symbolic_developments

August 13, 2020

1 Symbolic developments

One of the major advantages of the kernuller package is that the combiner matrices are managed in a symbolic format.

While it is not case for default propagation when using `kernuller.get_I()` method, this symbolic format enables to manipulate the symbolic representation, not only of the combiner matrix, but also of the propagated light.

Depending on the goal, one can develop the equations corresponding to the requested type of propagation including for axample different types of aberrations, or the location of the apertures.

[1]:
```
import numpy as np
import sympy as sp
import matplotlib.pyplot as plt
#import kernuller_class as kernuller
import kernuller
import astropy.coordinates
import astropy.units as u

from time import time

# ============================================================================
# ============================================================================
def mas2rad(x):
    ''' Convenient little function to convert milliarcsec to radians '''
    return(x * 4.8481368110953599e-09) # = x*np.pi/(180*3600*1000)

# ============================================================================
# ============================================================================
def rad2mas(x):
    '''  convert radians to mas'''
    return(x / 4.8481368110953599e-09) # = x / (np.pi/(180*3600*1000))

# ============================================================================
# ============================================================================
```

```
Building a model from scratch
```

[114]:

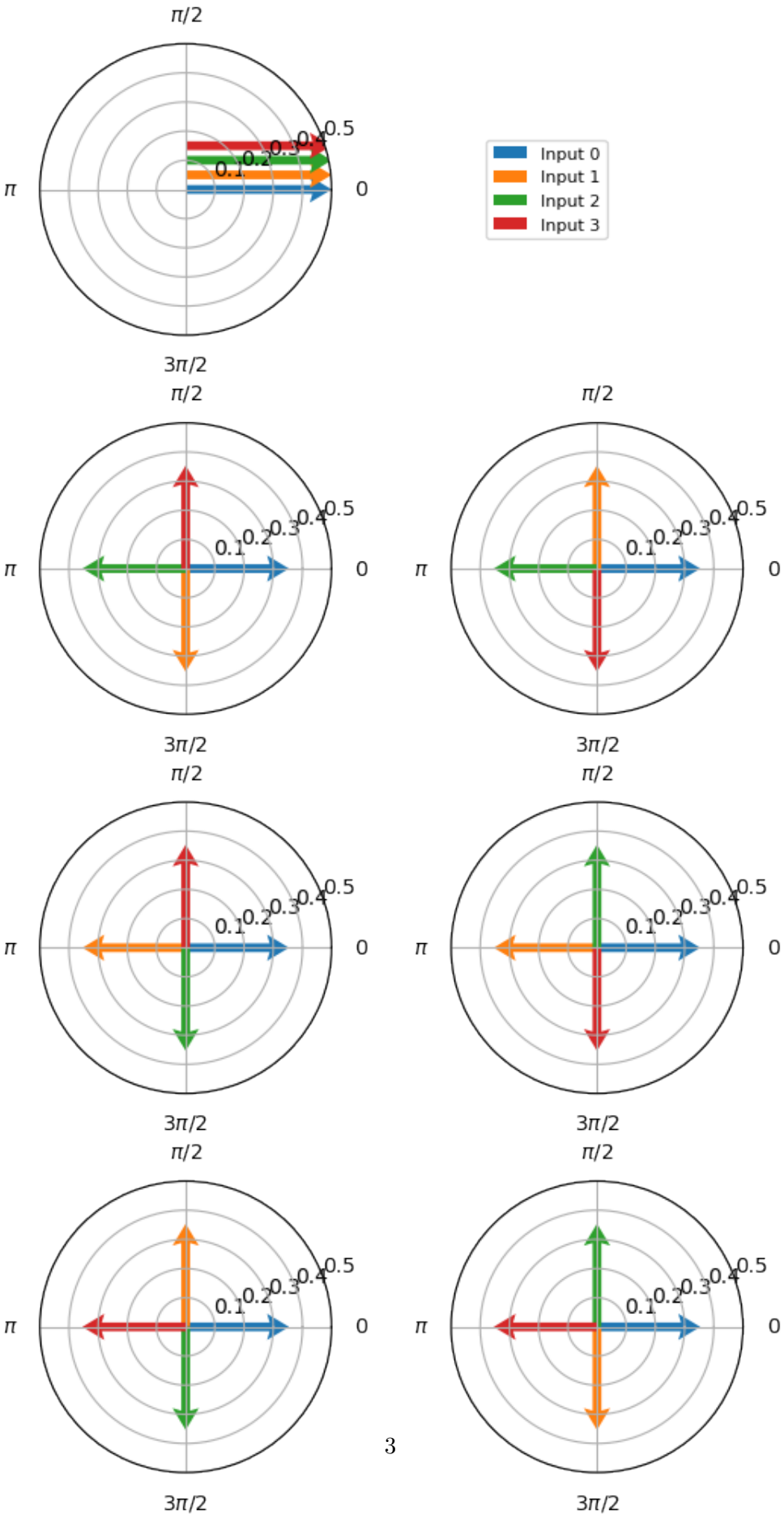
1.1 The first step is to build a model

[4]:
```
mymatrix = np.load("../data/4T_matrix.npy", allow_pickle=True)
mykernuller = kernuller.kernuller(kernuller.VLTI,5.e-6)
mykernuller.build_model_from_matrix(mymatrix)
mykernuller.Ks = sp.Matrix(kernuller.pairwise_kernel(6, blankcols=1))
```

```
Building a model from scratch
```

