files.      |
| <i>asdf_io</i>         | File input/output for localization data in ASDF files          |
| <i>smlm_io</i>         | File input/output for localization data in SMLM files.         |
| <i>decode_io</i>       | File input/output for localization data in DECODE files.       |
| <i>smap_io</i>         | File input/output for localization data in SMAP files.         |

**locan.locan\_io.locdata.io\_locdata**

File input/output for localization data.

**Functions**

|   |  |
|---|--|
| <i>load_locdata</i> (path[, file_type, nrows])          | Load data from localization file as specified by type. |
| <i>load_txt_file</i> (path[, sep, columns, nrows, ...]) | Load localization data from a txt file.                |

**locan.locan\_io.locdata.io\_locdata.load\_locdata**

`locan.locan_io.locdata.io_locdata.load_locdata(path, file_type=1, nrows=None, **kwargs)`

Load data from localization file as specified by type.

This function is a wrapper for read functions for the various types of SMLM data.

**Parameters**

- **path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a localization data file to load.

- **file\_type** (int | str | locan.constants.FileType | locan.data.metadata\_pb2.Metadata) – Indicator for the file type. Integer or string should be according to locan.constants.FileType.
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded.
- **kwargs** (Any) – kwargs passed to the specific load function.

#### Returns

A new instance of LocData with all localizations.

#### Return type

*LocData*

### locan.locan\_io.locdata.io\_locdata.load\_txt\_file

`locan.locan_io.locdata.io_locdata.load_txt_file(path, sep=',', columns=None, nrows=None, property_mapping=None, convert=True, **kwargs)`

Load localization data from a txt file.

Locan column names are either supplied or read from the first line header.

#### Parameters

- **path** (str | os.PathLike[str] | SupportsRead[Any]) – File path for a localization file to load.
- **sep** (str) – separator between column values (Default: ',')
- **columns** (list[str] | None) – Locan column names. If None the first line is interpreted as header (Default: None).
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded (Default: None).
- **property\_mapping** (dict[str, str] | list[dict[str, str]] | None) – Mappings between column names and locan property names
- **convert** (bool) – If True convert types by applying type specifications in locan.constants.PROPERTY\_KEYS.
- **kwargs** (Any) – Other parameters passed to `pandas.read_csv()`.

#### Returns

A new instance of LocData with all localizations.

#### Return type

*LocData*

## locan.locan\_io.locdata.utilities

Utility functions for file input/output of localization data.

### Functions

|   |   |
|---|---|
| <code>convert_property_names(properties[, ...])</code>        | Convert property names to standard locan property names if a mapping is provided. |
| <code>convert_property_types(dataframe, types[, ...])</code>  | Convert data types according to the column-type mapping in types.                 |
| <code>open_path_or_file_like(path_or_file_like[, ...])</code> | Provide open-file context from <i>path_or_file_like</i> input.                    |

## locan.locan\_io.locdata.utilities.convert\_property\_names

`locan.locan_io.locdata.utilities.convert_property_names(properties, property_mapping=None)`

Convert property names to standard locan property names if a mapping is provided. Otherwise, leave the property name as is and throw a warning.

#### Parameters

- **properties** (Iterable[str]) – Properties to be converted
- **property\_mapping** (UnionType[dict[str, str], Iterable[dict[str, str]], None]) – Mappings between other property names and locan property names

#### Returns

Converted property names

#### Return type

list[str]

## locan.locan\_io.locdata.utilities.convert\_property\_types

`locan.locan_io.locdata.utilities.convert_property_types(dataframe, types, loc_properties=None)`

Convert data types according to the column-type mapping in types. If the target type is one of ‘integer’, ‘signed’, ‘unsigned’, ‘float’ then `pandas.to_numeric()` will be applied. Otherwise, if the target type is any type object like *int*, *str*, *np.float64* or similar then `pandas.astype()` will be applied.

#### Parameters

- **dataframe** (DataFrame) – Data to be converted
- **types** (Mapping[str, str | type]) – Mapping of loc\_properties to types
- **loc\_properties** (Optional[Iterable[str]]) – The columns in dataframe to be converted. If None, all columns will be converted according to types.

**Returns**

A copy of dataframe with converted types

**Return type**

pandas.DataFrame

**locan.locan\_io.locdata.utilities.open\_path\_or\_file\_like**

`locan.locan_io.locdata.utilities.open_path_or_file_like(path_or_file_like, mode='r', encoding=None)`

Provide open-file context from *path\_or\_file\_like* input.

**Parameters**

- **path\_or\_file\_like** (str | bytes | os.PathLike[Any] | int | SupportsRead[Any]) – Identifier for file
- **mode** (str) – same as in *open()*
- **encoding** (str | None) – same as in *open()*

**Return type**

context for file object

**locan.locan\_io.locdata.rapidstorm\_io**

File input/output for localization data in rapidSTORM files.

**Functions**

|   |  |
|---|--|
| <code>load_rapidSTORM_file(path[, nrows, convert])</code>   | Load data from a rapidSTORM single-molecule localization file.   |
| <code>load_rapidSTORM_header(path)</code>                   | Load xml header from a rapidSTORM single-molecule localization file and identify column names.         |
| <code>load_rapidSTORM_track_file(path[, nrows, ...])</code> | Load data from a rapidSTORM single-molecule localization file with tracked localizations.              |
| <code>load_rapidSTORM_track_header(path)</code>             | Load xml header from a rapidSTORM (track) single-molecule localization file and identify column names. |

### locan.locan\_io.locdata.rapidstorm\_io.load\_rapidSTORM\_file

```
locan.locan_io.locdata.rapidstorm_io.load_rapidSTORM_file(path, nrows=None,  
                                                           convert=True, **kwargs)
```

Load data from a rapidSTORM single-molecule localization file.

#### Parameters

- **path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded.
- **convert** (bool) – If True convert types by applying type specifications in locan.constants.PROPERTY\_KEYS.
- **kwargs** (Any) – Other parameters passed to *pandas.read\_csv()*.

#### Returns

A new instance of LocData with all localizations.

#### Return type

*LocData*

### locan.locan\_io.locdata.rapidstorm\_io.load\_rapidSTORM\_header

```
locan.locan_io.locdata.rapidstorm_io.load_rapidSTORM_header(path)
```

Load xml header from a rapidSTORM single-molecule localization file and identify column names.

#### Parameters

**path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.

#### Returns

A list of valid dataset property keys as derived from the identifiers.

#### Return type

list[str]

### locan.locan\_io.locdata.rapidstorm\_io.load\_rapidSTORM\_track\_file

```
locan.locan_io.locdata.rapidstorm_io.load_rapidSTORM_track_file(path, nrows=None,  
                                                                  convert=True,  
                                                                  collection=True,  
                                                                  min_localization_count=1,  
                                                                  **kwargs)
```

Load data from a rapidSTORM single-molecule localization file with tracked localizations.

#### Parameters

- **path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded.

- **convert** (bool) – If True convert types by applying type specifications in `locan.constants.PROPERTY_KEYS`.
- **collection** (bool) – If True a collection of all tracks is returned. If False `LocData` with center positions is returned.
- **min\_localization\_count** (int) – If collection is True, only clusters with at least *min\_localization\_count* localizations are loaded.
- **kwargs** (Any) – Other parameters passed to *pandas.read\_csv()*.

**Returns**

A new instance of `LocData` with all localizations/tracks as a collection.

**Return type**

*LocData*

**locan.locan\_io.locdata.rapidstorm\_io.load\_rapidSTORM\_track\_header**

`locan.locan_io.locdata.rapidstorm_io.load_rapidSTORM_track_header(path)`

Load xml header from a rapidSTORM (track) single-molecule localization file and identify column names.

**Parameters**

**path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.

**Returns**

A list of valid dataset property keys as derived from the identifiers.

**Return type**

tuple[list[str], list[str]]

**locan.locan\_io.locdata.thunderstorm\_io**

File input/output for localization data in Thunderstorm files.

**Functions**

|   |  |
|---|--|
| <code>load_thunderstorm_file(path[, nrows, convert])</code> | Load data from a Thunderstorm single-molecule localization file.                                 |
| <code>load_thunderstorm_header(path)</code>                 | Load csv header from a Thunderstorm single-molecule localization file and identify column names. |
| <code>save_thunderstorm_csv(locdata, path)</code>           | Save <code>LocData</code> attributes Thunderstorm-readable csv-file.                             |

### locan.locan\_io.locdata.thunderstorm\_io.load\_thunderstorm\_file

```
locan.locan_io.locdata.thunderstorm_io.load_thunderstorm_file(path, nrows=None,  
                                                                convert=True,  
                                                                **kwargs)
```

Load data from a Thunderstorm single-molecule localization file.

#### Parameters

- **path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded.
- **convert** (bool) – If True convert types by applying type specifications in locan.constants.PROPERTY\_KEYS.
- **kwargs** (Any) – Other parameters passed to *pandas.read\_csv()*.

#### Returns

A new instance of LocData with all localizations.

#### Return type

*LocData*

### locan.locan\_io.locdata.thunderstorm\_io.load\_thunderstorm\_header

```
locan.locan_io.locdata.thunderstorm_io.load_thunderstorm_header(path)
```

Load csv header from a Thunderstorm single-molecule localization file and identify column names.

#### Parameters

**path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.

#### Returns

A list of valid dataset property keys as derived from the identifiers.

#### Return type

list[str]

### locan.locan\_io.locdata.thunderstorm\_io.save\_thunderstorm\_csv

```
locan.locan_io.locdata.thunderstorm_io.save_thunderstorm_csv(locdata, path)
```

Save LocData attributes Thunderstorm-readable csv-file.

In the Thunderstorm csv-file format we store only localization data with Thunderstorm-readable column names.

#### Parameters

- **locdata** (LocData) – The LocData object to be saved.
- **path** (str | os.PathLike[Any] | SupportsWrite[Any]) – File path including file name to save to.

#### Return type

None



## locan.locan\_io.locdata.elyra\_io

File input/output for localization data in Elyra files.

### Functions

|  |   |
|--|---|
| <code>load_Elyra_file(path[, nrows, convert])</code> | Load data from a rapidSTORM single-molecule localization file.                                  |
| <code>load_Elyra_header(path)</code>                 | Load xml header from a Zeiss Elyra single-molecule localization file and identify column names. |

## locan.locan\_io.locdata.elyra\_io.load\_Elyra\_file

`locan.locan_io.locdata.elyra_io.load_Elyra_file(path, nrows=None, convert=True, **kwargs)`

Load data from a rapidSTORM single-molecule localization file.

#### Parameters

- **path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded.
- **convert** (bool) – If True convert types by applying type specifications in `locan.constants.PROPERTY_KEYS`.
- **kwargs** (Any) – Other parameters passed to `pandas.read_csv()`.

#### Returns

A new instance of `LocData` with all localizations.

#### Return type

*LocData*

---

**Note:** Data is loaded with encoding = ‘latin-1’ and only data before the first NUL character is returned. Additional information appended at the end of the file is thus ignored.

---

## locan.locan\_io.locdata.elyra\_io.load\_Elyra\_header

`locan.locan_io.locdata.elyra_io.load_Elyra_header(path)`

Load xml header from a Zeiss Elyra single-molecule localization file and identify column names.

#### Parameters

**path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.

#### Returns

A list of valid dataset property keys as derived from the identifiers.

**Return type**

list[str]

**locan.locan\_io.locdata.nanoimager\_io**

File input/output for localization data Nanoimager files.

**Functions**

|  |  |
|--|--|
| <code>load_Nanoimager_file</code> (path[, nrows, convert]) | Load data from a Nanoimager single-molecule localization file.                                 |
| <code>load_Nanoimager_header</code> (path)                 | Load csv header from a Nanoimager single-molecule localization file and identify column names. |

**locan.locan\_io.locdata.nanoimager\_io.load\_Nanoimager\_file**

`locan.locan_io.locdata.nanoimager_io.load_Nanoimager_file`(path, nrows=None, convert=True, \*\*kwargs)

Load data from a Nanoimager single-molecule localization file.

**Parameters**

- **path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded.
- **convert** (bool) – If True convert types by applying type specifications in `locan.constants.PROPERTY_KEYS`.
- **kwargs** (Any) – Other parameters passed to `pandas.read_csv()`.

**Returns**

A new instance of `LocData` with all localizations.

**Return type***LocData***locan.locan\_io.locdata.nanoimager\_io.load\_Nanoimager\_header**

`locan.locan_io.locdata.nanoimager_io.load_Nanoimager_header`(path)

Load csv header from a Nanoimager single-molecule localization file and identify column names.

**Parameters**

**path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a Nanoimager file to load.

**Returns**

A list of valid dataset property keys as derived from the Nanoimager identifiers.

**Return type**

list[str]

**locan.locan\_io.locdata.asdf\_io**

File input/output for localization data in ASDF files

**Functions**

|  |  |
|--|--|
| <code>load_asdf_file</code> (path[, nrows, convert]) | Load data from ASDF localization file.   |
| <code>save_asdf</code> (locdata, path)               | Save LocData attributes in an asdf file. |

**locan.locan\_io.locdata.asdf\_io.load\_asdf\_file**

`locan.locan_io.locdata.asdf_io.load_asdf_file`(path, nrows=None, convert=True)

Load data from ASDF localization file.

**Parameters**

- **path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded.
- **convert** (bool) – If True convert types by applying type specifications in `locan.constants.PROPERTY_KEYS`.

**Returns**

A new instance of LocData with all localizations.

**Return type***LocData***locan.locan\_io.locdata.asdf\_io.save\_asdf**

`locan.locan_io.locdata.asdf_io.save_asdf`(locdata, path)

Save LocData attributes in an asdf file.

In the Advanced Scientific Data Format (ASDF) file format we store metadata, properties and column names as human-readable yaml header. The data is stored as binary numpy.ndarray.

---

**Note:** Only selected LocData attributes are saved. Currently these are: ‘data’, ‘columns’, ‘properties’, ‘meta’.

---

**Parameters**

- **locdata** (LocData) – The LocData object to be saved.

- **path** (str | os.PathLike[Any] | SupportsWrite[Any]) – File path including file name to save to.

**Return type**

None

**locan.locan\_io.locdata.smlm\_io**

File input/output for localization data in SMLM files.

File specifications are provided at <https://github.com/imodpasteur/smlm-file-format/blob/master/specification.md>.

Code is adapted from [https://github.com/imodpasteur/smlm-file-format/blob/master/implementations/Python/smlm\\_file.py](https://github.com/imodpasteur/smlm-file-format/blob/master/implementations/Python/smlm_file.py). (MIT license)

**Functions**

|  |   |
|--|---|
| <i>load_SMLM_file</i> (path[, nrows, convert])   | Load data from a SMLM single-molecule localization file.  |
| <i>load_SMLM_header</i> (path)                   | Read header (manifest) from a SMLM single-molecule localization file and identify column names.             |
| <i>load_SMLM_manifest</i> (path)                 | Read manifest.json (version 0.2) from a SMLM single-molecule localization (zip) file.                       |
| <i>manifest_file_info_from_locdata</i> (locdata) | Prepare a manifest["file_info"] protobuf from locdata.  |
| <i>manifest_format_from_locdata</i> (locdata)    | Prepare a manifest["format"] protobuf from locdata.   |
| <i>manifest_from_locdata</i> (locdata[, ...])    | Prepare a manifest protobuf from locdata.   |
| <i>save_SMLM</i> (locdata, path[, manifest])     | Save LocData attributes in a SMLM single-molecule localization (zip) file with manifest.json (version 0.2). |

**locan.locan\_io.locdata.smlm\_io.load\_SMLM\_file**

`locan.locan_io.locdata.smlm_io.load_SMLM_file(path, nrows=None, convert=True)`

Load data from a SMLM single-molecule localization file.

**Parameters**

- **path** (Union[str, PathLike[Any], IO[Any]]) – File path for a file to load.
- **nrows** (Optional[int]) – The number of localizations to load from file. None means that all available rows are loaded.
- **convert** (bool) – If True convert types by applying type specifications in `locan.constants.PROPERTY_KEYS`.

**Returns**

A new instance of `LocData` with all localizations. Returns a list of `LocData` if multiple tables are found.

**Return type**

`LocData` | list[`LocData`]

**locan.locan\_io.locdata.smlm\_io.load\_SMLM\_header**

`locan.locan_io.locdata.smlm_io.load_SMLM_header(path)`

Read header (manifest) from a SMLM single-molecule localization file and identify column names.

**Parameters**

**path** (Union[str, PathLike[str], IO[Any]]) – File path for a file to load.

**Returns**

A list of dataset property keys as derived from the identifiers.

**Return type**

list[str]

**locan.locan\_io.locdata.smlm\_io.load\_SMLM\_manifest**

`locan.locan_io.locdata.smlm_io.load_SMLM_manifest(path)`

Read manifest.json (version 0.2) from a SMLM single-molecule localization (zip) file.

**Parameters**

**path** (Union[str, PathLike[str], IO[Any]]) – File path for a file to load.

**Returns**

manifest in json format

**Return type**

dict[str, Any]

**locan.locan\_io.locdata.smlm\_io.manifest\_file\_info\_from\_locdata**

`locan.locan_io.locdata.smlm_io.manifest_file_info_from_locdata(locdata)`

Prepare a manifest[“file\_info”] protobuf from locdata. The manifest holds metadata for smlm files.

**Parameters**

**locdata** (`LocData`) – The `LocData` object.

**Returns**

The manifest file information

**Return type**

`locan.locan_io.locdata.manifest_pb2.FileInfo`

### locan.locan\_io.locdata.smlm\_io.manifest\_format\_from\_locdata

locan.locan\_io.locdata.smlm\_io.manifest\_format\_from\_locdata(*locdata*)

Prepare a manifest[“format”] protobuf from locdata. The manifest holds metadata for smlm files.

#### Parameters

**locdata** (*LocData*) – The LocData object.

#### Returns

The manifest format

#### Return type

locan.locan\_io.locdata.manifest\_pb2.Format

### locan.locan\_io.locdata.smlm\_io.manifest\_from\_locdata

locan.locan\_io.locdata.smlm\_io.manifest\_from\_locdata(*locdata*,  
*return\_json\_string=False*)

Prepare a manifest protobuf from locdata. The manifest holds metadata for smlm files.

#### Parameters

- **locdata** (*LocData*) – The LocData object.
- **return\_json\_string** (*bool*) – Flag for returning json string

#### Returns

The manifest

#### Return type

locan.locan\_io.locdata.manifest\_pb2.Manifest | str

### locan.locan\_io.locdata.smlm\_io.save\_SMLM

locan.locan\_io.locdata.smlm\_io.save\_SMLM(*locdata*, *path*, *manifest=None*)

Save LocData attributes in a SMLM single-molecule localization (zip) file with manifest.json (version 0.2).

In the smlm file format we store metadata as a human- readable manifest. The data is stored as byte string.

---

**Note:** Only selected LocData attributes are saved. These are: ‘data’, ‘columns’, ‘meta’.

---

#### Parameters

- **locdata** (*LocData*) – The LocData object to be saved.
- **path** (*Union[str, PathLike[Any], IO[Any]]*) – File path including file name to save to.
- **manifest** (*Optional[Manifest]*) – Protobuf with manifest to use instead of an autogenerated manifest.

**Return type**

None

**locan.locan\_io.locdata.decode\_io**

File input/output for localization data in DECODE files.

**Functions**

|  |  |
|--|--|
| <code>load_decode_file</code> (path[, nrows, convert]) | Load data from a DECODE single-molecule localization file.                             |
| <code>load_decode_header</code> (path)                 | Load header from a DECODE single-molecule localization file and identify column names. |

**locan.locan\_io.locdata.decode\_io.load\_decode\_file**

`locan.locan_io.locdata.decode_io.load_decode_file`(path, nrows=None, convert=True)

Load data from a DECODE single-molecule localization file.

**Parameters**

- **path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path or file-like object for a file to load.
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded.
- **convert** (bool) – If True convert types by applying type specifications in `locan.constants.PROPERTY_KEYS`.

**Returns**

A new instance of `LocData` with all localizations.

**Return type***LocData***locan.locan\_io.locdata.decode\_io.load\_decode\_header**

`locan.locan_io.locdata.decode_io.load_decode_header`(path)

Load header from a DECODE single-molecule localization file and identify column names.

The hdf5 file should contain the following keys: <KeysViewHDF5 [‘data’, ‘decode’, ‘meta’]>

**Parameters**

**path** (str | os.PathLike | SupportsRead) – File path or file-like object for a file to load.

**Returns**

Tuple with identifiers, meta and decode sections. Identifiers are list of valid dataset property keys as derived from the DECODE identifiers.

**Return type**

tuple[list[str], dict[str, Any], dict[str, Any]]

**locan.locan\_io.locdata.smap\_io**

File input/output for localization data in SMAP files.

**Functions**

|  |  |
|--|--|
| <code>load_SMAP_file</code> (path[, nrows, convert]) | Load data from a SMAP single-molecule localization file.             |
| <code>load_SMAP_header</code> (path)                 | Identify column names from a SMAP single-molecule localization file. |
| <code>save_SMAP_csv</code> (locdata, path)           | Save LocData to SMAP-readable csv-file.                              |

**locan.locan\_io.locdata.smap\_io.load\_SMAP\_file**`locan.locan_io.locdata.smap_io.load_SMAP_file(path, nrows=None, convert=True)`

Load data from a SMAP single-molecule localization file.

**Parameters**

- **path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.
- **nrows** (int | None) – The number of localizations to load from file. None means that all available rows are loaded.
- **convert** (bool) – If True convert types by applying type specifications in `locan.constants.PROPERTY_KEYS`.

**Returns**

A new instance of LocData with all localizations.

**Return type***LocData***locan.locan\_io.locdata.smap\_io.load\_SMAP\_header**`locan.locan_io.locdata.smap_io.load_SMAP_header(path)`

Identify column names from a SMAP single-molecule localization file.

**Parameters**

**path** (str | os.PathLike[Any] | SupportsRead[Any]) – File path for a file to load.

**Returns**

A list of valid dataset property keys as derived from the identifiers.

**Return type**

list[str]



### locan.locan\_io.locdata.smap\_io.save\_SMAP\_csv

locan.locan\_io.locdata.smap\_io.save\_SMAP\_csv(*locdata*, *path*)

Save LocData to SMAP-readable csv-file.

In the csv-file file format we store only localization data with SMAP-readable column names.

#### Parameters

- **locdata** (LocData) – The LocData object to be saved.
- **path** (str | os.PathLike[Any] | SupportsWrite[Any]) – File path including file name to save to.

#### Return type

None

### locan.locan\_io.utilities

Utility functions for file input/output.

#### Functions

|  |   |
|--|---|
| <code>find_file_upstream</code> ( <i>sub_directory</i> , <i>pattern</i> [, ...]) | Search for first upstream parent of <i>sub_directory</i> that contains <i>pattern</i> . |
|--|---|

### locan.locan\_io.utilities.find\_file\_upstream

locan.locan\_io.utilities.find\_file\_upstream(*sub\_directory*, *pattern*, *regex*=None, *top\_directory*=None)

Search for first upstream parent of *sub\_directory* that contains *pattern*. Return first pattern found. Return None if no pattern has been found when parent equals directory.

#### Parameters

- **sub\_directory** (str | PathLike[str]) – Directory or file path to start with.
- **pattern** (Optional[str]) – glob pattern passed to `Path.glob()`
- **regex** (Optional[str]) – regex pattern passed to `re.search()` and applied in addition to glob pattern
- **top\_directory** (UnionType[str, PathLike[str], None]) – Directory in which to stop the search.

#### Return type

Path | None

## 8.9 locan.rois

Region of interest.

This module provides functions for managing regions of interest in localization data.

### 8.9.1 Submodules:

|            |   |
|------------|---|
| <i>roi</i> | The Roi class is an object that defines a region of interest within a localization dataset. |
|------------|---|

### locan.rois.roi

The Roi class is an object that defines a region of interest within a localization dataset. It is therefore related to region specifications and a unique LocData object.

The Roi object provides methods for saving all specifications to a yaml file, for loading them, and for returning LocData with localizations selected to be within the roi region.

### Classes

|   |  |
|---|--|
| <i>Roi</i> (region[, reference, loc_properties])      | Class for defining a region of interest for a referenced LocData object. |
| <i>RoiLegacy_0</i> (region_type, region_specs[, ...]) | Class for a region of interest on LocData (roi).                         |

### locan.rois.roi.Roi

**class** locan.rois.roi.Roi(region, reference=None, loc\_properties=None)

Bases: object

Class for defining a region of interest for a referenced LocData object.

#### Parameters

- **region** (*Region*) – Geometrical region of interest.
- **reference** (*LocData* | *dict* | *locan.data.metadata\_pb2.Metadata* | *locan.data.metadata\_pb2.File* | *None*) – Reference to localization data for which the region of interest is defined. It can be a LocData object, a reference to a saved SMLM file, or None for indicating no specific reference. When dict it must have keys *file\_path* and *file\_type*. When Metadata message it must have keys *file.path* and *file.type* for a path pointing to a localization file and an integer or string indicating the file type. Integer or string should be according to locan.constants.FileType.
- **loc\_properties** (*Sequence[str]* | *None*) – Localization properties in LocData object on which the region selection will be applied (for instance the *coordinate\_keys*).

