Properimage Documentation

Release 0.7.1

Bruno Sánchez

Jun 25, 2022

Contents:

Properimage is an astronomical image processing code, specially written for coaddition, and image subtraction. It uses the mathematical developement published in the following papers [Zackay2016], [Zackay2017a], and [Zackay2017b], and a PSF estimation method published in [Lauer2002].

Most of the code is based on a class called *SingleImage*, which provides methods and properties for image processing such as PSF determination.

Note: A previous version of this code used the concept of *ensembles*, which was a class inheriting from python lists, grouping instances of SingleImage, providing all the mathematical operations over multiple images such as coadd, and subtract methods.

Now to offer more flexibility to user, there is only one class, the SingleImage. For coaddition, and subtraction the user must employ the functions provided, which take instances of SingleImage as input.

CHAPTER 1

Install Properimage

Install the latest release from PyPI

```
$ pip install properimage
```

CHAPTER 2

License

Properimage is released under The BSD-3 License.

This license allows unlimited redistribution for any purpose as long as its copyright notices and the license's disclaimers of warranty are maintained.

CHAPTER 3

Contents:

3.1 Tutorial

This section contains a step-by-step by example tutorial to analyze images with ProperImage.

Contents:

3.1.1 Tutorial - Part #1 - The SingleImage Class

In this first tutorial the basics of the methods and properties of the SingleImage class are explained.

This object class represents the basic unit of data which we will manipulate. Basically it starts containing pixel data, with its mask. First lets create an instance of this class with a numpy array.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

from properimage import single_image as s

%matplotlib inline
```

[2]:
```
pixel = np.random.random((128,128))*5.
# Add some stars to it
star = [[35, 38, 35],
        [38, 90, 39],
        [35, 39, 34]]
for i in range(25):
    x, y = np.random.randint(120, size=2)
    pixel[x:x+3,y:y+3] = star

mask = np.random.randint(2, size=(128,128))
for i in range(10):
    mask = mask & np.random.randint(2, size=(128,128))
```

(continues on next page)

(continued from previous page)

```
img  = s.SingleImage(pixel, mask)
```

We can see that the `img` object created automatically produces an output displaying the number of sources found. This just accounts for sources good enough for PSF estimation, which is the first step for any processing `ProperImage` is intended for.

If we try to print the instance, (or obtain the representation output) we find that the explicit origin of the data is being displayed

[3]:
```
print(img)
```

```
SingleImage instance for ndarray
```

If you would like to acces the data inside the object `img` just ask for `data`.

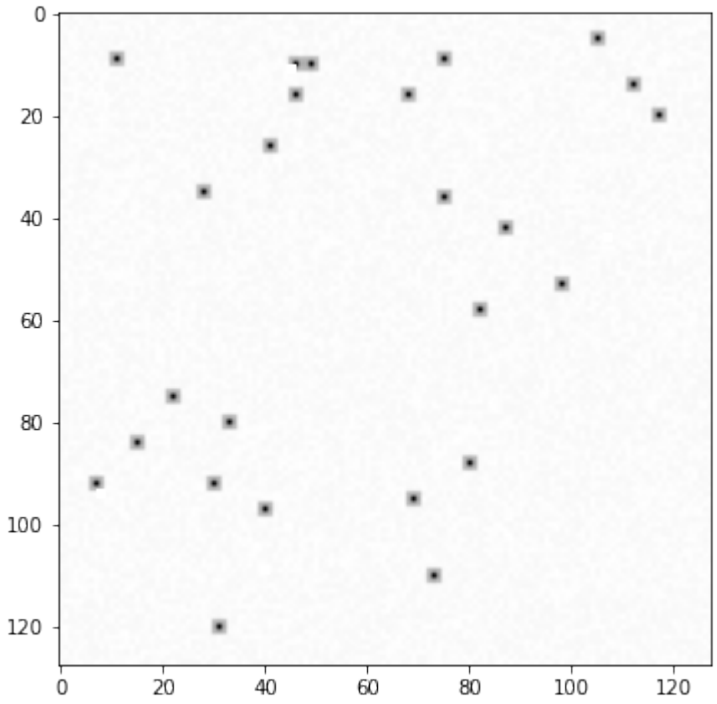
[4]:
```
img.data
```

[4]:
```
masked_array(
  data=[[4.05117654800415, 2.3367366790771484, 3.358428478240967, ...,
         2.9861879348754883, 2.9639270305633545, 2.420717954635562],
        [0.7913976907730103, 0.396319717168808, 4.990025997161865, ...,
         2.640835762023926, 0.19548970460891724, 4.581973075866699],
        [0.21659301221370697, 3.309115409851074, 3.2671613693237305, ...,
         0.602097749710083, 1.2853821516036987, 4.894009113311768],
        ...,
        [3.6120381355285645, 0.2359519749879837, 4.402842998504639, ...,
         3.145235300064087, 1.6198410987854004, 4.595581531524658],
        [1.655808687210083, 2.216202735900879, 2.922168254852295, ...,
         4.323221206665039, 4.086019515991211, 4.8711090087890625],
        [1.7609387636184692, 4.972808837890625, 2.632087469100952, ...,
         3.8503310680389404, 0.36555567383766174, 1.0644891262054443]],
  mask=[[False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        ...,
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False],
        [False, False, False, ..., False, False, False]],
  fill_value=1e+20,
  dtype=float32)
```

As can be seen it is a numpy masked array, with bad pixels flagged.

[5]:
```
plt.figure(figsize=(6,6))
plt.imshow(img.data, cmap='Greys')
```

[5]:
```
<matplotlib.image.AxesImage at 0x7f6d02315cd0>
```



We can check the best sources extracted.

[6]:
```
img.best_sources[['x', 'y', 'cflux']]
```

[6]:
```
array([(74.99950543,  8.9952085 , 269.88079834),
       (46.00495959, 16.00276395, 268.41549683),
       (67.99847428, 16.00627683, 269.9263916 ),
       (40.99845109, 25.99984788, 268.90808105),
       (28.00398889, 34.99543308, 267.75750732),
       (75.00020406, 36.00161855, 267.00131226),
       (86.99742868, 42.00051015, 268.46685791),
       (98.00059635, 53.01035352, 267.27404785),
       (81.99934787, 58.00604355, 269.30249023),
       (21.99991067, 74.99746784, 268.52197266),
       (32.999547  , 80.00032554, 267.3571167 ),
       (15.00032772, 84.00220434, 268.28329468),
       (80.00352989, 88.00013903, 266.56143188),
       ( 6.99951973, 92.00160289, 265.66516113),
       (30.0013353 , 91.99455509, 266.05792236),
       (40.00159237, 97.00203221, 266.15142822)],
      dtype={'names':['x','y','cflux'], 'formats':['<f8','<f8','<f8'], 'offsets':[56,
↪64,168], 'itemsize':240})
```

And also obtain the estimation of PSF.

As the PSF may vary across the field, we use the method described in Lauer T. (2002) for spatially variant PSF. The methodology returns a series of image-sized coefficients a_i and a series of basis PSF elements p_i.

[7]:
```
a_fields, psf_basis = img.get_variable_psf()
```

```
updating stamp shape to (15,15)
(16, 16) (225, 16)
```

As in our simple example we don't vary the PSF we obtain only a PSF element, and a `None` coefficient.

[8]:
```
len(psf_basis), psf_basis[0].shape
```

[8]:
```
(1, (15, 15))
```

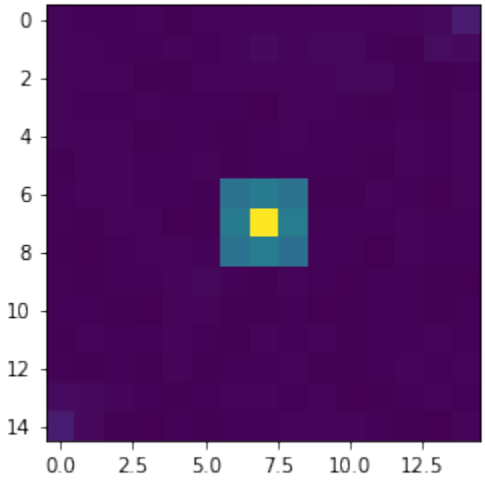
[9]:
```
a_fields
```

[9]:
```
[None]
```

We may check the looks of the `psf_basis` single element.

[10]:
```
plt.imshow(psf_basis[0])
```

[10]:
```
<matplotlib.image.AxesImage at 0x7f6d00a85a90>
```



3.1.2 Tutorial - Part #2 - Spatially variant PSF

Introduction

This package hosts a simple implementation of the proposed technique by Lauer 2002.

The technique performs a linear decomposition of the PSF into several *basis* elements of small size (much smaller than the image) normalized to unit sum. Each of these *autopsfs* come with a coefficient, which is not normalized, that have the same size of the image.

To perform the decomposition uses the *Karhunen-Loeve* transform, in order to calculate the *basis* as a linear expansion of the *psf observations* –which are cutout stamps of the stars in the image.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

[2]:
```
from properimage import simtools
from properimage import single_image as si
```

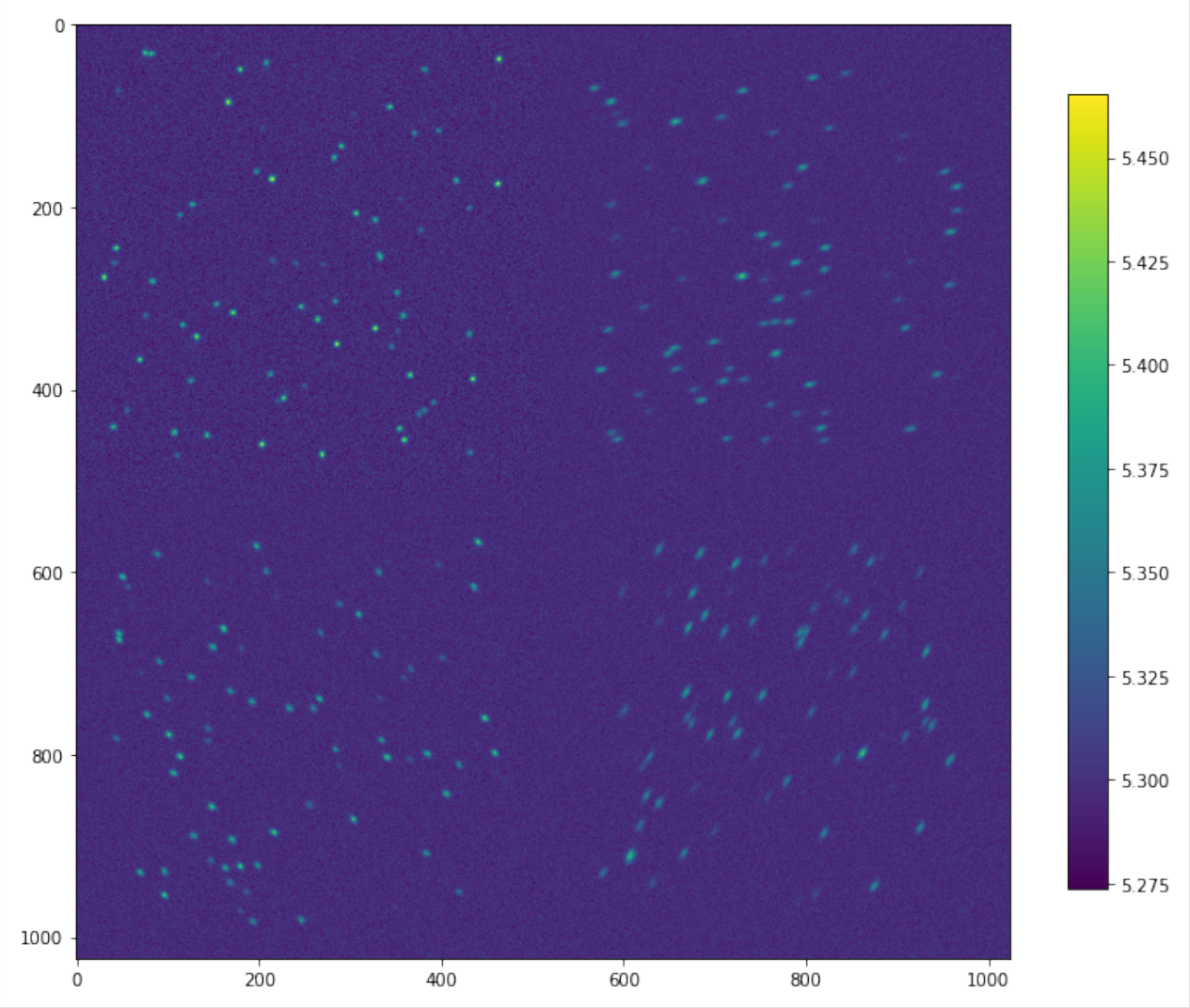
We simulate an image with several star shapes across the field

[3]:
```
frames = []
for theta in [0, 45, 105, 150]:
    N = 512  # side
    X_FWHM = 5 + 6.5*theta/180
    Y_FWHM = 5
    t_exp = 5
    max_fw = max(X_FWHM, Y_FWHM)
    #test_dir = os.path.abspath('./test/test_images/psf_basis_kl_gs')

    x = np.random.randint(low=6*max_fw, high=N-6*max_fw, size=80)
    y = np.random.randint(low=6*max_fw, high=N-6*max_fw, size=80)
    xy = [(x[i], y[i]) for i in range(80)]

    SN =  30. # SN para poder medir psf
    weights = list(np.linspace(10, 1000., len(xy)))
    m = simtools.delta_point(N, center=False, xy=xy, weights=weights)
    im = simtools.image(m, N, t_exp, X_FWHM, Y_FWHM=Y_FWHM, theta=theta,
                        SN=SN, bkg_pdf='gaussian')
    frames.append(im+100.)
```

[4]:
```
frame = np.zeros((1024, 1024))
for j in range(2):
    for i in range(2):
        frame[i*512:(i+1)*512, j*512:(j+1)*512] = frames[i+2*j]
```

[5]:
```
plt.figure(figsize=(12, 12))
plt.imshow(np.log(frame), interpolation='none')
plt.colorbar(shrink=0.7)
```

[5]:
```
<matplotlib.colorbar.Colorbar at 0x7f0a37c3c3d0>
```



We perform the psf extraction.

This is possible to do with a context manager, in order to *autoclean* the disk files where the star stamps are stored.

[6]:
```
%%time

with si.SingleImage(frame, smooth_psf=False) as sim:
    a_fields, psf_basis = sim.get_variable_psf(inf_loss=0.15)
    x, y = sim.get_afield_domain()
```

```
updating stamp shape to (25,25)
```

```
CPU times: user 1.74 s, sys: 277 ms, total: 2.01 s
Wall time: 1.14 s
```

[7]:
```
print(len(psf_basis))
```

```
3
```

This is equivalent to do the following:

[8]:
```
sim = si.SingleImage(frame, smooth_psf=False)
a_fields, psf_basis = sim.get_variable_psf(inf_loss=0.15)
x, y = sim.get_afield_domain()
```

```
updating stamp shape to (25,25)
```

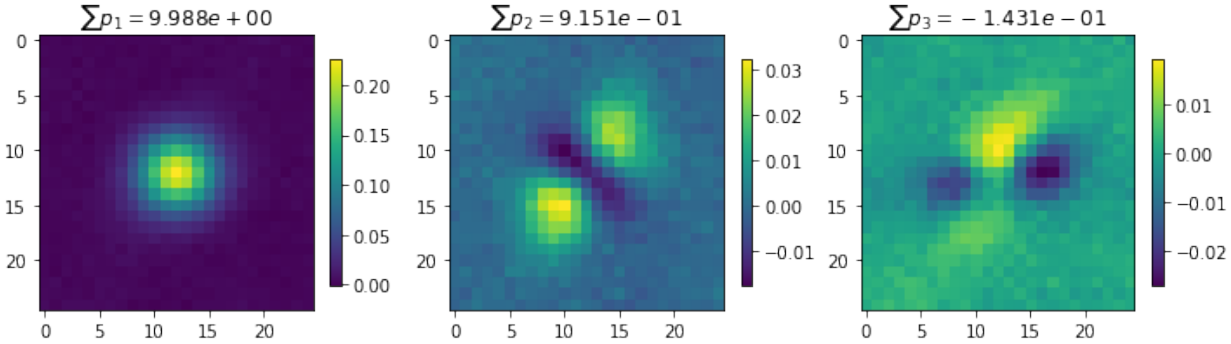
In this case, we will have the `sim` object in memory and we need to close it to erase the auxiliary files being generated.

We can peek into the variable PSF

The *autopsf* elements are small patches, like the ones below.

The principal component psf is the first element of this list of arrays.

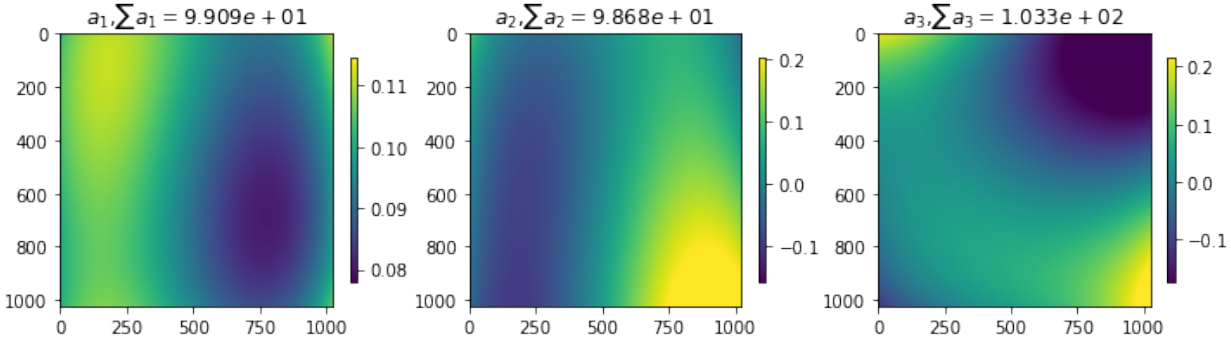
[9]:
```
ax = sim.plot.autopsf()
```



We can also plot the coefficients, they look as smooth fields of the same size of the image.

Whenever they grow they show where the corresponding *autopsf* element is more important than others.

[10]:
```
ax = sim.plot.autopsf_coef()
```



Rebuilding an image frame

In order to obtain the PSF value on any place of the image, we need to perform the summation of the *autopsf* elements weighted by the *afields* in the particular position we may desire.

[11]:
```
loc = (880, 995)
```

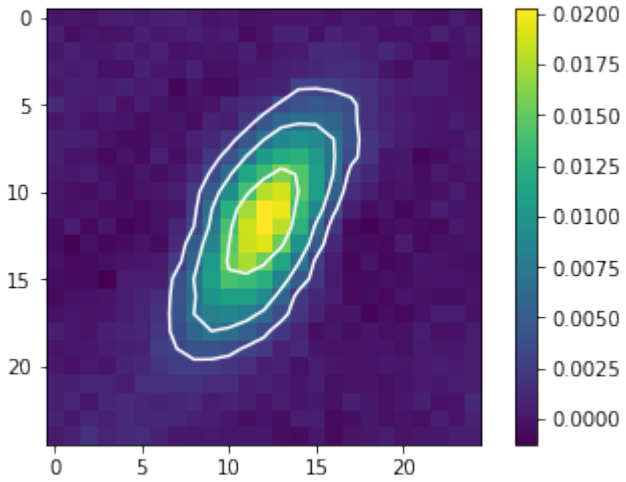
(continues on next page)

(continued from previous page)

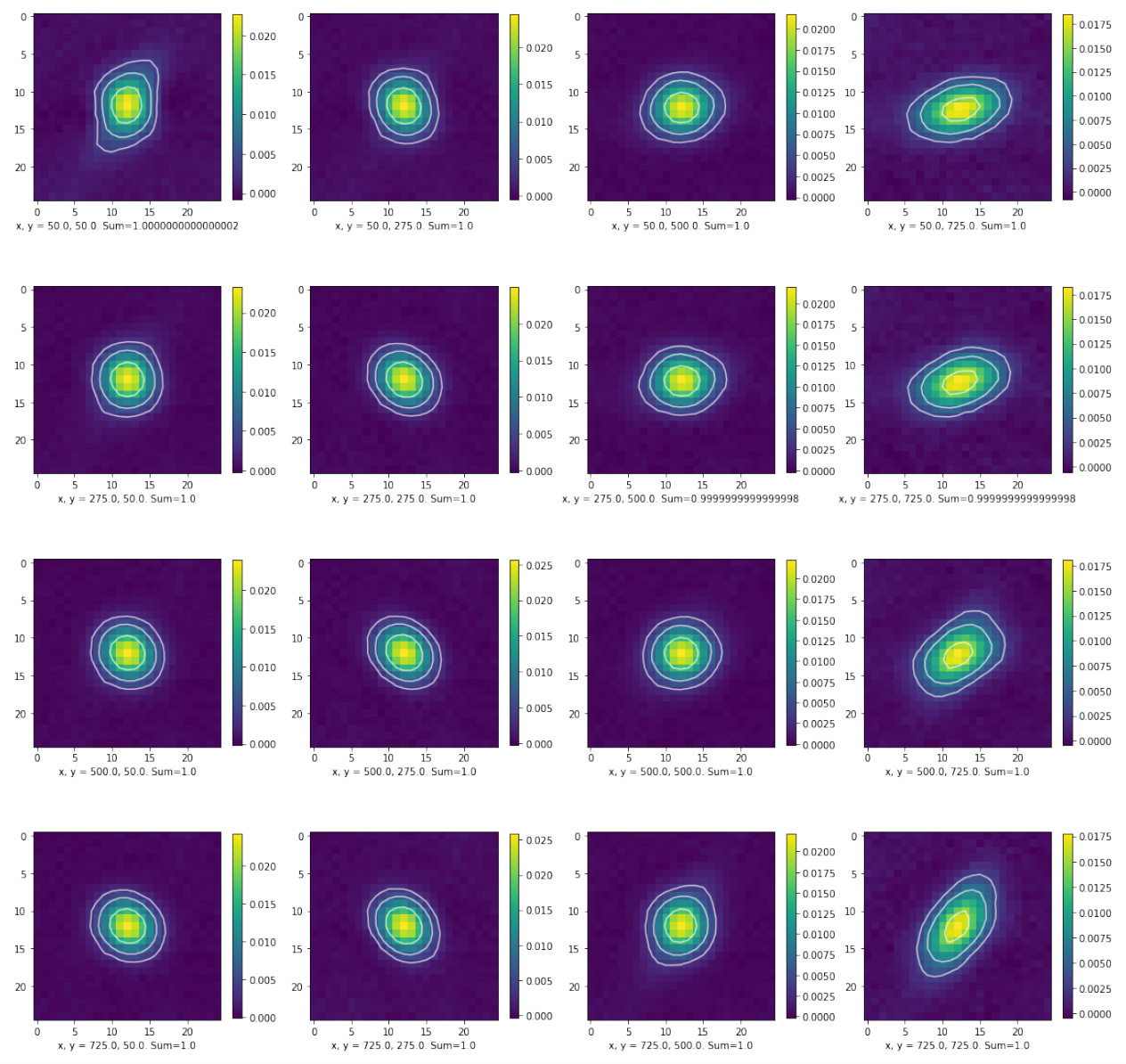
```
PSF_loc = np.zeros_like(psf_basis[0])
for apsf, afield in zip(psf_basis, a_fields):
    PSF_loc += apsf * afield(*loc)
```

[12]:
```
levels = np.logspace(-2.5, -0.5, num=7)