[74]:
```
Mn = np.array(sp.N(mykernuller.Ms), dtype=np.complex64)
Mn2 = np.vstack((Mn[:1,:], np.zeros_like(Mn[1,:]), Mn[1:,:]))
fig, axs = mykernuller.plot_outputs_smart(Mn2, nx=2,legendoffset=(1.5,0.5),␣
 ↪dpi=100,
                                          plotsize=3, osfrac=0.1, title=False,␣
 ↪mainlinewidth=0.03,
                                          labelsize=10,legendstring="center␣
 ↪left", outputontop=True,
                                          labels=False, onlyoneticklabel=False)
```

```
/home/rlaugier/opt/miniconda2/envs/p3-7/lib/python3.7/site-
packages/matplotlib/transforms.py:923: ComplexWarning: Casting complex values to
real discards the imaginary part
  self._points[:, 1] = interval
/home/rlaugier/Documents/kernel/kernuller/kernuller/kernuller.py:1409:
UserWarning: This figure includes Axes that are not compatible with
tight_layout, so results might be incorrect.
  fig.tight_layout()
removing the labels: [False False False False False False False False]
```


π/2
π
0
3π/2
0.1
0.2
0.3
0.4
0.5
Input 0
Input 1
Input 2
Input 3

1.2 Building the relevant equation

1.2.1 Position of the apertures

[8]:
```
X = sp.Matrix(sp.symbols('X:{}'.format(mykernuller.Na+1), real=True))
Y = sp.Matrix(sp.symbols('Y:{}'.format(mykernuller.Na+1), real=True))
```

1.3 Complex amplitude of light sampled by this array

We need λ (wavelength), α and β (sky position relative to optical axis) ζ (an amplitude term for each input) γ (an input phase error term we didn't use here).

[14]:
```
lamb = sp.symbols("lambda", real=True)
alpha = sp.symbols("alpha", real=True)
beta = sp.symbols("beta", real=True)
zeta = sp.Matrix(sp.symbols('zeta:{}'.format(mykernuller.Na), real=True))
gamma = sp.Matrix(sp.symbols('gamma:{}'.format(mykernuller.Na), real=True))
```

[18]:
```
zeta
```

[18]:
$$\begin{bmatrix}\zeta_0\\ \zeta_1\\ \zeta_2\\ \zeta_3\end{bmatrix}$$

For now they are just symbols, but now we can use it in an equation.

[20]:
```
z = sp.Matrix([zeta[i]*sp.exp(sp.I*(2*sp.pi/lamb)*(alpha*X[i] + beta*Y[i])) for␣
↪i in range(mykernuller.Na)])
z
```

[20]:
$$\begin{bmatrix}\zeta_0 e^{\frac{2i\pi(X_0\alpha+Y_0\beta)}{\lambda}}\\ \zeta_1 e^{\frac{2i\pi(X_1\alpha+Y_1\beta)}{\lambda}}\\ \zeta_2 e^{\frac{2i\pi(X_2\alpha+Y_2\beta)}{\lambda}}\\ \zeta_3 e^{\frac{2i\pi(X_3\alpha+Y_3\beta)}{\lambda}}\end{bmatrix}$$

