nikamap Documentation

Release 0.1

Alexandre Beelen

Jan 12, 2023

CONTENTS

nikamap is a python module to manipulate data produced by the IDL NIKA2 pipeline.

```
from nikamap import NikaMap

nm = NikaMap.read('map.fits', band='1mm')
nm.plot()
```

or alternatively

```
from nikamap import NikaFits

data = NikaFits.read('map.fits')
data['1mm'].plot()
```

CHAPTER

ONE

FEATURES

- reading, slicing, plotting
- match filtering, point source detection and photometry
- power spectra estimation
- bootstraping and jackknife

Please refer to the README file in the Git repository.

Contents:

1.1 NikaMap Examples

Click one of the images below to be taken its corresponding example.

1.1.1 Fake data

These are basic examples based on a generated fake dataset. They explain the most basic command available with NikaMap

1.1.2 G2 data

These are examples based on the G2 dataset, to explain the jackknife and bootstrap approach, but these can be applied to any dataset.

Fake data

These are basic examples based on a generated fake dataset. They explain the most basic command available with NikaMap

Basic usage

This example shows the basic operation on the *nikamap.NikaMap* object

```
import os
import numpy as np
import matplotlib.pyplot as plt
import astropy.units as u
from astropy.table import Table
from astropy.coordinates import SkyCoord, Angle

from nikamap import NikaMap
```

Generate fake map

```
from fake_map import create_dataset

create_dataset()
```

Read the data

By default the read routine will read the 1mm band, but any band can be read

Note: This fake dataset as been generated by the `fake_map.py` script

```
data_path = os.getcwd()
nm = NikaMap.read(os.path.join(data_path, "fake_map.fits"))
```

NikaMap is derived from the *astropy.NDData* class and thus you can access and and manipulate the data the same way

- *nm.data* : an np.array containing the brightness
- *nm.wcs* : a WCS object describing the astrometry of the image
- *nm.uncertainy.array* : a np.array containing the uncertainty array
- *nm.mask* : a boolean mask of the observations
- *nm.meta* : a copy of the header of the original map

```
print(nm)
```

```
[[nan nan nan ... nan nan nan]
 [nan nan nan ... nan nan nan]
 [nan nan nan ... nan nan nan]
 ...
 [nan nan nan ... nan nan nan]
 [nan nan nan ... nan nan nan]
 [nan nan nan ... nan nan nan]] Jy / beam
```

```
print(nm.wcs)
```

```
WCS Keywords

Number of WCS axes: 2
CTYPE : 'RA---TAN'  'DEC--TAN'
CRVAL : 0.0  0.0
CRPIX : 255.5  255.5
PC1_1 PC1_2  : 1.0  0.0
PC2_1 PC2_2  : 0.0  1.0
CDELT : -0.00055555555555556  0.00055555555555556
NAXIS : 512  512
```

NikaMap objects support slicing like numpy arrays, thus one can access part of the dataset

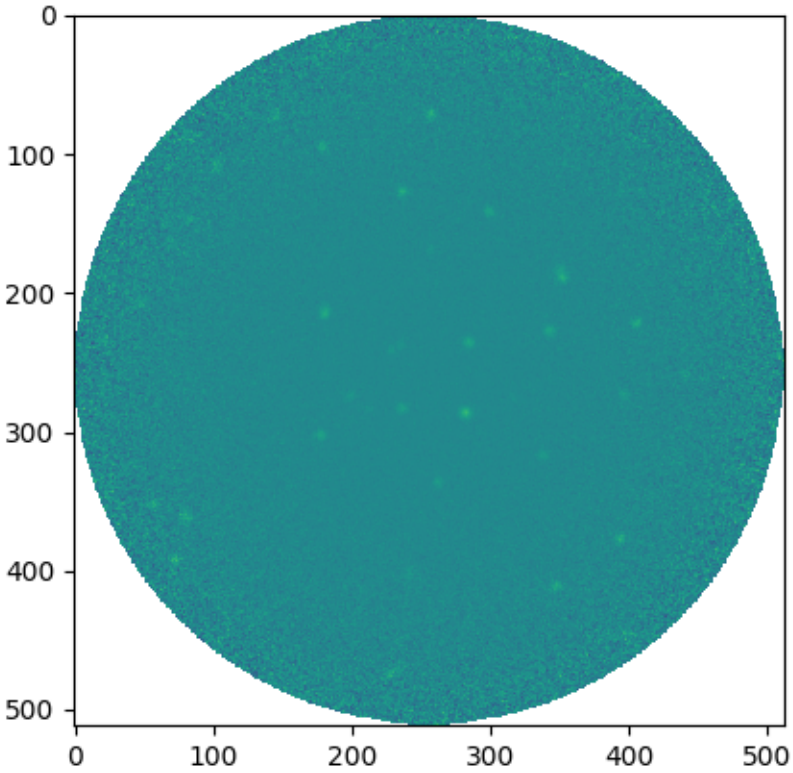
```
print(nm[96:128, 96:128])
```

```
[[-6.33690088e-04  1.69940188e-05  1.13807169e-02 ... -4.77849898e-04
  -4.42656104e-03  1.29046581e-03]
 [-2.72178236e-03 -3.45859875e-03  6.88742228e-03 ... -1.64266145e-03
  -2.13449370e-04  1.42029957e-03]
 [-1.61883365e-03 -1.94432669e-03  6.59878990e-03 ... -5.03622493e-03
  -3.17169283e-03 -1.80714630e-03]
 ...
 [-2.25541455e-03  7.91104592e-03 -7.27016993e-03 ... -4.20066714e-03
  -1.19154754e-03 -2.18287246e-04]
 [-5.37531694e-04 -1.29236258e-03 -2.83427495e-03 ...  5.39804507e-03
   1.34197023e-03  2.60073558e-03]
 [ 3.62899719e-03  2.33734785e-03 -3.51804099e-04 ... -2.20487005e-03
   2.53243571e-03  3.69477925e-03]] Jy / beam
```