plt.imshow(PSF_loc)
plt.colorbar()
plt.contour(PSF_loc, levels=levels, cmap='Greys')
```

[12]:
```
<matplotlib.contour.QuadContourSet at 0x7f0a377153d0>
```



[13]:
```
xs = np.repeat(np.arange(50, 950, 900/4), 4)
ys = np.array([np.repeat(50+i*900/4., 4) for i in range(4)]).T.flatten()

i = 0
plt.figure(figsize=(16, 16))
for xp, yp in zip(xs, ys):
    PSF_loc = sim.get_psf_xy(xp, yp)
    plt.subplot(4, 4, i+1)
    plt.imshow(PSF_loc/np.sum(PSF_loc))
    plt.colorbar(shrink=0.7)
    plt.contour(PSF_loc, cmap='Greys', alpha=0.7, levels=levels)
    plt.xlabel('x, y = {}, {}. Sum={}'.format(np.round(xp), np.round(yp), np.sum(PSF_
↪loc)))
    i += 1
plt.tight_layout()
```



[ ]:

3.1.3 Tutorial - Part #3 - Advanced SingleImage

In this tutorial a more advanced set of examples are presented on SingleImage class, which allows tod do more specific tasks with the instances.

We import the packages, and also a pair of sample images

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```
from astropy.visualization import LinearStretch, LogStretch, ZScaleInterval,␣
↪MinMaxInterval, ImageNormalize
```

[3]:
```
import properimage.single_image as si
```

[4]:
```
img_path = './../../../data/aligned_eso085-030-004.fit'
```

[5]:
```
img = si.SingleImage(img_path)
```

Quickly we get the answer for the number of sources *a priori* we would use and the *estimated* size of thw PSF cutout stamp.

If we want to know the different properties assigned to this instance we can enumerate them:

- The origin of the information:

[6]:
```
print(img.attached_to)
```

```
./../../../data/aligned_eso085-030-004.fit
```

- The header if a fits file

[7]:
```
img.header
```

[7]:
```
SIMPLE  =                    T / conforms to FITS standard
BITPIX  =                   16 / array data type
NAXIS   =                    2 / number of array dimensions
NAXIS1  =                 1024
NAXIS2  =                  682
DATE-OBS= '2015-12-27T06:26:24' /YYYY-MM-DDThh:mm:ss observation start, UT
EXPTIME =   60.000000000000000 /Exposure time in seconds
EXPOSURE=   60.000000000000000 /Exposure time in seconds
SET-TEMP=  -20.000000000000000 /CCD temperature setpoint in C
CCD-TEMP=  -20.091654000000002 /CCD temperature at start of exposure in C
XPIXSZ  =   27.000000000000000 /Pixel Width in microns (after binning)
YPIXSZ  =   27.000000000000000 /Pixel Height in microns (after binning)
XBINNING=                    3 /Binning factor in width
YBINNING=                    3 /Binning factor in height
XORGSUBF=                    0 /Subframe X position in binned pixels
YORGSUBF=                    0 /Subframe Y position in binned pixels
READOUTM= 'Monochrome (Preflash)' /Readout mode of image
FILTER  = 'Clear    ' /          Filter used when taking image
IMAGETYP= 'Light Frame' /        Type of image
SITELAT = '-31 35 48' /          Latitude of the imaging location
SITELONG= '-64 32 56' /          Longitude of the imaging location
JD      =   2457383.7683333335 /Julian Date at start of exposure
FOCALLEN=   7475.0000000000000 /Focal length of telescope in mm
APTDIA  =   1540.0000000000000 /Aperture diameter of telescope in mm
APTAREA =   1862650.3361463547 /Aperture area of telescope in mm^2
SWCREATE= 'MaxIm DL Version 5.24 130605 0QTH7' /Name of software that created
         the image
SBSTDVER= 'SBFITSEXT Version 1.0' /Version of SBFITSEXT standard in effect
OBJECT  = 'eso085-030'
TELESCOP= '        ' /           telescope used to acquire this image
INSTRUME= 'Apogee USB/Net'
OBSERVER= '        '
NOTES   = '        '
FLIPSTAT= '        '
```

(continues on next page)

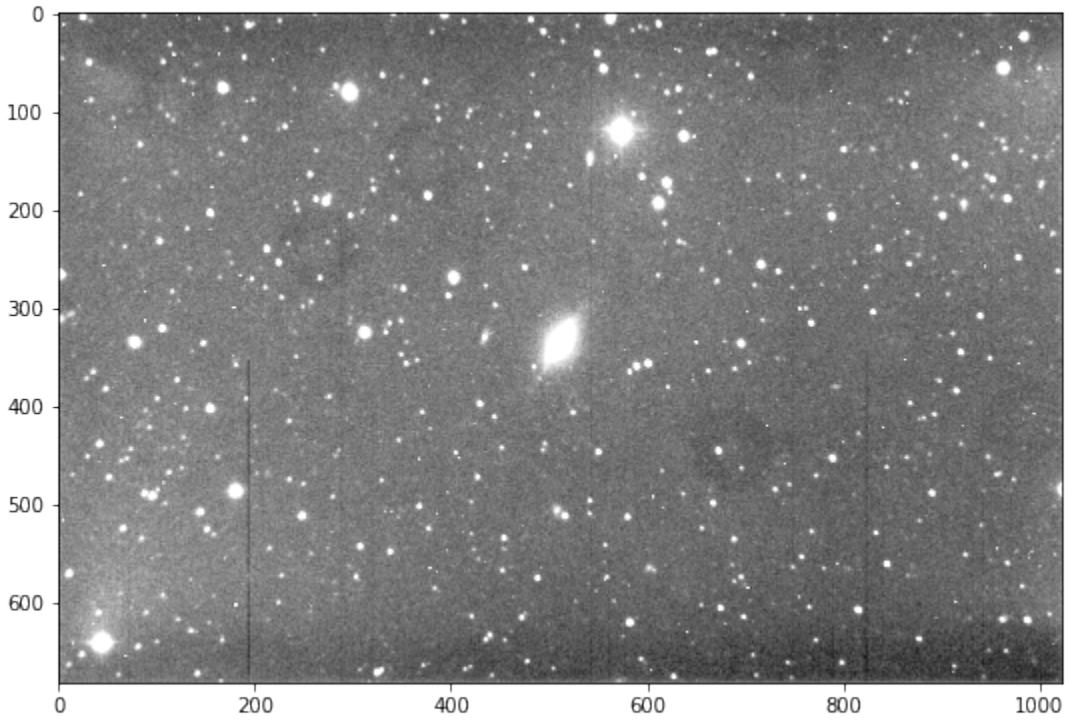
(continued from previous page)

```
SWOWNER = 'Mario C Diaz' /      Licensed owner of software
BSCALE  =                    1
BZERO   =                32768
COMMENT aligned img /home/bruno/Documentos/Data/ESO085-030/eso085-030-004.fit to
COMMENT  /home/bruno/Documentos/Data/ESO085-030/eso085-030-003.fit
```

- The pixel data

[8]:
```
norm = ImageNormalize(img.data, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(9,10))
plt.imshow(img.data, cmap='Greys_r', norm=norm)
```

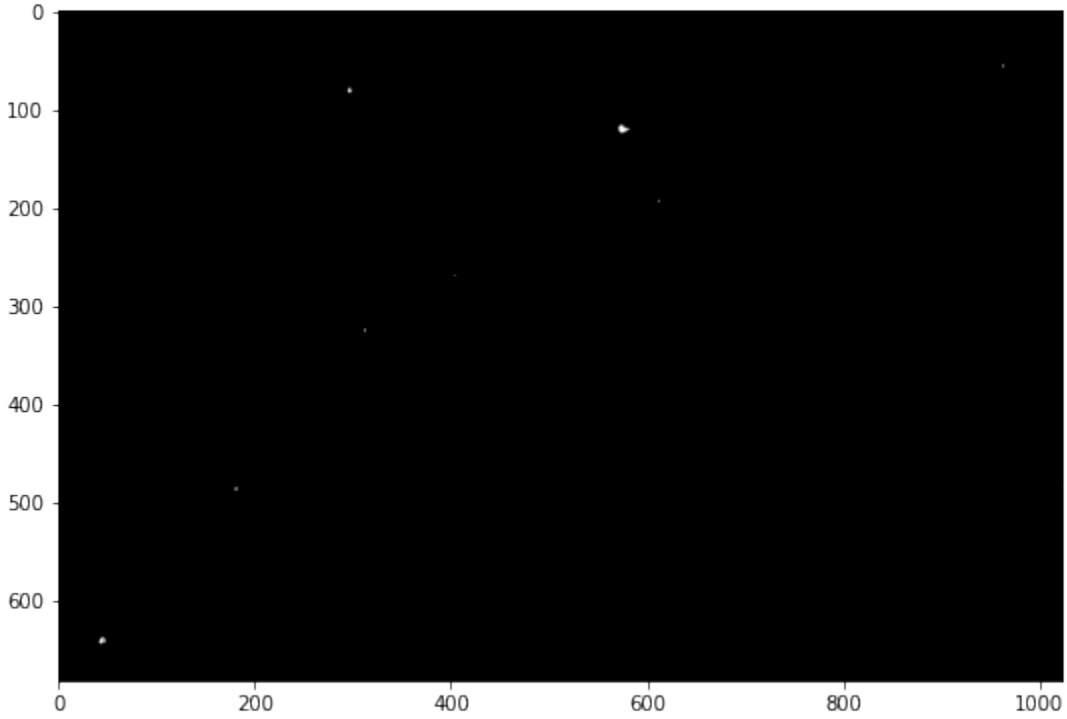
[8]:
```
<matplotlib.image.AxesImage at 0x7f9eacef5df0>
```



- The mask inferred or setted

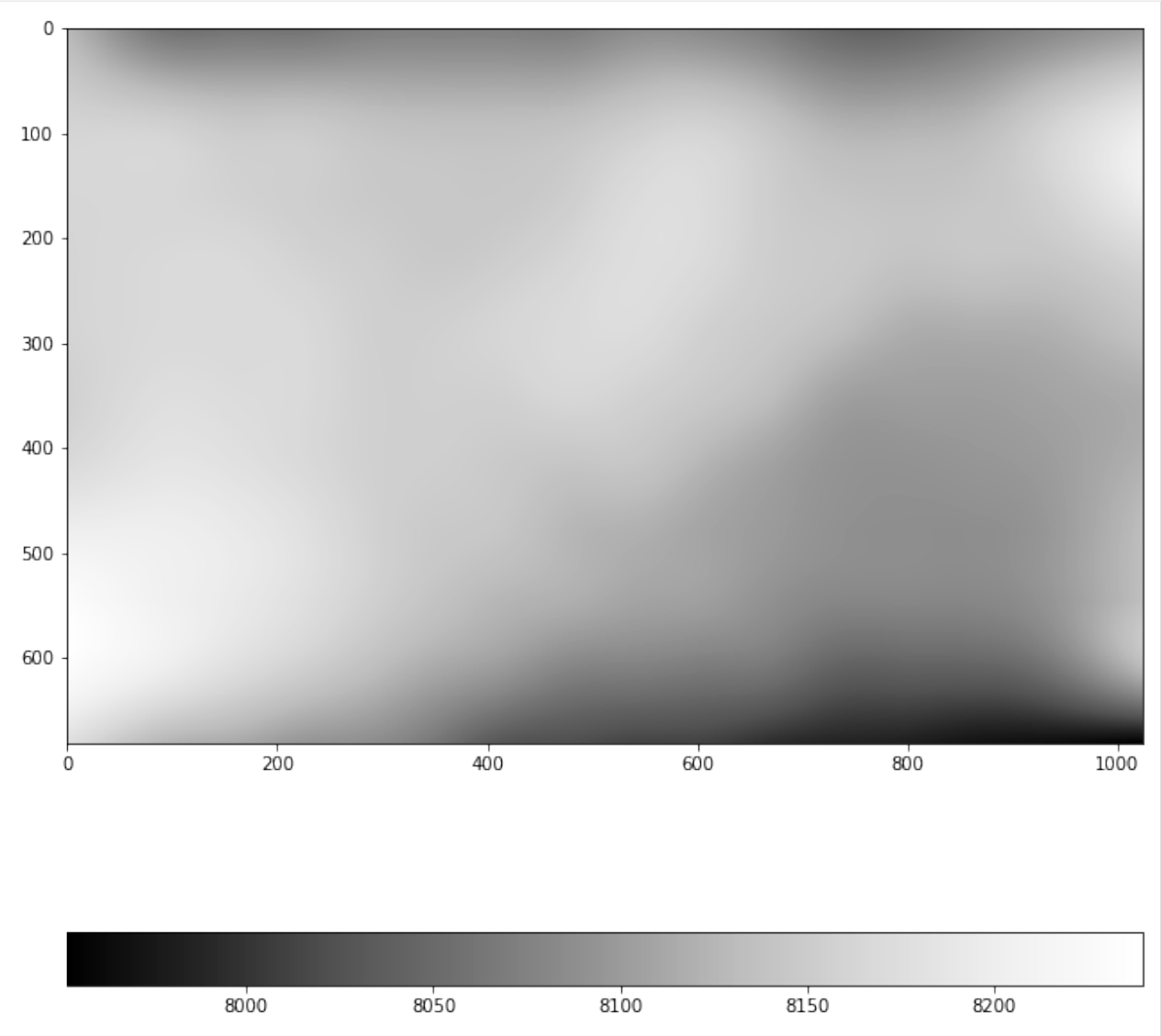
[9]:
```
plt.figure(figsize=(9,10))
plt.imshow(img.mask, cmap='Greys_r')
```

[9]:
```
<matplotlib.image.AxesImage at 0x7f9eace5d250>
```



- The background calculated

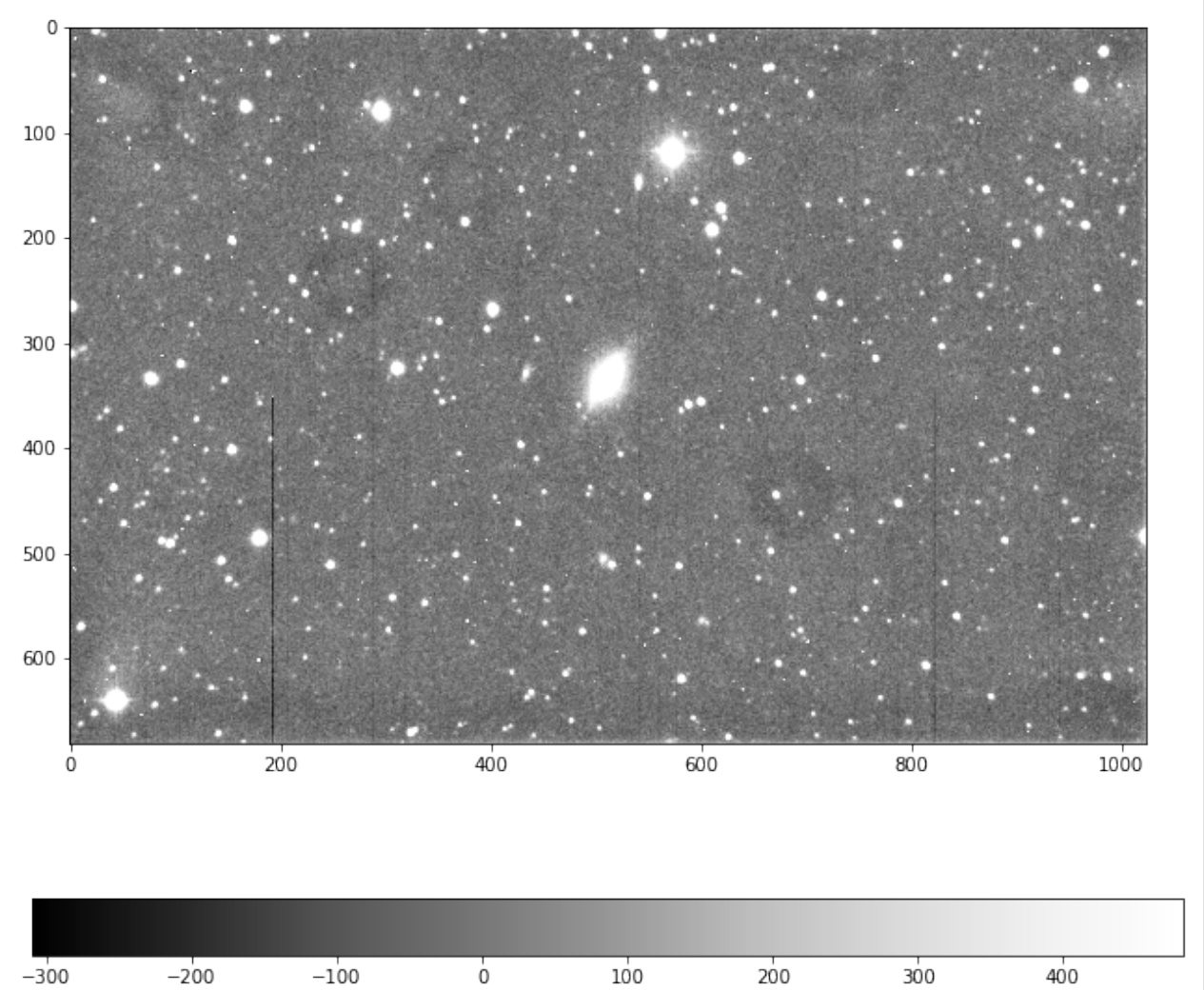
[10]:
```
plt.figure(figsize=(9,10))
plt.imshow(img.background, cmap='Greys_r')
plt.colorbar(orientation='horizontal')
plt.tight_layout()
```



As the background is being estimated only if accesed, then it prints the results.