## Variables

- **region** ([Region](#)) – Geometrical region of interest.
- **reference** ([LocData](#) | `locan.data.metadata_pb2.Metadata` | `None`) – Reference to localization data for which the region of interest is defined. It can be a `LocData` object, a reference to a saved SMLM file, or `None` for indicating no specific reference. When referencing a saved SMLM file, `reference` has attributes `file.path` and `file.type` for a path pointing to a localization file and an integer indicating the file type.
- **loc\_properties** (`tuple[str, ...]` | `None`) – Localization properties in `LocData` object on which the region selection will be applied (for instance the `coordinate_keys`).

## Methods

|   |  |
|---|--|
| <code>__init__(region[, loc_properties])</code> | <code>reference,</code>                            |
| <code>from_yaml(path)</code>                    | Read Roi object from yaml format.                  |
| <code>locdata([reduce])</code>                  | Localization data according to roi specifications. |
| <code>to_yaml([path])</code>                    | Save Roi object in yaml format.                    |

## Attributes

|                     |
|---------------------|
| <code>region</code> |
|---------------------|

### classmethod `from_yaml(path)`

Read Roi object from yaml format.

#### Parameters

**path** (`str` | `PathLike[Any]`) – Path for yaml file.

#### Return type

`TypeVar(T_Roi, bound= Roi)`

### `locdata(reduce=True)`

Localization data according to roi specifications.

The ROI is applied on `locdata` properties as specified in `self.loc_properties` or by taking the first applicable `locdata.coordinate_keys`.

#### Parameters

**reduce** (`bool`) – Return the reduced `LocData` object or keep references alive.

#### Returns

A new instance of `LocData` with all localizations within region of interest.

#### Return type

[LocData](#)

property region: *Region*

**to\_yaml**(*path=None*)

Save Roi object in yaml format.

#### Parameters

**path** (UnionType[str, PathLike[str], None]) – Path for yaml file. If None a roi file path is generated from the metadata.

#### Return type

None

## locan.rois.roi.RoiLegacy\_0

**class** locan.rois.roi.RoiLegacy\_0(*region\_type, region\_specs, reference=None, properties\_for\_roi=()*)

Bases: object

Class for a region of interest on LocData (roi).

Roi objects define a region of interest for a referenced LocData object.

#### Parameters

- **reference** (*LocData, dict, locan.data.metadata\_pb2.Metadata, None*) – Reference to localization data for which the region of interests are defined. It can be a LocData object, a reference to a saved SMLM file, or None for indicating no specific reference. When referencing a saved SMLM file, reference must be a dict or locan.data.metadata\_pb2.Metadata with keys *file\_path* and *file\_type* for a path pointing to a localization file and an integer or string indicating the file type. Integer or string should be according to locan.constants.FileType.
- **region\_type** (*str*) – A string indicating the roi shape. In 1D it can be *interval*. In 2D it can be either *rectangle*, *ellipse*, or closed *polygon*. In 3D it can be either *cuboid* or *ellipsoid* or *polyhedron* (not implemented yet).
- **region\_specs** (*tuple*) – 1D rois are defined by the following tuple: \* interval: (start, stop) 2D rois are defined by the following tuples: \* rectangle: ((corner\_x, corner\_y), width, height, angle) \* ellipse: ((center\_x, center\_y), width, height, angle) \* polygon: ((point1\_x, point1\_y), (point2\_x, point2\_y), ..., (point1\_x, point1\_y)) 3D rois are defined by the following tuples: \* cuboid: ((corner\_x, corner\_y, corner\_z), length, width, height, angle\_1, angle\_2, angle\_3) \* ellipsoid: ((center\_x, center\_y, center\_z), length, width, height, angle\_1, angle\_2, angle\_3) \* polyhedron: (...)
- **properties\_for\_roi** (*tuple[str, ...]*) – Localization properties in LocData object on which the region selection will be applied (for instance the *coordinate\_keys*).

#### Variables

- **reference** (*LocData / locan.data.metadata\_pb2.Metadata / None*) – Reference to localization data for which the regions of interest are defined. It can be a LocData object, a reference (locan.data.metadata\_pb2.Metadata) to a saved SMLM file, or None for

indicating no specific reference. When referencing a saved SMLM file, reference as attributes *file\_path* and *file\_type* for a path pointing to a localization file and an integer indicating the file type. The integer should be according to `locan.data.metadata_pb2.Metadata.file_type`.

- **\_region** (`RoiRegion` / `list[RoiRegion]`) – Object specifying the geometrical region of interest. In case a list of `RoiRegion` is provided it is the union that makes up the region of interest.
- **properties\_for\_roi** (`tuple[str, ...]`) – Localization properties in `LocData` object on which the region selection will be applied (for instance the `coordinate_keys`).

**Warning:** *RoiLegacy* is deprecated and should only be used to read legacy `_roi.yaml` files. Use `locan.Roi` instead.

## Methods

|   |  |
|---|--|
| <code>__init__(region_type, region_specs[, ...])</code> |  |
| <code>from_yaml(path)</code>                            | Read Roi object from yaml format.                  |
| <code>locdata([reduce])</code>                          | Localization data according to roi specifications. |
| <code>to_yaml([path])</code>                            | Save Roi object in yaml format.                    |

## Attributes

|                     |
|---------------------|
| <code>region</code> |
|---------------------|

**classmethod** `from_yaml(path)`

Read Roi object from yaml format.

### Parameters

**path** (`str` / `os.PathLike`) – Path for yaml file.

**locdata** (`reduce=True`)

Localization data according to roi specifications.

The ROI is applied on locdata properties as specified in `self.loc_properties` or by taking the first applicable `locdata.coordinate_keys`.

### Parameters

**reduce** (`bool`) – Return the reduced `LocData` object or keep references alive.

### Returns

A new instance of `LocData` with all localizations within region of interest.

### Return type

`LocData`

**property region**

**to\_yaml**(*path=None*)

Save Roi object in yaml format.

**Parameters**

**path** (*str* / *os.PathLike* / *None*) – Path for yaml file. If None a roi file path is generated from the metadata.

## Functions

|  |  |
|--|--|
| <b>rasterize</b> ( <i>locdata</i> [, <i>support</i> , <i>n_regions</i> , ...]) | Provide regions of interest by dividing the locdata support in equally sized rectangles. |
|--|--|

## locan.rois.roi.rasterize

**locan.rois.roi.rasterize**(*locdata*, *support=None*, *n\_regions*=(2, 2, 2), *loc\_properties*=())

Provide regions of interest by dividing the locdata support in equally sized rectangles.

**Parameters**

- **locdata** (*LocData*) – The localization data from which to select localization data.
- **support** (Optional[tuple[tuple[int], ...]]) – Coordinate intervals that are divided in *n\_regions* subintervals. For None intervals are taken from the bounding box.
- **n\_regions** (tuple[int, ...]) – Number of regions in each dimension. E.g. *n\_regions* = (2, 2) returns 4 rectangular Roi objects.
- **loc\_properties** (Iterable[str]) – Localization properties in LocData object on which the region selection will be applied. (Only implemented for coordinates labels)

**Returns**

A sequence of Roi objects

**Return type**

tuple[Roi, ...]

## 8.10 locan.scripts

Command-line utility scripts

The locan package provides a command-line interface. It is accessible by the base command **locan** and provides a number of compound commands. See available commands using **locan -h**.

This subpackage contains implementations of command-line scripts. Scripts are installed in **bin/** as simple wrappers for these modules. They can be run directly from a terminal through **locan command options** as long as the correct environment is activated.

### 8.10.1 Submodules:

|                                   |   |
|-----------------------------------|---|
| <code>script_check</code>         | Show original SMLM images overlaid with localization data.        |
| <code>script_rois</code>          | Define regions of interest with napari and save as roi files.     |
| <code>script_draw_roi</code>      | Define regions of interest with matplotlib and save as roi files. |
| <code>script_napari</code>        | Render localization data in napari.                               |
| <code>script_show_versions</code> | Show system information and dependency versions.                  |
| <code>script_test</code>          | Run project test suite with pytest.                               |

#### locan.scripts.script\_check

Show original SMLM images overlaid with localization data. Data is rendered in napari.

To run the script:

```
locan check <pixel size> -f <images file> -l <localization file> -t <file_
↪ type>
```

Try for instance:

```
locan check 133 -f "locan/tests/test_data/images.tif" -l "locan/tests/test_
↪ data/rapidStorm_from_images.txt" -t 2
```

### Functions

|   |  |
|---|--|
| <code>main([args])</code>                                     | <b>rtype</b><br>None   |
| <code>render_locs_per_frame_napari(images, ...[, ...])</code> | Display original recording and overlay localization spots in napari.                   |
| <code>sc_check(pixel_size[, file_images, ...])</code>         | Load and display original recording and load and overlay localization spots in napari. |

### locan.scripts.script\_check.main

locan.scripts.script\_check.main(*args=None*)

#### Return type

None

### locan.scripts.script\_check.render\_locs\_per\_frame\_napari

locan.scripts.script\_check.render\_locs\_per\_frame\_napari(*images, pixel\_size, locdata, viewer=None, transpose=True, kwargs\_image=None, kwargs\_points=None*)

Display original recording and overlay localization spots in napari.

#### Parameters

- **images** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Stack of raw data as recorded by camera.
- **pixel\_size** (float | tuple[float]) – Pixel size for images (in locdata units) with shape (2,).
- **transpose** (bool) – If True transpose x and y axis of *images*.
- **locdata** (*LocData*) – Localization data that corresponds to *images* raw data.
- **viewer** (Viewer) – The viewer object on which to add the image
- **kwargs\_image** (Optional[dict[str, Any]]) – Other parameters passed to napari.Viewer().add\_image().
- **kwargs\_points** (dict) – Other parameters passed to napari.Viewer().add\_points().

#### Returns

Viewer with the image.

#### Return type

napari.Viewer

### locan.scripts.script\_check.sc\_check

locan.scripts.script\_check.sc\_check(*pixel\_size, file\_images=None, file\_locdata=None, file\_type=FileType.RAPIDSTORM, viewer=None, transpose=True, kwargs\_image=None, kwargs\_points=None*)

Load and display original recording and load and overlay localization spots in napari.

#### Parameters



- **pixel\_size** (float | tuple[float]) – Pixel size for images (in locdata units) with shape (2,).
- **file\_images** (UnionType[str, PathLike[Any], None]) – File path for stack of raw data as recorded by camera.
- **file\_locdata** (UnionType[str, PathLike[Any], None]) – File path for localization data that corresponds to *images* raw data.
- **file\_type** (int | str | *FileType* | Metadata) – Indicator for the file type. Integer or string should be according to locan.constants.FileType.
- **transpose** (bool) – If True transpose x and y axis of *images*.
- **viewer** (*napari.Viewer*) – The viewer object on which to add the image
- **kwargs\_image** (*dict*) – Other parameters passed to *napari.Viewer.add\_image()*.
- **kwargs\_points** (*dict*) – Other parameters passed to *napari.Viewer.add\_points()*.

#### Return type

None

### locan.scripts.script\_rois

Define regions of interest with napari and save as roi files.

With this script you can choose a file name, open the localization file in napari. Draw regions of interest as additional shapes in napari. Upon closing napari each shape is taken as single roi and saved as *\_roi.yaml* file.

To run the script:

```
locan rois -f <file> -t <file type> -i <roi file indicator> --bin_size <>
--rescale <string with tuple or rescale>
```

Try for instance:

```
locan rois -f "locan/tests/test_data/five_blobs.txt" -t 1 -i "_roi" --bin_
↪size 10
```

See also:

*locan.visualize.render\_napari.utilities.select\_by\_drawing\_napari*

## Functions

```
main([args])
```

**rtype**  
None

```
sc_draw_roi_napari([file_path, file_type, ...]) Define regions of interest by drawing a boundary.
```

### locan.scripts.script\_rois.main

```
locan.scripts.script_rois.main(args=None)
```

**Return type**  
None

### locan.scripts.script\_rois.sc\_draw\_roi\_napari

```
locan.scripts.script_rois.sc_draw_roi_napari(file_path=None,  
                                              file_type=FileType.CUSTOM,  
                                              roi_file_indicator='_roi', **kwargs)
```

Define regions of interest by drawing a boundary.

#### Parameters

- **file\_path** (UnionType[str, PathLike[Any], None]) – File path to localization data.
- **file\_type** (int | str | *FileType* | Metadata) – Indicator for the file type. Integer or string should be according to locan.constants.FileType.
- **roi\_file\_indicator** (str) – Indicator to add to the localization file name and use as roi file name (with further extension .yaml).
- **kwargs** (Any) – Other parameters passed to `render_2d_napari()`.

#### Returns

File paths for all roi files

#### Return type

list[Path]

### locan.scripts.script\_draw\_roi

Define regions of interest with matplotlib and save as roi files.

With this script you can choose a file name, open the localization file and draw a rectangular region of interest. Within the matplotlib image draw a rectangle, type '+' to add the roi to the list, then type 'q' to quit. The roi is then saved as `_roi.yaml` file.

To run the script:

```
locan draw_roi_mpl -f <file> -t <file type> -i <roi file indicator> -r
↳<region type>
```

Try for instance:

```
locan draw_roi_mpl -f "locan/tests/test_data/five_blobs.txt" -t 1 -i "_roi" -
↳r "ellipse"
```

See also:

`locan.rois.select_by_drawing_mpl`

## Functions

|   |   |
|---|---|
| <code>main([args])</code>                                 |   |
|   | <b>rtype</b><br>None                              |
| <code>sc_draw_roi_mpl([file_path, file_type, ...])</code> | Define regions of interest by drawing a boundary. |

### locan.scripts.script\_draw\_roi.main

`locan.scripts.script_draw_roi.main(args=None)`

**Return type**  
None

### locan.scripts.script\_draw\_roi.sc\_draw\_roi\_mpl

`locan.scripts.script_draw_roi.sc_draw_roi_mpl(file_path=None, file_type=1, roi_file_indicator='_roi', region_type='rectangle')`

Define regions of interest by drawing a boundary.

#### Parameters

- **file\_path** (UnionType[str, PathLike[Any], None]) – File path to localization data.
- **file\_type** (int | str | *FileType* | Metadata) – Indicator for the file type. Integer or string should be according to `locan.constants.FileType`.
- **roi\_file\_indicator** (str) – Indicator to add to the localization file name and use as roi file name (with further extension .yaml).
- **region\_type** (Literal['rectangle', 'ellipse', 'polygon']) – rectangle, ellipse, or polygon specifying the selection widget to use.

**Return type**  
None

## locan.scripts.script\_napari

Render localization data in napari.

With this script you can choose a file name and render the localization file in napari.

To run the script:

```
locan napari -f <file> -t <file type> -k <string with kwargs for render_
↪function> --bin_size <>
--rescale <string with tuple or rescale>
```

Try for instance:

```
locan napari -f "locan/tests/test_data/five_blobs.txt" -t 1 --bin_size 10 --
↪rescale "0 1"
```

See also:

locan.render.render2d.render\_2d\_napari

## Functions

*main*([args])

**rtype**  
None

*sc\_napari*([file\_path, file\_type])

Render localization data in napari.

## locan.scripts.script\_napari.main

locan.scripts.script\_napari.**main**(args=None)

**Return type**  
None

## locan.scripts.script\_napari.sc\_napari

locan.scripts.script\_napari.**sc\_napari**(file\_path=None, file\_type=FileType.CUSTOM, \*\*kwargs)

Render localization data in napari.

### Parameters

- **file\_path** (UnionType[str, PathLike[Any], None]) – File path to localization data.
- **file\_type** (int | str | *FileType* | Metadata) – Indicator for the file type. Integer or string should be according to locan.constants.FileType.
- **kwargs** (Any) – Other parameters passed to `render_2d_napari()`.

**Return type**

None

**locan.scripts.script\_show\_versions**

Show system information and dependency versions.

To run the script:

```
locan show_versions -v -e -o <module name> [<module name>...]
```

Try for instance:

```
locan show_versions -v -e
```

**See also:**

[\*locan.utils.system\\_information.show\\_versions\*](#)

**Functions**

```
main([args])
```

**rtype**  
None

**locan.scripts.script\_show\_versions.main**

`locan.scripts.script_show_versions.main(args=None)`

**Return type**

None

**locan.scripts.script\_test**

Run project test suite with pytest.

To run the script:

```
test
```

Try for instance:

```
test
```

**See also:**

[\*locan.tests.test\*](#)

## Functions

|                           |                      |
|---------------------------|----------------------|
| <code>main([args])</code> | <b>rtype</b><br>None |
|---------------------------|----------------------|

### locan.scripts.script\_test.main

locan.scripts.script\_test.**main**(args=None)

**Return type**  
None

## 8.11 locan.simulation

Synthetic data

This module provides functions to simulate localization and other data that can be used for testing and development.

### 8.11.1 Submodules:

|                               |   |
|-------------------------------|---|
| <code>simulate_locdata</code> | Simulate localization data.                       |
| <code>simulate_drift</code>   | Simulate drift and apply it to localization data. |

### locan.simulation.simulate\_locdata

Simulate localization data.

This module provides functions to simulate localization data and return LocData objects. Localizations are often distributed either by a spatial process of complete-spatial randomness or following a Neyman-Scott process<sup>1</sup>. For a Neyman-Scott process parent events (representing single emitters) yield a random number of cluster\_mu events (representing localizations due to repeated blinking). Related spatial point processes include Matérn and Thomas processes.

Functions that are named as make\_\* provide point data arrays. Functions that are named as simulate\_\* provide locdata.

Parts of this code is adapted from scikit-learn/sklearn/datasets/\_samples\_generator.py . (BSD 3-Clause License, Copyright (c) 2007-2020 The scikit-learn developers.)

---

<sup>1</sup> Neyman, J. & Scott, E. L., A Theory of the Spatial Distribution of Galaxies. Astrophysical Journal 1952, vol. 116, p.144.

## References

## Functions

|  |   |
|--|---|
| <code>make_Matern([parent_intensity, region, ...])</code>      | Generate clustered point data following a Matern cluster random point process.  |
| <code>make_NeymanScott([parent_intensity, region, ...])</code> | Generate clustered point data following a Neyman-Scott random point process.  |
| <code>make_Poisson(intensity[, region, seed])</code>           | Provide points that are distributed by a uniform Poisson point process within the boundaries given by <i>region</i> .   |
| <code>make_Thomas([parent_intensity, region, ...])</code>      | Generate clustered point data following a Thomas random point process.  |
| <code>make_cluster([centers, region, ...])</code>              | Parent positions are taken from <i>centers</i> or are distributed according to a homogeneous Poisson process with exactly <i>centers</i> within the boundaries given by <i>region</i> expanded by the <i>expansion_distance</i> . |
| <code>make_dstorm([parent_intensity, region, ...])</code>      | Generate clustered point data following a Thomas-like random point process.   |
| <code>make_uniform(n_samples[, region, seed])</code>           | Provide points that are distributed by a uniform (complete spatial randomness) point process within the boundaries given by <i>region</i> .   |
| <code>randomize(locdata[, hull_region, seed])</code>           | Transform locdata coordinates into randomized coordinates that follow complete spatial randomness on the same region as the input locdata.  |
| <code>resample(locdata[, n_samples, seed])</code>              | Resample locdata according to localization uncertainty.   |
| <code>simulate_Matern([parent_intensity, region, ...])</code>  | Generate clustered point data following a Matern cluster random point process.  |
| <code>simulate_NeymanScott([parent_intensity, ...])</code>     | Generate clustered point data following a Neyman-Scott random point process.  |
| <code>simulate_Poisson(intensity[, region, seed])</code>       | Provide points that are distributed by a uniform Poisson point process within the boundaries given by <i>region</i> .   |
| <code>simulate_Thomas([parent_intensity, region, ...])</code>  | Generate clustered point data following a Thomas random point process.  |
| <code>simulate_cluster([centers, region, ...])</code>          | Generate clustered point data.  |
| <code>simulate_dstorm([parent_intensity, region, ...])</code>  | Generate clustered point data following a Thomas-like random point process.   |
| <code>simulate_frame_numbers(n_samples, lam[, seed])</code>    | Simulate Poisson-distributed frame numbers for a list of localizations.   |
| <code>simulate_tracks([n_walks, n_steps, ranges, ...])</code>  | Provide a dataset of localizations representing random walks with starting points being spatially-distributed on a rectangular shape or cubic volume by complete spatial randomness.  |
| <code>simulate_uniform(n_samples[, region, seed])</code>       | Provide points that are distributed by a uniform Poisson point process within the boundaries given by <i>region</i> .   |

## locan.simulation.simulate\_locdata.make\_Matern

```
locan.simulation.simulate_locdata.make_Matern(parent_intensity=1, region=(0, 1.0),
                                              cluster_mu=1, radius=1.0, clip=True,
                                              shuffle=True, seed=None)
```

Generate clustered point data following a Matern cluster random point process. Parent positions are distributed according to a homogeneous Poisson process with *parent\_intensity* within the boundaries given by *region* expanded by the maximum radius. Each parent position is then replaced by spots of size *radius* with Poisson distributed points inside. Offspring from parent events that are located outside the region are included.

### Parameters

- **parent\_intensity** (int | float) – The intensity (points per unit region measure) of the Poisson point process for parent events.
- **region** (Union[*Region*, \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for all features. If array-like it must provide upper and lower bounds for each feature.
- **cluster\_mu** (int | float) – The mean number of points of the Poisson point process for cluster(cluster\_mu) events.
- **radius** (float | Sequence[float]) – The radius for the spots. If tuple, the number of elements must be larger than the expected number of parents.
- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.
- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

### Returns

The generated samples, labels, parent\_samples of shape (n\_samples, n\_features) and region

### Return type

tuple[npt.NDArray[np.float64], npt.NDArray[np.int64], npt.NDArray[np.float64], *Region*]

## locan.simulation.simulate\_locdata.make\_NeymanScott

```
locan.simulation.simulate_locdata.make_NeymanScott(parent_intensity=100, region=(0,
1.0), expansion_distance=0,
offspring=None, clip=True,
shuffle=True, seed=None)
```

Generate clustered point data following a Neyman-Scott random point process. Parent positions are distributed according to a homogeneous Poisson process with *parent\_intensity* within the boundaries given by *region* expanded by the *expansion\_distance*. Each parent position is then replaced by offspring points as passed or generated by a given function. Offspring from parent events that are located outside the region are included.



## Parameters

- **parent\_intensity** (int | float) – The intensity (points per unit region measure) of the Poisson point process for parent events.
- **region** (Union[[Region](#), \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for all features. If array-like it must provide upper and lower bounds for each feature.
- **expansion\_distance** (float) – The distance by which region is expanded on all boundaries.
- **offspring** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], Callable[... Any], None]) – Points or function for point process to provide offspring points. Callable must take single parent point as parameter. If array-like it must have enough elements to fit the randomly generated number of parent events.
- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.
- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

## Returns

The generated samples, labels, parent\_samples of shape (n\_samples, n\_features) and region

## Return type

tuple[npt.NDArray[np.float64], npt.NDArray[np.int64], npt.NDArray[np.float64], [Region](#)]

## locan.simulation.simulate\_locdata.make\_Poisson

locan.simulation.simulate\_locdata.**make\_Poisson**(intensity, region=(0, 1), seed=None)

Provide points that are distributed by a uniform Poisson point process within the boundaries given by *region*.

## Parameters

- **intensity** (int | float) – The intensity (points per unit region measure) of the point process
- **region** (Union[[Region](#), \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for all features. If array-like it must provide upper and lower bounds for each feature.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

**Returns**

The generated samples of shape (n\_samples, n\_features).

**Return type**

npt.NDArray[np.float64]

**locan.simulation.simulate\_locdata.make\_Thomas**

```
locan.simulation.simulate_locdata.make_Thomas(parent_intensity=1, region=(0, 1.0),
                                              expansion_factor=6, cluster_mu=1,
                                              cluster_std=1.0, clip=True, shuffle=True,
                                              seed=None)
```

Generate clustered point data following a Thomas random point process. Parent positions are distributed according to a homogeneous Poisson process with *parent\_intensity* within the boundaries given by *region* expanded by an expansion distance that equals *expansion\_factor* \* *max(cluster\_std)*. Each parent position is then replaced by *n* offspring points where *n* is Poisson-distributed with mean number *cluster\_mu* and point coordinates are normal-distributed around the parent point with standard deviation *cluster\_std*. Offspring from parent events that are located outside the region are included.

**Parameters**

- **parent\_intensity** (int | float) – The intensity (points per unit region measure) of the Poisson point process for parent events.
- **region** (Union[[Region](#), \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for all features. If array-like it must provide upper and lower bounds for each feature.
- **expansion\_factor** (int | float) – Factor by which the *cluster\_std* is multiplied to set the region expansion distance.
- **cluster\_mu** (int | float | Sequence[float]) – The mean number of points for normal-distributed offspring points.
- **cluster\_std** (float | Sequence[float] | Sequence[Sequence[float]]) – The standard deviation for normal-distributed offspring points.
- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.
- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

**Returns**

The generated samples, labels, parent\_samples of shape (n\_samples, n\_features) and region

**Return type**

tuple[npt.NDArray[np.float64], npt.NDArray[np.int64], npt.NDArray[np.float64], [Region](#)]

## locan.simulation.simulate\_locdata.make\_cluster

```
locan.simulation.simulate_locdata.make_cluster(centers=3, region=(0, 1.0),
                                              expansion_distance=0, offspring=None,
                                              clip=True, shuffle=True, seed=None)
```

Parent positions are taken from *centers* or are distributed according to a homogeneous Poisson process with exactly *centers* within the boundaries given by *region* expanded by the *expansion\_distance*. Each parent position is then replaced by *cluster\_mu* offspring points as passed or generated by a given function. Offspring from parent events that are located outside the region are included.