Basic Plotting

thus they can be plotted directly using maplotlib routines

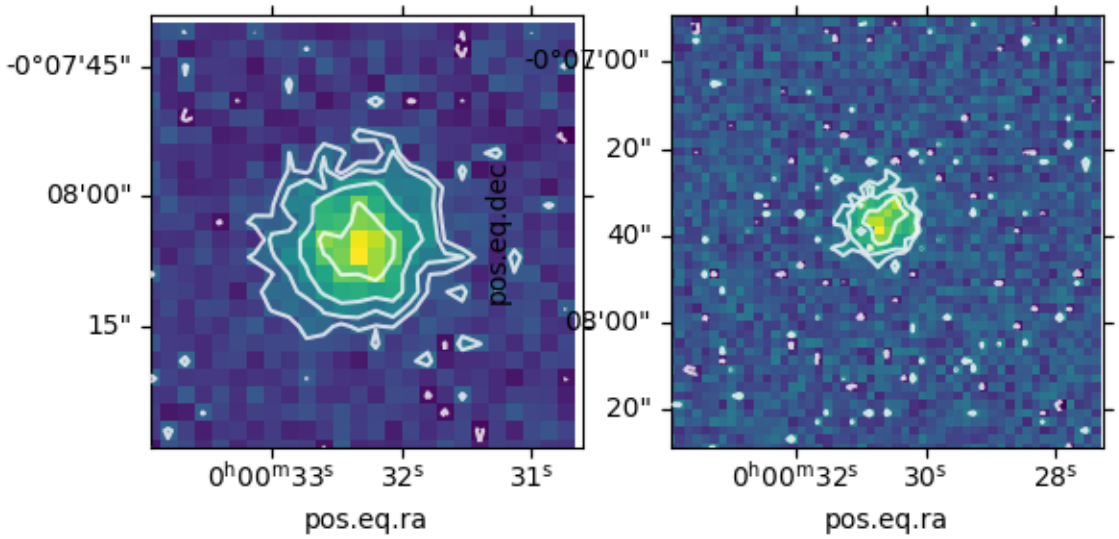
```
plt.imshow(nm.data)
```




```
<matplotlib.image.AxesImage object at 0x7fd17cf5ca50>
```

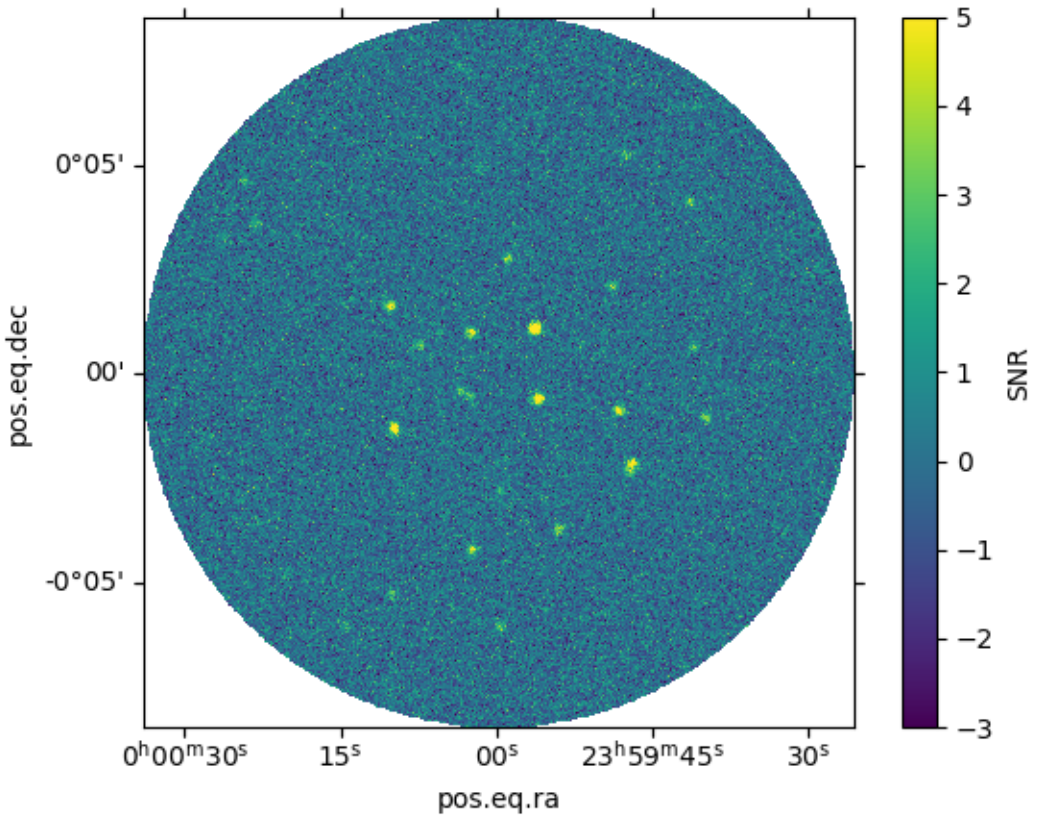
or using the convience routine of *nikamap.NikaMap*

```
fig, axes = plt.subplots(ncols=2, subplot_kw={"projection": nm.wcs})
levels = np.logspace(np.log10(2 * 1e-3), np.log10(10e-3), 4)
nm[275:300, 270:295].plot(ax=axes[0], levels=levels)
nm[210:260, 260:310].plot(ax=axes[1], levels=levels)
```




```
<matplotlib.image.AxesImage object at 0x7fd17aae4a10>
```

```
nm.plot_SNR(cbar=True)
```