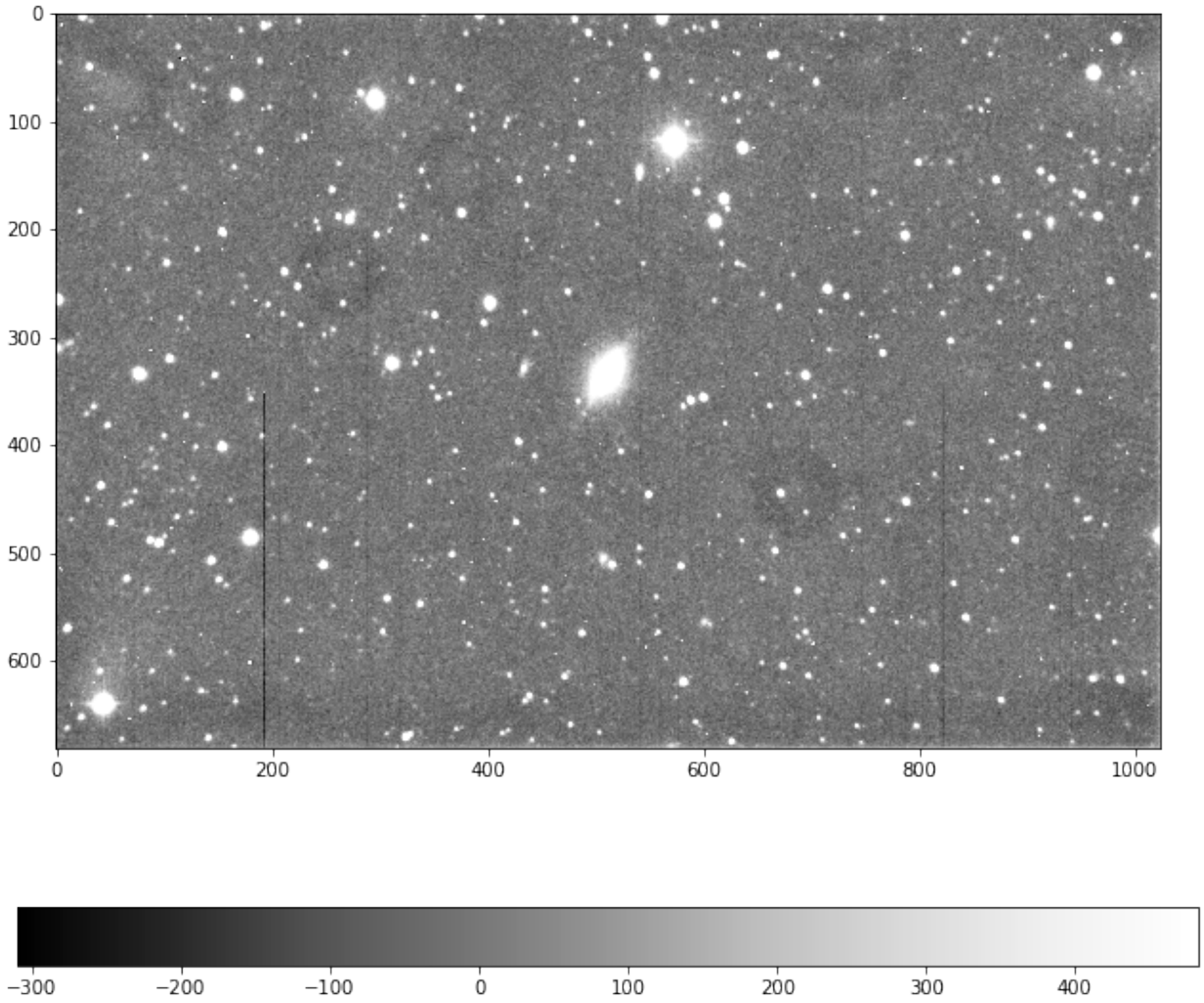
- The background subtracted image

[11]:
```
norm = ImageNormalize(img.bkg_sub_img, interval=ZScaleInterval(),
                      stretch=LinearStretch())
plt.figure(figsize=(9,8))
plt.imshow(img.bkg_sub_img, cmap='Greys_r', norm=norm)
plt.colorbar(orientation='horizontal')
plt.tight_layout()
```



- It also can be obtained a interpolated version of this image. The interpolated version is replacing masked pixels by using a box kernel convolutional interpolation:

[12]:
```
norm = ImageNormalize(img.bkg_sub_img, interval=ZScaleInterval(),
                      stretch=LinearStretch())
plt.figure(figsize=(9,8))
plt.imshow(img.interped, cmap='Greys_r', norm=norm)
plt.colorbar(orientation='horizontal')
plt.tight_layout()
```



- The stamp_shape to use (this is the final figure, after some exploring of the stars chosen)

[13]:
```
print(img.stamp_shape)
```

```
(15, 15)
```

- Get the stamp positions is also possible

[14]:
```
print(img.stamps_pos[0:10])
```

```
updating stamp shape to (21,21)
[[  4.00943632  25.26128582]
 [  7.14687073 480.9091281 ]
 [ 11.72774022 610.83903618]
 [ 12.82933202 193.25984592]
 [ 19.53525094 493.70670288]
 [ 41.5536534  548.48252542]
 [ 50.58756505  31.54122568]
 [ 64.9548974  704.69765449]
 [ 70.30182414 373.69091293]
 [ 74.9685698  282.51691292]]
```

- Obtaining the best sources was explained in Tutorial 01, but here we show it again just to be complete

[15]:
```
print(img.best_sources[0:10][['x', 'y', 'cflux']])
```

```
[( 25.26128582,  4.00943632, 53946.68359375)
 (480.9091281 ,  7.14687073, 24435.02539062)
 (610.83903618, 11.72774022, 40931.53515625)
 (193.25984592, 12.82933202, 47999.05859375)
 (493.70670288, 19.53525094, 30155.32421875)
 (548.48252542, 41.5536534 , 48259.6953125 )
 ( 31.54122568, 50.58756505, 22010.40234375)
 (704.69765449, 64.9548974 , 18889.1796875 )
 (373.69091293, 70.30182414, 19581.19335938)
 (282.51691292, 74.9685698 , 24461.18554688)]
```

- We can get the final number of sources used in PSF estimation

[16]:
```
print(img.n_sources)
```

```
83
```

- We can also print the covariance matrix from these objects

[17]:
```
print(img.cov_matrix)
```

```
[[8.98632497e-05 6.09003740e-05 7.33019848e-05 ... 5.39034293e-05
  5.81182182e-05 8.01227559e-05]
 [6.09003740e-05 5.48652651e-05 5.77634347e-05 ... 4.11557684e-05
  4.41151438e-05 5.80260826e-05]
 [7.33019848e-05 5.77634347e-05 7.42518817e-05 ... 4.85060699e-05
  5.77473931e-05 7.41490342e-05]
 ...
 [5.39034293e-05 4.11557684e-05 4.85060699e-05 ... 4.26306282e-05
  3.74503378e-05 5.01497669e-05]
 [5.81182182e-05 4.41151438e-05 5.77473931e-05 ... 3.74503378e-05
  5.33649456e-05 6.24053839e-05]
 [8.01227559e-05 5.80260826e-05 7.41490342e-05 ... 5.01497669e-05
  6.24053839e-05 9.33254922e-05]]
```

- As showed from Tutorial 02 we can get the PSF, depending on our level of approximation needed

[18]:
```
a_fields, psf_basis = img.get_variable_psf(inf_loss=0.01)
```

```
(83, 83) (441, 83)
```

[19]:
```
print(len(psf_basis), len(a_fields))
```

```
45 45
```

Check the information loss argument, which states the maximum amount of information lost in the basis expansion. If we change it the basis is updated:

[20]:
```
a_fields, psf_basis = img.get_variable_psf(inf_loss=0.10)
print(len(psf_basis), len(a_fields))
```

```
(83, 83) (441, 83)
3 3
```

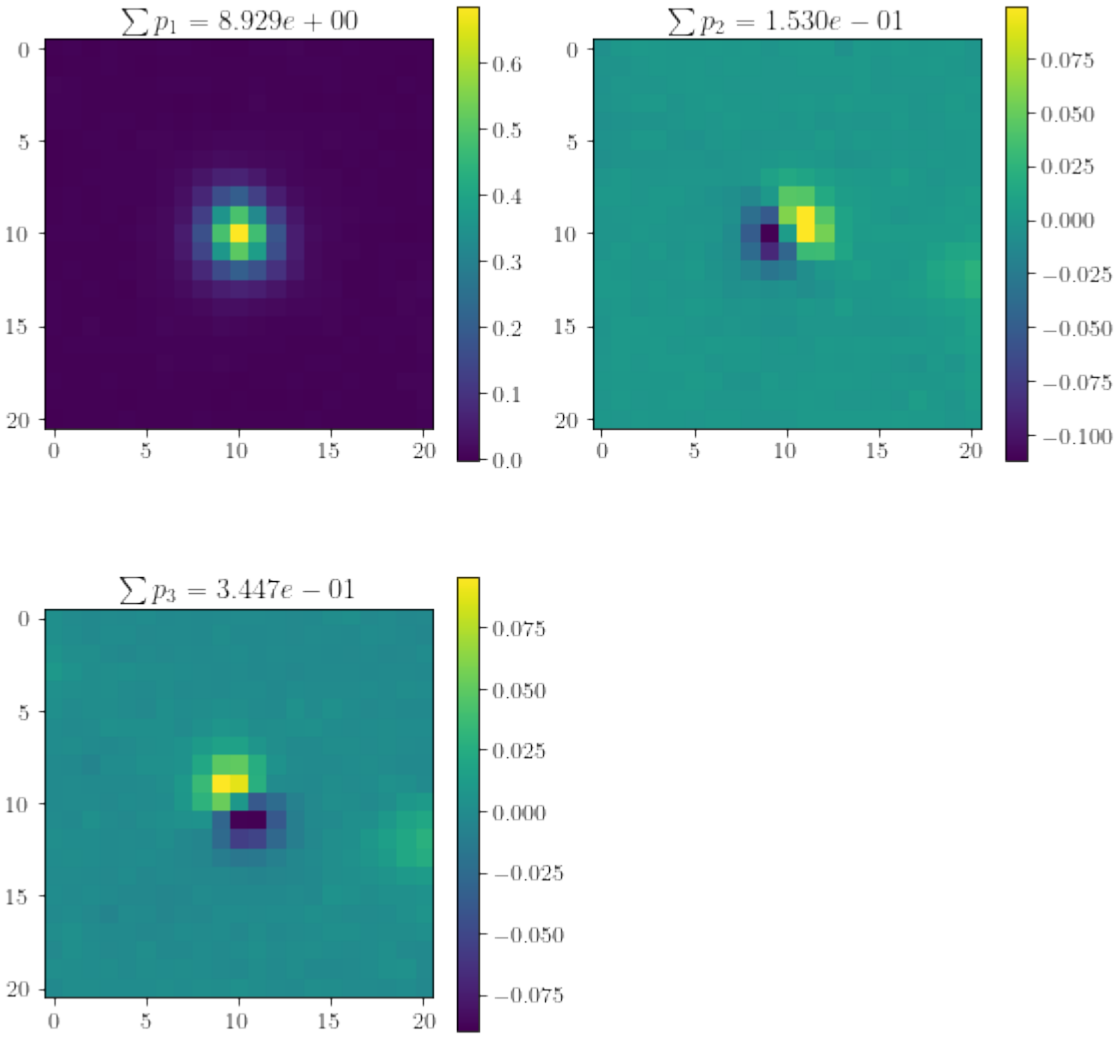
Of course the elements of the basis are unchanged, only a subset is returned. So going from small `inf_loss` to bigger values is the same as choosing less elements in the calculated basis.

Once obtained this basis and coefficient fields we can display them using some of the `plot` module functionalities:

[21]:
```
from properimage.plot import plot_afields, plot_psfbasis
```

[22]:
```
plot_psfbasis(psf_basis=psf_basis, nbook=True)
```

```
/home/bruno/Devel/zackay_code/properimage/properimage/plot.py:89:␣
↪MatplotlibDeprecationWarning: Passing non-integers as three-element position␣
↪specification is deprecated since 3.3 and will be removed two minor releases later.
  plt.subplot(subplots[1], subplots[0], i + 1)
```



For the `a_fields` object we need to give the coordinates where we evaluate this coefficients. A function is provided, inside `img` instance object.

[23]:
```
x, y = img.get_afield_domain()
```

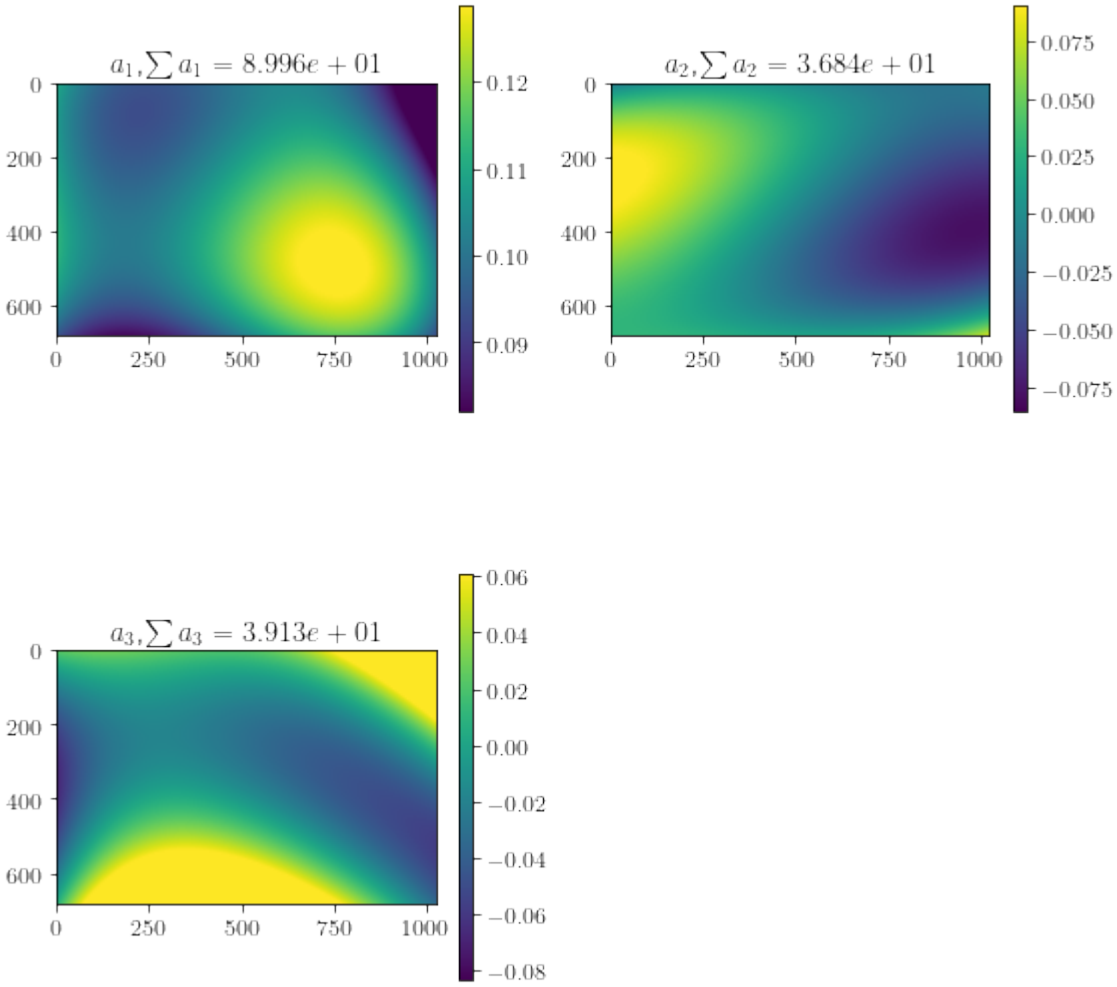
[24]:
```
plot_afields(a_fields=a_fields, x=x, y=y, nbook=True)
```

```
/home/bruno/Devel/zackay_code/properimage/properimage/plot.py:127:␣
↪MatplotlibDeprecationWarning: Passing non-integers as three-element position␣
↪specification is deprecated since 3.3 and will be removed two minor releases later.
  plt.subplot(subplots[1], subplots[0], i + 1)
/home/bruno/Devel/zackay_code/properimage/properimage/plot.py:127:␣
↪MatplotlibDeprecationWarning: Passing non-integers as three-element position␣
↪specification is deprecated since 3.3 and will be removed two minor releases later.
```

(continues on next page)

(continued from previous page)