1.4 The equation for the output electric field

Very easy, you just have to multiply it by the combiner matrix

[22]:
```
anE = mykernuller.Ms@z
anE
```

[22]:
$$\begin{bmatrix} \frac{\zeta_0 e^{\frac{2i\pi(X_0\alpha+Y_0\beta)}{\lambda}}}{2} + \frac{\zeta_1 e^{\frac{2i\pi(X_1\alpha+Y_1\beta)}{\lambda}}}{2} + \frac{\zeta_2 e^{\frac{2i\pi(X_2\alpha+Y_2\beta)}{\lambda}}}{2} + \frac{\zeta_3 e^{\frac{2i\pi(X_3\alpha+Y_3\beta)}{\lambda}}}{2} \\ \frac{\sqrt{2}\zeta_0 e^{\frac{2i\pi(X_0\alpha+Y_0\beta)}{\lambda}}}{4} - \frac{\sqrt{2}i\zeta_1 e^{\frac{2i\pi(X_1\alpha+Y_1\beta)}{\lambda}}}{4} - \frac{\sqrt{2}\zeta_2 e^{\frac{2i\pi(X_2\alpha+Y_2\beta)}{\lambda}}}{4} + \frac{\sqrt{2}i\zeta_3 e^{\frac{2i\pi(X_3\alpha+Y_3\beta)}{\lambda}}}{4} \\ \frac{\sqrt{2}\zeta_0 e^{\frac{2i\pi(X_0\alpha+Y_0\beta)}{\lambda}}}{4} + \frac{\sqrt{2}i\zeta_1 e^{\frac{2i\pi(X_1\alpha+Y_1\beta)}{\lambda}}}{4} - \frac{\sqrt{2}\zeta_2 e^{\frac{2i\pi(X_2\alpha+Y_2\beta)}{\lambda}}}{4} - \frac{\sqrt{2}i\zeta_3 e^{\frac{2i\pi(X_3\alpha+Y_3\beta)}{\lambda}}}{4} \\ \frac{\sqrt{2}\zeta_0 e^{\frac{2i\pi(X_0\alpha+Y_0\beta)}{\lambda}}}{4} - \frac{\sqrt{2}\zeta_1 e^{\frac{2i\pi(X_1\alpha+Y_1\beta)}{\lambda}}}{4} - \frac{\sqrt{2}i\zeta_2 e^{\frac{2i\pi(X_2\alpha+Y_2\beta)}{\lambda}}}{4} + \frac{\sqrt{2}i\zeta_3 e^{\frac{2i\pi(X_3\alpha+Y_3\beta)}{\lambda}}}{4} \\ \frac{\sqrt{2}\zeta_0 e^{\frac{2i\pi(X_0\alpha+Y_0\beta)}{\lambda}}}{4} - \frac{\sqrt{2}\zeta_1 e^{\frac{2i\pi(X_1\alpha+Y_1\beta)}{\lambda}}}{4} + \frac{\sqrt{2}i\zeta_2 e^{\frac{2i\pi(X_2\alpha+Y_2\beta)}{\lambda}}}{4} - \frac{\sqrt{2}i\zeta_3 e^{\frac{2i\pi(X_3\alpha+Y_3\beta)}{\lambda}}}{4} \\ \frac{\sqrt{2}\zeta_0 e^{\frac{2i\pi(X_0\alpha+Y_0\beta)}{\lambda}}}{4} + \frac{\sqrt{2}i\zeta_1 e^{\frac{2i\pi(X_1\alpha+Y_1\beta)}{\lambda}}}{4} - \frac{\sqrt{2}\zeta_2 e^{\frac{2i\pi(X_2\alpha+Y_2\beta)}{\lambda}}}{4} - \frac{\sqrt{2}\zeta_3 e^{\frac{2i\pi(X_3\alpha+Y_3\beta)}{\lambda}}}{4} \\ \frac{\sqrt{2}\zeta_0 e^{\frac{2i\pi(X_0\alpha+Y_0\beta)}{\lambda}}}{4} - \frac{\sqrt{2}i\zeta_1 e^{\frac{2i\pi(X_1\alpha+Y_1\beta)}{\lambda}}}{4} + \frac{\sqrt{2}i\zeta_2 e^{\frac{2i\pi(X_2\alpha+Y_2\beta)}{\lambda}}}{4} - \frac{\sqrt{2}\zeta_3 e^{\frac{2i\pi(X_3\alpha+Y_3\beta)}{\lambda}}}{4} \end{bmatrix}$$

One might prefer it under a different form:

1.5 Preparing substitutions

Now we want to replace the constants by the value they will take. This can be prepared in a list. For now let us only replace ζ and look at the result:

[30]:
```
thesubs = []
for i in range(mykernuller.Na):
    #thesubs.append((X[i],X[i]-X[0]))
    #thesubs.append((Y[i],Y[i]-Y[0]))
    thesubs.append((zeta[i], 1))
#thesubs.append((X[0],0))
#thesubs.append((Y[0],0))
```

[31]:
```
anE2 = sp.expand_mul(anE.subs(thesubs))
anE2
```

[31]:
$$\begin{bmatrix} \frac{e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{2} + \frac{e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{2} + \frac{e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{2} + \frac{e^{\frac{2i\pi X_3\alpha}{\lambda}+\frac{2i\pi Y_3\beta}{\lambda}}}{2} \\ \frac{\sqrt{2}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{4} - \frac{\sqrt{2}ie^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{4} - \frac{\sqrt{2}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{4} + \frac{\sqrt{2}ie^{\frac{2i\pi X_3\alpha}{\lambda}+\frac{2i\pi Y_3\beta}{\lambda}}}{4} \\ \frac{\sqrt{2}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{4} + \frac{\sqrt{2}ie^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{4} - \frac{\sqrt{2}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{4} - \frac{\sqrt{2}ie^{\frac{2i\pi X_3\alpha}{\lambda}+\frac{2i\pi Y_3\beta}{\lambda}}}{4} \\ \frac{\sqrt{2}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{4} - \frac{\sqrt{2}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{4} - \frac{\sqrt{2}ie^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{4} + \frac{\sqrt{2}ie^{\frac{2i\pi X_3\alpha}{\lambda}+\frac{2i\pi Y_3\beta}{\lambda}}}{4} \\ \frac{\sqrt{2}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{4} - \frac{\sqrt{2}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{4} + \frac{\sqrt{2}ie^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{4} - \frac{\sqrt{2}ie^{\frac{2i\pi X_3\alpha}{\lambda}+\frac{2i\pi Y_3\beta}{\lambda}}}{4} \\ \frac{\sqrt{2}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{4} + \frac{\sqrt{2}ie^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{4} - \frac{\sqrt{2}ie^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{4} - \frac{\sqrt{2}e^{\frac{2i\pi X_3\alpha}{\lambda}+\frac{2i\pi Y_3\beta}{\lambda}}}{4} \\ \frac{\sqrt{2}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{4} - \frac{\sqrt{2}ie^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{4} + \frac{\sqrt{2}ie^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{4} - \frac{\sqrt{2}e^{\frac{2i\pi X_3\alpha}{\lambda}+\frac{2i\pi Y_3\beta}{\lambda}}}{4} \end{bmatrix}$$

1.6 Computing the output intensities

[32]:
```
anI =  sp.matrix_multiply_elementwise(sp.conjugate(anE2), anE2)
anI = sp.expand_mul(anI)
anI
```