### Parameters

- **centers** (Union[int, \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The number of parents or coordinates for parent events, where each parent represents a cluster center.
- **region** (Union[[Region](#), \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for all features. If array-like it must provide upper and lower bounds for each feature.
- **expansion\_distance** (float) – The distance by which region is expanded on all boundaries.
- **offspring** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], Callable[... Any], None]) – Points or function for point process to provide cluster. Callable must take single parent point as parameter and return an iterable. If array-like it must have the same length as parent events.
- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.
- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

### Returns

The generated samples, labels, parent\_samples of shape (n\_samples, n\_features) and region

### Return type

tuple[npt.NDArray[np.float64], npt.NDArray[np.int64], npt.NDArray[np.float64], [Region](#)]

## locan.simulation.simulate\_locdata.make\_dstorm

```
locan.simulation.simulate_locdata.make_dstorm(parent_intensity=1, region=(0, 1.0),
                                              expansion_factor=6, cluster_mu=1,
                                              min_points=0, cluster_std=1.0, clip=True,
                                              shuffle=True, seed=None)
```

Generate clustered point data following a Thomas-like random point process. Parent positions are distributed according to a homogeneous Poisson process with *parent\_intensity* within the boundaries given by *region* expanded by an expansion distance that equals *expansion\_factor* \* *max(cluster\_std)*. Each parent position is then replaced by *n* offspring points where *n* is geometrically-distributed with mean number *cluster\_mu* and point coordinates are normal-distributed around the parent point with standard deviation *cluster\_std*. Offspring from parent events that are located outside the region are included.

### Parameters

- **parent\_intensity** (int | float) – The intensity (points per unit region measure) of the Poisson point process for parent events.
- **region** (Union[[Region](#), \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for all features. If array-like it must provide upper and lower bounds for each feature.
- **expansion\_factor** (int | float) – Factor by which the *cluster\_std* is multiplied to set the region expansion distance.
- **cluster\_mu** (int | float | Sequence[float]) – The mean number of points for normal-distributed offspring points.
- **min\_points** (int) – The minimum number of points per cluster.
- **cluster\_std** (float | Sequence[float] | Sequence[Sequence[float]]) – The standard deviation for normal-distributed offspring points.
- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.
- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

### Returns

The generated samples, labels, parent\_samples  
of shape (n\_samples, n\_features) and region

### Return type

tuple[npt.NDArray[np.float64], npt.NDArray[np.int64], npt.NDArray[np.float64],  
[Region](#)]

### locan.simulation.simulate\_locdata.make\_uniform

locan.simulation.simulate\_locdata.**make\_uniform**(*n\_samples*, *region*=(0, 1), *seed*=None)

Provide points that are distributed by a uniform (complete spatial randomness) point process within the boundaries given by *region*.

#### Parameters

- **n\_samples** (int) – The total number of localizations of the point process
- **region** (Union[[Region](#), \_SupportsArray[*dtype*[Any]], \_NestedSequence[\_SupportsArray[*dtype*[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for all features. If array-like it must provide upper and lower bounds for each feature.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – Random number generation seed

#### Returns

The generated samples of shape (n\_samples, n\_features).

#### Return type

npt.NDArray[np.float64]

### locan.simulation.simulate\_locdata.randomize

locan.simulation.simulate\_locdata.**randomize**(*locdata*, *hull\_region*='bb', *seed*=None)

Transform locdata coordinates into randomized coordinates that follow complete spatial randomness on the same region as the input locdata.

#### Parameters

- **locdata** ([LocData](#)) – Localization data to be randomized
- **hull\_region** (Union[[Region](#), Literal['bb', 'ch', 'as', 'obb']]) – Region of interest. String identifier can refer to the corresponding hull.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

#### Returns

New localization data with randomized coordinates.

#### Return type

[LocData](#)

## locan.simulation.simulate\_locdata.resample

locan.simulation.simulate\_locdata.**resample**(*locdata*, *n\_samples*=10, *seed*=None)

Resample locdata according to localization uncertainty. Per localization *n\_samples* new localizations are simulated normally distributed around the localization coordinates with a standard deviation set to the uncertainty in each dimension. Uncertainties are taken from “uncertainty\_c” or “uncertainty”. The resulting LocData object carries new localizations with the following new properties: position coordinates, ‘original\_index’.

### Parameters

- **locdata** (*LocData*) – Localization data to be resampled
- **n\_samples** (int) – The number of localizations generated for each original localization.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

### Returns

New localization data with simulated coordinates.

### Return type

*LocData*

## locan.simulation.simulate\_locdata.simulate\_Matern

locan.simulation.simulate\_locdata.**simulate\_Matern**(*parent\_intensity*=1, *region*=(0, 1.0),  
*cluster\_mu*=1, *radius*=1.0,  
*clip*=True, *shuffle*=True, *seed*=None)

Generate clustered point data following a Matern cluster random point process. Parent positions are distributed according to a homogeneous Poisson process with *parent\_intensity* within the boundaries given by *region* expanded by the maximum radius. Each parent position is then replaced by spots of size *radius* with Poisson distributed points inside. Offspring from parent events that are located outside the region are included.

### Parameters

- **parent\_intensity** (int | float) – The intensity (points per unit region measure) of the Poisson point process for parent events.
- **region** (Union[*Region*, \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for each feature. If array-like it must provide upper and lower bounds for each feature.
- **cluster\_mu** (int | float) – The mean number of points of the Poisson point process for cluster(cluster\_mu) events.
- **radius** (float | Sequence[float]) – The radius for the spots. If tuple, the number of elements must be larger than the expected number of parents.
- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.

- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

#### Returns

The generated samples.

#### Return type

*LocData*

### locan.simulation.simulate\_locdata.simulate\_NeymanScott

```
locan.simulation.simulate_locdata.simulate_NeymanScott(parent_intensity=100,
                                                         region=(0, 1.0),
                                                         expansion_distance=0,
                                                         offspring=None, clip=True,
                                                         shuffle=True, seed=None)
```

Generate clustered point data following a Neyman-Scott random point process. Parent positions are distributed according to a homogeneous Poisson process with *parent\_intensity* within the boundaries given by *region* expanded by the *expansion\_distance*. Each parent position is then replaced by offspring points as passed or generated by a given function. Offspring from parent events that are located outside the region are included.

#### Parameters

- **parent\_intensity** (int | float) – The intensity (points per unit region measure) of the Poisson point process for parent events.
- **region** (Union[*Region*, \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for each feature. If array-like it must provide upper and lower bounds for each feature.
- **expansion\_distance** (float) – The distance by which region is expanded on all boundaries.
- **offspring** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], Callable[... Any], None) – Points or function for point process to provide offspring points. Callable must take single parent point as parameter. If array-like it must have enough elements to fit the randomly generated number of parent events.
- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.
- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

#### Returns

The generated samples.

**Return type***LocData***locan.simulation.simulate\_locdata.simulate\_Poisson**

```
locan.simulation.simulate_locdata.simulate_Poisson(intensity, region=(0, 1),  
                                                    seed=None)
```

Provide points that are distributed by a uniform Poisson point process within the boundaries given by *region*.

**Parameters**

- **intensity** (int | float) – The intensity (points per unit region measure) of the point process
- **region** (Union[*Region*, \_SupportsArray[*dtype*[Any]], \_NestedSequence[\_SupportsArray[*dtype*[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for each feature. If array-like it must provide upper and lower bounds for each feature.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

**Returns**

The generated samples.

**Return type***LocData***locan.simulation.simulate\_locdata.simulate\_Thomas**

```
locan.simulation.simulate_locdata.simulate_Thomas(parent_intensity=1, region=(0, 1.0),  
                                                    expansion_factor=6, cluster_mu=1,  
                                                    cluster_std=1.0, clip=True,  
                                                    shuffle=True, seed=None)
```

Generate clustered point data following a Thomas random point process. Parent positions are distributed according to a homogeneous Poisson process with *parent\_intensity* within the boundaries given by *region* expanded by an expansion distance that equals *expansion\_factor* \* max(*cluster\_std*). Each parent position is then replaced by *n* offspring points where *n* is Poisson-distributed with mean number *cluster\_mu* and point coordinates are normal-distributed around the parent point with standard deviation *cluster\_std*. Offspring from parent events that are located outside the region are included.

**Parameters**

- **parent\_intensity** (int | float) – The intensity (points per unit region measure) of the Poisson point process for parent events.
- **region** (Union[*Region*, \_SupportsArray[*dtype*[Any]], \_NestedSequence[\_SupportsArray[*dtype*[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for each feature. If array-like it must provide upper and lower bounds for each feature.



- **expansion\_factor** (int | float) – Factor by which the cluster\_std is multiplied to set the region expansion distance.
- **cluster\_mu** (int | float | Sequence[float]) – The mean number of points for normal-distributed offspring points.
- **cluster\_std** (float | Sequence[float] | Sequence[Sequence[float]]) – The standard deviation for normal-distributed offspring points.
- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.
- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

**Returns**

The generated samples.

**Return type**

*LocData*

**locan.simulation.simulate\_locdata.simulate\_cluster**

```
locan.simulation.simulate_locdata.simulate_cluster(centers=3, region=(0, 1.0),  
                                                    expansion_distance=0,  
                                                    offspring=None, clip=True,  
                                                    shuffle=True, seed=None)
```

Generate clustered point data. Parent positions are taken from *centers* or are distributed according to a homogeneous Poisson process with exactly *centers* within the boundaries given by *region* expanded by the *expansion\_distance*. Each parent position is then replaced by offspring points as passed or generated by a given function. Offspring from parent events that are located outside the region are included.

**Parameters**

- **centers** (Union[int, \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The number of parents or coordinates for parent events, where each parent represents a cluster center.
- **region** (Union[*Region*, \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for each feature. If array-like it must provide upper and lower bounds for each feature.
- **expansion\_distance** (float) – The distance by which region is expanded on all boundaries.
- **offspring** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], Callable[..., Any], None]) – Points or function

for point process to provide cluster. Callable must take single parent point as parameter and return an iterable. If array-like it must have the same length as parent events.

- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.
- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

#### Returns

The generated samples.

#### Return type

*LocData*

### locan.simulation.simulate\_locdata.simulate\_dstorm

```
locan.simulation.simulate_locdata.simulate_dstorm(parent_intensity=1, region=(0, 1.0),  
                                                    expansion_factor=6, cluster_mu=1,  
                                                    min_points=0, cluster_std=1.0,  
                                                    clip=True, shuffle=True, seed=None)
```

Generate clustered point data following a Thomas-like random point process. Parent positions are distributed according to a homogeneous Poisson process with *parent\_intensity* within the boundaries given by *region* expanded by an expansion distance that equals *expansion\_factor* \* *max(cluster\_std)*. Each parent position is then replaced by *n* offspring points where *n* is geometrically-distributed with mean number *cluster\_mu* and point coordinates are normal-distributed around the parent point with standard deviation *cluster\_std*. Offspring from parent events that are located outside the region are included.

#### Parameters

- **parent\_intensity** (int | float) – The intensity (points per unit region measure) of the Poisson point process for parent events.
- **region** (Union[*Region*, \_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for each feature. If array-like it must provide upper and lower bounds for each feature.
- **expansion\_factor** (int | float) – Factor by which the *cluster\_std* is multiplied to set the region expansion distance.
- **cluster\_mu** (int | float | Sequence[float]) – The mean number of points for normal-distributed offspring points.
- **min\_points** (int) – The minimum number of points per cluster.
- **cluster\_std** (float | Sequence[float] | Sequence[Sequence[float]]) – The standard deviation for normal-distributed offspring points.
- **clip** (bool) – If True the result will be clipped to ‘region’. If False the extended region will be kept.

- **shuffle** (bool) – If True shuffle the samples.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

#### Returns

The generated samples.

#### Return type

*LocData*

### locan.simulation.simulate\_locdata.simulate\_frame\_numbers

locan.simulation.simulate\_locdata.**simulate\_frame\_numbers**(*n\_samples*, *lam*,  
*seed=None*)

Simulate Poisson-distributed frame numbers for a list of localizations.

Return numpy.ndarray of sorted integers with each integer *i* being repeated *n(i)* times where *n(i)* is drawn from a Poisson distribution with mean *lam*.

Use the following to add frame numbers to a given LocData object:

```
frames = simulate_frame_numbers(n_samples=len(locdata), lam=5)
locdata.dataframe = locdata.dataframe.assign(frame = frames)
```

#### Parameters

- **n\_samples** (int) – number of elements to be returned
- **lam** (float) – mean of the Poisson distribution (lambda)
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

#### Returns

The generated sequence of integers with shape (n\_samples,)

#### Return type

npt.NDArray[np.int64]

### locan.simulation.simulate\_locdata.simulate\_tracks

locan.simulation.simulate\_locdata.**simulate\_tracks**(*n\_walks=1*, *n\_steps=10*, *ranges=((0, 10000), (0, 10000))*,  
*diffusion\_constant=1*, *time\_step=10*,  
*seed=None*)

Provide a dataset of localizations representing random walks with starting points being spatially-distributed on a rectangular shape or cubic volume by complete spatial randomness.

#### Parameters

- **n\_walks** (int) – Number of walks
- **n\_steps** (int) – Number of time steps (i.e. frames)

- **ranges** (tuple[tuple[int, int], tuple[int, int]]) – the range for valid x[, y, z]-coordinates
- **diffusion\_constant** (int | float) – Diffusion constant with unit length per seconds<sup>2</sup>
- **time\_step** (float) – Time per frame (or simulation step) in seconds.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

**Returns**

A new LocData instance with localization data.

**Return type**

*LocData*

### locan.simulation.simulate\_locdata.simulate\_uniform

locan.simulation.simulate\_locdata.**simulate\_uniform**(*n\_samples*, *region*=(0, 1),  
seed=None)

Provide points that are distributed by a uniform Poisson point process within the boundaries given by *region*.

**Parameters**

- **n\_samples** (int) – The total number of localizations of the point process
- **region** (Union[*Region*, \_SupportsArray[*dtype*[Any]], \_NestedSequence[\_SupportsArray[*dtype*[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The region (or support) for each feature. If array-like it must provide upper and lower bounds for each feature.
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

**Returns**

The generated samples.

**Return type**

*LocData*

### locan.simulation.simulate\_drift

Simulate drift and apply it to localization data.

Drift can be linear in time or resemble a random walk.

## Functions

|   |  |
|---|--|
| <code>add_drift</code> (locdata[, diffusion_constant, ...]) | Compute position deltas as function of frame number. |
|---|--|

### `locan.simulation.simulate_drift.add_drift`

`locan.simulation.simulate_drift.add_drift`(locdata, diffusion\_constant=None, velocity=None, seed=None)

Compute position deltas as function of frame number.

#### Parameters

- **locdata** (*LocData*) – Original localization data
- **diffusion\_constant** (Optional[tuple[float, ...]]) – Diffusion constant for each dimension specifying the drift velocity with shape (point\_dimension,). The diffusion constant has the unit of square of localization coordinate unit per frame unit.
- **velocity** (Optional[tuple[float, ...]]) – Drift velocity in units of localization coordinate unit per frame unit with shape (point\_dimension,)
- **seed** (Union[None, int, Sequence[int], SeedSequence, BitGenerator, Generator]) – random number generation seed

#### Returns

A new *LocData* instance with localization data.

#### Return type

*LocData*

## 8.12 locan.tests

Test the locan package.

This module includes unit tests for all modules within the locan package. The tests are organized following the subpackage structure of locan.

## Functions

|                            |                            |
|----------------------------|----------------------------|
| <code>test</code> ([args]) | Running tests with pytest. |
|----------------------------|----------------------------|

### 8.12.1 locan.tests.test

`locan.tests.test(args=None)`

Running tests with pytest.

**Parameters**

**args** (UnionType[str, list[str], None]) – Parameters passed to `pytest.main()`

**Return type**

`int | ExitCode`

## 8.13 locan.utils

Utility functions.

### 8.13.1 Submodules:

|                           |   |
|---------------------------|---|
| <i>miscellaneous</i>      | Miscellaneous functions without a home.                     |
| <i>statistics</i>         | Statistics related tools.                                   |
| <i>system_information</i> | Utility methods to print system and dependency information. |

#### locan.utils.miscellaneous

Miscellaneous functions without a home.

#### Functions

|  |   |
|--|---|
| <i>iterate_2d_array</i> ([n_elements, n_cols]) | Iterator for 2-dimensional array iterating first over columns then over rows. |
|--|---|

#### locan.utils.miscellaneous.iterate\_2d\_array

`locan.utils.miscellaneous.iterate_2d_array(n_elements=5, n_cols=2)`

Iterator for 2-dimensional array iterating first over columns then over rows.

**Parameters**

- **n\_elements** (int) – Number of elements
- **n\_cols** (int) – Number of columns

**Returns**

Indices for (row, column) in each iteration.

**Return type**

Iterator[tuple[int, int]]

**locan.utils.statistics**

Statistics related tools.

**Classes**

---

*WeightedMeanVariance*(weighted\_mean, ...)

---

**locan.utils.statistics.WeightedMeanVariance****class** locan.utils.statistics.**WeightedMeanVariance**(*weighted\_mean*,  
*weighted\_mean\_variance*)

Bases: NamedTuple

**Methods**

---

`__init__()``count(value, /)` Return number of occurrences of value.`index(value[, start, stop])` Return first index of value.

---

**Attributes**

---

*weighted\_mean* Alias for field number 0*weighted\_mean\_variance* Alias for field number 1

---

**weighted\_mean:** float | ndarray[Any, dtype[float64]]

Alias for field number 0

**weighted\_mean\_variance:** float | ndarray[Any, dtype[float64]]

Alias for field number 1

## Functions

|  |  |
|--|--|
| <code>biased_variance(variance, n_samples)</code>    | The sample variance is biased if not corrected by Bessel's correction.   |
| <code>ratio_fwhm_to_sigma()</code>                   | The numeric value of the ratio between full-width-half-max (fwhm) width and standard deviation (sigma) of a normal distribution. |
| <code>weighted_mean_variance(values, weights)</code> | Compute weighted mean (average) and the corresponding weighted mean variance [1].  |

### locan.utils.statistics.biased\_variance

`locan.utils.statistics.biased_variance(variance, n_samples)`

The sample variance is biased if not corrected by Bessel's correction. This function yields the biased variance by applying the inverse correction.

$$E(\text{variance}_{\text{biased}}) = \text{variance} * (1 - 1/\text{localization\_counts}).$$

#### Parameters

- **variance** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – An unbiased variance.
- **n\_samples** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Number of samples from which the biased sample variance would be computed.

#### Return type

`npt.NDArray[np.float64]`

### locan.utils.statistics.ratio\_fwhm\_to\_sigma

`locan.utils.statistics.ratio_fwhm_to_sigma()`

The numeric value of the ratio between full-width-half-max (fwhm) width and standard deviation (sigma) of a normal distribution.

#### Return type

`float`



## locan.utils.statistics.weighted\_mean\_variance

locan.utils.statistics.**weighted\_mean\_variance**(*values*, *weights*)

Compute weighted mean (average) and the corresponding weighted mean variance<sup>1</sup>.

### Parameters

- **values** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Values from which to compute the weighted average.
- **weights** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]], None]) – Weights to use for weighted average.

### Returns

weighted\_mean, weighted\_mean\_variance

### Return type

*WeightedMeanVariance*

## References

## locan.utils.system\_information

Utility methods to print system and dependency information.

adapted from `pandas.show_versions()` (*locan/licences/PANDAS.rst*) (BSD 3-Clause License, Copyright (c) 2008-2011, AQR Capital Management, LLC, Lambda Foundry, Inc. and PyData Development Team, Copyright (c) 2011-2021, Open source contributors.) and from `scikit-learn.show_versions()` (*locan/licences/SCIKIT-LEARN.rst*) (BSD 3-Clause License, Copyright (c) 2007-2020 The scikit-learn developers.)

## Functions

|   |   |
|---|---|
| <i>dependency_info</i> ([extra_dependencies, ...])  | Overview of the installed version of main dependencies.               |
| <i>show_versions</i> ([locan, python, system, ...]) | Print useful debugging information on system and dependency versions. |
| <i>system_info</i> ([verbose])                      | Return system information.  |

<sup>1</sup> Donald F. Gatz, Luther Smith, The standard error of a weighted mean concentration — I. Bootstrapping vs other methods. Atmospheric Environment 29(11), 1995: 1185-1193, [https://doi.org/10.1016/1352-2310\(94\)00210-C](https://doi.org/10.1016/1352-2310(94)00210-C).

### **locan.utils.system\_information.dependency\_info**

`locan.utils.system_information.dependency_info(extra_dependencies=True,  
other_dependencies=None)`

Overview of the installed version of main dependencies.

#### **Parameters**

- **extra\_dependencies** (bool) – Include extra dependencies
- **other\_dependencies** (Optional[list[str]]) – Include other module names.

#### **Returns**

Version information on relevant Python libraries

#### **Return type**

dict

### **locan.utils.system\_information.show\_versions**

`locan.utils.system_information.show_versions(locan=True, python=True, system=True,  
dependencies=True, verbose=True,  
extra_dependencies=True,  
other_dependencies=None)`

Print useful debugging information on system and dependency versions.

#### **Parameters**

- **locan** (bool) – Show locan version
- **python** (bool) – Show python version
- **system** (bool) – Show system information
- **verbose** (bool) – If True information on node and executable path are added.
- **dependencies** (bool) – Show main dependencies
- **extra\_dependencies** (bool) – Include extra dependencies as specified in setup.py if True.
- **other\_dependencies** (Optional[list[str]]) – Include other module names.

#### **Return type**

None

## locan.utils.system\_information.system\_info

`locan.utils.system_information.system_info(verbose=True)`

Return system information.

### Parameters

**verbose** (bool) – If True information on node and executable path are added.

### Returns

System information

### Return type

dict

## 8.14 locan.visualize

Visualize localization data.

This module provides functions to visualize localization data with matplotlib or napari.

### 8.14.1 Submodules:

|                      |   |
|----------------------|---|
| <i>colormap</i>      | Provide colormaps for visualization.      |
| <i>render</i>        | Render localization data.                 |
| <i>render_mpl</i>    | Render localization data with matplotlib. |
| <i>render_napari</i> | Render localization data with napari.     |
| <i>transform</i>     | Transform intensity values.               |

## locan.visualize.colormap

Provide colormaps for visualization.

This module provides convenience functions for using colormap definitions from various visualization packages.

Default colormaps in locan are set through the *locan.configuration.COLORMAP\_DEFAULTS* mapping.

Named colormaps are registered through the `locan.visualization.colormap.colormap_registry` mapping.

In locan `locan.Colormap` serves as adapter class to provide an interface for various visualization functions. Instances of `locan.Colormap` can be requested through the `locan.visualization.colormap.get_colormap()` function and contain references to matplotlib and napari colormap instances.

## Examples

```
>>> colormap = locan.get_colormap("viridis")
>>> assert isinstance(colormap.matplotlib, mcolors.Colormap)
>>> colormap.name
viridis
```

```
>>> colormap = locan.Colormap.from_matplotlib(colormap="viridis")
>>> assert isinstance(colormap.matplotlib, mcolors.Colormap)
>>> colormap.name
viridis
```

## Variables

|                          |   |
|--------------------------|---|
| <i>colormap_registry</i> | A mapping of names onto Colormap instances. |
|--------------------------|---|

### locan.visualize.colormap.colormap\_registry

```
locan.visualize.colormap.colormap_registry: Mapping[str, Colormap] =
{'cet_coolwarm': <locan.visualize.colormap.Colormap object>, 'cet_fire':
<locan.visualize.colormap.Colormap object>, 'cet_fire_r':
<locan.visualize.colormap.Colormap object>, 'cet_glasbey_dark':
<locan.visualize.colormap.Colormap object>, 'cet_gray':
<locan.visualize.colormap.Colormap object>, 'cet_gray_r':
<locan.visualize.colormap.Colormap object>, 'coolwarm':
<locan.visualize.colormap.Colormap object>, 'gray':
<locan.visualize.colormap.Colormap object>, 'gray_r':
<locan.visualize.colormap.Colormap object>, 'tab20':
<locan.visualize.colormap.Colormap object>, 'turbo':
<locan.visualize.colormap.Colormap object>, 'viridis':
<locan.visualize.colormap.Colormap object>, 'viridis_r':
<locan.visualize.colormap.Colormap object>}
```

A mapping of names onto Colormap instances.

## Classes

|  |  |
|--|--|
| <i>Colormap</i> (colormap)                               | Container class for colormaps.                         |
| <i>Colormaps</i> (value[, names, module, qualname, ...]) | Preferred colormap types to be used for visualization. |

**locan.visualize.colormap.Colormap**

**class** locan.visualize.colormap.Colormap(*colormap*)

Bases: object

Container class for colormaps.

A locan Colormap can be instantiated from other colormaps and serves as adapter class.

**Methods**

|   |   |
|---|---|
| <code>__init__(colormap)</code>           |   |
| <code>from_colorcet(cls, colormap)</code> | <b>rtype</b><br>TypeVar(T_Colormap,<br>bound= Colormap) |
| <code>from_matplotlib(colormap)</code>    | <b>rtype</b><br>TypeVar(T_Colormap,<br>bound= Colormap) |
| <code>from_napari(cls, colormap)</code>   | <b>rtype</b><br>TypeVar(T_Colormap,<br>bound= Colormap) |
| <code>from_registry(colormap)</code>      | <b>rtype</b><br><i>Colormap</i>                         |

**Attributes**

|                   |
|-------------------|
| <i>matplotlib</i> |
| <i>name</i>       |
| <i>napari</i>     |

`__call__(*args, **kwargs)`

Call self as a function.