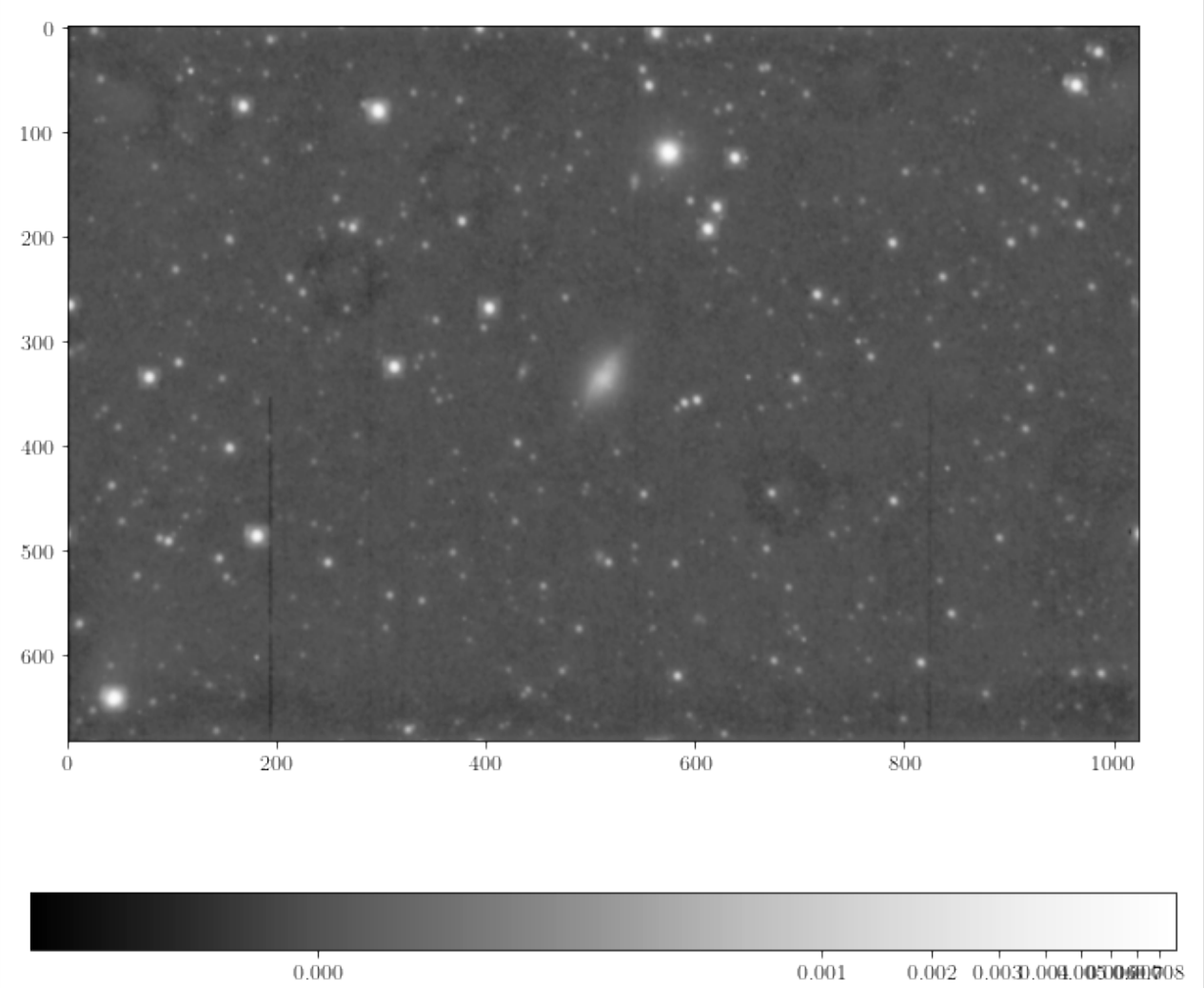
```
  plt.subplot(subplots[1], subplots[0], i + 1)
/home/bruno/Devel/zackay_code/properimage/properimage/plot.py:127:␣
↪MatplotlibDeprecationWarning: Passing non-integers as three-element position␣
↪specification is deprecated since 3.3 and will be removed two minor releases later.
  plt.subplot(subplots[1], subplots[0], i + 1)
```



- The instance is capable of calculating its own S component (Zackay et al. 2016 notation)

[25]:
```
S = img.s_component
```

[26]:
```
norm = ImageNormalize(S, interval=MinMaxInterval(),
                      stretch=LogStretch())
plt.figure(figsize=(9,8))
plt.imshow(S, cmap='Greys_r', norm=norm)
plt.colorbar(orientation='horizontal')
plt.tight_layout()
```



We can also attempt to place our PSF measurement on top of the stars of the image. This is done by placing a delta function in each star position, and convolving with the `autopsfs` obtained, and add them weighted with the `a(x,y)` fields.

[27]:
```
a_fields, psf_basis = img.get_variable_psf(inf_loss=0.05)
```

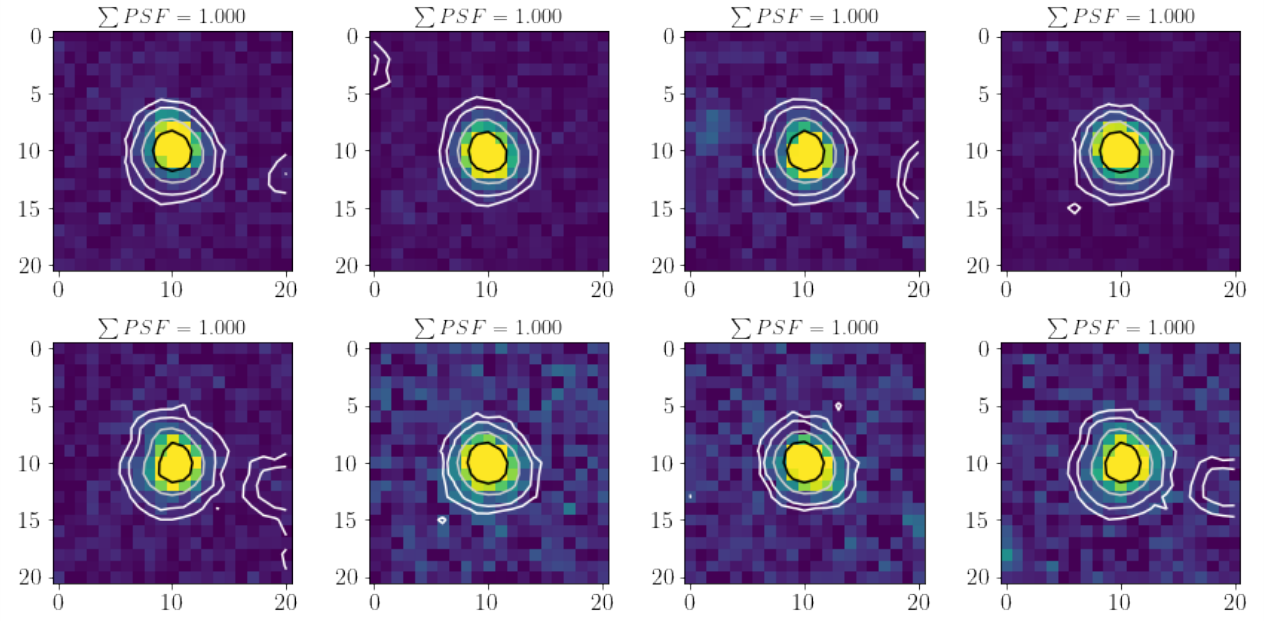
```
(83, 83) (441, 83)
```

[28]:
```
plt.figure(figsize=(12, 6))
for i in range(8):
    nsrc = np.random.randint(0, img.n_sources)
    xc, yc = img.best_sources[nsrc][['y', 'x']]
    try:
        patch = si.extract_array(img.data, img.stamp_shape,
                                 [xc, yc], fill_value=img._bkg.globalrms, mode='strict
↪')
    except:
        nsrc = np.random.randint(0, img.n_sources)
        xc, yc = img.best_sources[nsrc][['y', 'x']]
        patch = si.extract_array(img.data, img.stamp_shape,
                                 [xc, yc], fill_value=img._bkg.globalrms, mode='strict
↪')
```

(continues on next page)

(continued from previous page)

```
    plt.subplot(2, 4, i+1)
    patch = np.log10(patch)
    plt.imshow(patch, vmin=np.percentile(patch, q=10), vmax=np.percentile(patch,␣
↪q=98))
    plt.tick_params(labelsize=16)
    try:
        thepsf = img.get_psf_xy(xc, yc)
    except ValueError:
        print(xc, yc)
    #print(patch.shape, thepsf.shape)
    labels = {'sum': np.sum(thepsf)}
    plt.title(r'$\sum PSF = {sum:4.3f}$'.format(**labels))
    plt.contour(thepsf, levels=[0.0015, 0.003, 0.01, 0.03], cmap='Greys')

plt.tight_layout()
```



3.1.4 Tutorial - Part #4 - Image Subtraction

Introduction

For image subtraction the package has a module called `operations`, which implements a main `subtract` function to estimate pair-image subtractions.

[1]:
```
from copy import copy
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as colors

%matplotlib inline
```

[2]:
```
from astropy.io.fits import getdata
```

[3]:
```
from astropy.visualization import LinearStretch, LogStretch
from astropy.visualization import ZScaleInterval, MinMaxInterval
from astropy.visualization import ImageNormalize
```

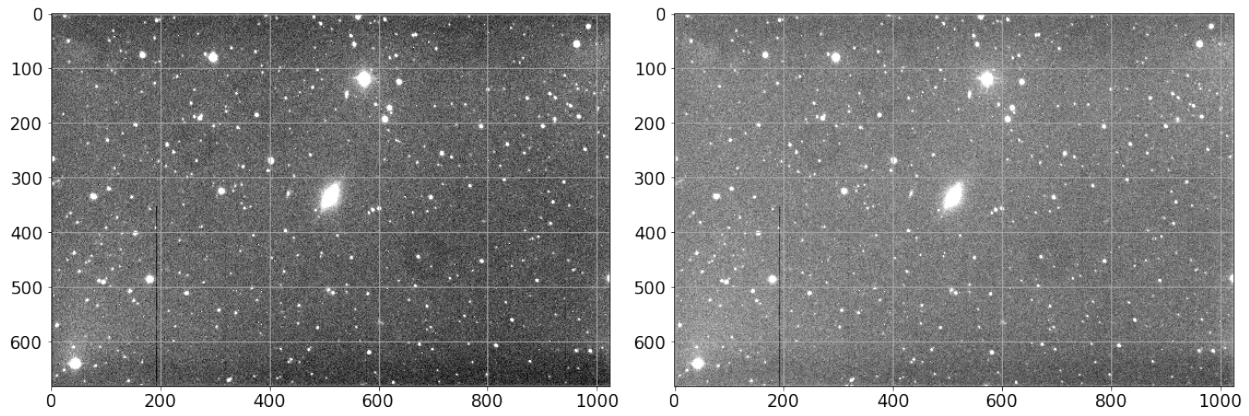
[4]:
```
palette = copy(plt.cm.gray)
palette.set_bad('r', 0.75)
```

[5]:
```
import properimage.single_image as si
from properimage.operations import subtract
```

[6]:
```
ref_path = './../../../data/aligned_eso085-030-004.fit'
new_path = './../../../data/aligned_eso085-030-005.fit'
```

To get the subtraction we need to run this function by using both paths for example:

[7]:
```
plt.figure(figsize=(16, 12))
plt.subplot(121)
ref = getdata(ref_path)
norm = ImageNormalize(ref, interval=ZScaleInterval(),
                      stretch=LinearStretch())
plt.imshow(ref, cmap=plt.cm.gray, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.grid()

plt.subplot(122)
ref = getdata(new_path)
norm = ImageNormalize(ref, interval=ZScaleInterval(),
                      stretch=LinearStretch())
plt.imshow(ref, cmap=plt.cm.gray, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.grid()

plt.tight_layout()
```



[8]:
```
%%time

result = subtract(
    ref=ref_path,
    new=new_path,
    smooth_psf=False,
    fitted_psf=True,
```

(continues on next page)

(continued from previous page)

```
    align=False,
    iterative=False,
    beta=False,
    shift=False
)
```

```
updating stamp shape to (21,21)
updating stamp shape to (21,21)
```

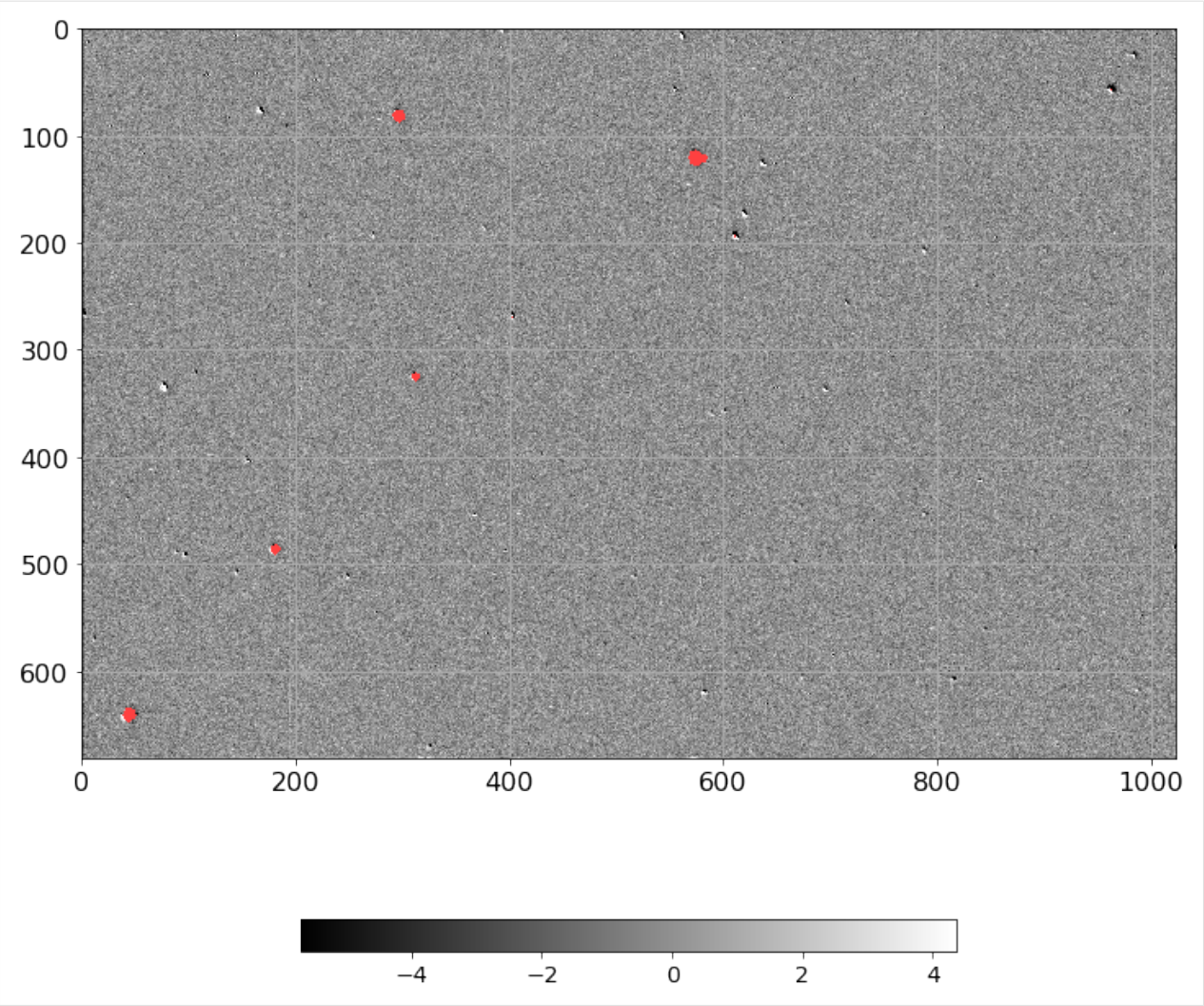
```
CPU times: user 21 s, sys: 1.55 s, total: 22.6 s
Wall time: 18.2 s
```

The result is a list of numpy arrays.

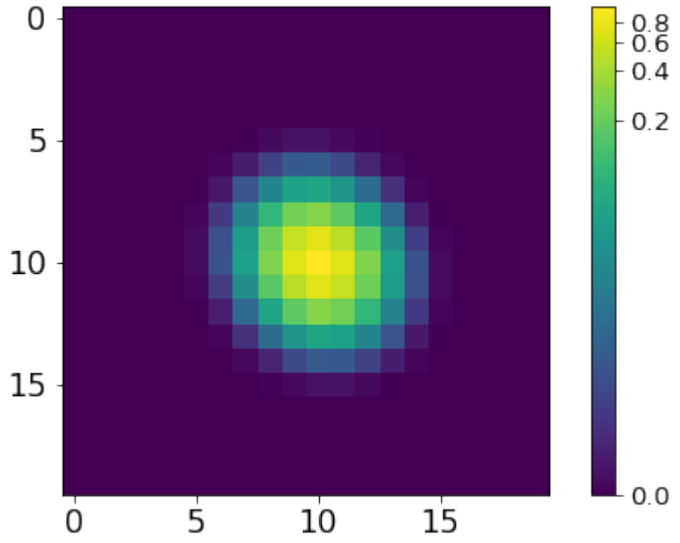
The arrays are in order: `D, P, Scorr, mask`

[9]:
```
D = result[0]
P = result[1]
Scorr = result[2]
mask = result[3]
```

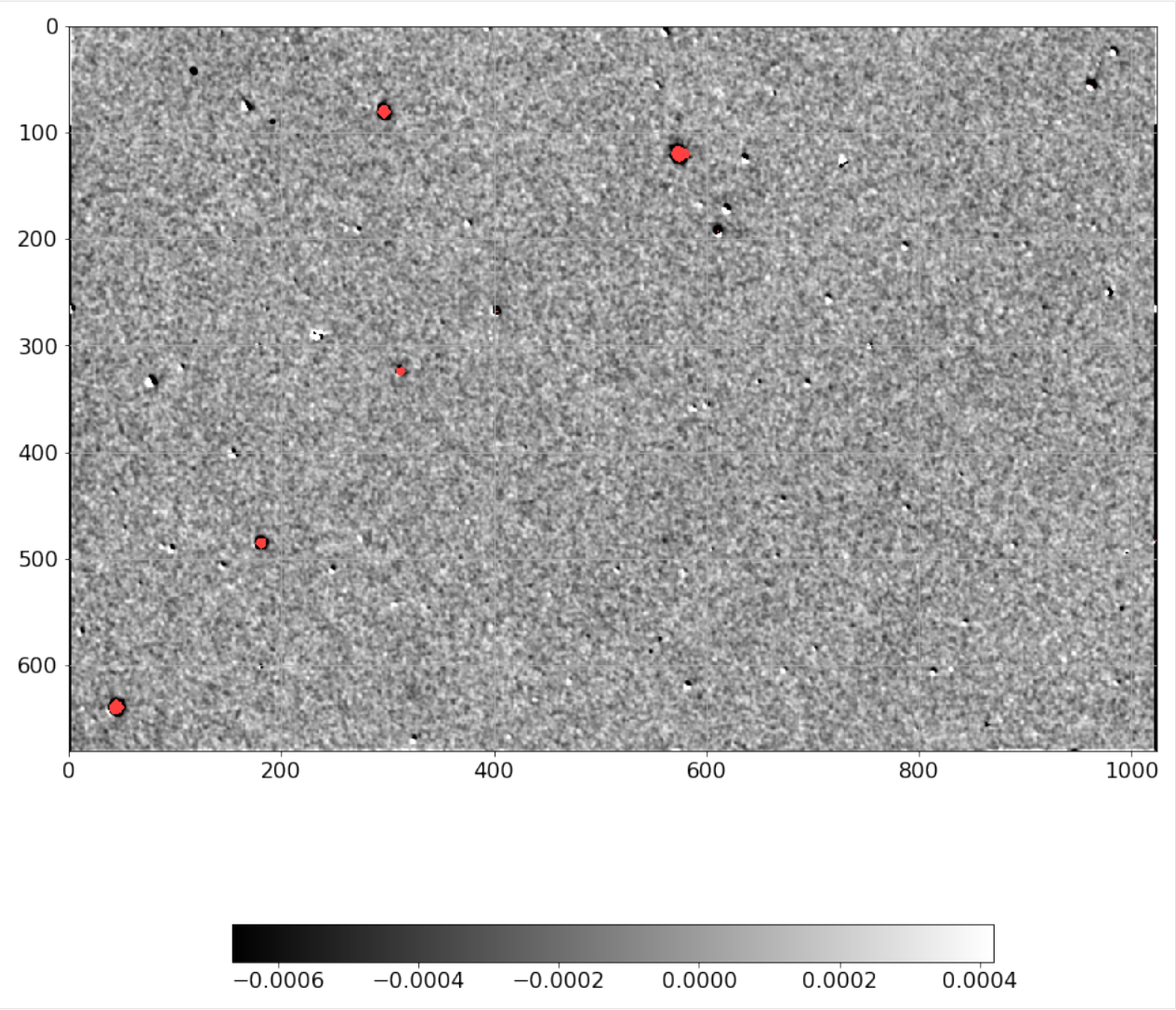
[10]:
```
norm = ImageNormalize(D.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(D.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.grid()
plt.colorbar(orientation='horizontal', shrink=0.6).ax.tick_params(labelsize=14)
```