[32]:
$$\begin{bmatrix} \frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{4}+\frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{4}+\frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{4}+\frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}}{4} \\ \frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{8}-\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{8}-\frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{8}+\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}}{8} \\ \frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{8}+\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{8}-\frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{8}-\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}}{8} \\ \frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{8}-\frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{8}-\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{8}+\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}}{8} \\ \frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{8}+\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{8}-\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{8}-\frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}}{8} \\ \frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}}{8}-\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}}{8}+\frac{ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}}{8}-\frac{e^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}}{8} \end{bmatrix}$$

1.7 Now let us look at the kernel vector

[34]:
```
kappa = mykernuller.Ks@anI
kappa
```

[34]:
$$\begin{bmatrix} -0.25ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}+0.25ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_3\alpha}{\lambda}+\frac{2i\pi Y_3\beta}{\lambda}}+0.25ie^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}e^{-\frac{2i\pi X_1\alpha}{\lambda}-\frac{2i\pi Y_1\beta}{\lambda}}- \\ -0.25ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}+0.25ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_3\alpha}{\lambda}+\frac{2i\pi Y_3\beta}{\lambda}}+0.25ie^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}e^{-\frac{2i\pi X_2\alpha}{\lambda}-\frac{2i\pi Y_2\beta}{\lambda}}- \\ 0.25ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_1\alpha}{\lambda}+\frac{2i\pi Y_1\beta}{\lambda}}-0.25ie^{-\frac{2i\pi X_0\alpha}{\lambda}-\frac{2i\pi Y_0\beta}{\lambda}}e^{\frac{2i\pi X_2\alpha}{\lambda}+\frac{2i\pi Y_2\beta}{\lambda}}-0.25ie^{\frac{2i\pi X_0\alpha}{\lambda}+\frac{2i\pi Y_0\beta}{\lambda}}e^{-\frac{2i\pi X_1\alpha}{\lambda}-\frac{2i\pi Y_1\beta}{\lambda}}+ \end{bmatrix}$$

1.8 Finer examination of a single kernel

The sympy method expand_complex() is helpful to show the trigonometric form of the output (We knew this had to be real!)

[36]:
```
aker = anI[1]-anI[2]
sinform = sp.expand_complex(aker)
sinform
```

[36]:
$$-\frac{\sin\left(\frac{2\pi X_0\alpha}{\lambda}+\frac{2\pi Y_0\beta}{\lambda}\right)\cos\left(\frac{2\pi X_1\alpha}{\lambda}+\frac{2\pi Y_1\beta}{\lambda}\right)}{2}+\frac{\sin\left(\frac{2\pi X_0\alpha}{\lambda}+\frac{2\pi Y_0\beta}{\lambda}\right)\cos\left(\frac{2\pi X_3\alpha}{\lambda}+\frac{2\pi Y_3\beta}{\lambda}\right)}{2}+ \\ \frac{\sin\left(\frac{2\pi X_1\alpha}{\lambda}+\frac{2\pi Y_1\beta}{\lambda}\right)\cos\left(\frac{2\pi X_0\alpha}{\lambda}+\frac{2\pi Y_0\beta}{\lambda}\right)}{2}-\frac{\sin\left(\frac{2\pi X_1\alpha}{\lambda}+\frac{2\pi Y_1\beta}{\lambda}\right)\cos\left(\frac{2\pi X_2\alpha}{\lambda}+\frac{2\pi Y_2\beta}{\lambda}\right)}{2}+ \\ \frac{\sin\left(\frac{2\pi X_2\alpha}{\lambda}+\frac{2\pi Y_2\beta}{\lambda}\right)\cos\left(\frac{2\pi X_1\alpha}{\lambda}+\frac{2\pi Y_1\beta}{\lambda}\right)}{2}-\frac{\sin\left(\frac{2\pi X_2\alpha}{\lambda}+\frac{2\pi Y_2\beta}{\lambda}\right)\cos\left(\frac{2\pi X_3\alpha}{\lambda}+\frac{2\pi Y_3\beta}{\lambda}\right)}{2}- \\ \frac{\sin\left(\frac{2\pi X_3\alpha}{\lambda}+\frac{2\pi Y_3\beta}{\lambda}\right)\cos\left(\frac{2\pi X_0\alpha}{\lambda}+\frac{2\pi Y_0\beta}{\lambda}\right)}{2}+\frac{\sin\left(\frac{2\pi X_3\alpha}{\lambda}+\frac{2\pi Y_3\beta}{\lambda}\right)\cos\left(\frac{2\pi X_2\alpha}{\lambda}+\frac{2\pi Y_2\beta}{\lambda}\right)}{2}$$

Of course, you can also print it out in latex source code with the kernuller.printlatex() method that is just repackaging the method from sympy (mostly to get the j symbol for complex unit.