**Return type**

float | tuple[float, float, float, float] | ndarray[Any,  
dtype[float64]]

**classmethod** `from_colorcet(cls, colormap)`

**Return type**

        TypeVar(T\_Colormap, bound= Colormap)

**classmethod** `from_matplotlib(colormap)`

**Return type**

        TypeVar(T\_Colormap, bound= Colormap)

**classmethod** `from_napari(cls, colormap)`

**Return type**

        TypeVar(T\_Colormap, bound= Colormap)

**classmethod** `from_registry(colormap)`

**Return type**

        Colormap

**property** `matplotlib`: Colormap

**property** `name`: str

**property** `napari`: Colormap

## locan.visualize.colormap.Colormaps

**class** `locan.visualize.colormap.Colormaps(value, names=None, *, module=None,  
  qualname=None, type=None, start=1,  
  boundary=None)`

Bases: Enum

Preferred colormap types to be used for visualization.

---

**Note:** This enum is automatically generated from COLORMAP\_DEFAULTS and should not be modified.

---

## Attributes

|                                |
|--------------------------------|
| <i>CONTINUOUS</i>              |
| <i>CONTINUOUS_REVERSE</i>      |
| <i>CONTINUOUS_GRAY</i>         |
| <i>CONTINUOUS_GRAY_REVERSE</i> |
| <i>DIVERGING</i>               |
| <i>CATEGORICAL</i>             |
| <i>TURBO</i>                   |

```

CATEGORICAL = 'cet_glasbey_dark'
CONTINUOUS = 'cet_fire'
CONTINUOUS_GRAY = 'cet_gray'
CONTINUOUS_GRAY_REVERSE = 'cet_gray_r'
CONTINUOUS_REVERSE = 'cet_fire_r'
DIVERGING = 'cet_coolwarm'
TURBO = 'turbo'

```

## Functions

|                                |  |
|--------------------------------|--|
| <i>get_colormap</i> (colormap) | Get a locan.Colormap instance from colormap searching string identifier through colormap_registry, matplotlib colormaps, napari_colormaps. |
|--------------------------------|--|

### locan.visualize.colormap.get\_colormap

locan.visualize.colormap.**get\_colormap**(colormap)

Get a locan.Colormap instance from colormap searching string identifier through colormap\_registry, matplotlib colormaps, napari\_colormaps.

#### Parameters

**colormap** (Union[str, *Colormaps*, *Colormap*, Colormap, Colormap]) – Colormap request

#### Return type

*Colormap*

## locan.visualize.render

Render localization data.

This module provides convenience functions for rendering localization data.

### Functions

|   |   |
|---|---|
| <code>render_2d</code> (locdata[, render_engine]) | Wrapper function to render localization data into a 2D image. |
| <code>render_3d</code> (locdata[, render_engine]) | Wrapper function to render localization data into a 3D image. |

## locan.visualize.render.render\_2d

`locan.visualize.render.render_2d`(locdata, render\_engine=*RenderEngine.MPL*, \*\*kwargs)

Wrapper function to render localization data into a 2D image. For complete signatures see `render_2d_mpl` or corresponding functions.

#### Return type

Any

## locan.visualize.render.render\_3d

`locan.visualize.render.render_3d`(locdata, render\_engine=*RenderEngine.MPL*, \*\*kwargs)

Wrapper function to render localization data into a 3D image. For complete signatures see `render_3d_mpl` or corresponding functions.

#### Return type

Any

## locan.visualize.render\_mpl

Render localization data with matplotlib.

This module provides functions to render and present localization data making use of matplotlib.

### Submodules:

|                       |  |
|-----------------------|--|
| <code>render2d</code> | This module provides functions for rendering locdata objects in 2D.        |
| <code>render3d</code> | This module provides functions for rendering <i>LocData</i> objects in 3D. |



## locan.visualize.render\_mpl.render2d

This module provides functions for rendering locdata objects in 2D.

### Functions

|  |  |
|--|--|
| <code>apply_window(image[, window_function])</code>            | Apply window function to image.  |
| <code>render_2d_mpl(locdata[, loc_properties, ...])</code>     | Render localization data into a 2D image by binning x,y-coordinates into regular bins.     |
| <code>render_2d_rgb_mpl(locdatas[, ...])</code>                | Render localization data into a 2D RGB image by binning x,y-coordinates into regular bins. |
| <code>render_2d_scatter_density(locdata[, ...])</code>         | Render localization data into a 2D image by binning x,y-coordinates into regular bins.     |
| <code>scatter_2d_mpl(locdata[, ax, index, text_kwargs])</code> | Scatter plot of locdata elements with text marker for each element.                        |
| <code>select_by_drawing_mpl(locdata[, re-gion_type])</code>    | Select region of interest from rendered image by drawing rois.                             |

## locan.visualize.render\_mpl.render2d.apply\_window

```
locan.visualize.render_mpl.render2d.apply_window(image, window_function='tukey',
                                                  **kwargs)
```

Apply window function to image.

### Parameters

- **image** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Image
- **window\_function** (str) – Window function to apply. One of ‘tukey’, ‘hann’ or any other in *scipy.signal.windows*.
- **kwargs** (Any) – Other parameters passed to the *scipy.signal.windows* window function.

### Return type

ndarray[Any, dtype[float64]]

## locan.visualize.render\_mpl.render2d.render\_2d\_mpl

```
locan.visualize.render_mpl.render2d.render_2d_mpl(locdata, loc_properties=None,
                                                    other_property=None, bins=None,
                                                    n_bins=None, bin_size=10,
                                                    bin_edges=None, bin_range=None,
                                                    rescale=None, ax=None,
                                                    cmap='cet_fire', cbar=True,
                                                    colorbar_kws=None,
                                                    interpolation='nearest', **kwargs)
```

Render localization data into a 2D image by binning x,y-coordinates into regular bins.

### Parameters

- **locdata** (*LocData*) – Localization data.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If None The coordinate\_values of *locdata* are used.
- **other\_property** (Optional[str]) – Localization property (columns in *locdata.data*) that is averaged in each pixel. If None, localization counts are shown.
- **bins** (UnionType[*Bins*, Axis, AxesTuple, None]) – The bin specification as defined in *Bins*
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If None (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **rescale** (UnionType[int, str, *Trafo*, Callable[... Any], bool, None]) – Transformation as defined in *locan.Trafo* or by transformation function. For None or False no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For 'equal' intensity values are rescaled by histogram equalization.
- **ax** (Optional[Axes]) – The axes on which to show the image
- **cmap** (Union[str, *Colormaps*, *Colormap*, Colormap, Colormap]) – The Colormap object used to map normalized data values to RGBA colors.
- **cbar** (bool) – If true draw a colorbar. The colobar axes is accessible using the *cax* property.

- **colorbar\_kws** (Optional[dict[str, Any]]) – Keyword arguments for `matplotlib.pyplot.colorbar()`.
- **interpolation** (str) – Keyword argument for `matplotlib.axes.Axes.imshow()`.
- **kwargs** (Any) – Other parameters passed to `matplotlib.axes.Axes.imshow()`.

#### Returns

Axes object with the image.

#### Return type

`matplotlib.axes.Axes`

### locan.visualize.render\_mpl.render2d.render\_2d\_rgb\_mpl

```
locan.visualize.render_mpl.render2d.render_2d_rgb_mpl(locdatas, loc_properties=None,
                                                    other_property=None,
                                                    bins=None, n_bins=None,
                                                    bin_size=10, bin_edges=None,
                                                    bin_range=None,
                                                    rescale=None, ax=None,
                                                    interpolation='nearest',
                                                    **kwargs)
```

Render localization data into a 2D RGB image by binning x,y-coordinates into regular bins.

---

**Note:** For `rescale=False` no normalization is carried out image intensities are clipped to (0, 1) for float value or (0, 255) for integer values according to the `matplotlib.imshow` behavior. For `rescale=None` we apply a normalization to (min, max) of all intensity values. For all other `rescale` options the normalization is applied to each individual image.

---

#### Parameters

- **locdatas** (list[[LocData](#)]) – Localization data.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If `None` The coordinate\_values of *locdata* are used.
- **other\_property** (Optional[str]) – Localization property (columns in *locdata.data*) that is averaged in each pixel. If `None`, localization counts are shown.
- **bins** (UnionType[[Bins](#), Axis, AxesTuple, None]) – The bin specification as defined in [Bins](#)
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If `None` (min, max) ranges are determined from data

and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.

- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **rescale** (UnionType[int, str, *Trafo*, Callable[... Any], bool, None]) – Transformation as defined in *locan.Trafo* or by transformation function. For None or False no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For 'equal' intensity values are rescaled by histogram equalization.
- **ax** (Optional[Axes]) – The axes on which to show the image
- **interpolation** (str) – Keyword argument for *matplotlib.axes.Axes.imshow()*.
- **kwargs** (Any) – Other parameters passed to *matplotlib.axes.Axes.imshow()*.

#### Returns

Axes object with the image.

#### Return type

*matplotlib.axes.Axes*

### **locan.visualize.render\_mpl.render2d.render\_2d\_scatter\_density**

```
locan.visualize.render_mpl.render2d.render_2d_scatter_density(locdata,
                                                                loc_properties=None,
                                                                other_property=None,
                                                                bin_range=None,
                                                                ax=None,
                                                                cmap='cet_fire',
                                                                cbar=True,
                                                                colorbar_kws=None,
                                                                **kwargs)
```

Render localization data into a 2D image by binning x,y-coordinates into regular bins.

Prepare *matplotlib.axes.Axes* with image.

---

**Note:** To rescale intensity values use norm keyword.

---

### Parameters

- **locdata** (*LocData*) – Localization data.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If None The coordinate\_values of *locdata* are used.
- **other\_property** (Optional[str]) – Localization property (columns in *locdata.data*) that is averaged in each pixel. If None, localization counts are shown.
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If None (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **ax** (Optional[Axes]) – The axes on which to show the image
- **cmap** (Union[str, *Colormaps*, *Colormap*, Colormap, Colormap]) – The Colormap object used to map normalized data values to RGBA colors.
- **cbar** (bool) – If true draw a colorbar. The colorbar axes is accessible using the cax property.
- **colorbar\_kws** (Optional[dict[str, Any]]) – Keyword arguments for matplotlib.pyplot.colorbar().
- **kwargs** (Any) – Other parameters passed to *mpl\_scatter\_density*. ScatterDensityArtist.

### Returns

Axes object with the image.

### Return type

matplotlib.axes.Axes

## locan.visualize.render\_mpl.render2d.scatter\_2d\_mpl

```
locan.visualize.render_mpl.render2d.scatter_2d_mpl(locdata, ax=None, index=True,
                                                    text_kwargs=None, **kwargs)
```

Scatter plot of locdata elements with text marker for each element.

### Parameters

- **locdata** (*LocData*) – Localization data.
- **ax** (Optional[Axes]) – The axes on which to show the plot
- **index** (bool) – Flag indicating if element indices are shown.

- **text\_kwargs** (Optional[dict[str, Any]]) – Keyword arguments for `matplotlib.axes.Axes.text()`.
- **kwargs** (Any) – Other parameters passed to `matplotlib.axes.Axes.scatter()`.

#### Returns

Axes object with the image.

#### Return type

`matplotlib.axes.Axes`

### locan.visualize.render\_mpl.render2d.select\_by\_drawing\_mpl

```
locan.visualize.render_mpl.render2d.select_by_drawing_mpl(locdata,
                                                         region_type='rectangle',
                                                         **kwargs)
```

Select region of interest from rendered image by drawing rois.

#### Parameters

- **locdata** (*LocData*) – The localization data from which to select localization data.
- **region\_type** (str) – rectangle, or ellipse specifying the selection widget to use.
- **kwargs** (Any) – Other parameters as specified for `render_2d()`.

#### Return type

list[*Roi*]

See also:

**locan.scripts.sc\_draw\_roi\_mpl()**

script for drawing rois

**matplotlib.widgets**

selector functions

### locan.visualize.render\_mpl.render3d

This module provides functions for rendering *LocData* objects in 3D.

#### Functions

|   |     |        |   |
|---|-----|--------|---|
| <code>scatter_3d_mpl</code> (locdata[,<br>text_kwargs]) | ax, | index, | Scatter plot of locdata elements with text marker for each element. |
|---|-----|--------|---|

## locan.visualize.render\_mpl.render3d.scatter\_3d\_mpl

`locan.visualize.render_mpl.render3d.scatter_3d_mpl(locdata, ax=None, index=True, text_kwargs=None, **kwargs)`

Scatter plot of locdata elements with text marker for each element.

### Parameters

- **locdata** (*LocData*) – Localization data.
- **ax** (Optional[Axes]) – The axes on which to show the plot
- **index** (bool) – Flag indicating if element indices are shown.
- **text\_kwargs** (Optional[dict[str, Any]]) – Keyword arguments for `matplotlib.axes.Axes.text()`.
- **kwargs** (Any) – Other parameters passed to `matplotlib.axes.Axes.scatter()`.

### Returns

Axes object with the image.

### Return type

`matplotlib.axes.Axes3D`

## locan.visualize.render\_napari

Render localization data with napari.

This module provides functions to interact with napari.

### Submodules:

|                  |  |
|------------------|--|
| <i>render2d</i>  | This module provides functions to interact with napari for rendering locdata objects in 2D.        |
| <i>render3d</i>  | This module provides functions to interact with napari for rendering <i>LocData</i> objects in 3D. |
| <i>utilities</i> | Utility functions for interacting with napari.   |

## locan.visualize.render\_napari.render2d

This module provides functions to interact with napari for rendering locdata objects in 2D.

## Functions

|  |  |
|--|--|
| <code>render_2d_napari</code> (locdata[, loc_properties, ...]) | Render localization data into a 2D image by binning x,y-coordinates into regular bins.     |
| <code>render_2d_napari_image</code> (locdata[, ...])           | Render localization data into a 2D image by binning x,y-coordinates into regular bins.     |
| <code>render_2d_rgb_napari</code> (locdatas[, ...])            | Render localization data into a 2D RGB image by binning x,y-coordinates into regular bins. |

### locan.visualize.render\_napari.render2d.render\_2d\_napari

```
locan.visualize.render_napari.render2d.render_2d_napari(locdata,
                                                         loc_properties=None,
                                                         other_property=None,
                                                         bins=None, n_bins=None,
                                                         bin_size=None,
                                                         bin_edges=None,
                                                         bin_range=None,
                                                         rescale=None, viewer=None,
                                                         cmap='cet_fire', **kwargs)
```

Render localization data into a 2D image by binning x,y-coordinates into regular bins. Render the data using napari.

#### Parameters

- **locdata** (*LocData*) – Localization data.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If None The coordinate\_values of *locdata* are used.
- **other\_property** (Optional[str]) – Localization property (columns in *locdata.data*) that is averaged in each pixel. If None, localization counts are shown.
- **bins** (UnionType[Bins, Axis, AxesTuple, None]) – The bin specification as defined in Bins
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If None (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.



- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **rescale** (UnionType[int, str, *Trafo*, Callable[... Any], bool, None]) – Transformation as defined in *locan.Trafo* or by transformation function. For None or False no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For ‘equal’ intensity values are rescaled by histogram equalization.
- **viewer** (Viewer) – The viewer object on which to add the image
- **cmap** (str / *Colormap*) – The Colormap object used to map normalized data values to RGBA colors.
- **kwargs** (Any) – Other parameters passed to *napari.Viewer.add\_image()*.

#### Return type

*napari.Viewer*

### *locan.visualize.render\_napari.render2d.render\_2d\_napari\_image*

```
locan.visualize.render_napari.render2d.render_2d_napari_image(locdata,
                                                                loc_properties=None,
                                                                other_property=None,
                                                                bins=None,
                                                                n_bins=None,
                                                                bin_size=None,
                                                                bin_edges=None,
                                                                bin_range=None,
                                                                rescale=None,
                                                                cmap='cet_fire',
                                                                **kwargs)
```

Render localization data into a 2D image by binning x,y-coordinates into regular bins. Provide layer data for *napari*.

#### Parameters

- **locdata** (*LocData*) – Localization data.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If None The *coordinate\_values* of *locdata* are used.
- **other\_property** (Optional[str]) – Localization property (columns in *locdata.data*) that is averaged in each pixel. If None, localization counts are shown.
- **bins** (UnionType[*Bins*, Axis, AxesTuple, None]) – The bin specification as defined in *Bins*

- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If None (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **rescale** (UnionType[int, str, [Trafo](#), Callable[... Any], bool, None]) – Transformation as defined in `locan.Trafo` or by transformation function. For None or False no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For 'equal' intensity values are rescaled by histogram equalization.
- **cmap** (Union[str, [Colormaps](#), [Colormap](#), Colormap, Colormap]) – The Colormap object used to map normalized data values to RGBA colors.
- **kwargs** (Any) – Other parameters passed to `napari.Viewer.add_image()`.

**Returns**

Tuple with data, image\_kwargs, layer\_type="image"

**Return type**

`napari.types.LayerData`

**locan.visualize.render\_napari.render2d.render\_2d\_rgb\_napari**

```
locan.visualize.render_napari.render2d.render_2d_rgb_napari(locdatas,
                                                            loc_properties=None,
                                                            other_property=None,
                                                            bins=None,
                                                            n_bins=None,
                                                            bin_size=10,
                                                            bin_edges=None,
                                                            bin_range=None,
                                                            rescale=None,
                                                            viewer=None,
                                                            **kwargs)
```

Render localization data into a 2D RGB image by binning x,y-coordinates into regular bins.

---

**Note:** For `rescale=False` no normalization is carried out image intensities are clipped to (0, 1) for float value or (0, 255) for integer values according to the `matplotlib.imshow` behavior. For `rescale=None` we apply a normalization to (min, max) of all intensity values. For all other `rescale` options the normalization is applied to each individual image.

---

**Parameters**

- **locdatas** (Iterable[[LocData](#)]) – Localization data.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If `None` The `coordinate_values` of `locdata` are used.
- **other\_property** (Optional[str]) – Localization property (columns in `locdata.data`) that is averaged in each pixel. If `None`, localization counts are shown.
- **bins** (UnionType[[Bins](#), Axis, AxesTuple, None]) – The bin specification as defined in `Bins`
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, `n_bin_edges`).
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link', None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If `None` (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, `n_bins`).

5 would describe `bin_size` of 5 for all bins in all dimensions. `((2, 5),)` yield bins of size (2, 5) for one dimension. `(2, 5)` yields bins of size 2 for one dimension and 5 for the other dimension. `((2, 5), (1, 3))` yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of `bin_size` use `bin_edges` instead.

- **rescale** (UnionType[int, str, [Trafo](#), Callable[... , Any], bool, None]) – Transformation as defined in `locan.Trafo` or by transformation function. For None or False no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For 'equal' intensity values are rescaled by histogram equalization.
- **viewer** (Viewer) – The viewer object on which to add the image
- **kwargs** (Any) – Other parameters passed to `napari.Viewer.add_image()`.

#### Return type

`napari.Viewer`

### locan.visualize.render\_napari.render3d

This module provides functions to interact with napari for rendering *LocData* objects in 3D.

#### Functions

|  |  |
|--|--|
| <code>render_3d_napari</code> (locdata[, loc_properties, ...]) | Render localization data into a 3D image by binning x,y,z-coordinates into regular bins.     |
| <code>render_3d_napari_image</code> (locdata[, ...])           | Render localization data into a 3D image by binning x,y,z-coordinates into regular bins.     |
| <code>render_3d_rgb_napari</code> (locdatas[, ...])            | Render localization data into a 3D RGB image by binning x,y,z-coordinates into regular bins. |

### locan.visualize.render\_napari.render3d.render\_3d\_napari

```
locan.visualize.render_napari.render3d.render_3d_napari(locdata,
                                                         loc_properties=None,
                                                         other_property=None,
                                                         bins=None, n_bins=None,
                                                         bin_size=10,
                                                         bin_edges=None,
                                                         bin_range=None,
                                                         rescale=None, viewer=None,
                                                         cmap='cet_fire', **kwargs)
```

Render localization data into a 3D image by binning x,y,z-coordinates into regular bins. Render the data using napari.

#### Parameters

- **locdata** ([LocData](#)) – Localization data.

- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If None The coordinate\_values of *locdata* are used.
- **other\_property** (Optional[str]) – Localization property (columns in *locdata.data*) that is averaged in each pixel. If None, localization counts are shown.
- **bins** (UnionType[Bins, Axis, AxesTuple, None]) – The bin specification as defined in Bins
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If None (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **rescale** (UnionType[int, str, Trafo, Callable[...], bool, None]) – Transformation as defined in *locan.Trafo* or by transformation function. For None or False no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For 'equal' intensity values are rescaled by histogram equalization.
- **viewer** (Viewer) – The viewer object on which to add the image
- **cmap** (Union[str, Colormaps, Colormap, Colormap, Colormap]) – The Colormap object used to map normalized data values to RGBA colors.
- **kwargs** (Any) – Other parameters passed to *napari.Viewer.add\_image()*.

#### Return type

napari.Viewer

**locan.visualize.render\_napari.render3d.render\_3d\_napari\_image**

```
locan.visualize.render_napari.render3d.render_3d_napari_image(locdata,
                                                                loc_properties=None,
                                                                other_property=None,
                                                                bins=None,
                                                                n_bins=None,
                                                                bin_size=10,
                                                                bin_edges=None,
                                                                bin_range=None,
                                                                rescale=None,
                                                                cmap='cet_fire',
                                                                **kwargs)
```

Render localization data into a 3D image by binning x,y,z-coordinates into regular bins. Provide layer data for napari.

**Parameters**

- **locdata** (*LocData*) – Localization data.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If None The coordinate\_values of *locdata* are used.
- **other\_property** (Optional[str]) – Localization property (columns in *locdata.data*) that is averaged in each pixel. If None, localization counts are shown.
- **bins** (UnionType[Bins, Axis, AxesTuple, None]) – The bin specification as defined in Bins
- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If None (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.

- **rescale** (UnionType[int, str, *Trafo*, Callable[... Any], bool, None]) – Transformation as defined in `locan.Trafo` or by transformation function. For None or False no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For ‘equal’ intensity values are rescaled by histogram equalization.
- **cmap** (Union[str, *Colormaps*, *Colormap*, Colormap, Colormap]) – The Colormap object used to map normalized data values to RGBA colors.
- **kwargs** (Any) – Other parameters passed to `napari.Viewer.add_image()`.

#### Returns

Tuple with data, image\_kwargs, “image”

#### Return type

`napari.types.LayerData`

### `locan.visualize.render_napari.render3d.render_3d_rgb_napari`

```
locan.visualize.render_napari.render3d.render_3d_rgb_napari(locdatas,
                                                             loc_properties=None,
                                                             other_property=None,
                                                             bins=None,
                                                             n_bins=None,
                                                             bin_size=10,
                                                             bin_edges=None,
                                                             bin_range=None,
                                                             rescale=None,
                                                             viewer=None,
                                                             **kwargs)
```

Render localization data into a 3D RGB image by binning x,y,z-coordinates into regular bins.

---

**Note:** For `rescale=False` no normalization is carried out image intensities are clipped to (0, 1) for float value or (0, 255) for integer values according to the `matplotlib.imshow` behavior. For `rescale=None` we apply a normalization to (min, max) of all intensity values. For all other `rescale` options the normalization is applied to each individual image.

---

#### Parameters

- **locdatas** (Iterable[*LocData*]) – Localization data.
- **loc\_properties** (Optional[list[str]]) – Localization properties to be grouped into bins. If None The coordinate\_values of *locdata* are used.
- **other\_property** (Optional[str]) – Localization property (columns in *locdata.data*) that is averaged in each pixel. If None, localization counts are shown.
- **bins** (UnionType[*Bins*, Axis, AxesTuple, None]) – The bin specification as defined in *Bins*

- **bin\_edges** (UnionType[Sequence[float], Sequence[Sequence[float]], None]) – Bin edges for all or each dimension with shape (dimension, n\_bin\_edges).
- **bin\_range** (Union[tuple[float, float], Sequence[float], Sequence[Sequence[float]], Literal['zero', 'link'], None]) – Minimum and maximum edge for all or each dimensions with shape (2,) or (dimension, 2). If None (min, max) ranges are determined from data and returned; if 'zero' (0, max) ranges with max determined from data are returned. if 'link' (min\_all, max\_all) ranges with min and max determined from all combined data are returned.
- **n\_bins** (UnionType[int, Sequence[int], None]) – The number of bins for all or each dimension. 5 yields 5 bins in all dimensions. (2, 5) yields 2 bins for one dimension and 5 for the other dimension.
- **bin\_size** (UnionType[float, Sequence[float], Sequence[Sequence[float]], None]) – The size of bins for all or each bin and for all or each dimension with shape (dimension,) or (dimension, n\_bins). 5 would describe bin\_size of 5 for all bins in all dimensions. ((2, 5),) yield bins of size (2, 5) for one dimension. (2, 5) yields bins of size 2 for one dimension and 5 for the other dimension. ((2, 5), (1, 3)) yields bins of size (2, 5) for one dimension and (1, 3) for the other dimension. To specify arbitrary sequence of *bin\_size* use *bin\_edges* instead.
- **rescale** (UnionType[int, str, *Trafo*, Callable[... Any], bool, None]) – Transformation as defined in *locan.Trafo* or by transformation function. For None or False no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For 'equal' intensity values are rescaled by histogram equalization.
- **viewer** (Viewer) – The viewer object on which to add the image
- **kwargs** (Any) – Other parameters passed to `napari.Viewer.add_image()`.