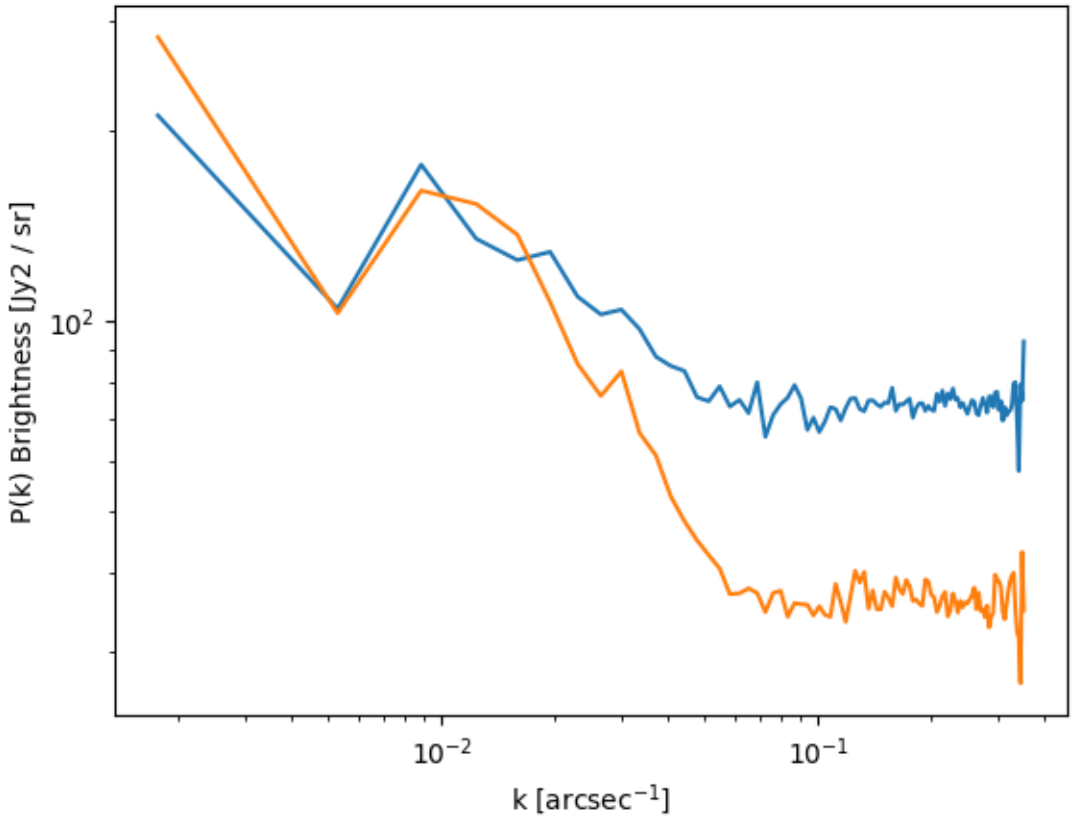



```
<matplotlib.image.AxesImage object at 0x7fd17a58f7d0>
```

or the power spectrum density of the data :

```
fig, ax = plt.subplots()
powspec, bins = nm.plot_PSD(ax=ax)

islice = nm.get_square_slice()
_ = nm[islice, islice].plot_PSD(ax=ax)
```

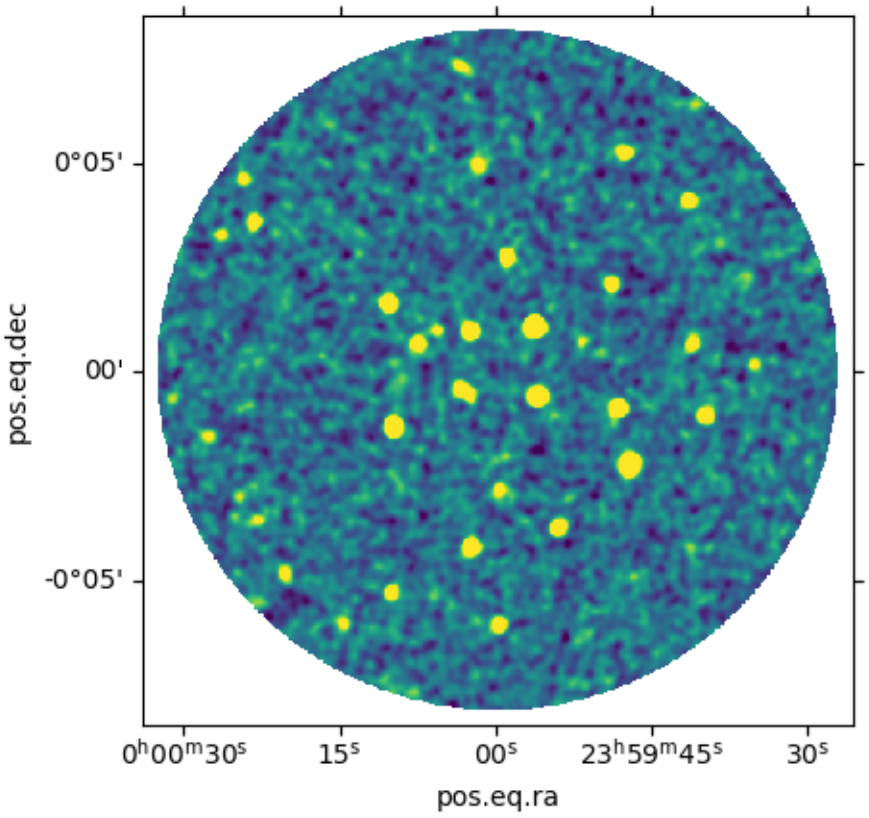


Beware that these PSD are based on an non-uniform noise, thus dominated by the largest noise part of the map

Match filtering

A match filter algorithm can be applied to the data to improve the detectability of sources. Here using the gaussian beam as the filter

```
mf_nm = nm.match_filter(nm.beam)
mf_nm.plot_SNR()
```




```
<matplotlib.image.AxesImage object at 0x7fd17a58f710>
```

Source detection & photometry

A peak finding algorithm can be applied to the SNR datasets

```
mf_nm.detect_sources(threshold=3)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/photutils/centroids/gaussian.py:199: AstropyUserWarning: Input data contains␣
↪non-finite values (e.g., NaN or infs) that were automatically masked.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
```

(continues on next page)

(continued from previous page)

```
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
```

(continues on next page)

(continued from previous page)

```
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
/home/docs/checkouts/readthedocs.org/user_builds/nikamap/envs/latest/lib/python3.7/site-
↪packages/astropy/modeling/fitting.py:1168: AstropyUserWarning: The fit may be␣
↪unsuccessful; check fit_info['message'] for more information.
  AstropyUserWarning)
```