[98]:
```
kernuller.printlatex(sinform)
```

```
\begin{equation}
  - \frac{\sin{\left(\frac{2 \pi X_{0} \alpha}{\lambda} + \frac{2 \pi Y_{0}
\beta}{\lambda} \right)} \cos{\left(\frac{2 \pi X_{1} \alpha}{\lambda} + \frac{2
\pi Y_{1} \beta}{\lambda} \right)}}{2} + \frac{\sin{\left(\frac{2 \pi X_{0}
\alpha}{\lambda} + \frac{2 \pi Y_{0} \beta}{\lambda} \right)} \cos{\left(\frac{2
\pi X_{3} \alpha}{\lambda} + \frac{2 \pi Y_{3} \beta}{\lambda} \right)}}{2} +
\frac{\sin{\left(\frac{2 \pi X_{1} \alpha}{\lambda} + \frac{2 \pi Y_{1}
\beta}{\lambda} \right)} \cos{\left(\frac{2 \pi X_{0} \alpha}{\lambda} + \frac{2
\pi Y_{0} \beta}{\lambda} \right)}}{2} - \frac{\sin{\left(\frac{2 \pi X_{1}
\alpha}{\lambda} + \frac{2 \pi Y_{1} \beta}{\lambda} \right)} \cos{\left(\frac{2
\pi X_{2} \alpha}{\lambda} + \frac{2 \pi Y_{2} \beta}{\lambda} \right)}}{2} +
\frac{\sin{\left(\frac{2 \pi X_{2} \alpha}{\lambda} + \frac{2 \pi Y_{2}
\beta}{\lambda} \right)} \cos{\left(\frac{2 \pi X_{1} \alpha}{\lambda} + \frac{2
\pi Y_{1} \beta}{\lambda} \right)}}{2} - \frac{\sin{\left(\frac{2 \pi X_{2}
\alpha}{\lambda} + \frac{2 \pi Y_{2} \beta}{\lambda} \right)} \cos{\left(\frac{2
\pi X_{3} \alpha}{\lambda} + \frac{2 \pi Y_{3} \beta}{\lambda} \right)}}{2} -
\frac{\sin{\left(\frac{2 \pi X_{3} \alpha}{\lambda} + \frac{2 \pi Y_{3}
\beta}{\lambda} \right)} \cos{\left(\frac{2 \pi X_{0} \alpha}{\lambda} + \frac{2
\pi Y_{0} \beta}{\lambda} \right)}}{2} + \frac{\sin{\left(\frac{2 \pi X_{3}
\alpha}{\lambda} + \frac{2 \pi Y_{3} \beta}{\lambda} \right)} \cos{\left(\frac{2
\pi X_{2} \alpha}{\lambda} + \frac{2 \pi Y_{2} \beta}{\lambda} \right)}}{2}
\end{equation}
```

1.9 Perspectives:

From there, it becomes really easy to extract * derivatives of those expressions sp.diff(expr, x) * taylor series expansions sp.series(expr,x, x0, n) And more!

From there, one could for example optimize the position of an aperture to maximize the kernel signal on a target.

2 A form that is usabale numerically

2.1 The lambdify method

As an example, let us create a kernel response map

First, we must substitute the parameters that are going to remain constant, constituting the expression sinapplied

[42]:
```
thesubs = []
for i in range(mykernuller.Na):
    #thesubs.append((X[i],X[i]-X[0]))
    #thesubs.append((Y[i],Y[i]-Y[0]))
    thesubs.append((X[i], mykernuller.pups[i, 0]))
    thesubs.append((Y[i], mykernuller.pups[i, 1]))
    thesubs.append((zeta[i], 1))
thesubs.append((lamb, 3.6e-6))
sinapplied = sinform.subs(thesubs)
sinapplied
```

[42]:
$$\frac{\sin\left(5513888.88888889\pi\alpha + 11297222.2222222\pi\beta\right)\cos\left(8270555.55555556\pi\alpha + 16945555.5555556\pi\beta\right)}{2} - \frac{\sin\left(5513888.88888889\pi\alpha + 11297222.2222222\pi\beta\right)\cos\left(57392222.2222222\pi\alpha + 24999444.4444444\pi\beta\right)}{2} + \frac{\sin\left(8270555.55555556\pi\alpha + 16945555.5555556\pi\beta\right)\cos\left(5513888.88888889\pi\alpha + 11297222.2222222\pi\beta\right)}{2} - \frac{\sin\left(8270555.55555556\pi\alpha + 16945555.5555556\pi\beta\right)\cos\left(24952777.7777778\pi\alpha + 36768333.3333333\pi\beta\right)}{2} + \frac{\sin\left(24952777.7777778\pi\alpha + 36768333.3333333\pi\beta\right)\cos\left(8270555.55555556\pi\alpha + 16945555.5555556\pi\beta\right)}{2} - \frac{\sin\left(24952777.7777778\pi\alpha + 36768333.3333333\pi\beta\right)\cos\left(57392222.2222222\pi\alpha + 24999444.4444444\pi\beta\right)}{2} - \frac{\sin\left(57392222.2222222\pi\alpha + 24999444.4444444\pi\beta\right)\cos\left(5513888.88888889\pi\alpha + 11297222.2222222\pi\beta\right)}{2} + \frac{\sin\left(57392222.2222222\pi\alpha + 24999444.4444444\pi\beta\right)\cos\left(24952777.7777778\pi\alpha + 36768333.3333333\pi\beta\right)}{2}$$

Then we must turn this expression into a numpy function that can be executed very fast.

[71]:
```
mykerout = sp.utilities.lambdify((alpha, beta), sp.expand_complex(kappa).
↪subs(thesubs))
```

Now mykerout is a numpy function that returns a kernel vector from the position of a source.