#### Return type

napari.Viewer

## locan.visualize.render\_napari.utilities

Utility functions for interacting with napari.

### Functions

|   |  |
|---|--|
| <code>get_rois(shapes_layer[, reference, ...])</code>         | Create rois from shapes in <code>napari.viewer.Shapes</code> .           |
| <code>save_rois(rois[, file_path, roi_file_indicator])</code> | Save list of Roi objects.  |
| <code>select_by_drawing_napari(locdata[, napari_run])</code>  | Select region of interest from rendered image by drawing rois in napari. |



## locan.visualize.render\_napari.utilities.get\_rois

```
locan.visualize.render_napari.utilities.get_rois(shapes_layer, reference=None,  
                                                  loc_properties=None)
```

Create rois from shapes in napari.viewer.Shapes.

### Parameters

- **shapes\_layer** (Shapes) – Napari shapes layer like *viewer.layers[“Shapes”]*
- **reference** (UnionType[*LocData*, dict[str, str], Metadata, File, None]) – Reference to localization data for which the region of interest is defined. It can be a *LocData* object, a reference to a saved SMLM file, or None for indicating no specific reference. When dict it must have keys *file\_path* and *file\_type*. When Metadata message it must have keys *file.path* and *file.type* for a path pointing to a localization file and an integer or string indicating the file type. Integer or string should be according to *locan.constants.FileType*.
- **loc\_properties** (Optional[Sequence[str]]) – Localization properties in *LocData* object on which the region selection will be applied (for instance the *coordinate\_keys*).

### Return type

list[*Roi*]

See also:

### locan.scripts.rois()

script for drawing rois

## locan.visualize.render\_napari.utilities.save\_rois

```
locan.visualize.render_napari.utilities.save_rois(rois, file_path=None,  
                                                  roi_file_indicator='_roi')
```

Save list of Roi objects.

### Parameters

- **rois** (Iterable[*Roi*]) – The rois to be saved.
- **file\_path** (Union[str, PathLike[Any], Literal['roi\_reference'], None]) – Base name for roi files or existing directory to save rois in. If “roi\_reference”, *roi.reference.file.path* is used. If None, a file dialog is opened.
- **roi\_file\_indicator** (str) – Indicator to add to the localization file name and use as roi file name (with further extension .yaml).

### Returns

New created roi file paths.

### Return type

list[Path]

## locan.visualize.render\_napari.utilities.select\_by\_drawing\_napari

```
locan.visualize.render_napari.utilities.select_by_drawing_napari(locdata, na-
                                                                pari_run=True,
                                                                **kwargs)
```

Select region of interest from rendered image by drawing rois in napari.

Rois will be created from shapes in `napari.viewer.layers['Shapes']`.

### Parameters

- **locdata** (*LocData*) – The localization data from which to select localization data.
- **napari\_run** (bool) – If *True* `napari.run` is called (set to *False* for testing).
- **kwargs** (Any) – Other parameters passed to `render_2d_napari()`.

### Return type

`list[Roi]`

See also:

**locan.scripts.rois()**  
script for drawing rois

## locan.visualize.transform

Transform intensity values.

This module provides functions and classes to rescale intensity values. In addition to the presented functions, rescaling can further be performed by third-party modules like: 1) `matplotlib.colors` 2) `skimage.exposure` 3) `astropy.visualization`.

Callable transformation classes that inherit from `matplotlib.colors.Normalize` and specify an inverse transformation can be passed to the *norm* parameter.

## Classes

|  |  |
|--|--|
| <i>HistogramEqualization</i> ([vmin, vmax, ...])     | Histogram equalization with power intensification. |
| <i>Trafo</i> (value[, names, module, qualname, ...]) | Standard definitions for intensity transformation. |
| <i>Transform</i> ()                                  | Abstract base class for transformation classes.    |

## locan.visualize.transform.HistogramEqualization

```
class locan.visualize.transform.HistogramEqualization(vmin=None, vmax=None,
                                                    reference=None, power=1,
                                                    n_bins=65536, mask=None)
```

Bases: `Normalize`, `Transform`

Histogram equalization with power intensification.

The transformation function as described in<sup>1</sup> is  $f(a, p)$  according to:

$$\frac{f(a) - f(a_{min})}{f(a_{max}) - f(a_{min})} = \frac{\int_{a_{min}}^a h^p(a') da'}{\int_{a_{min}}^{a_{max}} h^p(a') da'}$$

Here,  $a$  is an intensity value,  $p$  the power (a parameter), and  $h(a)$  the histogram of intensities.

---

**Note:** The default for `n_bins` is 65536 (16 bit). For most SMLM datasets this should be sufficient to resolve individual localizations despite a large dynamic range. Setting `n_bins` to 256 (8 bit) is too coarse for many SMLM datasets.

---

## References

### Parameters

- **vmin** (`UnionType[int, float, None]`) – min intensity
- **vmax** (`UnionType[int, float, None]`) – max intensity
- **reference** (`Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]], None]`) – The data values to define the transformation function. If `None` then the values in `__call__` are used.
- **power** (`float`) – The *power* intensification parameter.
- **n\_bins** (`int`) – Number of bins used to compute the intensity histogram.
- **mask** (`Union[_SupportsArray[dtype[Any]], _NestedSequence[_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, _NestedSequence[Union[bool, int, float, complex, str, bytes]], None]`) – A bool mask with shape equal to that of values. If `reference` is `None`, `reference` is set to `values[mask]`. The transformation determined from `reference` is then applied to all values.

---

<sup>1</sup> Yaroslavsky, L. (1985) Digital picture processing. Springer, Berlin.

## Methods

|  |   |
|--|---|
| <code>__init__([vmin, vmax, reference, power, ...])</code> | <b>type vmin</b><br>UnionType[int, float, None]                                     |
| <code>autoscale(A)</code>                                  | Set <i>vmin</i> , <i>vmax</i> to min, max of <i>A</i> .                             |
| <code>autoscale_None(A)</code>                             | If <i>vmin</i> or <i>vmax</i> are not set, use the min/max of <i>A</i> to set them. |
| <code><i>inverse</i>(values)</code>                        | A Transformation object that performs the inverse operation.                        |
| <code>process_value(value)</code>                          | Homogenize the input <i>value</i> for easy and efficient normalization.             |
| <code>scaled()</code>                                      | Return whether <i>vmin</i> and <i>vmax</i> are set.                                 |

## Attributes

|                   |
|-------------------|
| <code>clip</code> |
| <code>vmax</code> |
| <code>vmin</code> |

`__call__(values)`

Histogram equalization with power intensification.

### Parameters

**values** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The input values.

### Return type

npt.NDArray

`inverse(values)`

A Transformation object that performs the inverse operation.

### Return type

ndarray[Any, dtype[Any]]

**locan.visualize.transform.Trafo**

```
class locan.visualize.transform.Trafo(value, names=None, *, module=None,
                                     qualname=None, type=None, start=1,
                                     boundary=None)
```

Bases: Enum

Standard definitions for intensity transformation.

**Attributes**

|                               |   |
|-------------------------------|---|
| <i>NONE</i>                   | no transformation   |
| <i>STANDARDIZE</i>            | rescale (min, max) to (0, 1)  |
| <i>STANDARDIZE_UINT8</i>      | rescale (min, max) to (0, 255)  |
| <i>ZERO</i>                   | rescale (0, max) to (0, 1)  |
| <i>ZERO_UINT8</i>             | rescale (0, max) to (0, 255)  |
| <i>EQUALIZE</i>               | equalize histogram from all values > 0                                |
| <i>EQUALIZE_UINT8</i>         | equalize histogram from all values > 0 onto (0, 255)                  |
| <i>EQUALIZE_ALL</i>           | equalize histogram  |
| <i>EQUALIZE_ALL_UINT8</i>     | equalize histogram onto (0, 255)                                      |
| <i>EQUALIZE_OP3</i>           | equalize histogram with power = 0.3 from all values > 0               |
| <i>EQUALIZE_OP3_UINT8</i>     | equalize histogram with power = 0.3 from all values > 0 onto (0, 255) |
| <i>EQUALIZE_OP3_ALL</i>       | equalize histogram with power = 0.3                                   |
| <i>EQUALIZE_OP3_ALL_UINT8</i> | equalize histogram with power = 0.3 onto (0, 255)                     |

**EQUALIZE = 5**

equalize histogram from all values > 0

**EQUALIZE\_OP3 = 9**

equalize histogram with power = 0.3 from all values > 0

**EQUALIZE\_OP3\_ALL = 11**

equalize histogram with power = 0.3

**EQUALIZE\_OP3\_ALL\_UINT8 = 12**

equalize histogram with power = 0.3 onto (0, 255)

**EQUALIZE\_OP3\_UINT8 = 10**

equalize histogram with power = 0.3 from all values > 0 onto (0, 255)

**EQUALIZE\_ALL = 7**

equalize histogram

**EQUALIZE\_ALL\_UINT8 = 8**

equalize histogram onto (0, 255)

**EQUALIZE\_UINT8 = 6**

equalize histogram from all values > 0 onto (0, 255)

**NONE = 0**

no transformation

**STANDARDIZE = 1**

rescale (min, max) to (0, 1)

**STANDARDIZE\_UINT8 = 2**

rescale (min, max) to (0, 255)

**ZERO = 3**

rescale (0, max) to (0, 1)

**ZERO\_UINT8 = 4**

rescale (0, max) to (0, 255)

## locan.visualize.transform.Transform

**class** locan.visualize.transform.Transform

Bases: ABC

Abstract base class for transformation classes.

## Methods

|                              |   |
|------------------------------|---|
| <code>__init__()</code>      |   |
| <code>inverse(values)</code> | A transformation that performs the inverse operation. |

**abstract** `__call__(values, clip=True)`

Transform values.

### Parameters

- **values** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – The input values
- **clip** (bool) – If *True* values outside the [0:1] range are clipped to the [0:1] range.

### Returns

The transformed values.

### Return type

npt.NDArray[Any]

**abstract inverse**(*values*)

A transformation that performs the inverse operation.

**Return type**

ndarray[Any, dtype[Any]]

## Functions

*adjust\_contrast*(image[, rescale])

Adjust contrast of image by a predefined transformation:

### locan.visualize.transform.adjust\_contrast

locan.visualize.transform.**adjust\_contrast**(*image*, *rescale=True*, *\*\*kwargs*)

Adjust contrast of image by a predefined transformation:

#### Parameters

- **image** (Union[\_SupportsArray[dtype[Any]], \_NestedSequence[\_SupportsArray[dtype[Any]]], bool, int, float, complex, str, bytes, \_NestedSequence[Union[bool, int, float, complex, str, bytes]]]) – Values to be adjusted
- **rescale** (UnionType[int, str, *Trafo*, Callable[... Any], bool, None]) – Transformation as defined in `locan.constants.Trafo` or by transformation function. For None or False no rescaling occurs. Legacy behavior: For tuple with upper and lower bounds provided in percent, rescale intensity values to be within percentile of max and min intensities. For ‘equal’ intensity values are rescaled by histogram equalization.
- **kwargs** (Any) – Other parameters that are passed to the specific Transformation class.

#### Return type

npt.NDArray[np.float64]

## CHANGELOG

### 9.1 0.20 - dev

#### 9.1.1 New Features

- ...

#### 9.1.2 API Changes

- change color of contours to gray in `LocalizationProperty2d`
- sort `show_versions` output
- add `register` method based on `skimage` and refactor `register_cc`

#### 9.1.3 Bug Fixes

- ...

#### 9.1.4 Other Changes and Additions

- various changes to documentation
- lint for `numpy=2.0` and fix new `ruff` issues
- refactor: `gitignore`
- refactor `dockerfiles`
- add tool `show_test_data` to show test data
- remove `requirements.txt` and `requirements_dev.txt`
- fix tests to be compatible with `pandas 2.2.0`
- speed up tests for `localization_precision`
- optimize test data
- fix errors and deprecation warnings for `pandas 2.0`
- switch to `black ~24.0` format
- add more benchmarks



- add dockerfile to deploy locan with jupyter lab

## 9.2 0.19 - 2023-12-12

### 9.2.1 Other Changes and Additions

- add lazy\_imports of API
- introduce src-layout in code base

## 9.3 0.18 - 2023-12-06

### 9.3.1 API Changes

- add colormaps module and change management of colormaps
- remove outdated module with analysis example

### 9.3.2 Bug Fixes

- fix version readout with readthedocs

### 9.3.3 Other Changes and Additions

- add GitHub action for deploying to PyPI and TestPyPI
- configure setuptools\_scm for branching model

## 9.4 0.17 - 2023-10-26

### 9.4.1 New Features

- feat: adapt LocData.from\_dataframe to take any dataframe that supports the dataframe interchange protocol.
- feat: add gui dialog to set file path

### 9.4.2 API Changes

- remove deprecated LocalizationUncertaintyFromIntensity

### 9.4.3 Bug Fixes

- fix: tests for shapely 2.0.2 with geos 3.12.0 in conda environment
- fix: bin-to-pixel relation in render in napari
- fix: readthedocs version readout

### 9.4.4 Other Changes and Additions

- refactor: update to python 3.12

## 9.5 0.16 - 2023-10-05

### 9.5.1 API Changes

- refactor: delete qt and needs\_datasets marker for pytest

### 9.5.2 Bug Fixes

- fix: import of pytest for type checking

### 9.5.3 Other Changes and Additions

- refactor: reduce load in GitHub actions
- add locan to conda-forge

## 9.6 0.15 - 2023-09-28

### 9.6.1 New Features

- feat: Selector class to specify loc\_property selections

### 9.6.2 API Changes

- refactor: change return type of clustering algorithms for noise from None to LocData()

### 9.6.3 Bug Fixes

- fix: `convert_property_types` and add documentation.

### 9.6.4 Other Changes and Additions

- Add type hints
- Add type checking with mypy in pre-commit and GitHub actions CI workflow
- Extend ruff linting
- refactor(test): include pytest-qt in standard tests.
- drop support for python 3.8

## 9.7 0.14 - 2023-06-30

### 9.7.1 New Features

- Add analysis routine for *SubpixelBias*
- Add function *merge\_metadata* to merge metadata from class instance or file
- Add class *Files* for managing file selections to be used in batch processing
- Add *utilities/statistics* module with helper function to compute *WeightedMeanVariance*
- Add *locdata.utilities.statistics.ratio\_fwhm\_to\_sigma* function
- Add *locdata.utilities.statistics.biased\_variance* function
- Add analysis routine for *ConvexHullExpectation*, *GroupedPropertyExpectation* and *PositionVarianceExpectation*.
- Add function to standardize locdata.
- add *Locdata.update\_properties\_in\_references* method
- add analysis class *locdata.analysis.ConvexHullExpectationBatch*.

### 9.7.2 API Changes

- Change *find\_pattern\_upstream* into *find\_file\_upstream*
- Add analysis class *LocalizationUncertainty* and deprecate *LocalizationUncertaintyFromIntensity*
- Refactor *LocData.properties* to include weighted coordinate averages and properties for frame and intensity
- Refactor *locan.simulation.simulate\_locdata.resample*
- Change *LoData.coordinate\_properties* to *LocData.coordinate\_keys* and add *LocData.uncertainty\_keys* and corresponding functions in *locan.constants.PropertyKeys*

### 9.7.3 Bug Fixes

- Fix use of callables in *LocalizationUncertainty*
- Adapt to to bug fixes in lmfit 1.2.0

### 9.7.4 Other Changes and Additions

- Use ruff for linting
- Use pyproject.toml for all specifications and deprecate use of setup.cfg
- Use import open3d-cpu instead of open3d
- docs: add tutorial for analysis of grouped cluster properties

## 9.8 0.13 - 2023-02-15

### 9.8.1 New Features

- Add CLI for `-info` and `-version`
- Add `overlay` function to transform locdatas
- Add function to standardize locdata.
- Add function `load_metadata_from_toml` to read metadata from toml file
- Add function `find_pattern_upstream`.
- Add function `transform_counts_to_photons`.

### 9.8.2 API Changes

- Add `boost_histogram_axes` property to `Bins`

### 9.8.3 Bug Fixes

- Fix intensity transform with nan
- Fix simulation on region being a rotated rectangle
- Fix `Roi.from_yaml` for rotated rectangular rois
- Fix `Roi.from_yaml` to work with Polygon regions
- Fix bug in `localization_property_2d`.
- Fix bug in `localizations_per_frame` concerning the time units
- Fix histogram with `bins=Bins_instance` as input
- Fix conversion of `bin_edges` to `bin_size` for float numbers
- Fix tests by excluding shapely 2.0.0 and 2.0.1
- Use `np.random.default_rng` for random number generation in `simulate_drift.py`

## 9.8.4 Other Changes and Additions

- Use of pre-commit
- Adapt to isort, black, flake8, bandit
- Integrate qtpy
- Add benchmark setup for Airspeed Velocity
- Modify dockerfiles to run with slim-bullseye
- Add conditional import of tomllib to replace tomli for python>=3.11
- Add CITATION.cff file
- Add some type hints

## 9.9 0.12 - 2022-06-02

### 9.9.1 API Changes

#### **locan.configuration**

- Introduced module `locan.configuration` to hold user-specific configuration values

#### **locan.constants**

- Introduced enum *PropertyKey* that holds *PropertyDescription* dataclasses with information on name, type, units and description

#### **locan.data**

- Provided new scheme for metadata
- Added tutorial about metadata for LocData
- Introduced use of protobuf *Timestamp* and *Duration* types in metadata messages
- Added function in *locan.data.metadata\_utils* to provide scheme of default metadata
- Added function in *locan.data.metadata\_utils* to read metadata from toml file

#### **locan.io**

- Refactored `locan_io.locdata` module structure

## locan.render

- Changed default value for `n_bins` in `HistogramEqualization` to increase intensity resolution. Note: This modification changes the visual presentation of localization data with a large dynamic range.

## 9.9.2 Bug Fixes

### locan.utils.system\_information

- Fixed `show_version` to read out all dependency versions

### locan.data

- Fixed bug in cluster functions such that setting the region of empty collections does not raise a `TypeError` anymore
- Fixed protobuf issues related to protobuf 4.21

## 9.9.3 Other Changes and Additions

- Dropped support for python 3.7
- Various minor changes of documentation and code
- Removed numba as dependency
- Based conda-related dockerfiles on mambaforge
- Introduced use of fixture from `pytest-qt` for testing QT interfaces

## 9.10 0.11.1 - 2022-04-08

### 9.10.1 Bug Fixes

#### locan.scripts

- Fix a bug introduced in 0.11 in `napari` and `rois` script.

## 9.11 0.11 - 2022-03-22

### 9.11.1 New Features

#### locan.data

- Modified `Polygon.contains` function to increase performance.
- Implemented `randomize` function for all hull types.

## **locan.io**

- Added methods to load DECODE and SMAP files.

## **locan.render**

- Added rendering functions for 3D
- Added rendering functions for RGB image (multi-color overlay)

### **9.11.2 API Changes**

## **locan.io**

- Extended `load_txt_files` to convert property names to locan standard property names.

## **locan.render**

- Refactored intensity rescaling by introducing standard normalization procedures.

### **9.11.3 Bug Fixes**

## **locan.data**

- Fixed bunwarp transformation

## **locan.io**

- Fixed lineterminator in `load_rapidstorm_track_file`

### **9.11.4 Other Changes and Additions**

- Ensured support of locan and all optional dependencies for Python 3.9
- Ensured support of locan (without optional dependencies) for Python 3.10
- Turned `hdbscan` into optional dependency

## **9.12 0.10 - 2021-11-23**

### **9.12.1 New Features**

## **locan.io**

- Add function to load rapidSTORM file with tracked data.
- Add function to load and save SMLM file.

### 9.12.2 Other Changes and Additions

- Locan went public.
- Readthedocs was set up.
- Zenodo DOI was generated.

## 9.13 0.9 - 2021-11-11

### 9.13.1 API Changes

#### **locan.analysis**

- Refactor computation of blinking\_statistics

#### **locan.data**

- Restructured Region management introducing new classes in locan.data.region
- Rename function to computer inertia moments

#### **locan.render**

- Change image orientation in render\_2d\_napari to be consistent with points coordinates.

#### **locan.simulation**

- Refactored simulation functions to make use of numpy random number generator.
- Refactored simulation functions to generate Neyman-Scott point processes in expanded regions.
- Add function to simulate dSTORM data as localization clusters with normal-distributed coordinates and geometric-distributed number of localizations.

### 9.13.2 Other Changes and Additions

- Added or modified tutorials on multiprocessing, regions and simulation.
- Introduce pytest markers for test functions that are excluded from test run per default.



## 9.14 0.8 - 2021-05-06

### 9.14.1 API Changes

#### locan.scripts

- Default values for verbose and extra flags in script show\_versions were changed.

### 9.14.2 Bug Fixes

#### locan.analysis

- Fit procedure was fixed for NearestNeighborDistances.

### 9.14.3 Other Changes and Additions

- Library was renamed to LOCAN
- Documentation and tutorials were modified accordingly
- Test coverage was improved and use of coverage.py introduced
- \_future module was deprecated

## 9.15 0.7 - 2021-03-26

### 9.15.1 API Changes

#### locan.analysis

- Added new keyword parameters in LocData.from\_chunks and Drift.
- Extended class for blinking analysis.

### 9.15.2 Other Changes and Additions

- Turn warning into log for file io.
- Restructured documentation, added tutorials, and changed html-scheme to furo.

## 9.16 0.6 - 2021-03-04

### 9.16.1 New Features

- Introduced logging capability.
- Added script for running tests from command line interface.

#### **locan.analysis**

- Make all analysis classes pickleable.
- Refactored Pipeline class
- Enabled and tested multiprocessing based on multiprocessing or ray.
- Added more processing bars.
- Added drift analysis and correction based on imagecorrelation and iterative closest point registration.

#### **locan.data**

- Made LocData class pickleable.
- Added computation of inertia moments.
- Added orientation property based on oriented bounding box and inertia moments.
- Added elongation property based on oriented bounding box.
- Add transformation method to overlay LocData objects.

#### **locan.io**

- Added loading function for Nanoimager data.

#### **locan.render**

- Added windowing function for image data.

### 9.16.2 API Changes

#### **locan.analysis**

- Fixed and extended methods for Drift analysis and correction.

### **locan.data**

- Implemented copy and deepcopy for LocData.
- Changed noise output in clustering methods. Removed noise parameter.

### **locan.datasets**

- Added dataset for microtubules

### **locan.io**

- Added option for file-like objects in io\_locdata functions.
- Added Bins class, introduced use of boost-histogram package, and restructured binning.
- Introduced use of napari.run.
- Changed default value in render\_2d\_mpl to interpolation='nearest'.

### **locan.scripts**

- Added arguments for locan napari and locan rois.

### **locan.simulation**

- Added simulation of frame values.

## **9.16.3 Bug Fixes**

### **locan.data**

- Fixed treatment of empty LocData in clustering and hull functions.

### **locan.gui**

- Use PySide2 as default QT backend depending on QT\_API setting.

### **locan.io**

- Fixed encoding issues for loading Elyra data.

### 9.16.4 Other Changes and Additions

- Test data is included in distribution.
- New dockerfiles for test and deployment.
- Included pyproject.toml file

## 9.17 0.5.1 - 2020-03-25

- Update environment and requirement files

## 9.18 0.5 - 2020-03-22

### 9.18.1 New Features

#### locan.utils

- Module locan.utils.system\_information with methods to get debugging information is added.

#### locan.analysis

- LocalizationPropertyCorrelation analysis class is added.

#### locan.data

- LocData.from\_coordinates() is added.
- LocData.update() method is added to change dataframe with corresponding updates of hull, properties and metadata.
- Methods to compute alpha shape hulls are added.
- Pickling capability for LocData is added.

#### locan.render

- scatter\_2d\_mpl() is added. to show locdata as scatter plot

#### locan.scripts

- show\_versions()

## 9.18.2 API Changes

### locan.analysis

- LocalizationProperty2D was modified and fixed.

### locan.data

- locan.data.region\_utils module is added with utility functions to analyze locdata regions.
- RoiRegions are added that support shapely Polygon and MultiPolygon objects.

## 9.18.3 Bug Fixes

### locan.analysis

- Adapt colormap and rescaling in LocalizationProperty2D plot functions.

## 9.19 0.4.1 - 2020-02-16

### 9.19.1 Bug Fixes

#### locan.analysis

- Fix LocalizationProperty2d fit procedure

### 9.19.2 Other Changes and Additions

- Increase import performance

## 9.20 0.4 - 2020-02-13

### 9.20.1 New Features

- New function test() to run pytest on whole test suite.

#### locan.data

- New rasterize function to divide localization support into rectangular rois.
- New functions to perform affine transformation using open3d.
- New functions to perform registration using open3d.
- New function for drift correction using icp (from open3d).
- Increase performance of maximum distance computation of localization data.

### locan.datasets

- Added functions to load example datasets. The datasets will be provided in a separate directory (repository).