[11]:
```
norm = ImageNormalize(P, interval=MinMaxInterval(), stretch=LogStretch())
xc, yc = np.where(P==P.max())
xc, yc = np.round(xc[0]), np.round(yc[0])
plt.imshow(P[xc-10:xc+10, yc-10:yc+10], norm=norm,
           cmap='viridis', interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar().ax.tick_params(labelsize=13)
plt.tight_layout()
```

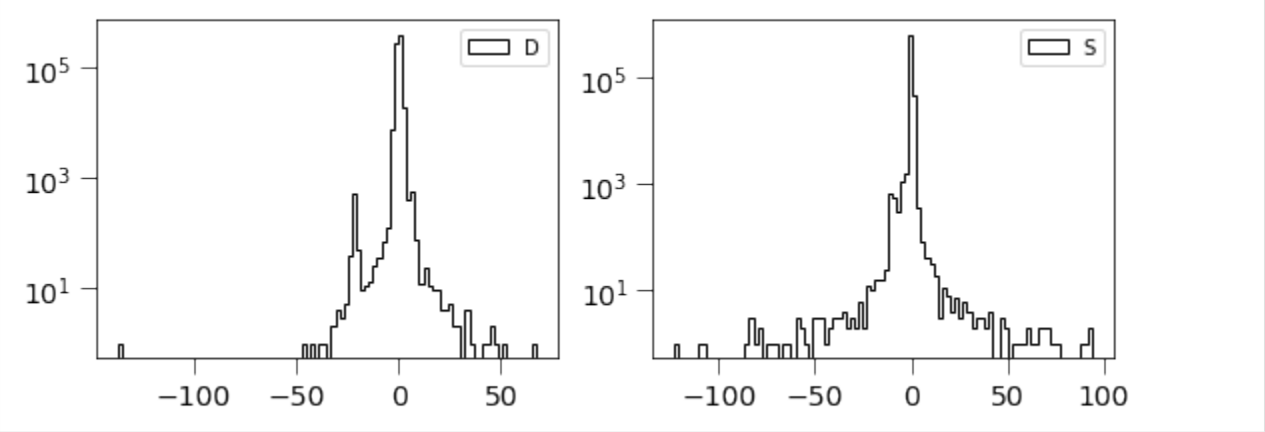


[12]:
```
norm = ImageNormalize(Scorr.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(Scorr.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar(orientation='horizontal', shrink=0.7).ax.tick_params(labelsize=16)
plt.grid()
plt.tight_layout()
```



[13]:
```
dimg = np.ma.MaskedArray(D.real, mask=mask).filled(0).flatten()
simage = np.ma.MaskedArray(Scorr.real, mask=mask).filled(0).flatten()
```

[14]:
```
plt.figure(figsize=(8, 3))

plt.subplot(121)
plt.hist(dimg, log=True, bins=100, histtype='step', label='D', color='k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.subplot(122)
plt.hist(simage/np.std(simage), log=True, bins=100, histtype='step', label='S', color=
↪'k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.tight_layout()
```



This is related to the quantities derived in Zackay et al. works. $S_{corr} = P_D \otimes D$

Different configurations:

We can run the subtraction using different configurations.

This parameters are the following:

```
align : bool
    Whether to align the images before subtracting, default to False
inf_loss : float
    Value of information loss in PSF estimation, lower limit is 0,
    upper is 1. Only valid if fitted_psf=False. Default is 0.25
smooth_psf : bool
    Whether to smooth the PSF, using a noise reduction technique.
    Default to False.
beta : bool
    Specify if using the relative flux scale estimation.
    Default to True.
shift : bool
    Whether to include a shift parameter in the iterative
    methodology, in order to correct for misalignments.
    Default to True.
iterative : bool
    Specify if an iterative estimation of the subtraction relative
    flux scale must be used. Default to False.
fitted_psf : bool
    Whether to use a Gaussian fitted PSF. Overrides the use of
    auto-psf determination. Default to True.
```

Example with `beta=True`, `iterative=False`

[15]:
```
%%time

D, P, Scorr, mask = subtract(
    ref=ref_path,
    new=new_path,
    align=False,
    iterative=False,
```

(continues on next page)

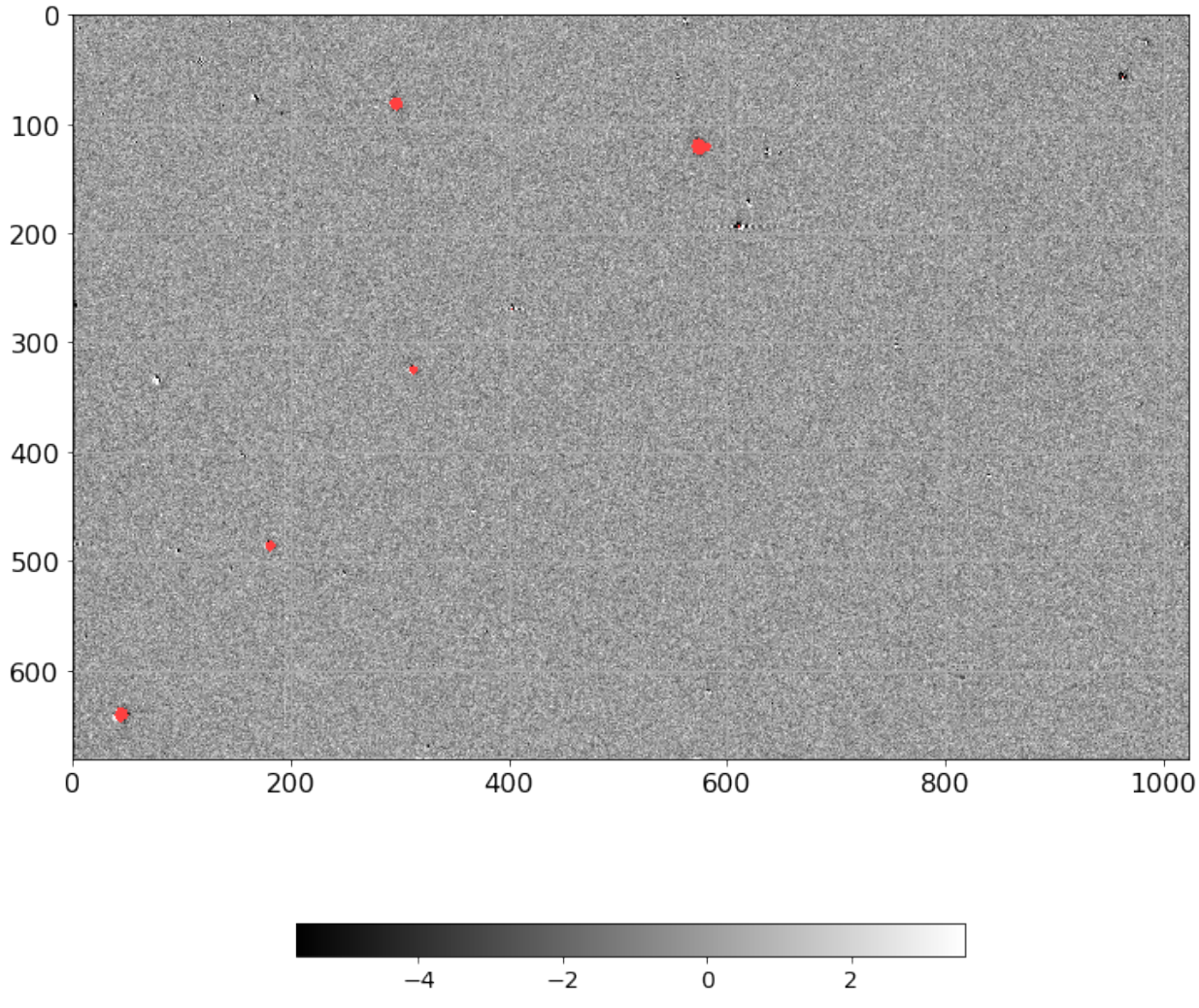
(continued from previous page)

```
    beta=True
)
```

```
updating stamp shape to (21,21)
updating stamp shape to (21,21)
```

```
CPU times: user 38.2 s, sys: 2.89 s, total: 41.1 s
Wall time: 37.5 s
```

[16]:
```
norm = ImageNormalize(D.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(D.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.grid()
plt.colorbar(orientation='horizontal', shrink=0.6).ax.tick_params(labelsize=14)
```

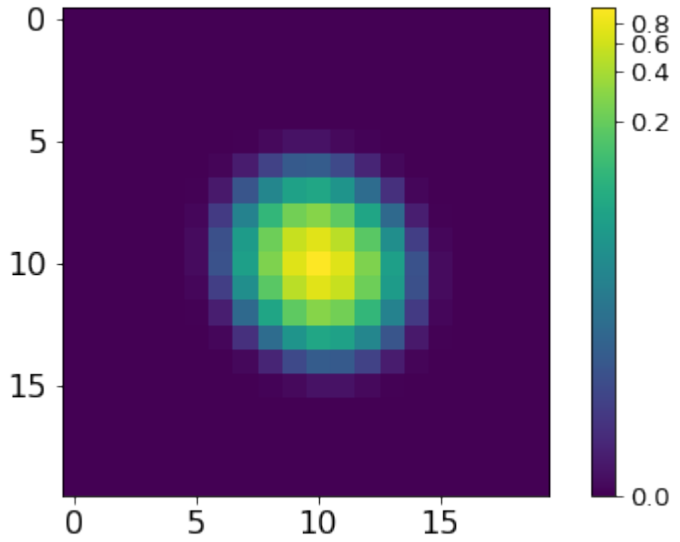


[17]:
```
norm = ImageNormalize(P, interval=MinMaxInterval(), stretch=LogStretch())
xc, yc = np.where(P==P.max())
```

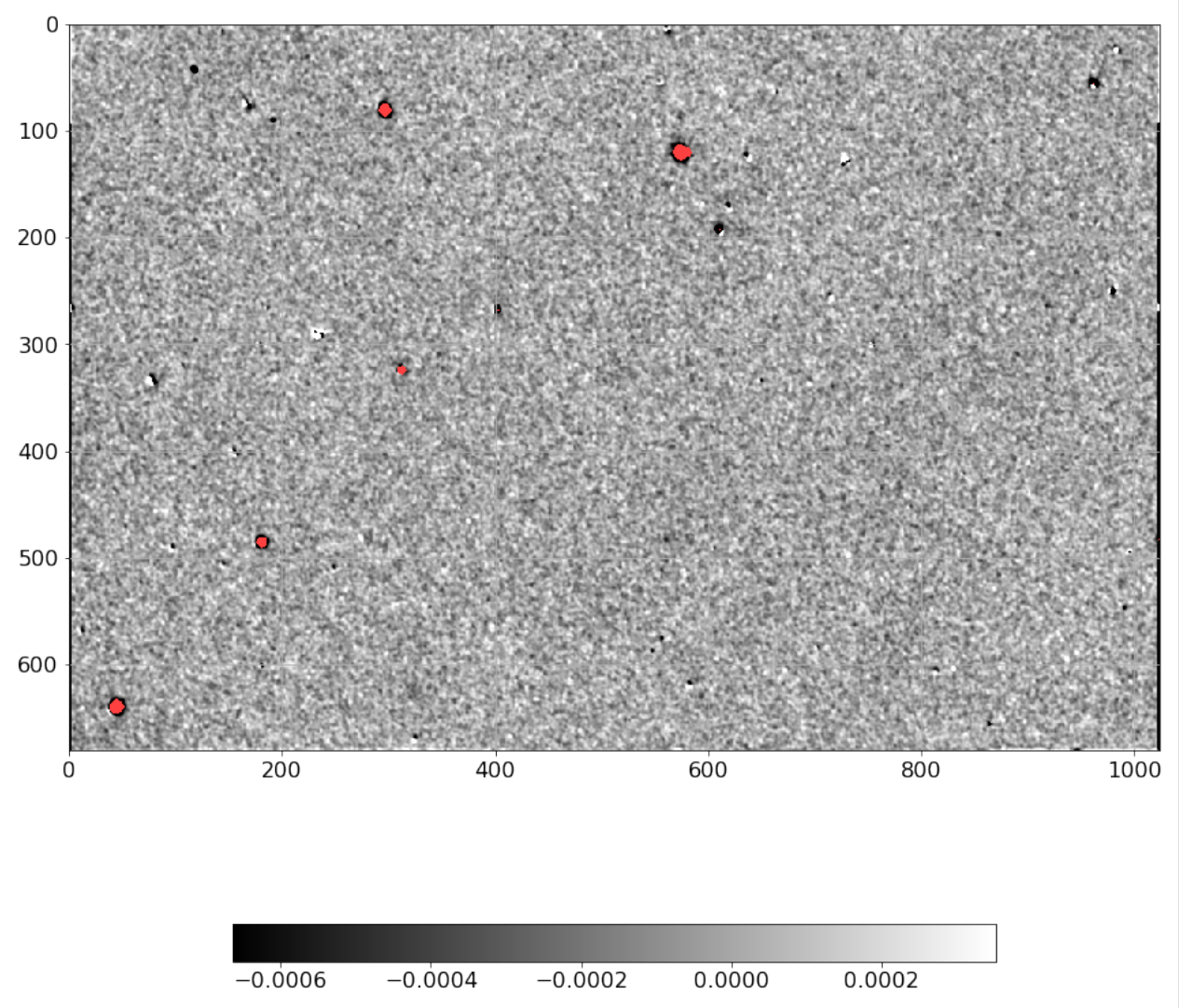
(continues on next page)

(continued from previous page)

```
xc, yc = np.round(xc[0]), np.round(yc[0])
plt.imshow(P[xc-10:xc+10, yc-10:yc+10], norm=norm,
           cmap='viridis', interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar().ax.tick_params(labelsize=13)
plt.tight_layout()
```

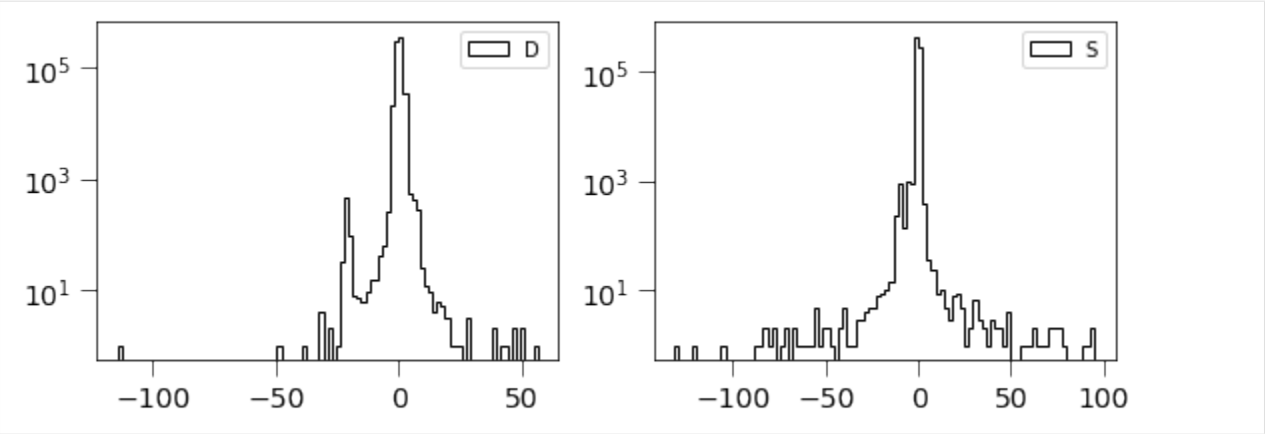


[18]:
```
norm = ImageNormalize(Scorr.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(Scorr.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar(orientation='horizontal', shrink=0.7).ax.tick_params(labelsize=16)
plt.grid()
plt.tight_layout()
```



[19]:
```
dimg = np.ma.MaskedArray(D.real, mask=mask).filled(0).flatten()
simage = np.ma.MaskedArray(Scorr.real, mask=mask).filled(0).flatten()
```

[20]:
```
plt.figure(figsize=(8, 3))

plt.subplot(121)
plt.hist(dimg, log=True, bins=100, histtype='step', label='D', color='k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.subplot(122)
plt.hist(simage/np.std(simage), log=True, bins=100, histtype='step', label='S', color=
↪'k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.tight_layout()
```



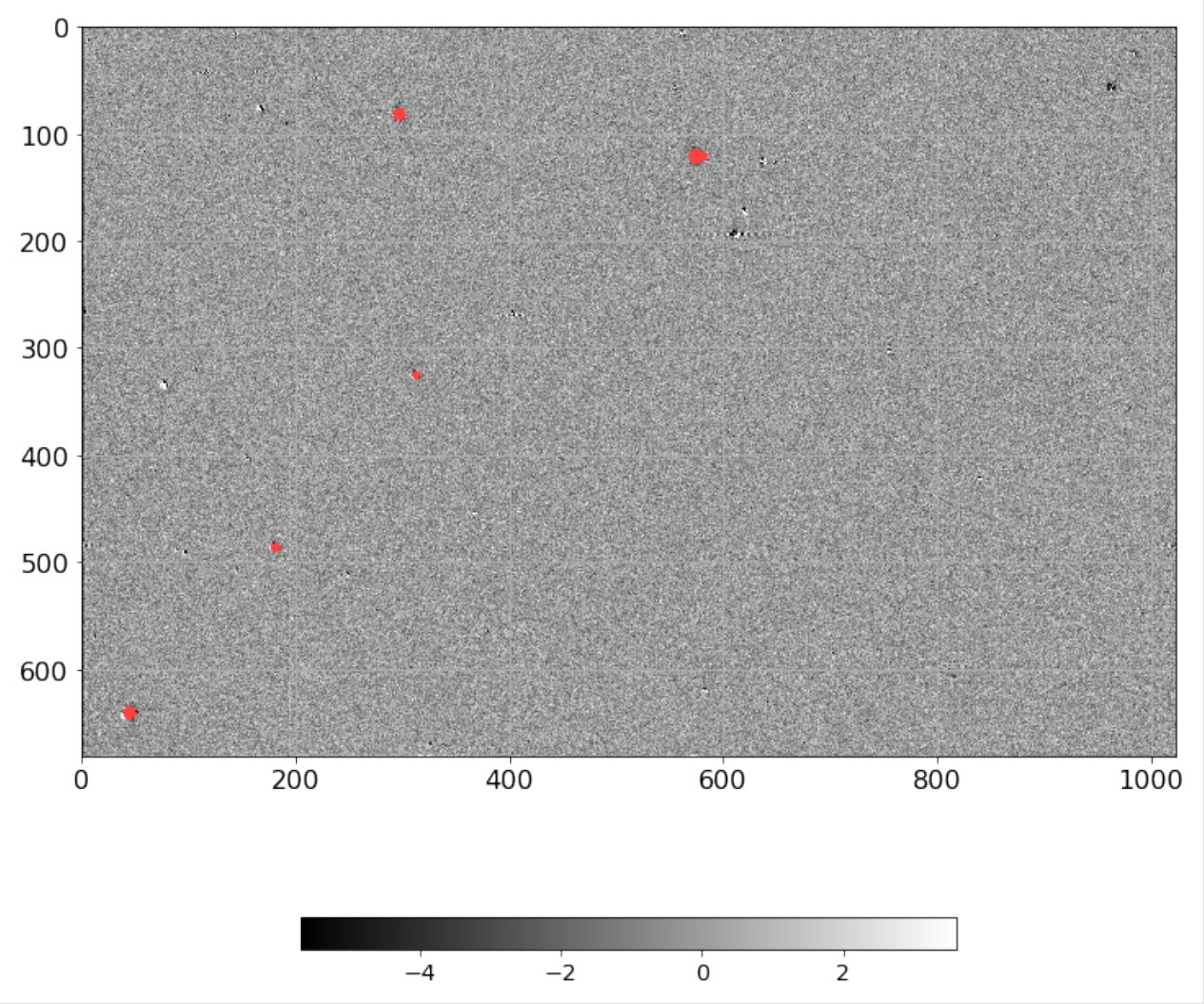
Example with `iterative=True` but `beta=False`

[21]:
```
%%time

D, P, Scorr, mask = subtract(
    ref=ref_path,
    new=new_path,
    align=False,
    iterative=True,
    beta=False
)
```

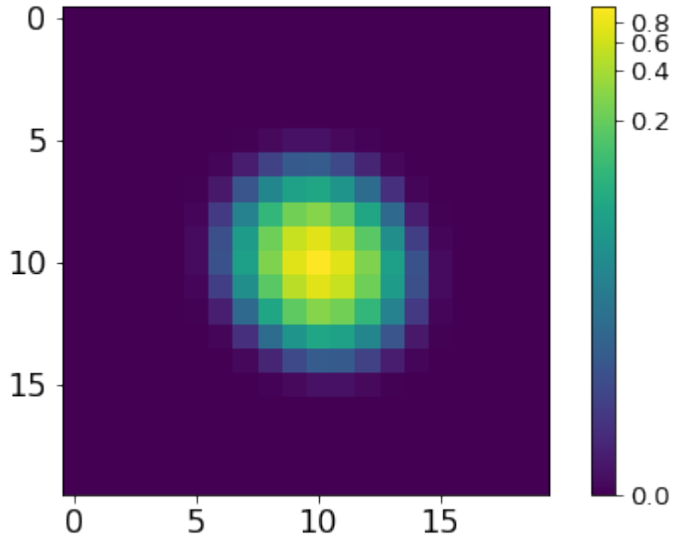
```
updating stamp shape to (21,21)
updating stamp shape to (21,21)
```

```
CPU times: user 24.7 s, sys: 1.28 s, total: 25.9 s
Wall time: 22.6 s
```

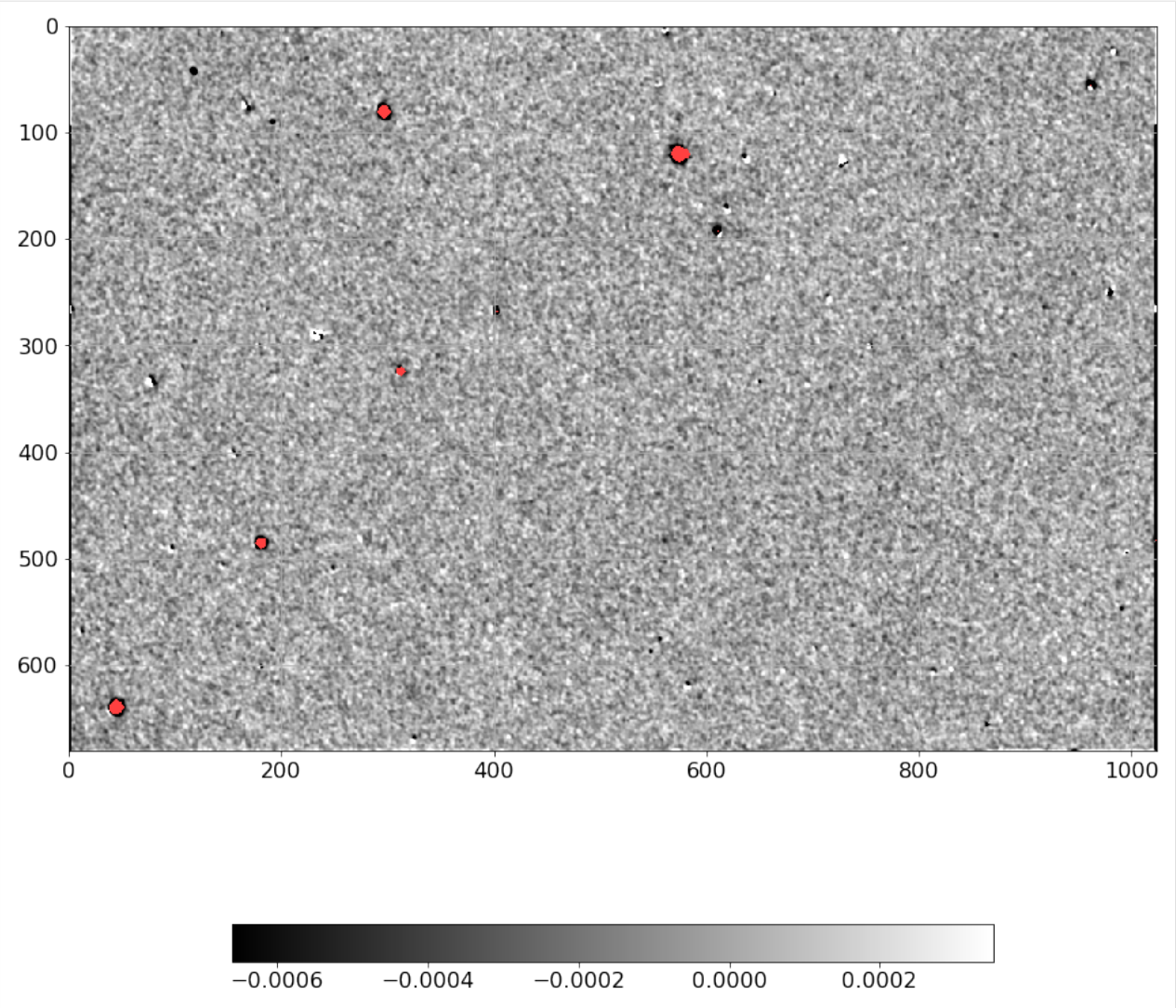
[22]:
```
norm = ImageNormalize(D.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(D.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.grid()
plt.colorbar(orientation='horizontal', shrink=0.6).ax.tick_params(labelsize=14)
```



[23]:
```
norm = ImageNormalize(P, interval=MinMaxInterval(), stretch=LogStretch())
xc, yc = np.where(P==P.max())
xc, yc = np.round(xc[0]), np.round(yc[0])
plt.imshow(P[xc-10:xc+10, yc-10:yc+10], norm=norm,
           cmap='viridis', interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar().ax.tick_params(labelsize=13)
plt.tight_layout()
```

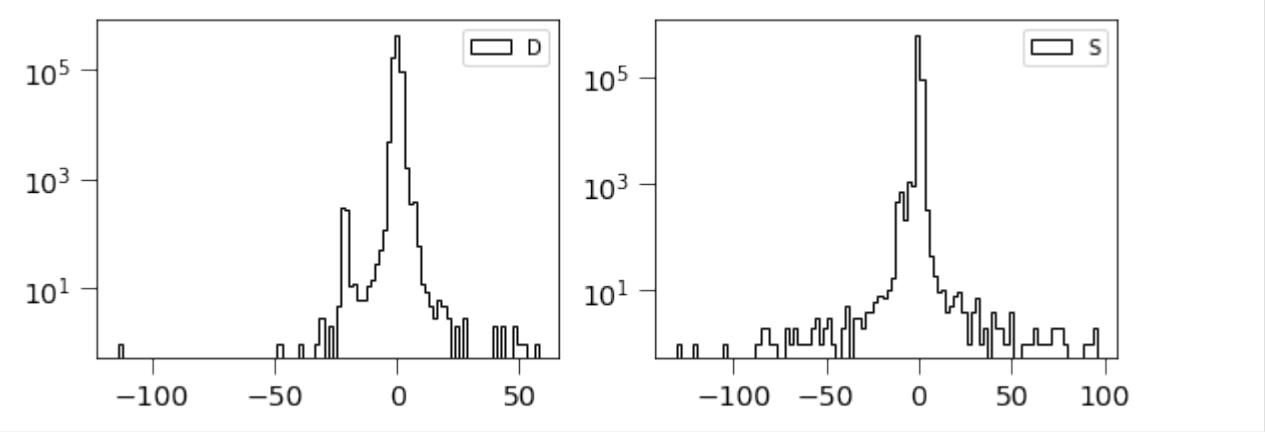


[24]:
```
norm = ImageNormalize(Scorr.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(Scorr.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar(orientation='horizontal', shrink=0.7).ax.tick_params(labelsize=16)
plt.grid()
plt.tight_layout()
```



[25]:
```
dimg = np.ma.MaskedArray(D.real, mask=mask).filled(0).flatten()
simage = np.ma.MaskedArray(Scorr.real, mask=mask).filled(0).flatten()
```

[26]:
```
plt.figure(figsize=(8, 3))

plt.subplot(121)
plt.hist(dimg, log=True, bins=100, histtype='step', label='D', color='k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.subplot(122)
plt.hist(simage/np.std(simage), log=True, bins=100, histtype='step', label='S', color=
↪'k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.tight_layout()
```



Example with `iterative=True` and `beta=True`

[27]:
```
%%time

D, P, Scorr, mask = subtract(
    ref=ref_path,
    new=new_path,
    smooth_psf=False,
    fitted_psf=True,
    align=False,
    iterative=True,
    beta=True,
    shift=True
)
```

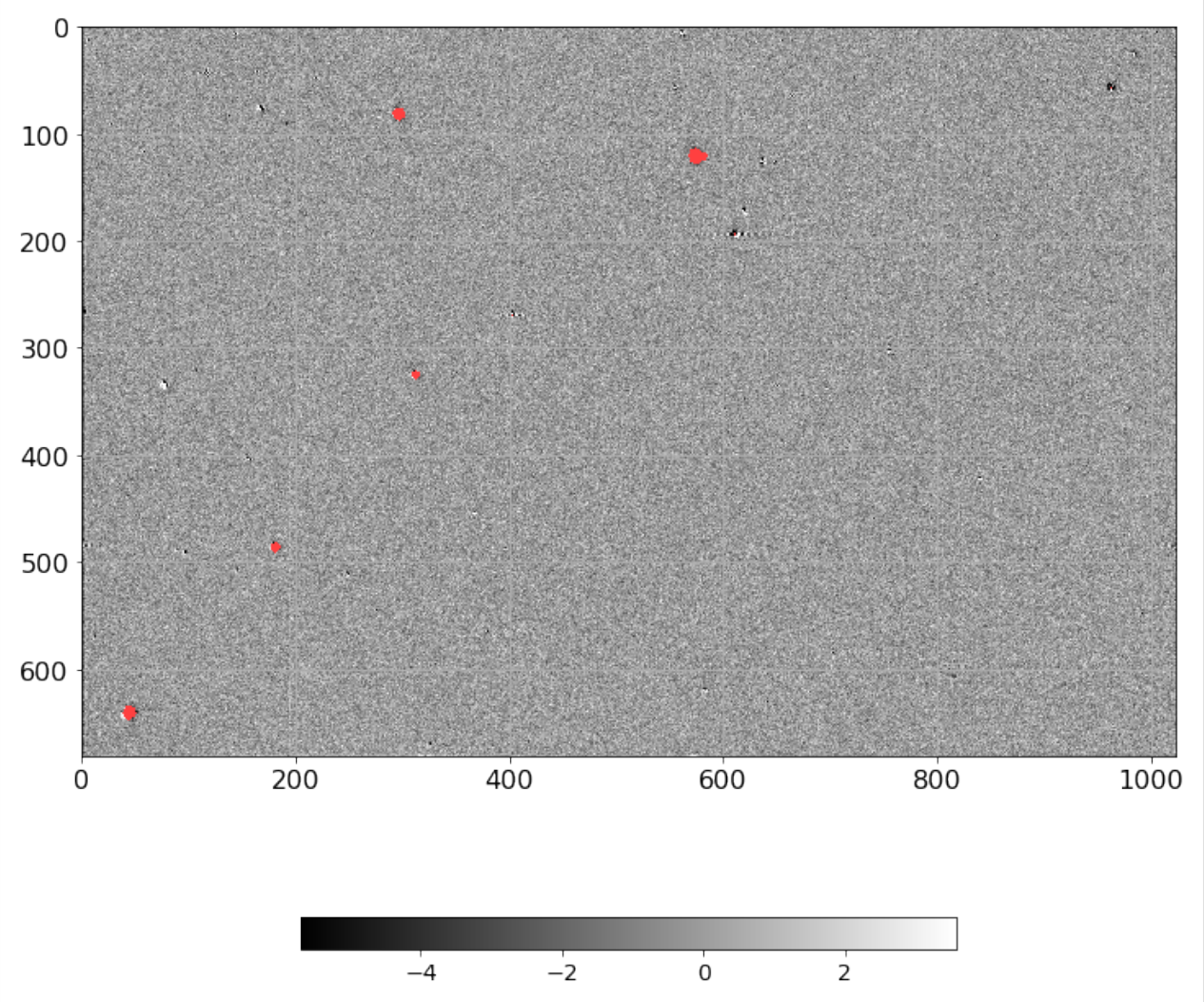
```
updating stamp shape to (21,21)
updating stamp shape to (21,21)
```

```
CPU times: user 39 s, sys: 2.52 s, total: 41.5 s
Wall time: 38.2 s
```

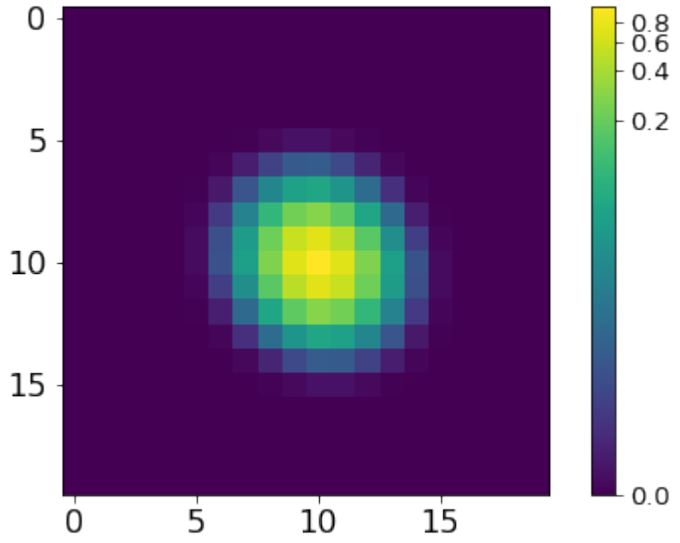
The result is a list of numpy arrays.

The arrays are in order: `D, P, Scorr, mask`

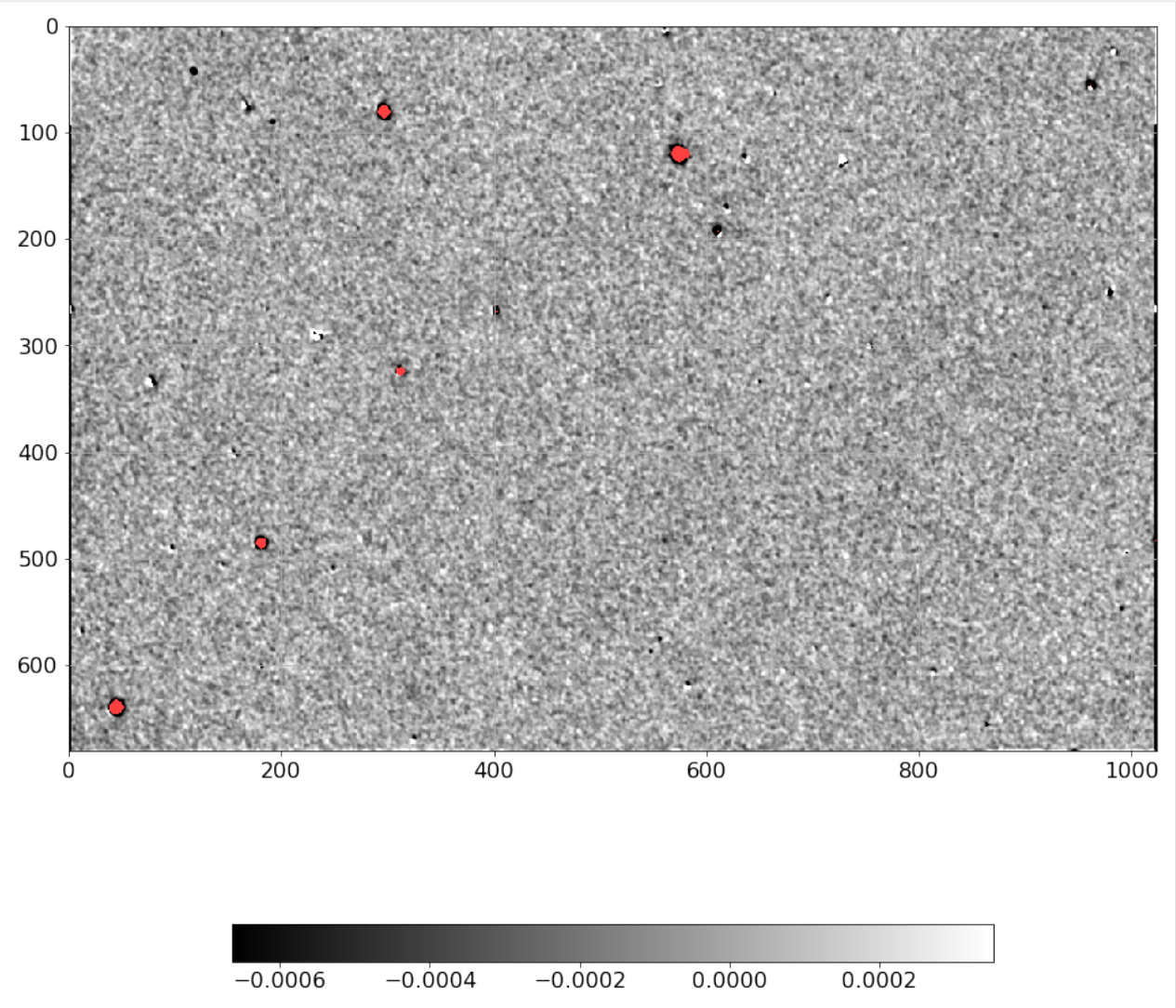
[28]:
```
norm = ImageNormalize(D.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(D.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.grid()
plt.colorbar(orientation='horizontal', shrink=0.6).ax.tick_params(labelsize=14)
```



[29]:
```
norm = ImageNormalize(P, interval=MinMaxInterval(), stretch=LogStretch())
xc, yc = np.where(P==P.max())
xc, yc = np.round(xc[0]), np.round(yc[0])
plt.imshow(P[xc-10:xc+10, yc-10:yc+10], norm=norm,
           cmap='viridis', interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar().ax.tick_params(labelsize=13)
plt.tight_layout()
```

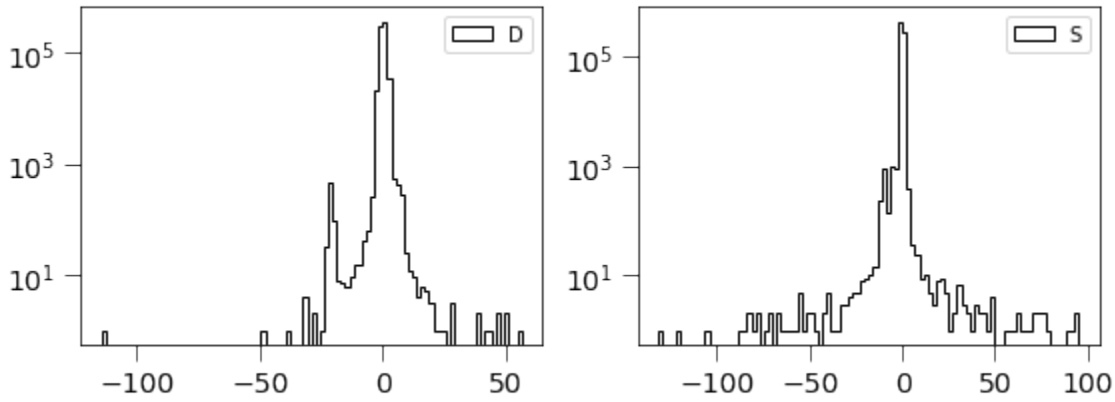


[30]:
```
norm = ImageNormalize(Scorr.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(Scorr.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar(orientation='horizontal', shrink=0.7).ax.tick_params(labelsize=16)
plt.grid()
plt.tight_layout()
```