- It is *fast*
- It is vectorized

[95]:
```
%timeit akernel = mykerout(1e-6, 2e-6)
```

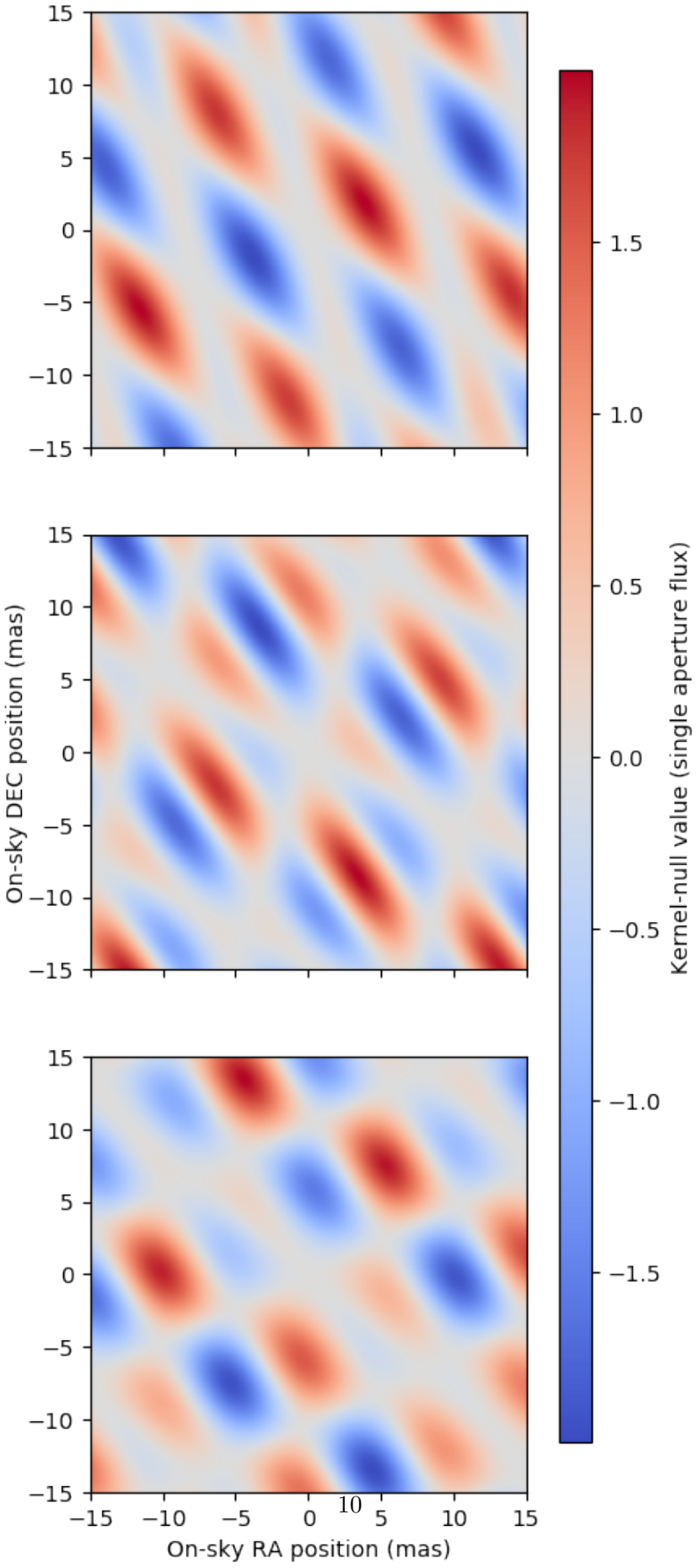
44.3 µs ± 798 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)

[96]:
```
xx, yy = np.meshgrid(np.linspace(-15,15,1024), np.linspace(-15,15,1024))
xxr = mas2rad(xx)
yyr = mas2rad(yy)
# Here we check how long it takes to compute the map
start = time()
amap = mykerout(xxr,yyr)
amap = np.squeeze(amap) # Removing an unwanted dimension
```

```
print("Map computed in %.2f seconds"%(time() - start))
```

```
Map computed in 1.49 seconds
```

[97]:
```
fig, axs = mykernuller.plot_response_maps(amap,␣
↪title=False,cbar_label="Kernel-null value (single aperture flux)",
                                          extent=[np.min(xx),np.max(xx), np.
↪min(yy), np.max(yy)],
                                         plotsize=4, dpi=100)
```


On-sky DEC position (mas)
On-sky RA position (mas)
Kernel-null value (single aperture flux)
15
10
5
0
−5
−10
−15
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

[68]:
```
from lmfit import minimize, Parameters
from tqdm import tqdm
from xara import mas2rad
```

2.2 Build a function that returns model signal

Just a little bit of packaging, unit conversion…

[69]:
```
def get_kn_signal(params):
    alpha = mas2rad(params["alpha"])
    beta = mas2rad(params["beta"])
    ic = params["ic"]
    kn_sig = ic * mykerout(alpha, beta).flatten()
    return kn_sig
```

2.3 Build a function that returns the residual

[70]:
```
def get_kn_residual(params, y):
    return y -  get_kn_signal(params)
```

2.4 Build a parameter object

[47]:
```
params = Parameters()
params.add("alpha", value=5, min=-10, max=10)
params.add("beta", value=8, min=-10, max=10)
params.add("ic",value=3, min=0, max=20)
```

2.5 A Monte Carlo simulation for noisy data

The measured data will be represented by ys

Here, I only simulate read noise on the measured intensities

[54]:
```
noisevec = np.random.normal(scale=0.1, size=(10000,3))
ys = get_kn_signal(params)
noisy = noisevec + ys.T
```

Let us change the starting point slightly, so that it is not too easy

[62]:
```
params = Parameters()
params.add("alpha", value=4.5, min=-10, max=10)
```

```
params.add("beta", value=7, min=-10, max=10)
params.add("ic",value=2.5, min=0, max=20)
```