### locan.scripts

- Introduced command-line interface with compound commands.
- New script to render localization data in napari
- New script to define and save rois using napari
- New script to render localizations onto raw data images

## 9.20.2 API Changes

### locan.analysis

- New analysis class for drift estimation.
- New analysis class for analysing 2d distribution of localization property.

### locan.data

- Deprecate *update\_convex\_hull\_in\_collection()*. Use *LocData.update\_convex\_hulls\_in\_references()*.
- Metadata on time is changed from timestamp to formatted time expression.

### locan.render

- Default colormaps are set to selected ones from colorcet or matplotlib.
- Add histogram function for rendering localization data.
- Add render functions to work with mpl, mpl-scatter-density, napari

### locan.scripts

- Add selection option for ellipse roi.

### locan.simulation

- Add functions for drift simulation.

### 9.20.3 Bug Fixes

#### locan.data

- Fixed update of bounding\_box, convex\_hull and oriented bounding box.

### 9.20.4 Other Changes and Additions

- Added centroid and dimension property to LocData.
- Implemented use of QT\_API to set the QT bindings and work in combination with napari.
- Make shapely a required dependency.

## 9.21 0.3 - 2019-07-09

### 9.21.1 New Features

#### locan.analysis

- Added analysis class BlinkStatistics to compute on/off times in localization cluster.

#### locan.data

- Introduced global variable locdata\_id that serves as standard running ID for LocData objects.
- Added function update\_convex\_hulls\_in\_collection

### 9.21.2 API Changes

#### locan.analysis

- Refactored all analysis class names to CamelCode.
- Refactored handling of LocData input in analysis classes to better resemble the scikit-learn API.

#### locan.simulation

- Deleted deprecated simulation functions.

### 9.21.3 Other Changes and Additions

- Refactored all localization property names to follow the convention to start with small letters.
- Changed import organization by adding `__add__` to enable import locan as `sp`.
- Added dockerfiles for using and testing locan.
- various other small changes and fixes as documented in the version control log.

## 9.22 0.2 - 2019-03-22

### 9.22.1 New Features

#### **locan.analysis**

- implemented an analysis class `CoordinateBasedColocalization`.
- implemented an analysis class `AccumulationClusterCheck`.

#### **locan.data**

- implemented a function `exclude_sparse_points` to eliminate localizations in low local density regions.
- implemented a function to apply affine coordinate transformations.
- implemented a function to apply a Bunwarp-transformation based on the raw transformation matrix from the ImageJ plugin `BUnwarpJ`

#### **locan.simulation**

- implemented functions to simulate localization data based on complete spatial randomness, Thomas, or Matern processes.
- implemented functions `simulate_xxx` to provided `LocData` objects.
- implemented functions `make_xxx` to provide point coordinates.

### 9.22.2 API Changes

#### **locan.data**

- implemented a new region of interest management. A `RoiRegion` class was defined as region object in `Roi` objects.



### 9.22.3 Bug Fixes

#### locan.data

- corrected index handling in track.track(), LocData.data and LocData.reduce().

#### locan.io

- changed types for column values returned from load\_thunderstorm\_file.

## 9.23 0.1 - 2018-12-09

### 9.23.1 New Features

#### locan.analysis

- localization\_precision
- localization\_property
- localizations\_per\_frame
- nearest\_neighbor
- pipeline
- ripley
- uncertainty

#### locan.data

- cluster
- properties
- filter
- hulls
- locdata
- rois
- track
- transformation

## **locan.gui**

- io

## **locan.io**

- io\_locdata

## **locan.render**

- render2d

## **locan.scripts**

- sc\_draw\_roi\_mpl

## **locan.simulation**

- simulate\_locdata

## **9.23.2 Other Changes and Additions**

### **locan.tests**

- corresponding unit tests

### **docs**

- rst files for sphinx documentation.

### **locan**

- CHANGES.rst
- LICENSE.md
- README.md
- environment.yml
- environment\_dev.yml

## CONTRIBUTIONS

### 10.1 Idea and Lead

- Sören Doose

### 10.2 Various Contributions

(in alphabetical order)

- Sarah Aufmkolk
- Philip Kollmansberger
- Sebastian Reinhard
- Felix Repp
- Sven Proppert
- Felix Wäldchen
- Steve Wolter

## **LICENSE**

### BSD 3-Clause License

Copyright (c) 2018-2021, Biotechnologie und Biophysik - Universität Würzburg All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## **OTHER LICENSES**

Locan makes use of libraries and code that are compatibly licensed as listed in the licences directory.

## DOCUMENTATION

Documentation is provided as restructured text, myst markdown, and as docstrings within the code. HTML pages and other documentation formats are build using [Sphinx](#).

We follow standard recommendations for [python documentation](#) and the [numpy conventions](#).

### 13.1 Update documentation

To update the documentation from sources delete `/docs/sources/generated` and run:

```
sphinx-build -b html -E YOUR_PATH\Locan\docs YOUR_PATH\Locan\docs\_build
```

### 13.2 Type hints

We try to make use of type checking using mypy as much as possible.

Type hints should be given as annotations to enable type checking.

Using the sphinx extension `sphinx_autodoc_typehints` (and `napoleon`), we convert type hints from function annotations automatically into the numpy-style documentation.

In case of duplicate documentation (annotation and docstring) the docstring is transferred into the sphinx-generated documentation but annotations are used for type-checking.

### 13.3 Example docstring

We try to follow standard docstrings as illustrated here:

```
def function(  
    par: None | UserClass = None, **kwargs  
) -> None:  
    """  
    Short title.  
  
    Long description about parameter `par` with some literature reference [1]_.  
  
    This paragraph might describe some ``code`` and include an equation:
```

(continues on next page)

(continued from previous page)

```

.. math::

    \\f(x) = \\frac{a}{(b + c)}

Parameters
-----
par : None | UserClass
    Description
kwargs : dict
    Parameters passed to some other documented function :func:`function_
↪name`

Returns
-----
None

Examples
-----
>>> 1 + 2
3

See Also
-----
:func:`locan.tests.test`

Note
----
Whatever there is to note.

References
-----
.. [1] <authors>, <title>. <journal> <year>, <volume>:<pages>. <doi>
"""
return None

class SomeClass:
    """
    Short title.

    Long description.

    Parameters
    -----
    par : None, other type
        Description
    kwargs : dict
        Parameters passed to some other documented function :func:`function_

```

(continues on next page)

(continued from previous page)

```
↪name`  
  
    Attributes  
    -----  
    arg : None, other type  
        Description  
    ""  
    return None
```



## DEVELOPMENT

We welcome any contributions for improving or further developing this package. However, please excuse that we are limited in time for development and support. Some things to keep in mind when adding code...

### 14.1 Install

A few extra libraries are needed for development:

```
pip install .[dev]
```

Alternatively, you may use the requirement files *requirements\_dev.txt* or *environment\_dev.yml*.

### 14.2 Import Conventions

The following import conventions are used throughout Locan source code and documentation:

- `import numpy as np`
- `import pandas as pd`
- `import matplotlib as mpl`
- `import matplotlib.pyplot as plt`
- `import scipy as sp`
- `import open3d as o3d`
- `import networkx as nx`
- `import boost_histogram as bh`

This is enforced through ruff following specifications in *pyproject.toml*.

## 14.3 Unit tests

For testing we use `py.test`.

A test suite is provided in `locan/tests`.

For unit testing we supply test data as data files located in `locan/tests/test_data`.

## 14.4 Coverage

For measuring code coverage in testing we use `coverage.py`.

Configurations are kept in `pyproject.toml`.

## 14.5 Versioning

We use `SemVer` for versioning. For all versions available, see the [releases in this repository](#).

## 14.6 To remember:

- The package makes use of third party packages that are only used by a few routines.

These optional packages are treated as `extra_dependencies`. Import of optional packages is tried in the `constants` module and a `_has_package` variable is defined. In each module that makes use of an optional import a conditional import is carried out

```
if _has_package: import package
```

In addition any user-accessible function that makes use of the optional package raises an import error if the optional package is not available.

Test functions that require optional dependencies should be marked with:

```
@pytest.mark.skipif(not _has_package, reason="requires optional package")
```

- The length of code and docstring lines should be kept to 88 and 75 characters, respectively.
- Use notation `n_something` for *number\_of\_something*.
- Provide commit messages with subject in imperative style (see [Chris Beams, How to Write a Git Commit Message](#)). If possible follow the specifications for [conventional commits](#).

## INDICES AND TABLES

- [genindex](#)
- [modindex](#)
- [search](#)
- [glossary](#)

## PYTHON MODULE INDEX

|  
locan, 259  
locan.analysis, 259  
locan.analysis.accumulation\_analysis, 260  
locan.analysis.blinking, 263  
locan.analysis.cbc, 265  
locan.analysis.convex\_hull\_expectation, 268  
locan.analysis.drift, 275  
locan.analysis.grouped\_property\_expectation, 282  
locan.analysis.localization\_precision, 285  
locan.analysis.localization\_property, 303  
locan.analysis.localization\_property\_2d, 305  
locan.analysis.localization\_property\_correlations, 310  
locan.analysis.localizations\_per\_frame, 311  
locan.analysis.nearest\_neighbor, 314  
locan.analysis.pipeline, 322  
locan.analysis.position\_variance\_expectation, 324  
locan.analysis.ripley, 328  
locan.analysis.subpixel\_bias, 333  
locan.analysis.uncertainty, 335  
locan.configuration, 352  
locan.constants, 339  
locan.data, 353  
locan.data.aggregate, 431  
locan.data.cluster, 445  
locan.data.cluster.clustering, 445  
locan.data.cluster.utils, 448  
locan.data.filter, 435  
locan.data.hulls, 366  
locan.data.hulls.alpha\_shape, 369  
locan.data.hulls.hull, 366  
locan.data.locdata, 353  
locan.data.metadata\_utils, 450  
locan.data.properties, 362  
locan.data.properties.misc, 362  
locan.data.region, 375  
locan.data.region\_utils, 426  
locan.data.register, 428  
locan.data.tracking, 449  
locan.data.transform, 440  
locan.data.transform.bunwarpj, 441  
locan.data.transform.intensity\_transformation, 444  
locan.data.transform.spatial\_transformation, 442  
locan.data.validation, 452  
locan.datasets, 453  
locan.dependencies, 455  
locan.gui, 458  
locan.gui.io, 458  
locan.locan\_io, 459  
locan.locan\_io.files, 459  
locan.locan\_io.locdata, 464  
locan.locan\_io.locdata.asdf\_io, 474  
locan.locan\_io.locdata.decode\_io, 478  
locan.locan\_io.locdata.elyra\_io, 472  
locan.locan\_io.locdata.io\_locdata, 465  
locan.locan\_io.locdata.nanoimager\_io, 473  
locan.locan\_io.locdata.rapidstorm\_io, 468  
locan.locan\_io.locdata.smap\_io, 479  
locan.locan\_io.locdata.smlm\_io, 475  
locan.locan\_io.locdata.thunderstorm\_io, 470  
locan.locan\_io.locdata.utilities, 467  
locan.locan\_io.utilities, 480  
locan.rois, 481  
locan.rois.roi, 481  
locan.scripts, 485  
locan.scripts.script\_check, 486  
locan.scripts.script\_draw\_roi, 489  
locan.scripts.script\_napari, 491

- `locan.scripts.script_rois`, [488](#)
- `locan.scripts.script_show_versions`, [492](#)
- `locan.scripts.script_test`, [492](#)
- `locan.simulation`, [493](#)
- `locan.simulation.simulate_drift`, [507](#)
- `locan.simulation.simulate_locdata`, [493](#)
- `locan.tests`, [508](#)
- `locan.utils`, [509](#)
- `locan.utils.miscellaneous`, [509](#)
- `locan.utils.statistics`, [510](#)
- `locan.utils.system_information`, [512](#)
- `locan.visualize`, [514](#)
- `locan.visualize.colormap`, [514](#)
- `locan.visualize.render`, [519](#)
- `locan.visualize.render_mpl`, [519](#)
- `locan.visualize.render_mpl.render2d`,  
[520](#)
- `locan.visualize.render_mpl.render3d`,  
[525](#)
- `locan.visualize.render_napari`, [526](#)
- `locan.visualize.render_napari.render2d`,  
[526](#)
- `locan.visualize.render_napari.render3d`,  
[531](#)
- `locan.visualize.render_napari.utilities`,  
[535](#)
- `locan.visualize.transform`, [537](#)

## Symbols

`__call__()` (*locan.visualize.colormap.Colormap* method), 516

`__call__()` (*locan.visualize.transform.HistogramEqualization* method), 539

`__call__()` (*locan.visualize.transform.Transform* method), 541

## A

`AccumulationClusterCheck` (class in *locan.analysis.accumulation\_analysis*), 261

`add_drift()` (in module *locan.simulation.simulate\_drift*), 508

`add_glob()` (*locan.locan\_io.files.Files* method), 461

`adjust_contrast()` (in module *locan.visualize.transform*), 542

`alpha` (*locan.data.hulls.alpha\_shape.AlphaShape* property), 374

`alpha` (*locan.data.region.Cuboid* property), 386

`ALPHA_SHAPE` (*locan.constants.HullType* attribute), 344

`alpha_shape` (*locan.data.hulls.alpha\_shape.AlphaShape* property), 375

`alpha_shape` (*locan.data.locdata.LocData* property), 356

`alpha_shape()` (*locan.data.hulls.alpha\_shape.AlphaComplex* method), 371

`AlphaComplex` (class in *locan.data.hulls.alpha\_shape*), 370

`alphas()` (*locan.data.hulls.alpha\_shape.AlphaComplex* method), 371

`AlphaShape` (class in *locan.data.hulls.alpha\_shape*), 373

`angle` (*locan.data.region.Ellipse* property), 391

`angle` (*locan.data.region.Rectangle* property), 409

`apply_correction()` (*locan.analysis.drift.Drift* method), 278

`apply_window()` (in module *locan.visualize.render\_mpl.render2d*), 520

`as_artist()` (*locan.data.region.AxisOrientedCuboid* method), 377

`as_artist()` (*locan.data.region.Cuboid* method), 386

`as_artist()` (*locan.data.region.Ellipse* method), 391

`as_artist()` (*locan.data.region.EmptyRegion* method), 394

`as_artist()` (*locan.data.region.Interval* method), 398

`as_artist()` (*locan.data.region.MultiPolygon* method), 402

`as_artist()` (*locan.data.region.Polygon* method), 405

`as_artist()` (*locan.data.region.Rectangle* method), 409

`as_artist()` (*locan.data.region.Region1D* method), 416

`as_artist()` (*locan.data.region.Region2D* method), 418

`as_artist()` (*locan.data.region.Region3D* method), 421

`as_artist()` (*locan.data.region.RoiRegion* method), 425

`ASDF` (*locan.constants.FileType* attribute), 343

`AxisOrientedCuboid` (class in *locan.data.region*), 376

`AxisOrientedHyperCuboid` (class in *locan.data.region*), 380

## B

`beta` (*locan.data.region.Cuboid* property), 387

`biased_variance()` (in module *locan.utils.statistics*), 511

`bin_centers` (*locan.data.aggregate.Bins* prop-

- `erty`), 433
  - `bin_edges` (*locan.data.aggregate.Bins* property), 433
  - `bin_range` (*locan.data.aggregate.Bins* property), 433
  - `bin_size` (*locan.data.aggregate.Bins* property), 433
  - `Bins` (class in *locan.data.aggregate*), 431
  - `bins` (*locan.analysis.localization\_property\_2d.FitImage* attribute), 306
  - `BlinkStatistics` (class in *locan.analysis.blinking*), 263
  - `boost_histogram_axes` (*locan.data.aggregate.Bins* property), 433
  - `BOUNDING_BOX` (*locan.constants.HullType* attribute), 344
  - `bounding_box` (*locan.data.locdata.LocData* property), 356
  - `bounding_box` (*locan.data.region.AxisOrientedCuboid* property), 378
  - `bounding_box` (*locan.data.region.AxisOrientedHypercuboid* property), 381
  - `bounding_box` (*locan.data.region.Cuboid* property), 387
  - `bounding_box` (*locan.data.region.EmptyRegion* property), 394
  - `bounding_box` (*locan.data.region.Interval* property), 398
  - `bounding_box` (*locan.data.region.Region* property), 412
  - `bounding_box` (*locan.data.region.Region2D* property), 418
  - `BoundingBox` (class in *locan.data.hulls.hull*), 366
  - `bounds` (*locan.data.region.AxisOrientedCuboid* property), 378
  - `bounds` (*locan.data.region.AxisOrientedHypercuboid* property), 381
  - `bounds` (*locan.data.region.Cuboid* property), 387
  - `bounds` (*locan.data.region.EmptyRegion* property), 395
  - `bounds` (*locan.data.region.Interval* property), 398
  - `bounds` (*locan.data.region.Region* property), 413
  - `bounds` (*locan.data.region.Region2D* property), 418
  - `buffer()` (*locan.data.region.AxisOrientedCuboid* method), 378
  - `buffer()` (*locan.data.region.AxisOrientedHypercuboid* method), 382
  - `buffer()` (*locan.data.region.Cuboid* method), 387
  - `buffer()` (*locan.data.region.EmptyRegion* method), 395
  - `buffer()` (*locan.data.region.Interval* method), 399
  - `buffer()` (*locan.data.region.Region* method), 413
  - `buffer()` (*locan.data.region.Region2D* method), 418
  - `bunwarp()` (in module *locan.data.transform.bunwarpj*), 441
- ## C
- `CATEGORICAL` (*locan.visualize.colormap.Colormaps* attribute), 518
  - `center` (*locan.data.region.Ellipse* property), 391
  - `centroid` (*locan.data.locdata.LocData* property), 356
  - `centroid` (*locan.data.region.AxisOrientedCuboid* property), 378
  - `centroid` (*locan.data.region.AxisOrientedHypercuboid* property), 382
  - `centroid` (*locan.data.region.Cuboid* property), 387
  - `centroid` (*locan.data.region.Ellipse* property), 392
  - `centroid` (*locan.data.region.EmptyRegion* property), 395
  - `centroid` (*locan.data.region.Interval* property), 399
  - `centroid` (*locan.data.region.MultiPolygon* property), 402
  - `centroid` (*locan.data.region.Polygon* property), 406
  - `centroid` (*locan.data.region.Rectangle* property), 410
  - `centroid` (*locan.data.region.Region* property), 413
  - `channel` (*locan.constants.PropertyKey* attribute), 347
  - `chi_square` (*locan.constants.PropertyKey* attribute), 347
  - `cluster_by_bin()` (in module *locan.data.cluster.clustering*), 446
  - `cluster_dbscan()` (in module *locan.data.cluster.clustering*), 447
  - `cluster_hdbscan()` (in module *locan.data.cluster.clustering*), 447
  - `cluster_label` (*locan.constants.PropertyKey* attribute), 347

tribute), 348

Collection (class in *locan.data.validation*), 453

Colormap (class in *locan.visualize.colormap*), 516

COLORMAP\_DEFAULTS (in module *locan.configuration*), 352

colormap\_registry (in module *locan.visualize.colormap*), 515

Colormaps (class in *locan.visualize.colormap*), 517

computation\_as\_string() (*locan.analysis.pipeline.Pipeline* method), 323

computation\_test() (in module *locan.analysis.pipeline*), 324

compute() (*locan.analysis.accumulation\_analysis.AccumulationAnalysis* method), 262

compute() (*locan.analysis.blinking.BlinkStatistics* method), 264

compute() (*locan.analysis.cbc.CoordinateBasedColocalization* method), 267

compute() (*locan.analysis.convex\_hull\_expectation.ConvexHullExpectation* method), 269

compute() (*locan.analysis.convex\_hull\_expectations.ConvexHullExpectations* method), 272

compute() (*locan.analysis.drift.Drift* method), 278

compute() (*locan.analysis.grouped\_property\_expectation.GroupedPropertyExpectation* method), 283

compute() (*locan.analysis.localization\_precision.LocalizationPrecision* method), 287

compute() (*locan.analysis.localization\_property.LocalizationProperty* method), 304

compute() (*locan.analysis.localization\_property\_2d.LocalizationProperty2d* method), 308

compute() (*locan.analysis.localization\_property\_covariance.LocalizationPropertyCovariance* method), 311

compute() (*locan.analysis.localizations\_per\_frame.LocalizationsPerFrame* method), 313

compute() (*locan.analysis.nearest\_neighbor.NearestNeighbor* method), 320

compute() (*locan.analysis.pipeline.Pipeline* method), 323

compute() (*locan.analysis.position\_variance\_expectation.PositionVarianceExpectation* method), 326

compute() (*locan.analysis.ripley.RipleysHFunction* method), 329

compute() (*locan.analysis.ripley.RipleysKFunction* method), 331

compute() (*locan.analysis.ripley.RipleysLFunction* method), 332

compute() (*locan.analysis.subpixel\_bias.SubpixelBias* method), 334

compute() (*locan.analysis.uncertainty.LocalizationUncertainty* method), 337

compute\_convex\_hull\_region\_measure\_2d() (in module *locan.analysis.convex\_hull\_expectation*), 275

concat() (*locan.data.locdata.LocData* class method), 356

concatenate() (*locan.locan\_io.files.Files* class method), 461

condition (*locan.data.filter.Selector* property), 436

connected\_components (*locan.analysis.alpha\_shape.AlphaShape* property), 375

contains() (*locan.data.region.AxisOrientedCuboid* method), 378

contains() (*locan.data.region.AxisOrientedHyperCuboid* method), 382

contains() (*locan.data.region.Cuboid* method), 387

contains() (*locan.data.region.Ellipse* method), 392

contains() (*locan.data.region.EmptyRegion* method), 395

contains() (*locan.data.region.Interval* method), 399

contains() (*locan.data.region.MultiPolygon* method), 402

contains() (*locan.data.region.Polygon* method), 406

contains() (*locan.data.region.Rectangle* method), 410

contains() (*locan.data.region.Region* method), 413

contains() (*locan.data.region.RoiRegion* method), 426

CONTINUOUS (*locan.visualize.colormap.Colormaps* attribute), 518

CONTINUOUS\_GRAY (*locan.visualize.colormap.Colormaps* attribute), 518

CONTINUOUS\_GRAY\_REVERSE (*locan.visualize.colormap.Colormaps* attribute), 518

CONTINUOUS\_REVERSE (*locan.visualize.colormap.Colormaps* attribute), 518



- attribute*), 518
  - `convert_property_names()` (in module *locan.locan\_io.locdata.utilities*), 467
  - `convert_property_types()` (in module *locan.locan\_io.locdata.utilities*), 467
  - `CONVEX_HULL` (*locan.constants.HullType* attribute), 344
  - `convex_hull` (*locan.data.locdata.LocData* property), 357
  - `ConvexHull` (class in *locan.data.hulls.hull*), 367
  - `ConvexHullExpectation` (class in *locan.analysis.convex\_hull\_expectation*), 268
  - `ConvexHullExpectationBatch` (class in *locan.analysis.convex\_hull\_expectation*), 271
  - `ConvexHullExpectationResults` (class in *locan.analysis.convex\_hull\_expectation*), 273
  - `ConvexHullExpectationValues` (class in *locan.analysis.convex\_hull\_expectation*), 273
  - `ConvexHullProperty` (class in *locan.analysis.convex\_hull\_expectation*), 274
  - `coordinate_keys` (*locan.data.locdata.LocData* property), 357
  - `coordinate_keys` (*locan.data.validation.Collection* attribute), 453
  - `coordinate_keys()` (*locan.constants.PropertyKey* class method), 348
  - `coordinate_properties()` (*locan.constants.PropertyKey* class method), 348
  - `CoordinateBasedColocalization` (class in *locan.analysis.cbc*), 266
  - `coordinates` (*locan.data.locdata.LocData* property), 357
  - `corner` (*locan.data.region.AxisOrientedCuboid* property), 379
  - `corner` (*locan.data.region.AxisOrientedHypercuboid* property), 382
  - `corner` (*locan.data.region.Cuboid* property), 388
  - `corner` (*locan.data.region.Rectangle* property), 410
  - `count` (*locan.analysis.accumulation\_analysis.AccumulationAnalysis* attribute), 262
  - `count` (*locan.analysis.blinking.BlinkStatistics* attribute), 264
  - `count` (*locan.analysis.cbc.CoordinateBasedColocalization* attribute), 267
  - `count` (*locan.analysis.drift.Drift* attribute), 278
  - `count` (*locan.analysis.localizations\_per\_frame.LocalizationsPerFrame* attribute), 313
  - `count` (*locan.analysis.nearest\_neighbor.NearestNeighborDistances* attribute), 320
  - `count` (*locan.analysis.ripley.RipleysHFunction* attribute), 330
  - `count` (*locan.analysis.ripley.RipleysKFunction* attribute), 331
  - `count` (*locan.analysis.ripley.RipleysLFunction* attribute), 332
  - `count` (*locan.analysis.subpixel\_bias.SubpixelBias* attribute), 334
  - `count` (*locan.analysis.uncertainty.LocalizationUncertainty* attribute), 337
  - `count` (*locan.data.locdata.LocData* attribute), 357
  - `Cuboid` (class in *locan.data.region*), 384
  - `CUSTOM` (*locan.constants.FileType* attribute), 343
- ## D
- `data` (*locan.data.locdata.LocData* property), 357
  - `data` (*locan.data.validation.Collection* attribute), 453
  - `DATASETS_DIR` (in module *locan.configuration*), 352
  - `DECODE` (*locan.constants.FileType* attribute), 343
  - `DECODE_KEYS` (in module *locan.constants*), 340
  - `dependency_info()` (in module *locan.utils.system\_information*), 513
  - `description` (*locan.constants.PropertyDescription* attribute), 345
  - `dimension` (*locan.data.aggregate.Bins* property), 433
  - `dimension` (*locan.data.region.AxisOrientedHypercuboid* property), 382
  - `dimension` (*locan.data.region.EmptyRegion* property), 395
  - `dimension` (*locan.data.region.Region* property), 413
  - `dimension` (*locan.data.region.Region1D* property), 416
  - `dimension` (*locan.data.region.Region2D* property), 419
  - `dimension` (*locan.data.region.Region3D* property), 422
  - `distance_to_region()` (in module *locan.data.properties.misc*), 364