[31]:
```
dimg = np.ma.MaskedArray(D.real, mask=mask).filled(0).flatten()
simage = np.ma.MaskedArray(Scorr.real, mask=mask).filled(0).flatten()
```

[32]:
```
plt.figure(figsize=(8, 3))

plt.subplot(121)
plt.hist(dimg, log=True, bins=100, histtype='step', label='D', color='k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.subplot(122)
plt.hist(simage/np.std(simage), log=True, bins=100, histtype='step', label='S', color=
↪'k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.tight_layout()
```



Example with `fitted_psf=False` and `beta=True`

[33]:
```
%%time

D, P, Scorr, mask = subtract(
    ref=ref_path,
    new=new_path,
    smooth_psf=False,
    fitted_psf=False,
    align=False,
    iterative=False,
    beta=True,
    shift=True
)
```

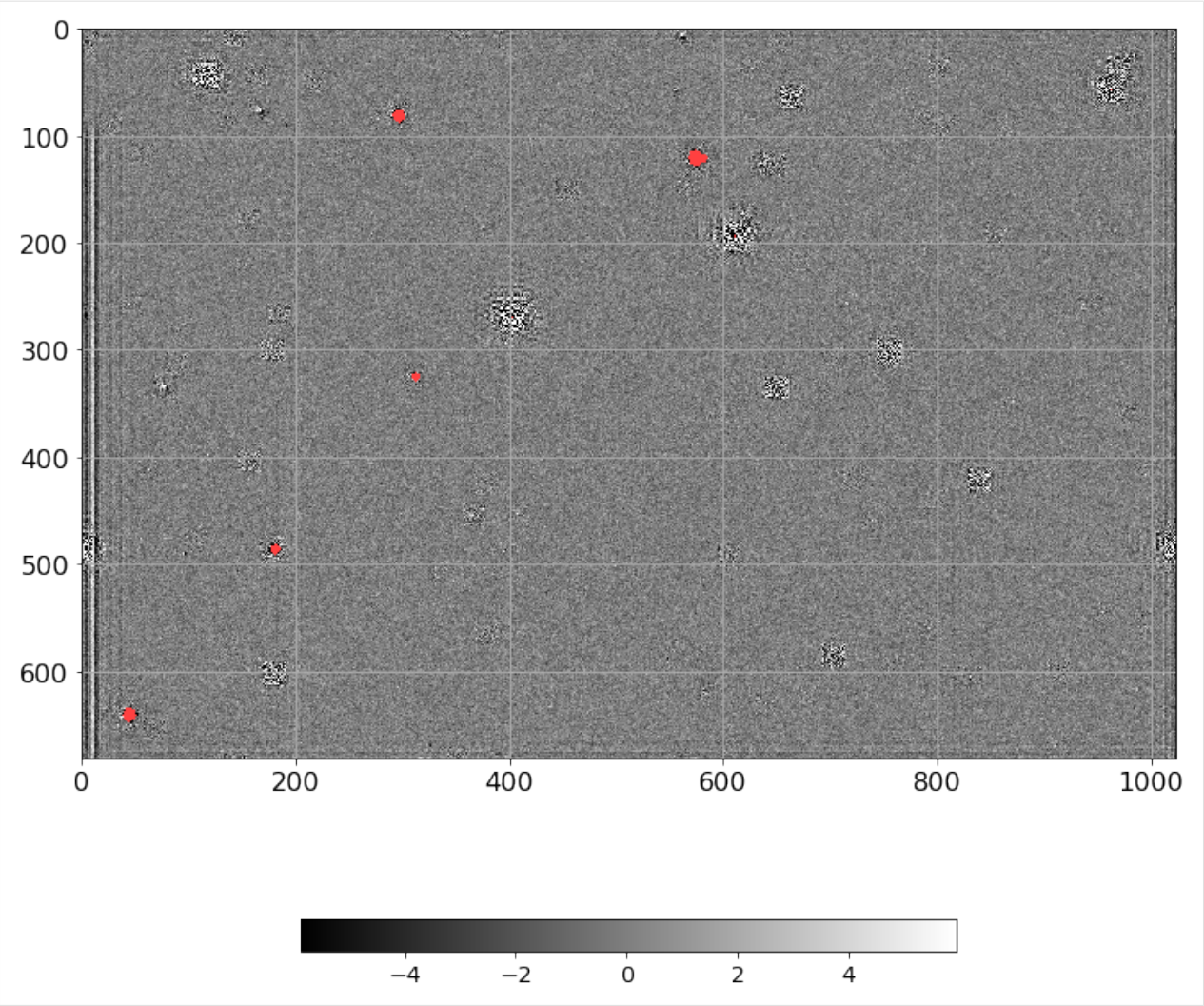
```
updating stamp shape to (21,21)
updating stamp shape to (21,21)
```

```
CPU times: user 37.1 s, sys: 1.48 s, total: 38.6 s
Wall time: 33.9 s
```

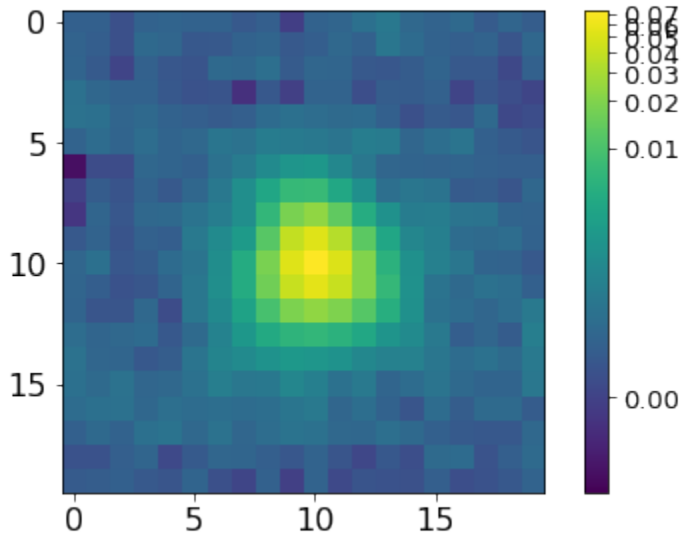
The result is a list of numpy arrays.

The arrays are in order: `D, P, Scorr, mask`

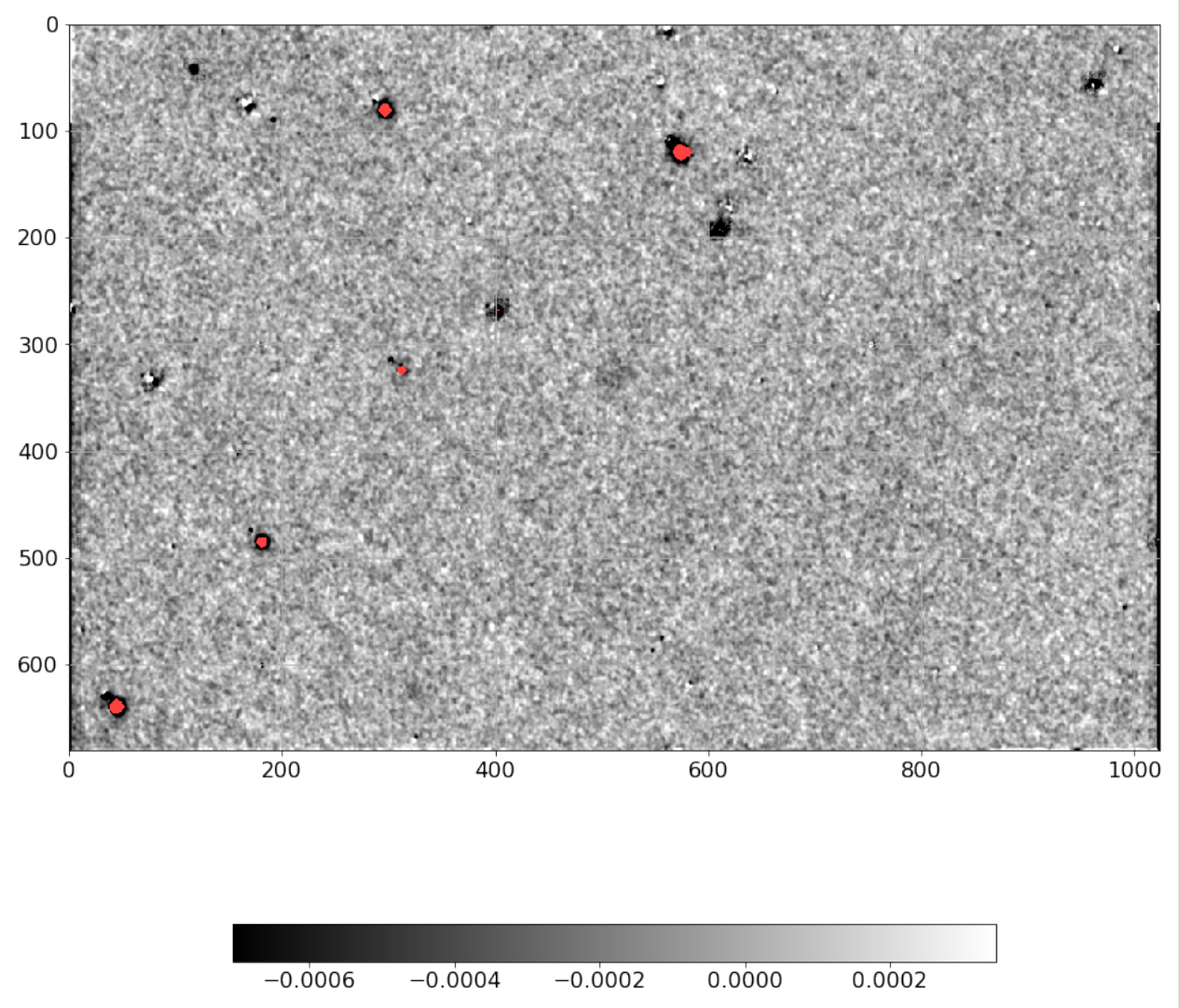
[34]:
```
norm = ImageNormalize(D.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(D.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.grid()
plt.colorbar(orientation='horizontal', shrink=0.6).ax.tick_params(labelsize=14)
```



[35]:
```
norm = ImageNormalize(P, interval=MinMaxInterval(), stretch=LogStretch())
xc, yc = np.where(P==P.max())
xc, yc = np.round(xc[0]), np.round(yc[0])
plt.imshow(P[xc-10:xc+10, yc-10:yc+10], norm=norm,
           cmap='viridis', interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar().ax.tick_params(labelsize=13)
plt.tight_layout()
```

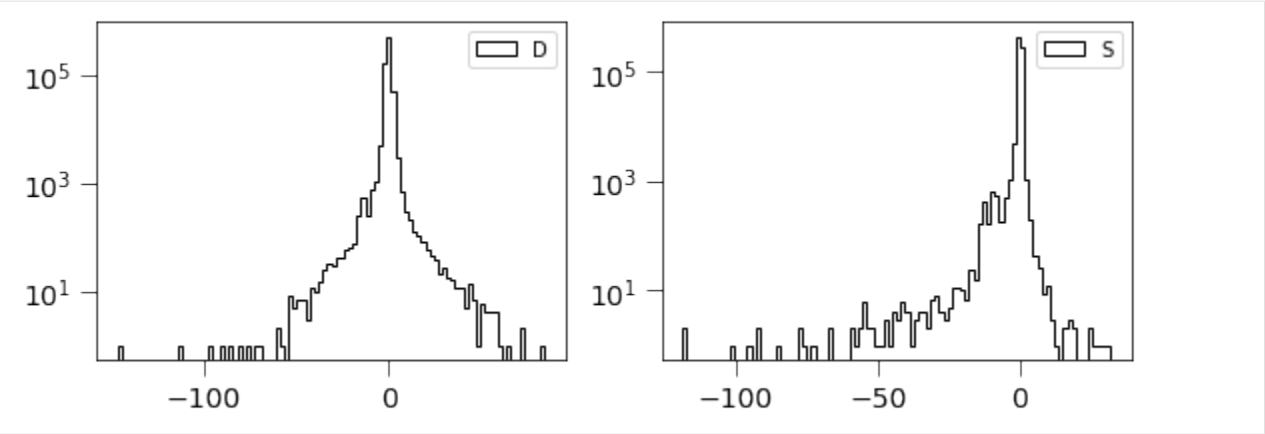


[36]:
```
norm = ImageNormalize(Scorr.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(Scorr.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar(orientation='horizontal', shrink=0.7).ax.tick_params(labelsize=16)
plt.grid()
plt.tight_layout()
```



[37]:
```
dimg = np.ma.MaskedArray(D.real, mask=mask).filled(0).flatten()
simage = np.ma.MaskedArray(Scorr.real, mask=mask).filled(0).flatten()
```

[38]:
```
plt.figure(figsize=(8, 3))

plt.subplot(121)
plt.hist(dimg, log=True, bins=100, histtype='step', label='D', color='k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.subplot(122)
plt.hist(simage/np.std(simage), log=True, bins=100, histtype='step', label='S', color=
↪'k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.tight_layout()
```



Example with `smooth_psf=True` and `fitted_psf=False`, using `beta=True`, `iterative=True`.

[39]:
```
%%time

D, P, Scorr, mask = subtract(
    ref=ref_path,
    new=new_path,
    smooth_psf=True,
    fitted_psf=False,
    align=False,
    iterative=True,
    beta=True,
    shift=True
)
```

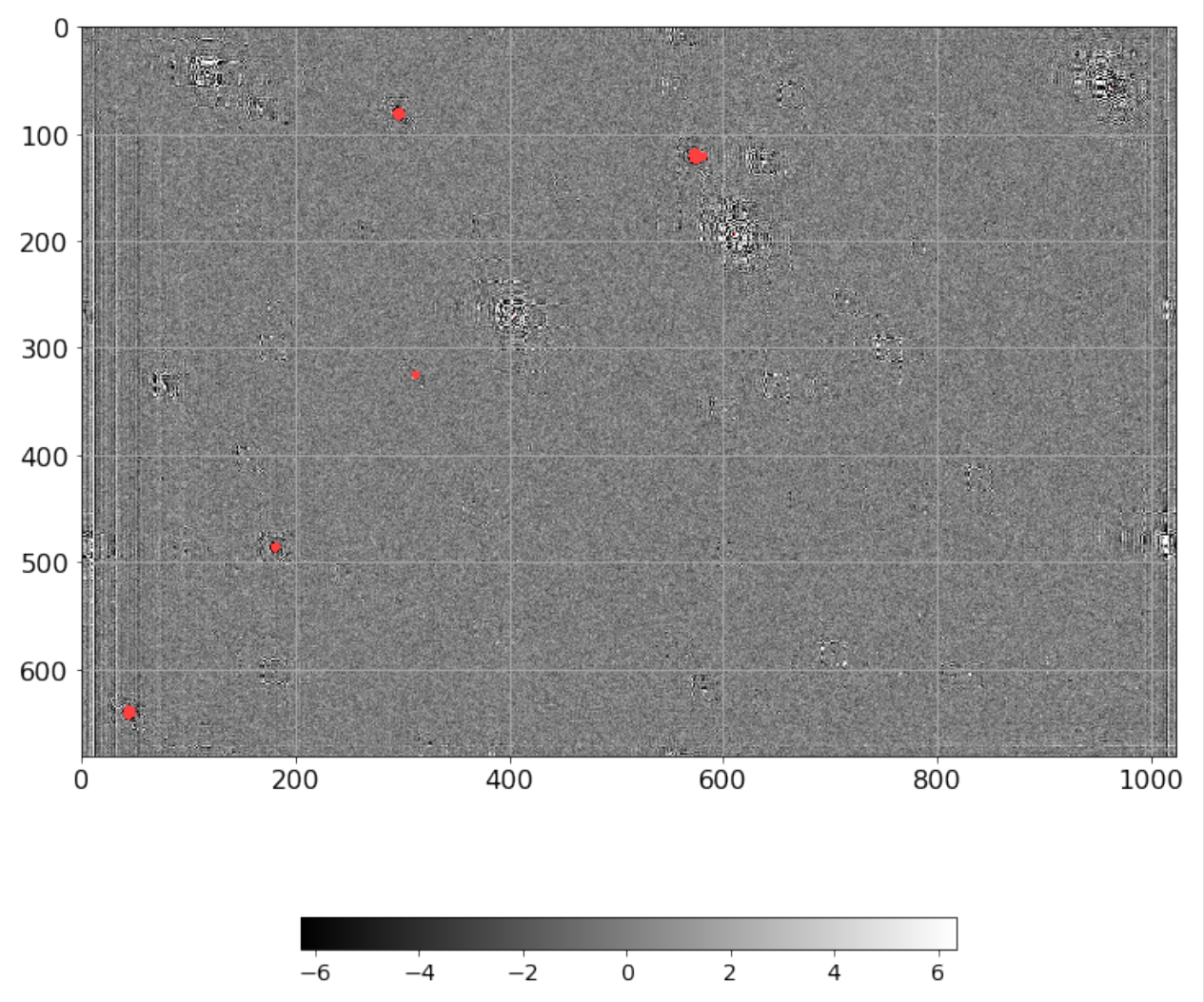
```
updating stamp shape to (21,21)
updating stamp shape to (21,21)
```

```
CPU times: user 28 s, sys: 2.13 s, total: 30.1 s
Wall time: 25.8 s
```

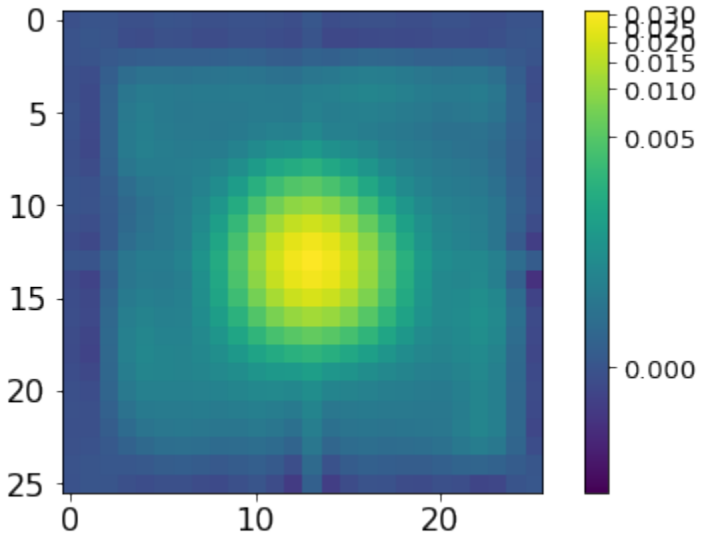
The result is a list of numpy arrays.

The arrays are in order: `D, P, Scorr, mask`

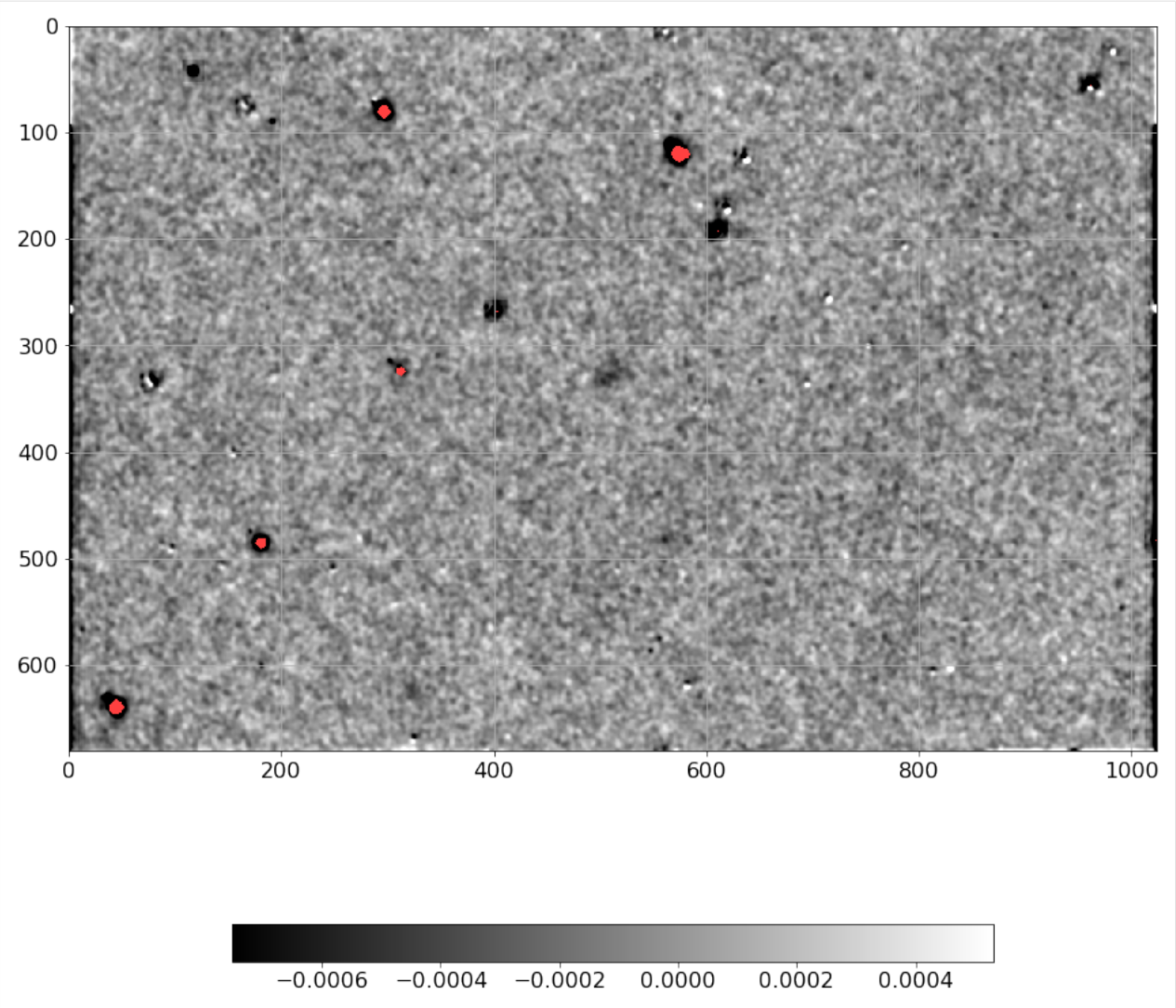
[40]:
```
norm = ImageNormalize(D.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(D.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.grid()
plt.colorbar(orientation='horizontal', shrink=0.6).ax.tick_params(labelsize=14)
```



[41]:
```
norm = ImageNormalize(P, interval=MinMaxInterval(), stretch=LogStretch())
xc, yc = np.where(P==P.max())
xc, yc = np.round(xc[0]), np.round(yc[0])
plt.imshow(P[xc-13:xc+13, yc-13:yc+13], norm=norm,
           cmap='viridis', interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar().ax.tick_params(labelsize=13)
plt.tight_layout()
```

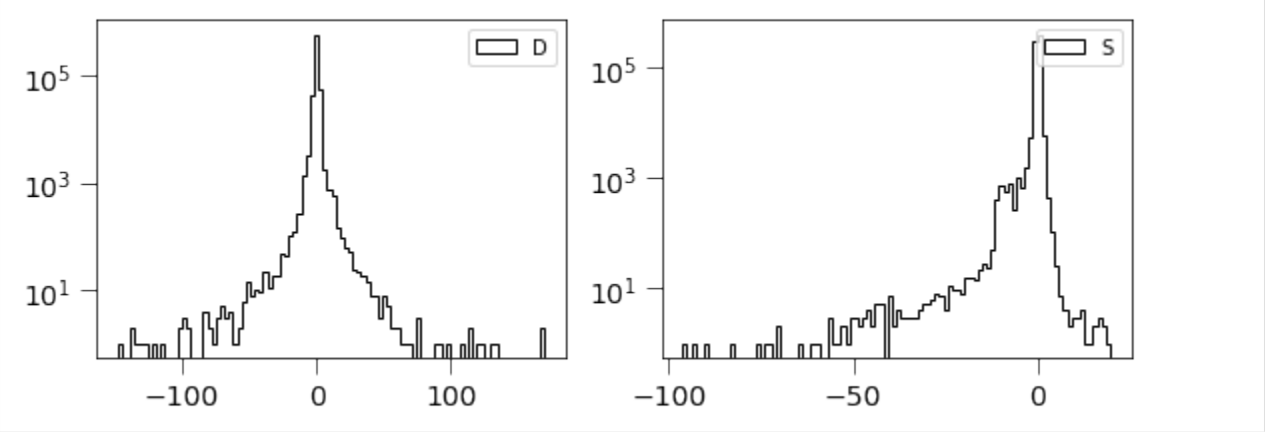


[42]:
```
norm = ImageNormalize(Scorr.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(Scorr.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar(orientation='horizontal', shrink=0.7).ax.tick_params(labelsize=16)
plt.grid()
plt.tight_layout()
```



[43]:
```
dimg = np.ma.MaskedArray(D.real, mask=mask).filled(0).flatten()
simage = np.ma.MaskedArray(Scorr.real, mask=mask).filled(0).flatten()
```

[44]:
```
plt.figure(figsize=(8, 3))

plt.subplot(121)
plt.hist(dimg, log=True, bins=100, histtype='step', label='D', color='k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.subplot(122)
plt.hist(simage/np.std(simage), log=True, bins=100, histtype='step', label='S', color=
↪'k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.tight_layout()
```



Example with `smooth_psf=True` and `fitted_psf=False`, using `beta=False`, `iterative=False`.

[45]:
```
%%time

D, P, Scorr, mask = subtract(
    ref=ref_path,
    new=new_path,
    smooth_psf=True,
    fitted_psf=False,
    align=False,
    iterative=False,
    beta=False,
    shift=True
)
```

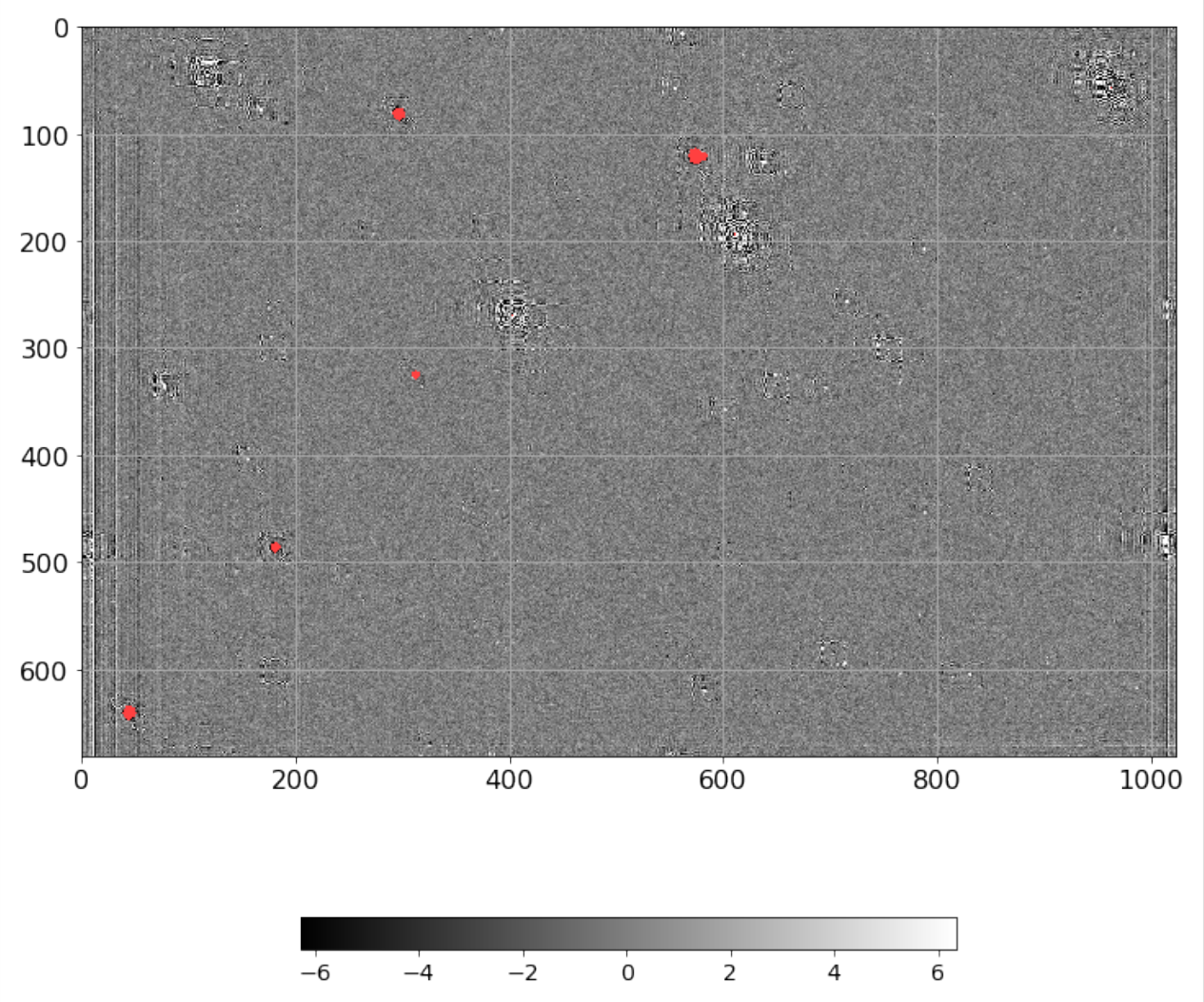
```
updating stamp shape to (21,21)
updating stamp shape to (21,21)
```

```
CPU times: user 20.7 s, sys: 1.83 s, total: 22.5 s
Wall time: 17 s
```

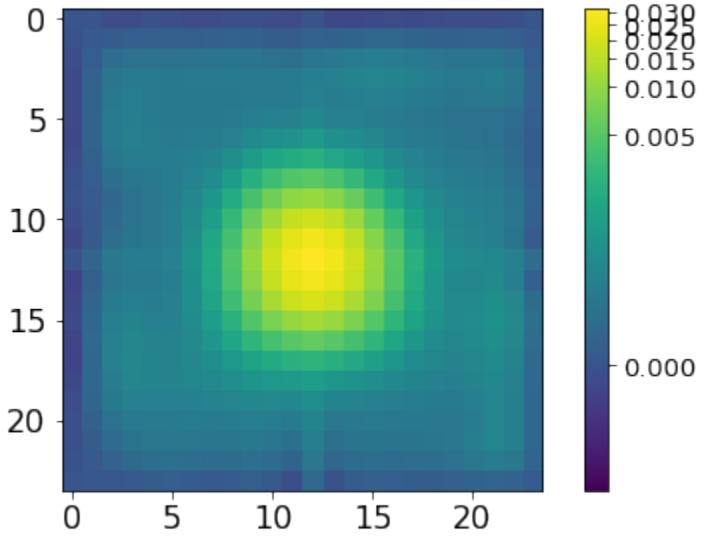
The result is a list of numpy arrays.

The arrays are in order: `D, P, Scorr, mask`

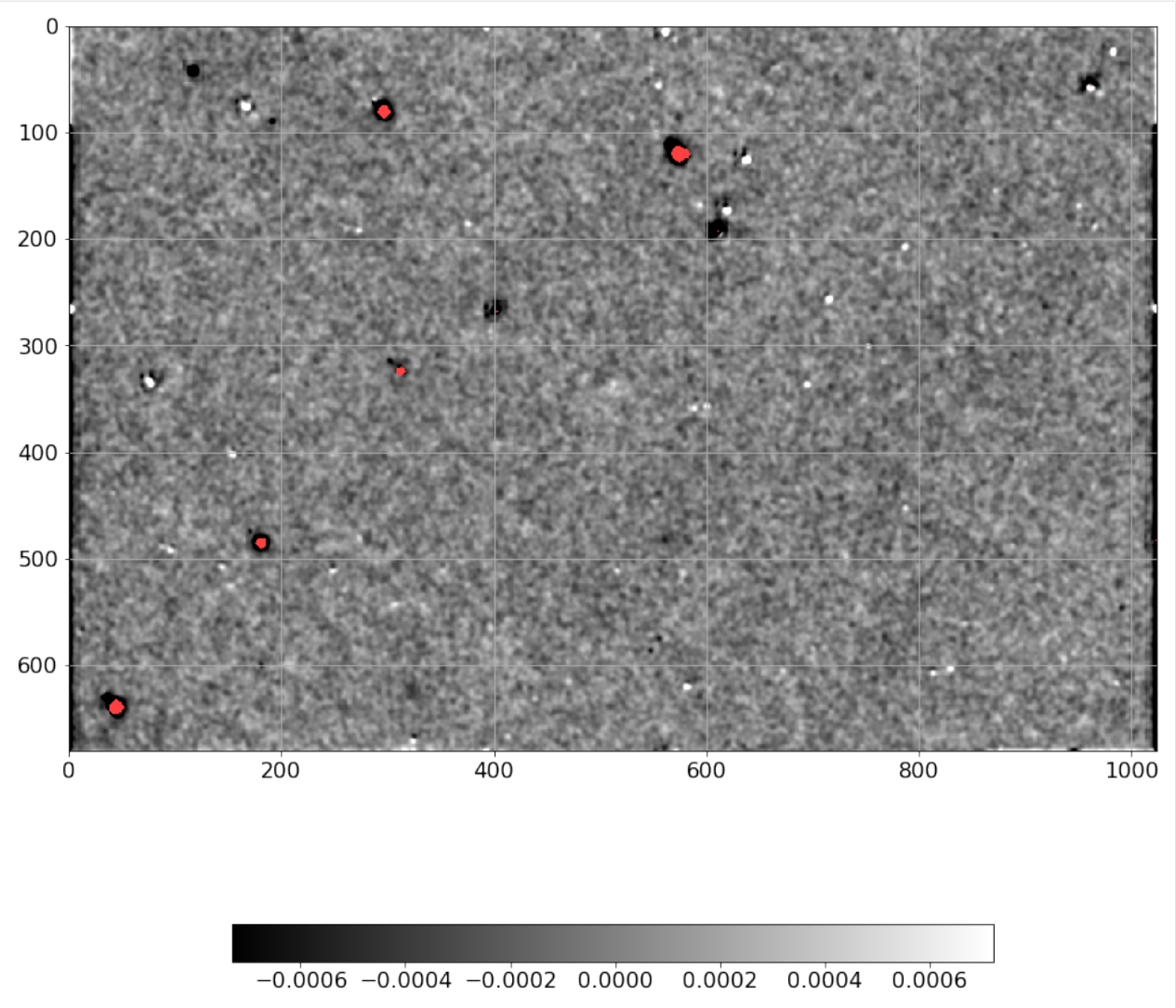
[46]:
```
norm = ImageNormalize(D.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(D.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.grid()
plt.colorbar(orientation='horizontal', shrink=0.6).ax.tick_params(labelsize=14)
```



[47]:
```
norm = ImageNormalize(P, interval=MinMaxInterval(), stretch=LogStretch())
xc, yc = np.where(P==P.max())
xc, yc = np.round(xc[0]), np.round(yc[0])
plt.imshow(P[xc-12:xc+12, yc-12:yc+12], norm=norm,
           cmap='viridis', interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar().ax.tick_params(labelsize=13)
plt.tight_layout()
```

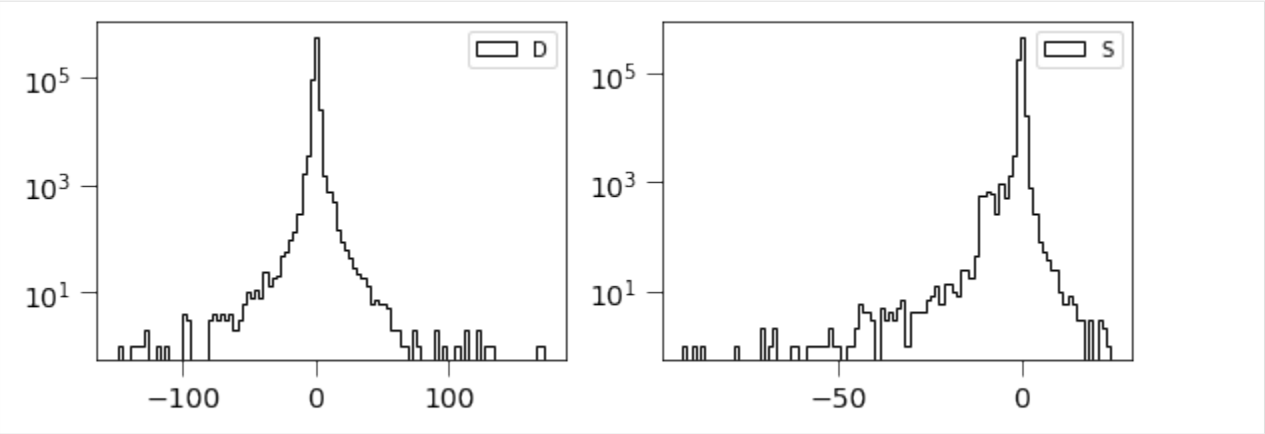


[48]:
```
norm = ImageNormalize(Scorr.real, interval=ZScaleInterval(),
                      stretch=LinearStretch())

plt.figure(figsize=(12, 12))
plt.imshow(np.ma.MaskedArray(Scorr.real, mask=mask),
           cmap=palette, norm=norm, interpolation='none')
plt.tick_params(labelsize=16)
plt.colorbar(orientation='horizontal', shrink=0.7).ax.tick_params(labelsize=16)
plt.grid()
plt.tight_layout()
```



[49]:
```
dimg = np.ma.MaskedArray(D.real, mask=mask).filled(0).flatten()
simage = np.ma.MaskedArray(Scorr.real, mask=mask).filled(0).flatten()
```

[50]:
```
plt.figure(figsize=(8, 3))

plt.subplot(121)
plt.hist(dimg, log=True, bins=100, histtype='step', label='D', color='k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.subplot(122)
plt.hist(simage/np.std(simage), log=True, bins=100, histtype='step', label='S', color=
↪'k')
plt.legend(loc='best')
plt.tick_params(labelsize=14)
plt.tick_params(size=8)

plt.tight_layout()
```



3.2 Glossary

There are some terminology which is used through thte documentation, and it is strictly related to astronomical image processing techniques, and implementation details.

source A light source, often used to name stars and galaxies in images.

covariance matrix The matrix C formed such as $C_{ij} =< o_i, o_j >$, where $<,>$ is the inner product operator, and each o_i is an observation.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

4.1 References

Bibliography

[Zackay2016] http://adsabs.harvard.edu/abs/2016ApJ...830...27Z

[Zackay2017a] http://adsabs.harvard.edu/abs/2017ApJ...836..187Z

[Zackay2017b] http://adsabs.harvard.edu/abs/2017ApJ...836..188Z

[Lauer2002] http://adsabs.harvard.edu/abs/2002SPIE.4847..167L

Index

C

S