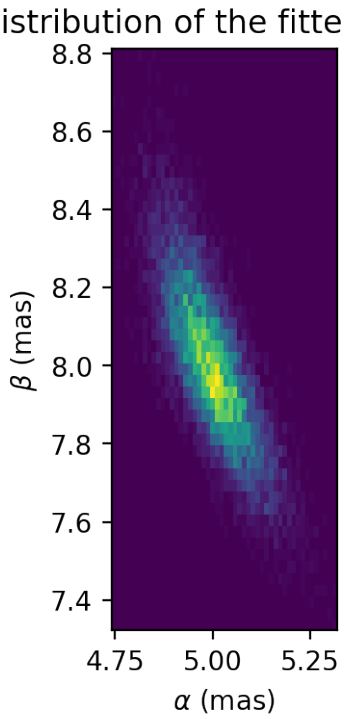
[63]:
```
soluce = minimize(get_kn_residual, params, args=(ys,))
soluce
```

[63]:
```
<lmfit.minimizer.MinimizerResult at 0x7f261161def0>
```

2.6 Now to do the model-fit for each of the realizations

[64]:
```
sols = []
alphas = []
betas = []
ics = []
res = []
for y in tqdm(noisy):
    soluce = minimize(get_kn_residual, params, args=(y,))
    sols.append(soluce.params)
    alphas.append(soluce.params["alpha"].value)
    betas.append(soluce.params["beta"].value)
    ics.append(soluce.params["ic"].value)
    res.append(get_kn_residual(params, y))
alphas = np.array(alphas)
betas = np.array(betas)
ics = np.array(ics)
plt.figure(dpi=200)
plt.hist2d(alphas, betas, bins=50)
plt.xlabel(r"$\alpha$ (mas)")
plt.ylabel(r"$\beta$ (mas)")
plt.gca().set_aspect("equal")
plt.title("A distribution of the fitted position")
plt.show()
get_kn_residual(params, y)
```

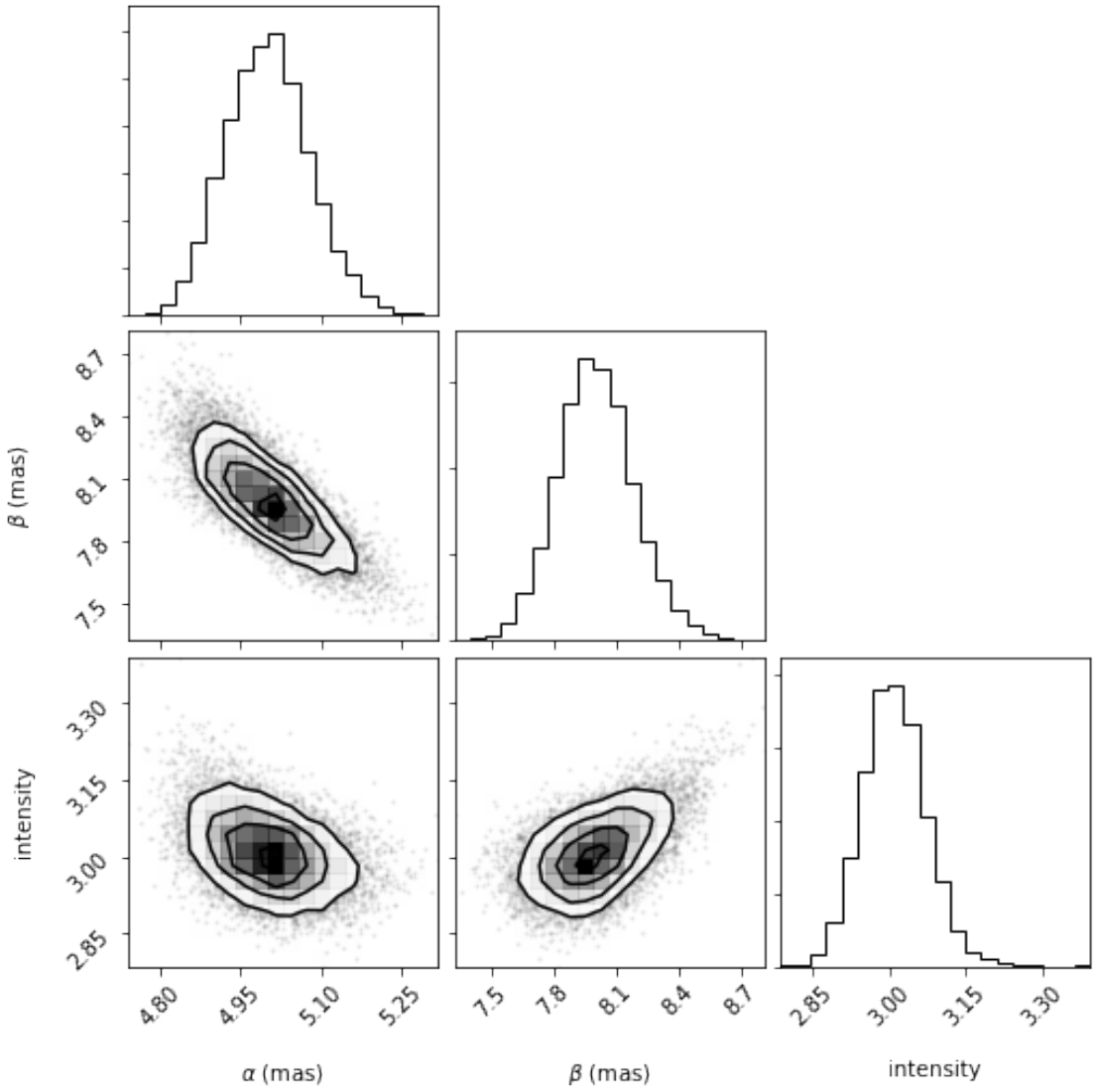
```
100%|      | 10000/10000 [00:52<00:00, 191.83it/s]
```