- `distance_to_region_boundary()` (in module `locan.data.properties.misc`), 364
- `DIVERGING` (`locan.visualize.colormap.Colormaps` attribute), 518
- `Drift` (class in `locan.analysis.drift`), 276
- `DriftComponent` (class in `locan.analysis.drift`), 280
- `DriftModel` (class in `locan.analysis.drift`), 281
- ## E
- `eccentricity` (`locan.data.properties.misc.InertiaMoments` attribute), 363
- `eigenvalues` (`locan.data.properties.misc.InertiaMoments` attribute), 363
- `eigenvectors` (`locan.data.properties.misc.InertiaMoments` attribute), 363
- `Ellipse` (class in `locan.data.region`), 389
- `ELYRA` (`locan.constants.FileType` attribute), 343
- `ELYRA_KEYS` (in module `locan.constants`), 340
- `EmptyRegion` (class in `locan.data.region`), 393
- `EQUALIZE` (`locan.visualize.transform.Trafo` attribute), 540
- `EQUALIZE_OP3` (`locan.visualize.transform.Trafo` attribute), 540
- `EQUALIZE_OP3_ALL` (`locan.visualize.transform.Trafo` attribute), 540
- `EQUALIZE_OP3_ALL_UINT8` (`locan.visualize.transform.Trafo` attribute), 540
- `EQUALIZE_OP3_UINT8` (`locan.visualize.transform.Trafo` attribute), 540
- `EQUALIZE_ALL` (`locan.visualize.transform.Trafo` attribute), 540
- `EQUALIZE_ALL_UINT8` (`locan.visualize.transform.Trafo` attribute), 540
- `equalize_bin_size()` (`locan.data.aggregate.Bins` method), 433
- `EQUALIZE_UINT8` (`locan.visualize.transform.Trafo` attribute), 540
- `eval()` (`locan.analysis.drift.DriftComponent` method), 280
- `eval()` (`locan.analysis.drift.DriftModel` method), 281
- `exclude()` (`locan.locan_io.files.Files` method), 462
- `exclude_sparse_points()` (in module `locan.data.filter`), 437
- `expand_region()` (in module `locan.data.region_utils`), 426
- `expectation` (`locan.analysis.convex_hull_expectation.ConvexHullExpectation` attribute), 274
- `extent` (`locan.data.region.AxisOrientedCuboid` property), 379
- `extent` (`locan.data.region.AxisOrientedHyperCuboid` property), 382
- `extent` (`locan.data.region.Cuboid` property), 388
- `extent` (`locan.data.region.EmptyRegion` property), 395
- `extent` (`locan.data.region.Interval` property), 399
- `extent` (`locan.data.region.Region` property), 413
- `extent` (`locan.data.region.Region2D` property), 419
- `EXTRAS_REQUIRE` (in module `locan.dependencies`), 455
- ## F
- `file_dialog()` (in module `locan.gui.io`), 458
- `Files` (class in `locan.locan_io.files`), 460
- `FileType` (class in `locan.constants`), 342
- `filter_condition()` (in module `locan.data.filter`), 438
- `find_file_upstream()` (in module `locan.locan_io.utilities`), 480
- `fit()` (`locan.analysis.drift.DriftComponent` method), 281
- `fit()` (`locan.analysis.drift.DriftModel` method), 282
- `fit_distributions()` (`locan.analysis.blinking.BlinkStatistics` method), 264
- `fit_distributions()` (`locan.analysis.localization_precision.LocalizationPrecision` method), 287
- `fit_distributions()` (`locan.analysis.localization_property.LocalizationProperty` method), 304
- `fit_distributions()` (`locan.analysis.localizations_per_frame.LocalizationsPerFrame` method), 313
- `fit_distributions()` (`locan.analysis.nearest_neighbor.NearestNeighborDistances` method), 320
- `fit_transformation()` (lo-

*can.analysis.drift.Drift method*), 278

`fit_transformations()` (locan.analysis.drift.Drift method), 279

`FitImageResults` (class in locan.analysis.localization\_property\_2d), 306

`frame` (locan.constants.PropertyKey attribute), 348

`frames_missing` (locan.constants.PropertyKey attribute), 348

`frames_number` (locan.constants.PropertyKey attribute), 348

`from_batch()` (locan.analysis.convex\_hull\_expectation.ConvexHullExpectationBatch method), 272

`from_chunks()` (locan.data.locdata.LocData class method), 357

`from_collection()` (locan.data.locdata.LocData class method), 358

`from_colorcet()` (locan.visualize.colormap.Colormap class method), 516

`from_coordinates()` (locan.data.locdata.LocData class method), 358

`from_dataframe()` (locan.data.locdata.LocData class method), 358

`from_glob()` (locan.locan\_io.files.Files class method), 462

`from_intervals()` (locan.data.region.AxisOrientedCuboid class method), 379

`from_intervals()` (locan.data.region.AxisOrientedHyperCuboid class method), 383

`from_intervals()` (locan.data.region.Interval class method), 399

`from_intervals()` (locan.data.region.Rectangle class method), 410

`from_intervals()` (locan.data.region.Region static method), 414

`from_matplotlib()` (locan.visualize.colormap.Colormap class method), 517

`from_napari()` (locan.visualize.colormap.Colormap class method), 517

`from_path()` (locan.locan\_io.files.Files class method), 462

`from_registry()` (locan.visualize.colormap.Colormap class method), 517

`from_selection()` (locan.data.locdata.LocData class method), 359

`from_shapely()` (locan.data.region.EmptyRegion class method), 396

`from_shapely()` (locan.data.region.MultiPolygon class method), 403

`from_shapely()` (locan.data.region.Polygon class method), 406

`from_shapely()` (locan.data.region.Region2D class method), 419

`from_shapely()` (locan.data.region.RoiRegion class method), 426

`from_yaml()` (locan.rois.roi.Roi class method), 482

`from_yaml()` (locan.rois.roi.RoiLegacy\_0 class method), 484

## G

`gamma` (locan.data.region.Cuboid property), 388

`get_alpha_complex_lines()` (locan.data.hulls.alpha\_shape.AlphaComplex method), 371

`get_alpha_complex_triangles()` (locan.data.hulls.alpha\_shape.AlphaComplex method), 371

`get_colormap()` (in module locan.visualize.colormap), 518

`get_rois()` (in module locan.visualize.render\_napari.utilities), 536

`graph_from_lines()` (locan.data.hulls.alpha\_shape.AlphaComplex method), 372

`graph_from_triangles()` (locan.data.hulls.alpha\_shape.AlphaComplex method), 372

`group_identifiers()` (locan.locan\_io.files.Files method), 462

`grouped` (locan.analysis.convex\_hull\_expectation.ConvexHullExpectation attribute), 273

`grouped` (locan.analysis.grouped\_property\_expectation.GroupedPropertyExpectation attribute), 285

`grouped` (locan.analysis.position\_variance\_expectation.PositionVarianceExpectation attribute), 327

`grouped()` (locan.locan\_io.files.Files method), 462

`GroupedPropertyExpectation` (class in locan.analysis.grouped\_property\_expectation), 285

`can.analysis.grouped_property_expectation.index` (`locan.constants.PropertyKey` attribute), 282, 348

`GroupedPropertyExpectationResults` (class in `locan.analysis.grouped_property_expectation`), 285

`inertia_moments` (`locan.data.locdata.LocData` property), 359

`inertia_moments()` (in module `locan.data.properties.misc`), 365

`InertiaMoments` (class in `locan.data.properties.misc`), 363

`INSTALL_REQUIRES` (in module `locan.dependencies`), 455

`intensity` (`locan.constants.PropertyKey` attribute), 348

`intensity_keys()` (`locan.constants.PropertyKey` class method), 348

`intensity_properties()` (`locan.constants.PropertyKey` class method), 348

`intersection()` (`locan.data.region.EmptyRegion` method), 414

`intersection()` (`locan.data.region.Region` method), 414

`intersection()` (`locan.data.region.Region1D` method), 419

`intersection()` (`locan.data.region.Region2D` method), 419

`intersection()` (`locan.data.region.Region3D` method), 422

`intersection()` (`locan.data.region.RegionND` method), 423

`Interval` (class in `locan.data.region`), 397

`IntervalDates` (`locan.data.region.AxisOrientedCuboid` property), 379

`intervals` (`locan.data.region.AxisOrientedHypercuboid` property), 383

`intervals` (`locan.data.region.Interval` property), 400

`intervals` (`locan.data.region.Rectangle` property), 410

`inverse()` (`locan.visualize.transform.HistogramEqualization` method), 539

`inverse()` (`locan.visualize.transform.Transform` method), 541

`is_array_like()` (in module `locan.data.aggregate`), 435

`is_equally_sized` (`locan.data.aggregate.Bins` property), 433

`iterate_2d_array()` (in module `locan.utils.miscellaneous`), 509

`H`

`HAS_DEPENDENCY` (in module `locan.dependencies`), 456

`height` (`locan.data.region.AxisOrientedCuboid` property), 379

`height` (`locan.data.region.Cuboid` property), 388

`height` (`locan.data.region.Ellipse` property), 392

`height` (`locan.data.region.Rectangle` property), 410

`hist()` (`locan.analysis.blinking.BlinkStatistics` method), 265

`hist()` (`locan.analysis.cbc.CoordinateBasedColocalization` method), 267

`hist()` (`locan.analysis.convex_hull_expectation.ConvexHullExpectation` method), 269

`hist()` (`locan.analysis.convex_hull_expectation.ConvexHullExpectationBatch` method), 272

`hist()` (`locan.analysis.grouped_property_expectation.GroupedPropertyExpectation` method), 283

`hist()` (`locan.analysis.localization_precision.LocalizationPrecision` method), 287

`hist()` (`locan.analysis.localization_property.LocalizationProperty` method), 304

`hist()` (`locan.analysis.localizations_per_frame.LocalizationsPerFrame` method), 313

`hist()` (`locan.analysis.nearest_neighbor.NearestNeighborDistances` method), 320

`hist()` (`locan.analysis.position_variance_expectation.PositionVarianceExpectation` method), 326

`hist()` (`locan.analysis.subpixel_bias.SubpixelBias` method), 335

`histogram()` (in module `locan.data.aggregate`), 434

`HistogramEqualization` (class in `locan.visualize.transform`), 538

`holes` (`locan.data.region.MultiPolygon` property), 403

`holes` (`locan.data.region.Polygon` property), 406

`HullType` (class in `locan.constants`), 344

`I`

`image` (`locan.analysis.localization_property_2d.FitImageResults` attribute), 306

`IMPORT_NAMES` (in module `locan.dependencies`), 456



## L

- `label` (*locan.analysis.localization\_property\_2d.FitImageResults* attribute), 306
- `labels` (*locan.data.aggregate.Bins* property), 434
- `length` (*locan.data.region.AxisOrientedCuboid* property), 379
- `length` (*locan.data.region.Cuboid* property), 388
- `lengths` (*locan.data.region.AxisOrientedHypercuboid* property), 383
- `link_locdata()` (in module *locan.data.tracking*), 449
- `load_asdf_file()` (in module *locan.locan\_io.locdata.asdf\_io*), 474
- `load_decode_file()` (in module *locan.locan\_io.locdata.decode\_io*), 478
- `load_decode_header()` (in module *locan.locan\_io.locdata.decode\_io*), 478
- `load_Elyra_file()` (in module *locan.locan\_io.locdata.elyra\_io*), 472
- `load_Elyra_header()` (in module *locan.locan\_io.locdata.elyra\_io*), 472
- `load_locdata()` (in module *locan.locan\_io.locdata.io\_locdata*), 465
- `load_metadata_from_toml()` (in module *locan.data.metadata\_utils*), 451
- `load_Nanoimager_file()` (in module *locan.locan\_io.locdata.nanoimager\_io*), 473
- `load_Nanoimager_header()` (in module *locan.locan\_io.locdata.nanoimager\_io*), 473
- `load_npc()` (in module *locan.datasets*), 454
- `load_rapidSTORM_file()` (in module *locan.locan\_io.locdata.rapidstorm\_io*), 469
- `load_rapidSTORM_header()` (in module *locan.locan\_io.locdata.rapidstorm\_io*), 469
- `load_rapidSTORM_track_file()` (in module *locan.locan\_io.locdata.rapidstorm\_io*), 469
- `load_rapidSTORM_track_header()` (in module *locan.locan\_io.locdata.rapidstorm\_io*), 470
- `load_SMAP_file()` (in module *locan.locan\_io.locdata.smap\_io*), 479
- `load_SMAP_header()` (in module *locan.locan\_io.locdata.smap\_io*), 479
- `load_SMLM_file()` (in module *locan.locan\_io.locdata.smlm\_io*), 475
- `load_SMLM_header()` (in module *locan.locan\_io.locdata.smlm\_io*), 476
- `load_SMLM_manifest()` (in module *locan.locan\_io.locdata.smlm\_io*), 476
- `load_thunderstorm_file()` (in module *locan.locan\_io.locdata.thunderstorm\_io*), 471
- `load_thunderstorm_header()` (in module *locan.locan\_io.locdata.thunderstorm\_io*), 471
- `load_tubulin()` (in module *locan.datasets*), 454
- `load_txt_file()` (in module *locan.locan\_io.locdata.io\_locdata*), 466
- `local_background` (*locan.constants.PropertyKey* attribute), 348
- `local_background_sigma` (*locan.constants.PropertyKey* attribute), 348
- `localization_precision_model_1()` (in module *locan.analysis.uncertainty*), 337
- `localization_precision_model_2()` (in module *locan.analysis.uncertainty*), 338
- `localization_precision_model_3()` (in module *locan.analysis.uncertainty*), 338
- `LocalizationPrecision` (class in *locan.analysis.localization\_precision*), 286
- `LocalizationProperty` (class in *locan.analysis.localization\_property*), 303
- `LocalizationProperty2d` (class in *locan.analysis.localization\_property\_2d*), 307
- `LocalizationPropertyCorrelations` (class in *locan.analysis.localization\_property\_correlations*), 310
- `localizations_in_cluster_regions()` (in module *locan.data.filter*), 438
- `LocalizationsPerFrame` (class in *locan.analysis.localizations\_per\_frame*), 312
- `LocalizationUncertainty` (class in *locan.analysis.uncertainty*), 336
- `locan` module, 259
- `locan.analysis` module, 259
- `locan.analysis.accumulation_analysis` module, 260
- `locan.analysis.blinking`

- module, [263](#)
- locan.analysis.cbc
  - module, [265](#)
- locan.analysis.convex\_hull\_expectation
  - module, [268](#)
- locan.analysis.drift
  - module, [275](#)
- locan.analysis.grouped\_property\_expectation
  - module, [282](#)
- locan.analysis.localization\_precision
  - module, [285](#)
- locan.analysis.localization\_property
  - module, [303](#)
- locan.analysis.localization\_property\_2d
  - module, [305](#)
- locan.analysis.localization\_property\_correlation
  - module, [310](#)
- locan.analysis.localizations\_per\_frame
  - module, [311](#)
- locan.analysis.nearest\_neighbor
  - module, [314](#)
- locan.analysis.pipeline
  - module, [322](#)
- locan.analysis.position\_variance\_expectation
  - module, [324](#)
- locan.analysis.ripley
  - module, [328](#)
- locan.analysis.subpixel\_bias
  - module, [333](#)
- locan.analysis.uncertainty
  - module, [335](#)
- locan.configuration
  - module, [352](#)
- locan.constants
  - module, [339](#)
- locan.data
  - module, [353](#)
- locan.data.aggregate
  - module, [431](#)
- locan.data.cluster
  - module, [445](#)
- locan.data.cluster.clustering
  - module, [445](#)
- locan.data.cluster.utils
  - module, [448](#)
- locan.data.filter
  - module, [435](#)
- locan.data.hulls
  - module, [366](#)
- locan.data.hulls.alpha\_shape
  - module, [369](#)
- locan.data.hulls.hull
  - module, [366](#)
- locan.data.locdata
  - module, [353](#)
- locan.data.metadata\_utils
  - module, [450](#)
- locan.data.properties
  - module, [362](#)
- locan.data.properties.misc
  - module, [362](#)
- locan.data.region
  - module, [375](#)
- locan.data.region\_utils
  - module, [426](#)
- locan.data.register
  - module, [428](#)
- locan.data.tracking
  - module, [449](#)
- locan.data.transform
  - module, [440](#)
- locan.data.transform.bunwarpj
  - module, [441](#)
- locan.data.transform.intensity\_transformation
  - module, [444](#)
- locan.data.transform.spatial\_transformation
  - module, [442](#)
- locan.data.validation
  - module, [452](#)
- locan.datasets
  - module, [453](#)
- locan.dependencies
  - module, [455](#)
- locan.gui
  - module, [458](#)
- locan.gui.io
  - module, [458](#)
- locan.locan\_io
  - module, [459](#)
- locan.locan\_io.files
  - module, [459](#)
- locan.locan\_io.locdata
  - module, [464](#)
- locan.locan\_io.locdata.asdf\_io
  - module, [474](#)
- locan.locan\_io.locdata.decode\_io
  - module, [478](#)
- locan.locan\_io.locdata.elyra\_io
  - module, [472](#)
- locan.locan\_io.locdata.io\_locdata
  - module, [465](#)
- locan.locan\_io.locdata.nanoimager\_io

- module, 473
- locan.locan\_io.locdata.rapidstorm\_io
  - module, 468
- locan.locan\_io.locdata.smap\_io
  - module, 479
- locan.locan\_io.locdata.smlm\_io
  - module, 475
- locan.locan\_io.locdata.thunderstorm\_io
  - module, 470
- locan.locan\_io.locdata.utilities
  - module, 467
- locan.locan\_io.utilities
  - module, 480
- locan.rois
  - module, 481
- locan.rois.roi
  - module, 481
- locan.scripts
  - module, 485
- locan.scripts.script\_check
  - module, 486
- locan.scripts.script\_draw\_roi
  - module, 489
- locan.scripts.script\_napari
  - module, 491
- locan.scripts.script\_rois
  - module, 488
- locan.scripts.script\_show\_versions
  - module, 492
- locan.scripts.script\_test
  - module, 492
- locan.simulation
  - module, 493
- locan.simulation.simulate\_drift
  - module, 507
- locan.simulation.simulate\_locdata
  - module, 493
- locan.tests
  - module, 508
- locan.utils
  - module, 509
- locan.utils.miscellaneous
  - module, 509
- locan.utils.statistics
  - module, 510
- locan.utils.system\_information
  - module, 512
- locan.visualize
  - module, 514
- locan.visualize.colormap
  - module, 514

- locan.visualize.render
  - module, 519
- locan.visualize.render\_mpl
  - module, 519
- locan.visualize.render\_mpl.render2d
  - module, 520
- locan.visualize.render\_mpl.render3d
  - module, 525
- locan.visualize.render\_napari
  - module, 526
- locan.visualize.render\_napari.render2d
  - module, 526
- locan.visualize.render\_napari.render3d
  - module, 531
- locan.visualize.render\_napari.utilities
  - module, 535
- locan.visualize.transform
  - module, 537
- LocData (*class in locan.data.locdata*), 354
- locdata() (*locan.rois.roi.Roi method*), 482
- locdata() (*locan.rois.roi.RoiLegacy\_0 method*), 484
- lower\_bound (*locan.data.filter.Selector property*), 436
- lower\_bound (*locan.data.region.Interval property*), 400

## M

- main() (*in module locan.scripts.script\_check*), 487
- main() (*in module locan.scripts.script\_draw\_roi*), 490
- main() (*in module locan.scripts.script\_napari*), 491
- main() (*in module locan.scripts.script\_rois*), 489
- main() (*in module locan.scripts.script\_show\_versions*), 492
- main() (*in module locan.scripts.script\_test*), 493
- make\_cluster() (*in module locan.simulation.simulate\_locdata*), 498
- make\_dstorm() (*in module locan.simulation.simulate\_locdata*), 499
- make\_Matern() (*in module locan.simulation.simulate\_locdata*), 495
- make\_NeymanScott() (*in module locan.simulation.simulate\_locdata*), 495

[make\\_Poisson\(\)](#) (in module [locan.simulation.simulate\\_locdata](#)), 496  
[make\\_Thomas\(\)](#) (in module [locan.simulation.simulate\\_locdata](#)), 497  
[make\\_uniform\(\)](#) (in module [locan.simulation.simulate\\_locdata](#)), 500  
[manifest\\_file\\_info\\_from\\_locdata\(\)](#) (in module [locan.locan\\_io.locdata.smlm\\_io](#)), 476  
[manifest\\_format\\_from\\_locdata\(\)](#) (in module [locan.locan\\_io.locdata.smlm\\_io](#)), 477  
[manifest\\_from\\_locdata\(\)](#) (in module [locan.locan\\_io.locdata.smlm\\_io](#)), 477  
[match\\_file\\_upstream\(\)](#) ([locan.locan\\_io.files.Files](#) method), 463  
[match\\_files\(\)](#) ([locan.locan\\_io.files.Files](#) method), 463  
[matplotlib](#) ([locan.visualize.colormap.Colormap](#) property), 517  
[matrix](#) ([locan.data.register.Transformation](#) attribute), 428  
[max\\_distance](#) ([locan.data.region.AxisOrientedCuboid](#) property), 380  
[max\\_distance](#) ([locan.data.region.AxisOrientedHypercube](#) property), 383  
[max\\_distance](#) ([locan.data.region.Cuboid](#) property), 388  
[max\\_distance](#) ([locan.data.region.Ellipse](#) property), 392  
[max\\_distance](#) ([locan.data.region.EmptyRegion](#) property), 396  
[max\\_distance](#) ([locan.data.region.Interval](#) property), 400  
[max\\_distance](#) ([locan.data.region.MultiPolygon](#) property), 403  
[max\\_distance](#) ([locan.data.region.Polygon](#) property), 406  
[max\\_distance](#) ([locan.data.region.Rectangle](#) property), 411  
[max\\_distance](#) ([locan.data.region.Region](#) property), 414  
[max\\_distance\(\)](#) (in module [locan.data.properties.misc](#)), 365  
[merge\\_metadata\(\)](#) (in module [locan.data.metadata\\_utils](#)), 451  
[message\\_scheme\(\)](#) (in module [locan.data.metadata\\_utils](#)), 451  
[metadata\\_from\\_toml\\_string\(\)](#) (in module [locan.data.metadata\\_utils](#)), 452  
[metadata\\_to\\_formatted\\_string\(\)](#) (in module [locan.data.metadata\\_utils](#)), 452  
[model\\_result](#) ([locan.analysis.localization\\_property\\_2d.FitImageResults](#) attribute), 306  
[module](#)  
[locan](#), 259  
[locan.analysis](#), 259  
[locan.analysis.accumulation\\_analysis](#), 260  
[locan.analysis.blinking](#), 263  
[locan.analysis.cbc](#), 265  
[locan.analysis.convex\\_hull\\_expectation](#), 268  
[locan.analysis.drift](#), 275  
[locan.analysis.grouped\\_property\\_expectation](#), 282  
[locan.analysis.localization\\_precision](#), 285  
[locan.analysis.localization\\_property](#), 303  
[locan.analysis.localization\\_property\\_2d](#), 305  
[locan.analysis.localization\\_property\\_correlation](#), 310  
[locan.analysis.localizations\\_per\\_frame](#), 311  
[locan.analysis.nearest\\_neighbor](#), 314  
[locan.analysis.pipeline](#), 322  
[locan.analysis.position\\_variance\\_expectation](#), 324  
[locan.analysis.ripley](#), 328  
[locan.analysis.subpixel\\_bias](#), 333  
[locan.analysis.uncertainty](#), 335  
[locan.configuration](#), 352  
[locan.constants](#), 339  
[locan.data](#), 353  
[locan.data.aggregate](#), 431  
[locan.data.cluster](#), 445  
[locan.data.cluster.clustering](#), 445  
[locan.data.cluster.utils](#), 448  
[locan.data.filter](#), 435  
[locan.data.hulls](#), 366  
[locan.data.hulls.alpha\\_shape](#), 369  
[locan.data.hulls.hull](#), 366  
[locan.data.locdata](#), 353  
[locan.data.metadata\\_utils](#), 450  
[locan.data.properties](#), 362



- locan.data.properties.misc, 362
  - locan.data.region, 375
  - locan.data.region\_utils, 426
  - locan.data.register, 428
  - locan.data.tracking, 449
  - locan.data.transform, 440
  - locan.data.transform.bunwarpj, 441
  - locan.data.transform.intensity\_transform, 444
  - locan.data.transform.spatial\_transformation, 442
  - locan.data.validation, 452
  - locan.datasets, 453
  - locan.dependencies, 455
  - locan.gui, 458
  - locan.gui.io, 458
  - locan.locan\_io, 459
  - locan.locan\_io.files, 459
  - locan.locan\_io.locdata, 464
  - locan.locan\_io.locdata.asdf\_io, 474
  - locan.locan\_io.locdata.decode\_io, 478
  - locan.locan\_io.locdata.elyra\_io, 472
  - locan.locan\_io.locdata.io\_locdata, 465
  - locan.locan\_io.locdata.nanoimager\_io, 473
  - locan.locan\_io.locdata.rapidstorm\_io, 468
  - locan.locan\_io.locdata.smap\_io, 479
  - locan.locan\_io.locdata.smlm\_io, 475
  - locan.locan\_io.locdata.thunderstorm\_io, 470
  - locan.locan\_io.locdata.utilities, 467
  - locan.locan\_io.utilities, 480
  - locan.rois, 481
  - locan.rois.roi, 481
  - locan.scripts, 485
  - locan.scripts.script\_check, 486
  - locan.scripts.script\_draw\_roi, 489
  - locan.scripts.script\_napari, 491
  - locan.scripts.script\_rois, 488
  - locan.scripts.script\_show\_versions, 492
  - locan.scripts.script\_test, 492
  - locan.simulation, 493
  - locan.simulation.simulate\_drift, 507
  - locan.simulation.simulate\_locdata, 493
  - locan.tests, 508
  - locan.utils, 509
  - locan.utils.miscellaneous, 509
  - locan.utils.statistics, 510
  - locan.utils.system\_information, 512
  - locan.visualize, 514
  - locan.visualize.colormap, 514
  - locan.visualize.render, 519
  - locan.visualize.render\_mpl, 519
  - locan.visualize.render\_mpl.render2d, 525
  - locan.visualize.render\_mpl.render3d, 525
  - locan.visualize.render\_napari, 526
  - locan.visualize.render\_napari.render2d, 526
  - locan.visualize.render\_napari.render3d, 531
  - locan.visualize.render\_napari.utilities, 535
  - locan.visualize.transform, 537
  - MPL (*locan.constants.RenderEngine* attribute), 351
  - MPL\_SCATTER\_DENSITY (*locan.constants.RenderEngine* attribute), 351
  - MultiPolygon (*class in locan.data.region*), 401
- ## N
- n\_bins (*locan.data.aggregate.Bins* property), 434
  - N\_JOBS (*in module locan.configuration*), 352
  - n\_points (*locan.analysis.convex\_hull\_expectation.ConvexHullExp* attribute), 274
  - name (*locan.constants.PropertyDescription* attribute), 345
  - name (*locan.visualize.colormap.Colormap* property), 517
  - NANOIMAGER (*locan.constants.FileType* attribute), 343
  - NANOIMAGER\_KEYS (*in module locan.constants*), 340
  - NAPARI (*locan.constants.RenderEngine* attribute), 351
  - napari (*locan.visualize.colormap.Colormap* property), 517
  - NearestNeighborDistances (*class in locan.analysis.nearest\_neighbor*), 319
  - needs\_package() (*in module locan.dependencies*), 457
  - NNDistances\_csr\_2d (*class in locan.analysis.nearest\_neighbor*), 315
  - NNDistances\_csr\_3d (*class in locan.analysis.nearest\_neighbor*), 317

NONE (*locan.dependencies.QtBindings* attribute), 457

NONE (*locan.visualize.transform.Trafo* attribute), 541

## O

offset (*locan.data.register.Transformation* attribute), 428

open\_path\_or\_file\_like() (in module *locan.locan\_io.locdata.utilities*), 468

optimal\_alpha() (*locan.data.hulls.alpha\_shape.AlphaComplex* method), 372

orientation (*locan.data.properties.misc.InertiaMoments* attribute), 363

ORIENTED\_BOUNDING\_BOX (*locan.constants.HullType* attribute), 344

oriented\_bounding\_box (*locan.data.locdata.LocData* property), 359

OrientedBoundingBox (class in *locan.data.hulls.hull*), 368

original\_index (*locan.constants.PropertyKey* attribute), 349

overlay() (in module *locan.data.transform.spatial\_transformation*), 442

## P

PairwiseDistanceId (class in *locan.analysis.localization\_precision*), 288

PairwiseDistanceIdIdenticalSigmaZeroMu (class in *locan.analysis.localization\_precision*), 291

PairwiseDistance2d (class in *locan.analysis.localization\_precision*), 293

PairwiseDistance2dIdenticalSigma (class in *locan.analysis.localization\_precision*), 295

PairwiseDistance2dIdenticalSigmaZeroMu (class in *locan.analysis.localization\_precision*), 297

PairwiseDistance3d (class in *locan.analysis.localization\_precision*), 299

PairwiseDistance3dIdenticalSigmaZeroMu (class in *locan.analysis.localization\_precision*), 301

pdf\_nnDistances\_csr\_2D() (in module *locan.analysis.nearest\_neighbor*), 321

pdf\_nnDistances\_csr\_3D() (in module *locan.analysis.nearest\_neighbor*), 321

Pipeline (class in *locan.analysis.pipeline*), 322

plane (*locan.constants.PropertyKey* attribute), 349

plot() (in module *locan.analysis.ripley*), 333

plot() (*locan.analysis.accumulation\_analysis.AccumulationCluster* method), 262

plot() (*locan.analysis.convex\_hull\_expectation.ConvexHullExpectation* method), 270

plot() (*locan.analysis.convex\_hull\_expectation.ConvexHullExpectation* method), 272

plot() (*locan.analysis.drift.Drift* method), 279

plot() (*locan.analysis.drift.DriftModel* method), 282

plot() (*locan.analysis.grouped\_property\_expectation.GroupedPropertyExpectation* method), 284

plot() (*locan.analysis.localization\_precision.LocalizationPrecision* method), 288

plot() (*locan.analysis.localization\_property.LocalizationProperty* method), 305

plot() (*locan.analysis.localization\_property\_2d.LocalizationProperty2D* method), 308

plot() (*locan.analysis.localization\_property\_correlations.LocalizationPropertyCorrelations* method), 311

plot() (*locan.analysis.localizations\_per\_frame.LocalizationsPerFrame* method), 314

plot() (*locan.analysis.position\_variance\_expectation.PositionVarianceExpectation* method), 327

plot() (*locan.analysis.ripley.RipleysHFunction* method), 330

plot() (*locan.analysis.ripley.RipleysKFunction* method), 331

plot() (*locan.analysis.ripley.RipleysLFunction* method), 333

plot() (*locan.data.region.Region2D* method), 419

plot() (*locan.data.region.Region3D* method), 422

plot\_deviation\_from\_mean() (*locan.analysis.localization\_property\_2d.LocalizationProperty2D* method), 309

plot\_deviation\_from\_median() (*locan.analysis.localization\_property\_2d.LocalizationProperty2D* method), 309

[plot\\_residuals\(\)](#) (locan.analysis.localization\_property\_2d.LocalizationProperty2d method), 309  
[points](#) (locan.data.region.AxisOrientedCuboid property), 380  
[points](#) (locan.data.region.AxisOrientedHypercuboid property), 383  
[points](#) (locan.data.region.Cuboid property), 388  
[points](#) (locan.data.region.Ellipse property), 392  
[points](#) (locan.data.region.EmptyRegion property), 396  
[points](#) (locan.data.region.Interval property), 400  
[points](#) (locan.data.region.MultiPolygon property), 403  
[points](#) (locan.data.region.Polygon property), 407  
[points](#) (locan.data.region.Rectangle property), 411  
[points](#) (locan.data.region.Region property), 414  
[Polygon](#) (class in locan.data.region), 404  
[polygons](#) (locan.data.region.MultiPolygon property), 403  
[position\\_x](#) (locan.constants.PropertyKey attribute), 349  
[position\\_y](#) (locan.constants.PropertyKey attribute), 349  
[position\\_z](#) (locan.constants.PropertyKey attribute), 349  
[PositionVarianceExpectation](#) (class in locan.analysis.position\_variance\_expectation), 325  
[PositionVarianceExpectationResults](#) (class in locan.analysis.position\_variance\_expectation), 327  
[print\\_meta\(\)](#) (locan.data.locdata.LocData method), 359  
[print\\_summary\(\)](#) (locan.data.locdata.LocData method), 359  
[print\\_summary\(\)](#) (locan.locan\_io.files.Files method), 463  
[projection\(\)](#) (locan.data.locdata.LocData method), 359  
[PROPERTY\\_KEYS](#) (in module locan.constants), 339  
[PropertyDescription](#) (class in locan.constants), 344  
[PropertyKey](#) (class in locan.constants), 345  
[psf\\_amplitude](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_half\\_width](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_half\\_width\\_x](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_half\\_width\\_y](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_half\\_width\\_z](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_sigma](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_sigma\\_x](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_sigma\\_y](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_sigma\\_z](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_width](#) (locan.constants.PropertyKey attribute), 349  
[psf\\_width\\_x](#) (locan.constants.PropertyKey attribute), 350  
[psf\\_width\\_y](#) (locan.constants.PropertyKey attribute), 350  
[psf\\_width\\_z](#) (locan.constants.PropertyKey attribute), 350  
[PYQT5](#) (locan.dependencies.QtBindings attribute), 457  
[PYQT6](#) (locan.dependencies.QtBindings attribute), 457  
[PYSIDE2](#) (locan.dependencies.QtBindings attribute), 457  
[PYSIDE6](#) (locan.dependencies.QtBindings attribute), 457  
**Q**  
[QtBindings](#) (class in locan.dependencies), 456  
**R**  
[random\\_subset\(\)](#) (in module locan.data.filter), 438  
[randomize\(\)](#) (in module locan.simulation.simulate\_locdata), 500  
[RAPIDSTORM](#) (locan.constants.FileType attribute), 343  
[RAPIDSTORM\\_KEYS](#) (in module locan.constants), 341  
[RAPIDSTORMTRACK](#) (locan.constants.FileType attribute), 343  
[rasterize\(\)](#) (in module locan.rois.roi), 485  
[ratio\\_fwhm\\_to\\_sigma\(\)](#) (in module locan.utils.statistics), 511  
[Rectangle](#) (class in locan.data.region), 407

`reduce()` (*locan.data.locdata.LocData* method), 360

`references` (*locan.data.validation.Collection* attribute), 453

`Region` (class in *locan.data.region*), 412

`region` (*locan.data.hulls.alpha\_shape.AlphaShape* property), 375

`region` (*locan.data.hulls.hull.BoundingBox* property), 367

`region` (*locan.data.hulls.hull.OrientedBoundingBox* property), 369

`region` (*locan.data.locdata.LocData* property), 360

`region` (*locan.data.region.RoiRegion* property), 426

`region` (*locan.rois.roi.Roi* property), 482

`region` (*locan.rois.roi.RoiLegacy\_0* property), 484

`Region1D` (class in *locan.data.region*), 415

`Region2D` (class in *locan.data.region*), 417

`Region3D` (class in *locan.data.region*), 420

`region_measure` (*locan.data.region.AxisOrientedCuboid* property), 380

`region_measure` (*locan.data.region.AxisOrientedHypercuboid* property), 383

`region_measure` (*locan.data.region.Cuboid* property), 388

`region_measure` (*locan.data.region.Ellipse* property), 392

`region_measure` (*locan.data.region.EmptyRegion* property), 396

`region_measure` (*locan.data.region.Interval* property), 400

`region_measure` (*locan.data.region.MultiPolygon* property), 403

`region_measure` (*locan.data.region.Polygon* property), 407

`region_measure` (*locan.data.region.Rectangle* property), 411

`region_measure` (*locan.data.region.Region* property), 414

`REGION_MEASURE_2D` (*locan.analysis.convex\_hull\_expectation.ConvexHullProperty* attribute), 274

`REGION_MEASURE_3D` (*locan.analysis.convex\_hull\_expectation.ConvexHullProperty* attribute), 274

`region_specs` (*locan.data.region.Ellipse* property), 393

`region_specs` (*locan.data.region.Interval* property), 400

`region_specs` (*locan.data.region.MultiPolygon* property), 403

`region_specs` (*locan.data.region.Polygon* property), 407

`region_specs` (*locan.data.region.Rectangle* property), 411

`RegionND` (class in *locan.data.region*), 423

`regions_union()` (in module *locan.data.region\_utils*), 427

`register_cc()` (in module *locan.data.register*), 429

`register_icp()` (in module *locan.data.register*), 430

`render_2d()` (in module *locan.visualize.render*), 519

`render_2d_mpl()` (in module *locan.visualize.render\_mpl.render2d*), 520

`render_2d_napari()` (in module *locan.visualize.render\_napari.render2d*), 527

`render_2d_napari_image()` (in module *locan.visualize.render\_napari.render2d*), 528

`render_2d_rgb_mpl()` (in module *locan.visualize.render\_mpl.render2d*), 522

`render_2d_rgb_napari()` (in module *locan.visualize.render\_napari.render2d*), 530

`render_2d_scatter_density()` (in module *locan.visualize.render\_mpl.render2d*), 523

`render_3d()` (in module *locan.visualize.render*), 519

`render_3d_napari()` (in module *locan.visualize.render\_napari.render3d*), 531

`render_3d_napari_image()` (in module *locan.visualize.render\_napari.render3d*), 533

`render_3d_rgb_napari()` (in module *locan.visualize.render\_napari.render3d*), 534

`RENDER_ENGINE` (in module *locan.configuration*), 352

`render_hyper_frame_napari()` (in module *locan.scripts.script\_check*), 487

[RenderEngine \(class in locan.constants\), 351](#)  
[report\(\) \(locan.analysis.localization\\_property\\_2d.LocalizationProperty2d method\), 309](#)  
[report\(\) \(locan.analysis.localization\\_property\\_correlation.LocalizationPropertyCorrelation method\), 311](#)  
[resample\(\) \(in module locan.simulation.simulate\\_locdata\), 501](#)  
[reset\(\) \(locan.data.locdata.LocData method\), 360](#)  
[Ripley\\_h\\_maximum \(locan.analysis.ripley.RipleysHFunction property\), 329](#)  
[RipleysHFunction \(class in locan.analysis.ripley\), 328](#)  
[RipleysKFunction \(class in locan.analysis.ripley\), 330](#)  
[RipleysLFunction \(class in locan.analysis.ripley\), 331](#)  
[Roi \(class in locan.rois.roi\), 481](#)  
[RoiLegacy\\_0 \(class in locan.rois.roi\), 483](#)  
[RoiRegion \(class in locan.data.region\), 424](#)  
[ROOT\\_DIR \(in module locan.constants\), 339](#)

## S

[save\\_asdf\(\) \(in module locan.locan\\_io.locdata.asdf\\_io\), 474](#)  
[save\\_computation\(\) \(locan.analysis.pipeline.Pipeline method\), 324](#)  
[save\\_rois\(\) \(in module locan.visualize.render\\_napari.utilities\), 536](#)  
[save\\_SMAP\\_csv\(\) \(in module locan.locan\\_io.locdata.smap\\_io\), 480](#)  
[save\\_SMLM\(\) \(in module locan.locan\\_io.locdata.smlm\\_io\), 477](#)  
[save\\_thunderstorm\\_csv\(\) \(in module locan.locan\\_io.locdata.thunderstorm\\_io\), 471](#)  
[sc\\_check\(\) \(in module locan.scripts.script\\_check\), 487](#)  
[sc\\_draw\\_roi\\_mpl\(\) \(in module locan.scripts.script\\_draw\\_roi\), 490](#)  
[sc\\_draw\\_roi\\_napari\(\) \(in module locan.scripts.script\\_rois\), 489](#)  
[sc\\_napari\(\) \(in module locan.scripts.script\\_napari\), 491](#)  
[scatter\\_2d\\_mpl\(\) \(in module locan.visualize.render\\_mpl.render2d\), 524](#)  
[scatter\\_3d\\_mpl\(\) \(in module locan.visualize.render\\_mpl.render3d\), 526](#)  
[select\\_by\\_condition\(\) \(locan.data.filter\), 439](#)  
[select\\_by\\_drawing\\_mpl\(\) \(in module locan.visualize.render\\_mpl.render2d\), 525](#)  
[select\\_by\\_drawing\\_napari\(\) \(in module locan.visualize.render\\_napari.utilities\), 537](#)  
[select\\_by\\_image\\_mask\(\) \(in module locan.data.filter\), 439](#)  
[select\\_by\\_region\(\) \(in module locan.data.filter\), 440](#)  
[Selector \(class in locan.data.filter\), 436](#)  
[serial\\_clustering\(\) \(in module locan.data.cluster.utils\), 448](#)  
[set\\_file\\_path\\_dialog\(\) \(in module locan.gui.io\), 459](#)  
[set\\_group\\_identifier\(\) \(locan.locan\\_io.files.Files method\), 463](#)  
[shapely\\_object \(locan.data.region.Ellipse property\), 393](#)  
[shapely\\_object \(locan.data.region.MultiPolygon property\), 404](#)  
[shapely\\_object \(locan.data.region.Polygon property\), 407](#)  
[shapely\\_object \(locan.data.region.Rectangle property\), 411](#)  
[shapely\\_object \(locan.data.region.Region2D property\), 420](#)  
[show\\_versions\(\) \(in module locan.utils.system\\_information\), 513](#)  
[signal\\_background\\_ratio \(locan.constants.PropertyKey attribute\), 350](#)  
[signal\\_noise\\_ratio \(locan.constants.PropertyKey attribute\), 350](#)  
[simulate\\_cluster\(\) \(in module locan.simulation.simulate\\_locdata\), 504](#)  
[simulate\\_dstorm\(\) \(in module locan.simulation.simulate\\_locdata\), 505](#)  
[simulate\\_frame\\_numbers\(\) \(in module locan.simulation.simulate\\_locdata\), 506](#)  
[simulate\\_Matern\(\) \(in module locan.simulation.simulate\\_locdata\),](#)



501  
 simulate\_NeymanScott() (in module locan.simulation.simulate\_locdata), 502  
 simulate\_Poisson() (in module locan.simulation.simulate\_locdata), 503  
 simulate\_Thomas() (in module locan.simulation.simulate\_locdata), 503  
 simulate\_tracks() (in module locan.simulation.simulate\_locdata), 506  
 simulate\_uniform() (in module locan.simulation.simulate\_locdata), 507  
 slice\_z (locan.constants.PropertyKey attribute), 350  
 SMAP (locan.constants.FileType attribute), 343  
 SMAP\_KEYS (in module locan.constants), 341  
 SMLM (locan.constants.FileType attribute), 343  
 SMLM\_KEYS (in module locan.constants), 341  
 STANDARDIZE (locan.visualize.transform.Trafo attribute), 541  
 standardize() (in module locan.data.transform.spatial\_transformation), 443  
 STANDARDIZE\_UINT8 (locan.visualize.transform.Trafo attribute), 541  
 statistics() (in module locan.data.properties), 365  
 std\_neg (locan.analysis.convex\_hull\_expectation.ConvexHullExpectationValues attribute), 274  
 std\_pos (locan.analysis.convex\_hull\_expectation.ConvexHullExpectationValues attribute), 274  
 SubpixelBias (class in locan.analysis.subpixel\_bias), 334  
 subregion\_measure (locan.data.region.AxisOrientedCuboid property), 380  
 subregion\_measure (locan.data.region.AxisOrientedHypercuboid property), 383  
 subregion\_measure (locan.data.region.Cuboid property), 388  
 subregion\_measure (locan.data.region.Ellipse property), 393  
 subregion\_measure (locan.data.region.EmptyRegion property), 396  
 subregion\_measure (locan.data.region.Interval property), 400  
 subregion\_measure (locan.data.region.MultiPolygon property), 404  
 subregion\_measure (locan.data.region.Polygon property), 407  
 subregion\_measure (locan.data.region.Rectangle property), 411  
 subregion\_measure (locan.data.region.Region property), 414  
 SUBREGION\_MEASURE\_2D (locan.analysis.convex\_hull\_expectation.ConvexHullProperty attribute), 274  
 SUBREGION\_MEASURE\_3D (locan.analysis.convex\_hull\_expectation.ConvexHullProperty attribute), 274  
 summary() (locan.constants.PropertyKey class method), 350  
 surrounding\_region() (in module locan.data.region\_utils), 427  
 symmetric\_difference() (locan.data.region.EmptyRegion method), 396  
 symmetric\_difference() (locan.data.region.Region method), 414  
 symmetric\_difference() (locan.data.region.Region1D method), 416  
 symmetric\_difference() (locan.data.region.Region2D method), 420  
 symmetric\_difference() (locan.data.region.Region3D method), 422  
 symmetric\_difference() (locan.data.region.RegionND method), 424  
 system\_info() (in module locan.utils.system\_information), 514

**T**  
 test() (in module locan.tests), 509  
 THUNDERSTORM (locan.constants.FileType attribute), 343  
 THUNDERSTORM\_KEYS (in module locan.constants), 341  
 time (locan.constants.PropertyKey attribute), 350  
 to\_shapely() (locan.data.region.RoiRegion method), 426  
 to\_yaml() (locan.rois.roi.Roi method), 483

- `to_yaml()` (*locan.rois.roi.RoiLegacy\_0* method), 485
- `TQDM_DISABLE` (in module *locan.configuration*), 353
- `TQDM_LEAVE` (in module *locan.configuration*), 353
- `track()` (in module *locan.data.tracking*), 450
- `Trafo` (class in *locan.visualize.transform*), 540
- `Transform` (class in *locan.visualize.transform*), 541
- `transform_affine()` (in module *locan.data.transform.spatial\_transformation*), 443
- `transform_counts_to_photons()` (in module *locan.data.transform.intensity\_transformation*), 444
- `Transformation` (class in *locan.data.register*), 428
- `TURBO` (*locan.visualize.colormap.Colormaps* attribute), 518
- `two_kernel_improvement` (*locan.constants.PropertyKey* attribute), 350
- `type` (*locan.constants.PropertyDescription* attribute), 345
- ## U
- `uncertainty` (*locan.constants.PropertyKey* attribute), 350
- `uncertainty_keys` (*locan.data.locdata.LocData* property), 360
- `uncertainty_keys()` (*locan.constants.PropertyKey* class method), 350
- `uncertainty_properties()` (*locan.constants.PropertyKey* class method), 351
- `uncertainty_x` (*locan.constants.PropertyKey* attribute), 351
- `uncertainty_y` (*locan.constants.PropertyKey* attribute), 351
- `uncertainty_z` (*locan.constants.PropertyKey* attribute), 351
- `union()` (*locan.data.region.EmptyRegion* method), 396
- `union()` (*locan.data.region.Region* method), 415
- `union()` (*locan.data.region.Region1D* method), 417
- `union()` (*locan.data.region.Region2D* method), 420
- `union()` (*locan.data.region.Region3D* method), 422
- `union()` (*locan.data.region.RegionND* method), 424
- `unit` (*locan.constants.PropertyDescription* attribute), 345
- `unit_SI` (*locan.constants.PropertyDescription* attribute), 345
- `UNKNOWN_FILE_TYPE` (*locan.constants.FileType* attribute), 343
- `update()` (*locan.data.locdata.LocData* method), 360
- `update_alpha_shape()` (*locan.data.locdata.LocData* method), 361
- `update_alpha_shape_in_references()` (*locan.data.locdata.LocData* method), 361
- `update_convex_hulls_in_references()` (*locan.data.locdata.LocData* method), 361
- `update_inertia_moments_in_references()` (*locan.data.locdata.LocData* method), 361
- `update_oriented_bounding_box_in_references()` (*locan.data.locdata.LocData* method), 362
- `update_properties_in_references()` (*locan.data.locdata.LocData* method), 362
- `upper_bound` (*locan.data.filter.Selector* property), 436
- `upper_bound` (*locan.data.region.Interval* property), 400
- ## V
- `values` (*locan.analysis.convex\_hull\_expectation.ConvexHullExpectation* attribute), 273
- `values` (*locan.analysis.grouped\_property\_expectation.GroupedPropertyExpectation* attribute), 285
- `values` (*locan.analysis.position\_variance\_expectation.PositionVarianceExpectation* attribute), 327
- `variance_explained` (*locan.data.properties.misc.InertiaMoments* attribute), 363
- `vertex_alpha_shape_indices` (*locan.data.hulls.alpha\_shape.AlphaShape* property), 375
- `vertex_indices` (*locan.data.hulls.alpha\_shape.AlphaShape* property), 375
- `vertices` (*locan.data.hulls.alpha\_shape.AlphaShape* property), 375

`vertices` (*locan.data.hulls.hull.BoundingBox* property), 367

`vertices` (*locan.data.hulls.hull.OrientedBoundingBox* property), 369

`vertices_alpha_shape` (*locan.data.hulls.alpha\_shape.AlphaShape* property), 375

`vertices_connected_components_indices` (*locan.data.hulls.alpha\_shape.AlphaShape* property), 375

## W

`weighted_mean` (*locan.utils.statistics.WeightedMeanVariance* attribute), 510

`weighted_mean_variance` (*locan.utils.statistics.WeightedMeanVariance* attribute), 510

`weighted_mean_variance()` (in module *locan.utils.statistics*), 512

`WeightedMeanVariance` (class in *locan.utils.statistics*), 510

`width` (*locan.data.region.AxisOrientedCuboid* property), 380

`width` (*locan.data.region.Cuboid* property), 389

`width` (*locan.data.region.Ellipse* property), 393

`width` (*locan.data.region.Rectangle* property), 411

## Z

`ZERO` (*locan.visualize.transform.Trafo* attribute), 541

`ZERO_UINT8` (*locan.visualize.transform.Trafo* attribute), 541