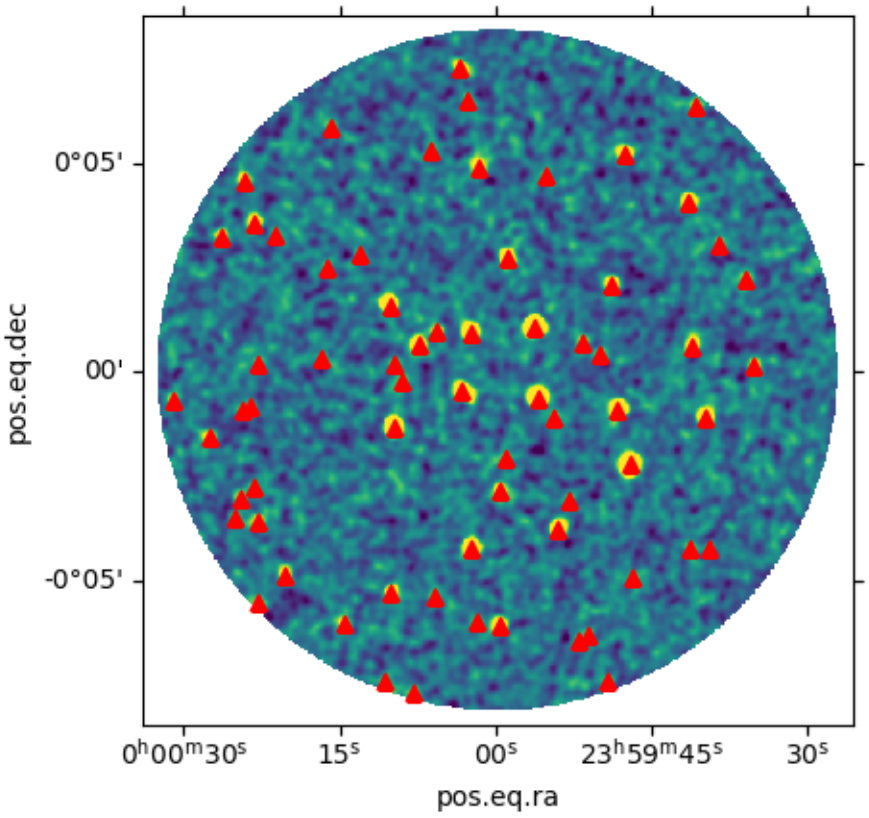
The resulting catalog is stored in the *sources* property of the *nikamap.NikaMap* object

```
print(mf_nm.sources)
```

```
 ID x_peak y_peak ...         _ra                  _dec
                  ...         deg                  deg
--- ------ ------ ... -------------------- ---------------------
  0    282    287 ...    359.9847114124959  0.017900476458951042
  1    284    236 ...    359.9833675785458 -0.010234896450712015
  2    181    215 ... 0.041035360941564326  -0.02207805913640279
  3    352    189 ...   359.94610412881894  -0.03665689518647971
  4    342    228 ...    359.951118338902 -0.014896908375759216
  5    236    283 ...  0.01013171408890115  0.016091889135546964
  6    178    303 ...  0.04263583229590677  0.026825164975406882
  7    237    128 ... 0.009989769624177525  -0.07051531787966656
  8    299    142 ...   359.97516728788645  -0.06253261597441508
  9    405    222 ...    359.9161735974608  -0.01793110948605165
...    ...    ... ...                  ...                  ...
 57    208    414 ...  0.02587453118074121   0.08891347353936153
 58    132    330 ...  0.06783784639683338   0.04179008331327504
 59    321     66 ...   359.96305002180287  -0.10488835800678183
 60     71    227 ...  0.10188849954444089 -0.015000829188970714
 61    290    396 ...   359.98029743623783    0.0786471628187289
 62     80    172 ...  0.09679612217264584  -0.04605495744483694
 63    336     32 ...    359.9550266025933  -0.12365120500549592
 64    408    127 ...   359.91480923866493  -0.07074502706394185
 65    314     61 ...   359.96703365977106  -0.10753107805188127
 66    174     32 ...  0.04467994501436867  -0.12338114817921203
Length = 67 rows
```

and can be overploted on the SNR maplotlib

```
mf_nm.plot_SNR(cat=True)
```




```
<matplotlib.image.AxesImage object at 0x7fd179ab5f90>
```

There is two available photometries : * **peak_flux** : to retrieve point sources flux directly on the pixel value of the map, ideadlly on the matched filtered map * **psf_flux** : which perfom psf fitting on the pixels at the given position

```
mf_nm.phot_sources(peak=True, psf=False)
```

catalog which can be transfered to the un-filtered dataset, where psf fitting can be performed

```
nm.phot_sources(sources=mf_nm.sources, peak=False, psf=True)
```

the *sources* attribute now contains both photometries

```
print(nm.sources)
```

```
 ID x_peak y_peak ...      flux_psf           eflux_psf      group_id
                  ...        mJy                mJy
--- ------ ------ ... ------------------ ------------------- --------
  0    282    287 ... 12.984409611567687 0.23217106371628127        1
  1    284    236 ...  8.182805551154722 0.22934325218363477        2
  2    181    215 ...  9.661357086264521 0.29139997278160235        3
  3    352    189 ... 10.457973917045644  0.4298943775516725        4
  4    342    228 ...  7.577516424910544  0.2674305539143367        5
  5    236    283 ...  5.782277915403067 0.23685526468283805        6
```

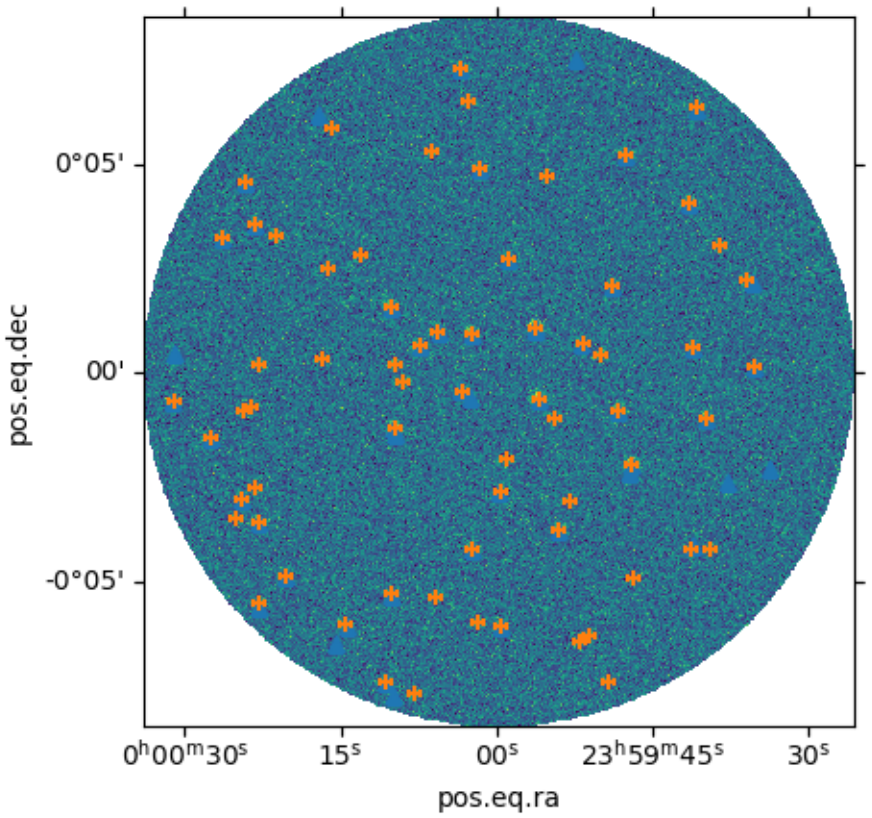
(continues on next page)

(continued from previous page)

```
  6    178    303 ...  7.020773945248183  0.2796755877878895        7
  7    237    128 ...  8.919819542708254  0.3713460064206581        8
  8    299    142 ... 6.7926022006199265  0.3411501103748405        9
  9    405    222 ...  8.641542180781649 0.46672616632831176       10
...    ...    ... ...                ...                 ...      ...
 57    208    414 ...  1.603340583355253  0.4696551670013086       50
 58    132    330 ...  1.250682931788955 0.40901592850128904       51
 59    321     66 ... 2.3053457016310177  0.7797812507892499       52
 60     71    227 ... 1.5662203466932614  0.6753829135107068       41
 61    290    396 ... 1.2720581723660047 0.43890018240144213       53
 62     80    172 ...  2.153016907492351  0.8815522274850082       29
 63    336     32 ...  4.169882788844715  1.3282650048823894       54
 64    408    127 ... 2.4106728978205854  0.7359750418734657       34
 65    314     61 ... 2.2591824113113095  0.7799074642328158       52
 66    174     32 ...  4.023546288906554  1.3723639824292304       55
Length = 67 rows
```

which can be compared to the original fake source catalog

```
fake_sources = Table.read("fake_map.fits", "FAKE_SOURCES")
fake_sources.meta["name"] = "fake sources"
nm.plot_SNR(cat=[(fake_sources, {"marker": "^"}), (nm.sources, {"marker": "+"})])
```



```
<matplotlib.image.AxesImage object at 0x7fd17a576910>
```

or in greater details :

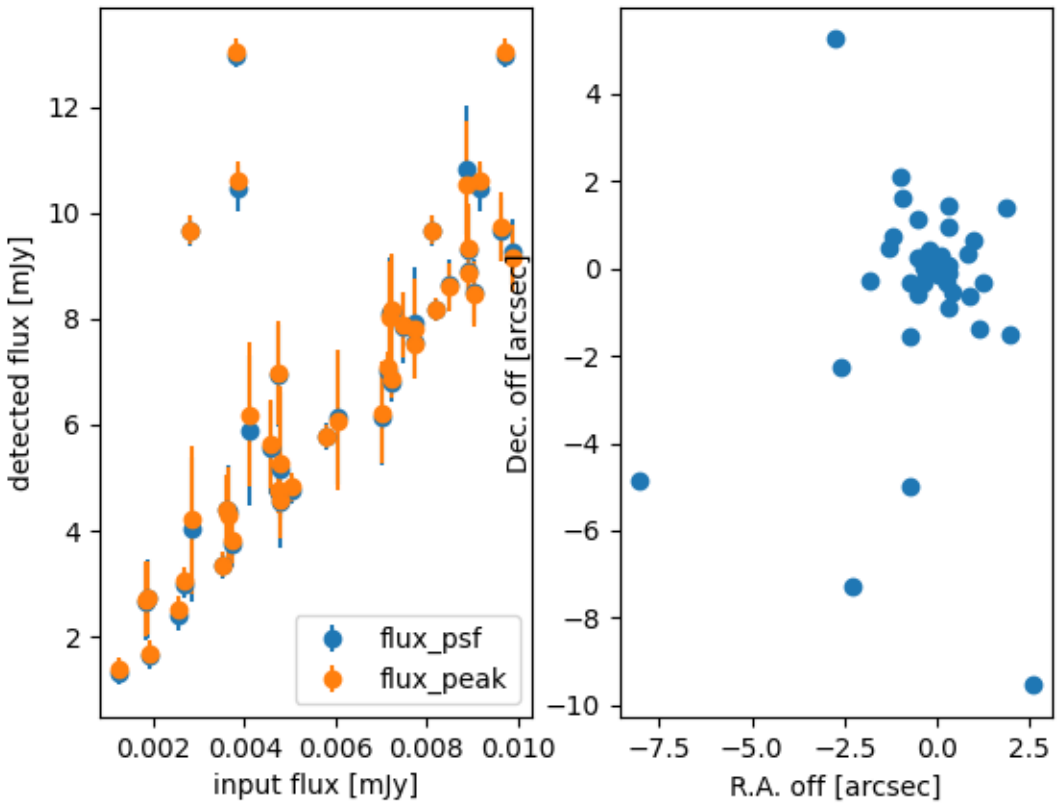
```
fake_coords = SkyCoord(fake_sources["ra"], fake_sources["dec"], unit="deg")
detected_coords = SkyCoord(nm.sources["ra"], nm.sources["dec"], unit="deg")

idx, sep2d, _ = fake_coords.match_to_catalog_sky(detected_coords)
good = sep2d < 10 * u.arcsec
idx = idx[good]
sep2d = sep2d[good]

ra_off = Angle(fake_sources[good]["ra"] - nm.sources[idx]["ra"], "deg")
dec_off = Angle(fake_sources[good]["dec"] - nm.sources[idx]["dec"], "deg")

fig, axes = plt.subplots(ncols=2)
for method in ["flux_psf", "flux_peak"]:
    axes[0].errorbar(
        fake_sources[good]["amplitude"],
        nm.sources[idx][method],
        yerr=nm.sources[idx]["e{}".format(method)],
        fmt="o",
        label=method,
    )
axes[0].legend(loc="best")
axes[0].set_xlabel("input flux [mJy]")
axes[0].set_ylabel("detected flux [mJy]")

axes[1].scatter(ra_off.arcsecond, dec_off.arcsecond)
axes[1].set_xlabel("R.A. off [arcsec]")
axes[1].set_ylabel("Dec. off [arcsec]")
```




```
Text(306.01767676767673, 0.5, 'Dec. off [arcsec]')
```

Total running time of the script: (0 minutes 7.095 seconds)

Create a fake fits file

Create a fully fake fits file for test purposes, note that this is basically a copy of the function *nikamap.fake_data()*

```
import numpy as np
import astropy.units as u
from astropy.io import fits
from astropy.wcs import WCS
from astropy.table import Table
from astropy.stats import gaussian_fwhm_to_sigma

from photutils.datasets import make_gaussian_sources_image

Jypb = u.Jy / u.beam
mJypb = u.mJy / u.beam

np.random.seed(1)
```

Create a fake dataset

Create a fake dataset with uniformly distributed point sources

Note: This is mostly inspired by *nikamap.fake_data()*

```
def create_dataset(
    shape=(512, 512),
    fwhm=12.5 * u.arcsec,
    pixsize=2 * u.arcsec,
    noise_level=1 * mJypb,
    n_sources=50,
    flux_min=1 * mJypb,
    flux_max=10 * mJypb,
    filename="fake_map.fits",
):

    hits, uncertainty, mask = create_ancillary(shape, fwhm=None, noise_level=1 * mJypb)
    wcs = create_wcs(shape, pixsize=2 * u.arcsec, center=None)

    beam_std_pix = (fwhm / pixsize).decompose().value * gaussian_fwhm_to_sigma
    sources = create_fake_source(shape, wcs, beam_std_pix, flux_min=flux_min, flux_
↪max=flux_max, n_sources=n_sources)

    sources_map = make_gaussian_sources_image(shape, sources) * sources["amplitude"].unit

    data = add_noise(sources_map, uncertainty)
    hdus = create_hdulist(data, hits, uncertainty, mask, wcs, sources, fwhm, noise_level)

    hdus.writeto(filename, overwrite=True)
```

Define the hits and uncertainty map

```
def create_ancillary(shape, fwhm=None, noise_level=1 * mJypb):
    if fwhm is None:
        fwhm = np.asarray(shape) / 2.5

    y_idx, x_idx = np.indices(shape, dtype=float)
    hits = (
        np.exp(
            -(
                (x_idx - shape[1] / 2) ** 2 / (2 * (gaussian_fwhm_to_sigma * fwhm[1]) **␣
↪2)
                + (y_idx - shape[0] / 2) ** 2 / (2 * (gaussian_fwhm_to_sigma * fwhm[0])␣
↪** 2)
            )
        )
        * 100
    )

    uncertainty = noise_level.to(u.Jy / u.beam).value / np.sqrt(hits / 100)
```

(continues on next page)

(continued from previous page)

```
    # with a circle for the mask
    xx, yy = np.indices(shape)
    mask = np.sqrt((xx - (shape[1] - 1) / 2) ** 2 + (yy - (shape[0] - 1) / 2) ** 2) >␣
↪shape[0] / 2

    return hits.astype(int), uncertainty, mask
```

and a fake `astropy.wcs.WCS`

```
def create_wcs(shape, pixsize=2 * u.arcsec, center=None):
    if center is None:
        center = np.asarray([0, 0]) * u.deg

    wcs = WCS(naxis=2)
    wcs.wcs.crval = center.to(u.deg).value
    wcs.wcs.crpix = np.asarray(shape) / 2 - 0.5  # Center of pixel
    wcs.wcs.cdelt = np.asarray([-1, 1]) * pixsize.to(u.deg)
    wcs.wcs.ctype = ("RA---TAN", "DEC--TAN")

    return wcs
```

Construct a fake source catalog

This define an uniformly distribued catalog of sources in a disk as an `astropy.table.Table`

```
def create_fake_source(shape, wcs, beam_std_pix, flux_min=1 * mJypb, flux_max=10 * mJypb,
↪ n_sources=1):

    peak_fluxes = np.random.uniform(flux_min.to(Jypb).value, flux_max.to(Jypb).value, n_
↪sources) * Jypb

    sources = Table(masked=True)
    sources["amplitude"] = peak_fluxes

    # Uniformly distributed sources on a disk
    theta = np.random.uniform(0, 2 * np.pi, n_sources)
    r = np.sqrt(np.random.uniform(0, 1, n_sources)) * (shape[0] / 2 - 10)

    sources["x_mean"] = r * np.cos(theta) + shape[1] / 2
    sources["y_mean"] = r * np.sin(theta) + shape[0] / 2

    sources["x_stddev"] = np.ones(n_sources) * beam_std_pix
    sources["y_stddev"] = np.ones(n_sources) * beam_std_pix
    sources["theta"] = np.zeros(n_sources)

    ra, dec = wcs.all_pix2world(sources["x_mean"], sources["y_mean"], 0)
    sources["ra"] = ra * u.deg
    sources["dec"] = dec * u.deg
    sources["_ra"] = ra * u.deg
    sources["_dec"] = dec * u.deg
    sources.meta = {"name": "fake catalog"}
```

(continues on next page)

(continued from previous page)

```
    return sources
```

Construct the map with noise

```
def add_noise(sources_map, uncertainty):
    data = sources_map.to(Jypb).value + np.random.normal(loc=0, scale=1, size=sources_
↪map.shape) * uncertainty
    return data
```

Pack everything into `astropy.io.fits.HDUList`

```
def create_hdulist(data, hits, uncertainty, mask, wcs, sources, fwhm, noise_level):

    data[mask] = np.nan
    hits[mask] = 0
    uncertainty[mask] = 0

    header = wcs.to_header()
    header["UNIT"] = "Jy / beam", "Fake Unit"

    primary_header = fits.header.Header()
    primary_header["f_sampli"] = 10.0, "Fake the f_sampli keyword"

    # Old keyword for compatilibity
    primary_header["FWHM_260"] = fwhm.to(u.arcsec).value, "[arcsec] Fake the FWHM_260␣
↪keyword"
    primary_header["FWHM_150"] = fwhm.to(u.arcsec).value, "[arcsec] Fake the FWHM_150␣
↪keyword"

    # Traceback of the fake sources
    primary_header["nsources"] = len(sources), "Number of fake sources"
    primary_header["noise"] = noise_level.to(u.Jy / u.beam).value, "[Jy/beam] noise␣
↪level per map"

    primary = fits.hdu.PrimaryHDU(header=primary_header)

    hdus = fits.hdu.HDUList(hdus=[primary])
    for band in ["1mm", "2mm"]:
        hdus.append(fits.hdu.ImageHDU(data, header=header, name="Brightness_{}".
↪format(band)))
        hdus.append(fits.hdu.ImageHDU(uncertainty, header=header, name="Stddev_{}".
↪format(band)))
        hdus.append(fits.hdu.ImageHDU(hits, header=header, name="Nhits_{}".format(band)))
        hdus.append(fits.hdu.BinTableHDU(sources, name="fake_sources"))

    return hdus
```

Finally, run the `create_dataset()` function, only if called directly

```
if __name__ == "__main__":
    create_dataset()
```

Total running time of the script: (0 minutes 0.000 seconds)

G2 data

These are examples based on the G2 dataset, to explain the jackknife and bootstrap approach, but these can be applied to any dataset.

Jackknife dataset

Simple example of jacknnife generation on G2 dataset from the N2CLS GTO.

First import the `nikamap.jackkinfe` class

```
from pathlib import Path
from nikamap import Jackknife
```

This define the root directory where all the data ...

```
DATA_DIR = Path("/data/NIKA/Reduced/G2_COMMON_MODE_ONE_BLOCK/v_1")
```

can be retrieved using a simple regular expression

```
filenames = list(DATA_DIR.glob('*/map.fits'))
filenames
```

Create a jackknife object for future use in, for e.g., a simulation,

---

Note: by default the constructor will read all the maps in memory

---

```
jacks = Jackknife(filenames, n=10)
```

The *jacks* object can be iterated upon, each of the items is a different jackknife with noise properties corresponding to the original dataset

```
for nm in jacks:
    print(nm.check_SNR())
```

Total running time of the script: (0 minutes 0.000 seconds)

Bootstraping error

Simple example of bootstrap on G2 dataset from the N2CLS GTO.

First import the *nikamap.analysis.Bootstrap* class

```
from pathlib import Path
from nikamap import Bootstrap
```

This define the root directory where all the data ...

```
DATA_DIR = Path("/data/NIKA/Reduced/G2_COMMON_MODE_ONE_BLOCK/v_1")
```

can be retrieved using a simple regular expression

```
filenames = list(DATA_DIR.glob("*/map.fits"))
filenames
```

Generate a bootstrap dataset for all the maps.

Note: At the moment, this is very memory demanding as we use `map_size * (len(filenames) + n_bootstrap))`

```
nm = Bootstrap(filenames, n_bootstrap=200, ipython_widget=True)
```

In the resulting *nikamap.NikaMap* object, the uncertainty as been computed using the bootstrap technique.

Warning: The resulting uncertainty map could still be biased, see https://gitlab.lam.fr/N2CLS/NikaMap/issues/4

```
_ = nm.plot_SNR(cbar=True)
```

Total running time of the script: (0 minutes 0.000 seconds)

1.2 API

1.2.1 Classes

ContMap(data, *args, **kwargs)	A ContMap object represent a continuum map with additionnal capabilities.
NikaBeam(*args, **kwargs)	NikaBeam describe the beam of a NikaMap.
NikaMap(data, *args, **kwargs)	A NikaMap object represent a nika map with additionnal capabilities.
HalfDifference(filenames[, parity_threshold])	A class to create weighted half differences uncertainty maps from a list of scans.
Jackknife(filenames[, n_samples, ...])	A class to create weighted Jackknife maps from a list of scans.
Bootstrap(filenames[, n_bootstrap])	A class to create bootstraped maps from a list of IDL fits files.

nikamap.ContMap

class nikamap.**ContMap**(*data*, **args*, ***kwargs*)

A ContMap object represent a continuum map with additionnal capabilities.

It contains the metadata, wcs, and all attribute (data/stddev/time/unit/mask) as well as potential source list detected in these maps.

Parameters

- **data** (ndarray or astropy.nddata.NDData) – The actual data contained in this *NDData* object. Not that this will always be copies by *reference* , so you should make copy the `data` before passing it in if that's the desired behavior.
- **uncertainty** (astropy.nddata.NDUncertainty, optional) – Uncertainties on the data.
- **mask** (ndarray-like, optional) – Mask for the data, given as a boolean Numpy array or any object that can be converted to a boolean Numpy array with a shape matching that of the data. The values must be `False` where the data is *valid* and `True` when it is not (like Numpy masked arrays). If `data` is a numpy masked array, providing `mask` here will causes the mask from the masked array to be ignored.
- **hits** (ndarray-like, optional) – The hit per pixel on the map
- **sampling_freq** (float or Quantity) – the sampling frequency of the experiment, default 1 Hz
- **wcs** (*undefined, optional*) – WCS-object containing the world coordinate system for the data.
- **meta** (*dict*-like object, optional) – Metadata for this object. "Metadata" here means all information that is included with this object but not part of any other attribute of this particular object. e.g., creation date, unique identifier, simulation parameters, exposure time, telescope name, etc.
- **unit** (astropy.units.UnitBase instance or str, optional) – The units of the data.
- **beam** (ContBeam) – The beam corresponding to the data, by default a gaussian constructed from the header 'BMAJ' 'BMIN', 'PA' keyword.
- **fake_source** (astropy.table.Table, optional) – The table of potential fake sources included in the data

  Note: The table must contain at least 3 columns: ['ID', 'ra', 'dec']

- **sources** (*:class`astropy.table.Table`, optional*) – The table of detected sources in the data.

__hash__()

Return hash(self).

Examples using `nikamap.ContMap`

- *Basic usage*

nikamap.NikaBeam

class nikamap.**NikaBeam**(**args*, ***kwargs*)

NikaBeam describe the beam of a NikaMap.

This class is for back-compatibility, returning a ContBeam object

Parameters

- **fwhm** (astropy.units.Quantity) – Full width half maximum of the Gaussian kernel.
- **pixel_scale** (*astropy.units.equivalencies.pixel_scale*) – The pixel scale either in units of angle/pixel or pixel/angle.

See also:

astropy.convolution.Gaussian2DKernel

__hash__()

Return hash(self).

nikamap.NikaMap

class nikamap.**NikaMap**(*data*, **args*, ***kwargs*)

A NikaMap object represent a nika map with additionnal capabilities.

It contains the metadata, wcs, and all attribute (data/stddev/time/unit/mask) as well as potential source list detected in these maps.

Parameters

- **data** (ndarray or astropy.nddata.NDData) – The actual data contained in this *NDData* object. Not that this will always be copies by *reference* , so you should make copy the `data` before passing it in if that's the desired behavior.
- **uncertainty** (astropy.nddata.NDUncertainty, optional) – Uncertainties on the data.
- **mask** (ndarray-like, optional) – Mask for the data, given as a boolean Numpy array or any object that can be converted to a boolean Numpy array with a shape matching that of the data. The values must be `False` where the data is *valid* and `True` when it is not (like Numpy masked arrays). If `data` is a numpy masked array, providing `mask` here will causes the mask from the masked array to be ignored.
- **hits** (ndarray-like, optional) – The hit per pixel on the map
- **sampling_freq** (float or Quantity) – the sampling frequency of the experiment, default 1 Hz
- **wcs** (*undefined, optional*) – WCS-object containing the world coordinate system for the data.
- **meta** (*dict*-like object, optional) – Metadata for this object. "Metadata" here means all information that is included with this object but not part of any other attribute of this particular object. e.g., creation date, unique identifier, simulation parameters, exposure time, telescope name, etc.

- **unit** (astropy.units.UnitBase instance or str, optional) – The units of the data.
- **beam** (radio_beam.Beam) – The beam corresponding to the data, by default a gaussian constructed from the header 'BMAJ' 'BMIN', 'PA' keyword.
- **fake_source** (astropy.table.Table, optional) – The table of potential fake sources included in the data

  Note: The table must contain at least 3 columns: ['ID', 'ra', 'dec']

- **sources** (*:class`astropy.table.Table`, optional*) – The table of detected sources in the data.

__hash__()

Return hash(self).

Examples using nikamap.NikaMap

- *Bootstraping error*

nikamap.HalfDifference

class nikamap.**HalfDifference**(*filenames*, *parity_threshold=1*, ***kwd*)

A class to create weighted half differences uncertainty maps from a list of scans.

This acts as a python lazy iterator and/or a callable

Parameters

- **filenames** (*list*) – the list of fits files to produce the Jackknifes
- **ipython_widget** (*bool, optional*) – If True, the progress bar will display as an IPython notebook widget.
- **n** (*int*) – the number of Jackknifes maps to be produced in the iterator

  if set to *None*, produce only one weighted average of the maps
- **parity_threshold** (*float*) – mask threshold between 0 and 1 to keep partially jackknifed area * 1 pure jackknifed * 0 partially jackknifed, keep all

Notes

A crude check is made on the wcs of each map when instanciated

__hash__()

Return hash(self).

nikamap.Jackknife

class nikamap.**Jackknife**(*filenames*, *n_samples=None*, *parity_threshold=1*, ***kwd*)

A class to create weighted Jackknife maps from a list of scans.

This acts as a python lazy iterator and/or a callable

Parameters

- **filenames** (*list*) – the list of fits files to produce the Jackknifes
- **n_samples** (*int*) – The number of (sub) samples to use (from 2 to len(filenames))
- **parity_threshold** (*float*) – mask threshold between 0 and 1 to keep partially jackknifed area * 1 pure jackknifed * 0 partially jackknifed, keep all
- **ipython_widget** (*bool, optional*) – If True, the progress bar will display as an IPython notebook widget.
- **n** (*int*) – the number of Jackknifes maps to be produced by the iterator

Notes

A crude check is made on the wcs of each map when instanciated

__hash__()

Return hash(self).

Examples using nikamap.Jackknife

- *Jackknife dataset*

nikamap.Bootstrap

class nikamap.**Bootstrap**(*filenames*, *n_bootstrap=None*, ***kwd*)

A class to create bootstraped maps from a list of IDL fits files.

This acts as a python lazy iterator and/or a callable

Parameters

- **filenames** (*list*) – the list of fits files to produce the Jackknifes
- **n_bootstrap** (*int*) – the number of realization to produce a bootsrapped map, by default 20 times the length of the input filename list
- **ipython_widget** (*bool, optional*) – If True, the progress bar will display as an IPython notebook widget.
- **n** (*int*) – the number of bootstrap maps to be produced by the iterator

Notes

A crude check is made on the wcs of each map when instanciated

__hash__()
Return hash(self).

Examples using `nikamap.Bootstrap`

- *Bootstraping error*

1.2.2 Functions

fake_data([shape, beam_fwhm, pixsize, nefd, ...])	Build fake dataset

nikamap.fake_data

nikamap.**fake_data**(*shape=(512, 512)*, *beam_fwhm=<Quantity 12.5 arcsec>*, *pixsize=<Quantity 2. arcsec>*, *nefd=<Quantity 0.05 Jy s(1/2) / beam>*, *sampling_freq=<Quantity 25. Hz>*, *time_fwhm=0.2*, *jk_data=None*, *e_data=None*, *nsources=32*, *peak_flux=None*, *pos_gen=<function pos_uniform>*, ***kwargs*)

Build fake dataset

Examples using `nikamap.fake_data`

- *Create a fake fits file*

INDEX

Symbols

B

C

F

H

J

N