[64]:
```
array([-0.02523429,  1.51276893,  1.74639696])
```

2.7 For a more standard way of looking at that kind of data, use corner plots

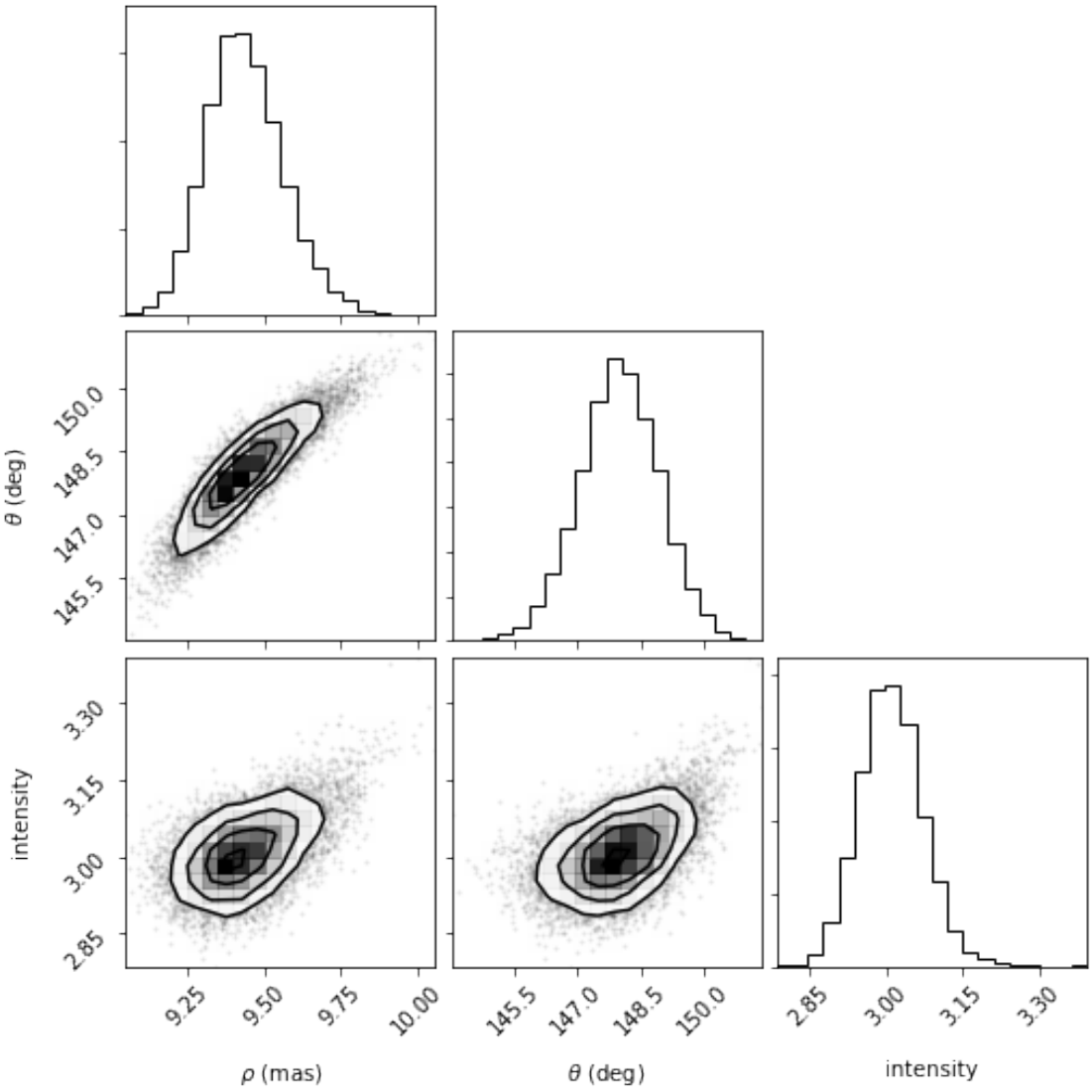
[65]:
```
import corner
figure = corner.corner(np.vstack((alphas, betas, ics)).T,labels=[r"$\alpha$␣
↪(mas)", r"$\beta$ (mas)", "intensity"])
```



2.8 Can also express the result in separation and position angle

[66]:
```
rhos = np.sqrt(alphas**2+betas**2)
cpform = alphas + 1j * betas
rhos = np.abs(cpform)
thetas = (np.angle(cpform)+np.pi/2)*180/np.pi
```

[67]:
```
figure = corner.corner(np.vstack((rhos, thetas, ics)).T,labels=[r"$\rho$␣
↪(mas)", r"$\theta$ (deg)", "intensity"])
```


θ (deg)
150.0
148.5
147.0
145.5
intensity
3.30
3.15
3.00
2.85
9.25
9.50
9.75
10.00
ρ (mas)
145.5
147.0
148.5
150.0
θ (deg)
2.85
3.00
3.15
3.30
intensity

[ ]:

[ ]: