basic_usage

August 13, 2020

[2]:
```
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
```

```
Building a model from scratch
```

1 Common use cases

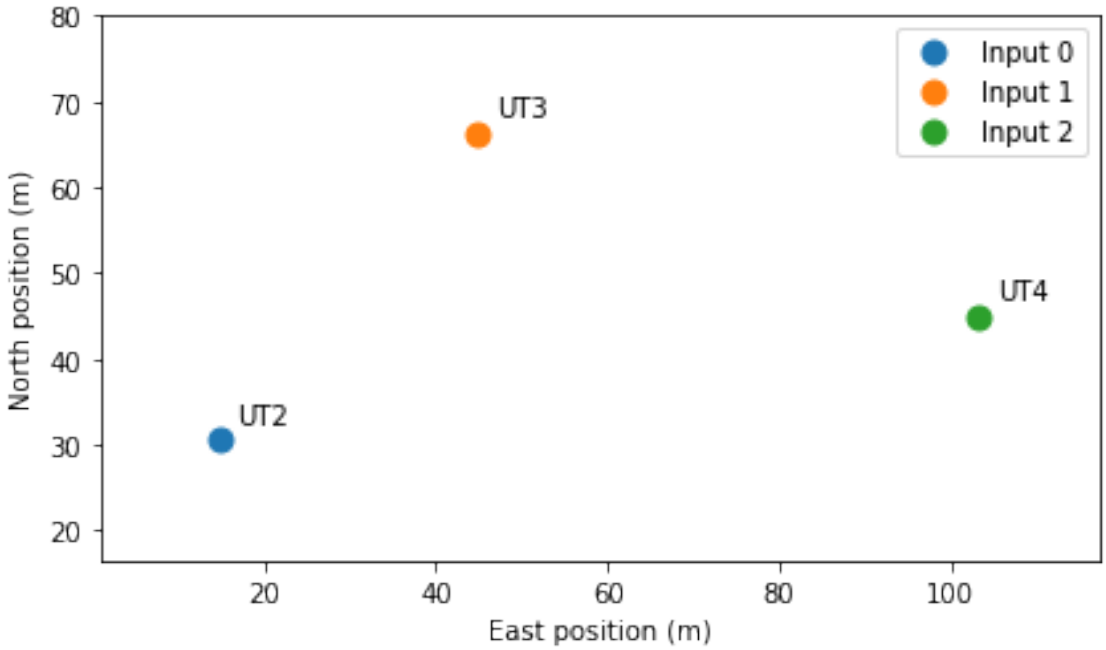
1.1 Three-input kernel nuller

To maintain flexibility, things are done in several steps.

1. Choosing an array of aperture coordinates (here by cropping the list of VLTI apertures
2. Creating the kernuller object
3. Building the model with the `build_procedural_model()` method
4. Building a generic kernel matrix for 2 inputs

[24]:
```
statlocs = kernuller.VLTI[1:]
statnames = ["UT2", "UT3", "UT4"] #Giving names to the apertures for the plot
mykernuller = kernuller.kernuller(statlocs,3.6e-6)
mykernuller.build_procedural_model(verbose=False)
mykernuller.K = kernuller.pairwise_kernel(2)

fig = mykernuller.plot_pupils(offset=2,s=80,
                              marginratio=7.,
                              title=False,
                              pupil_indices=statnames,
                              showlegend=True)
#fig.axes[0].set_xlim(-20,350)
plt.show()
```

Building a model from scratch



1.2 Plotting the sagital Complex Matrix Plot

TODO: It does not include the bright row.

This procedure builds **N** and **S** matrices. We have to add the bright output manually

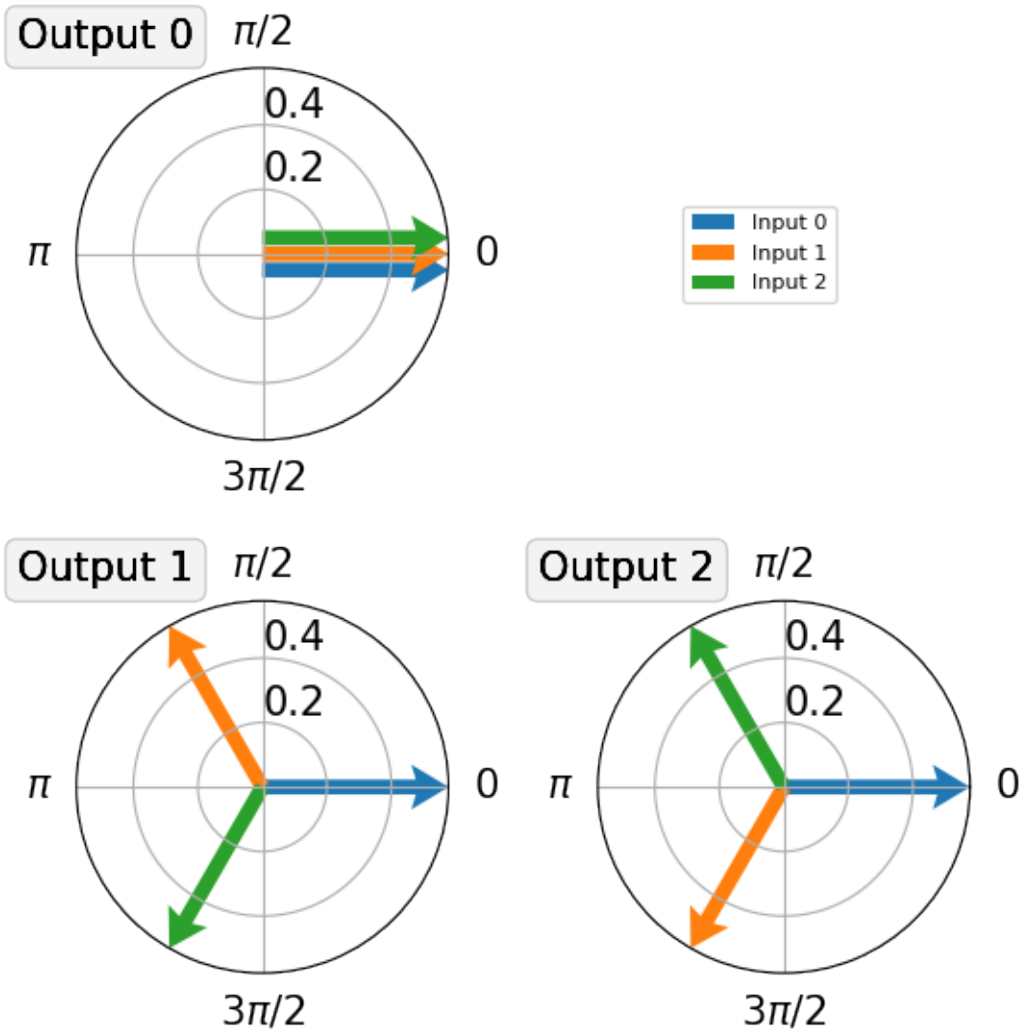
Here is the procedure:

1. We use a vstack to add the bright row
2. We use a vstack to add a blank row that will be used to have a blank spot in the grid of plots
3. We create the `base_preoffset` that is an array of offsets to manually move the overlapping arrows in plot of the bright row (by default the plotting routine makes incremental offsets)
4. We create labels for the outputs (`outlabeld`) and insert a blank item to match the blankrow of the matrix

[25]:
```
Mn = np.vstack([1/np.sqrt(3)*np.ones(3),mykernuller.S.dot(mykernuller.Np)])
Mn2 = np.vstack((Mn[:1,:], np.zeros_like(Mn[1,:]), Mn[1:,:]))

base_preoffset = np.zeros_like(Mn2)
base_preoffset[0,:] = np.linspace(0-0.05j,0+0.05j,base_preoffset.shape[1])
outlabels = ["Output %d"%(i) for i in range(7)]
outlabels.insert(1, "")
fig, axs = mykernuller.plot_outputs_smart(Mn2,base_preoffset=base_preoffset,
                                          nx=2,plotsize=3,
```

```
                                        osfrac=0.1, title=False,
                                        labels=False,
                                        legendoffset=(1.6,0.5),
                                        out_label=outlabels,
                                        thealpha=0.05,
                                        onlyoneticklabel=False,
                                        rlabelpos=90, dpi=100)
```

```
removing the labels: [False False False False]
```



1.3 Plotting CMP including the output combination

Here done for three on-sky positions (three phasings of the inputs) that are: 1. Unresolved 1. Partially resolved 1. On the peak of throughput

To do that:

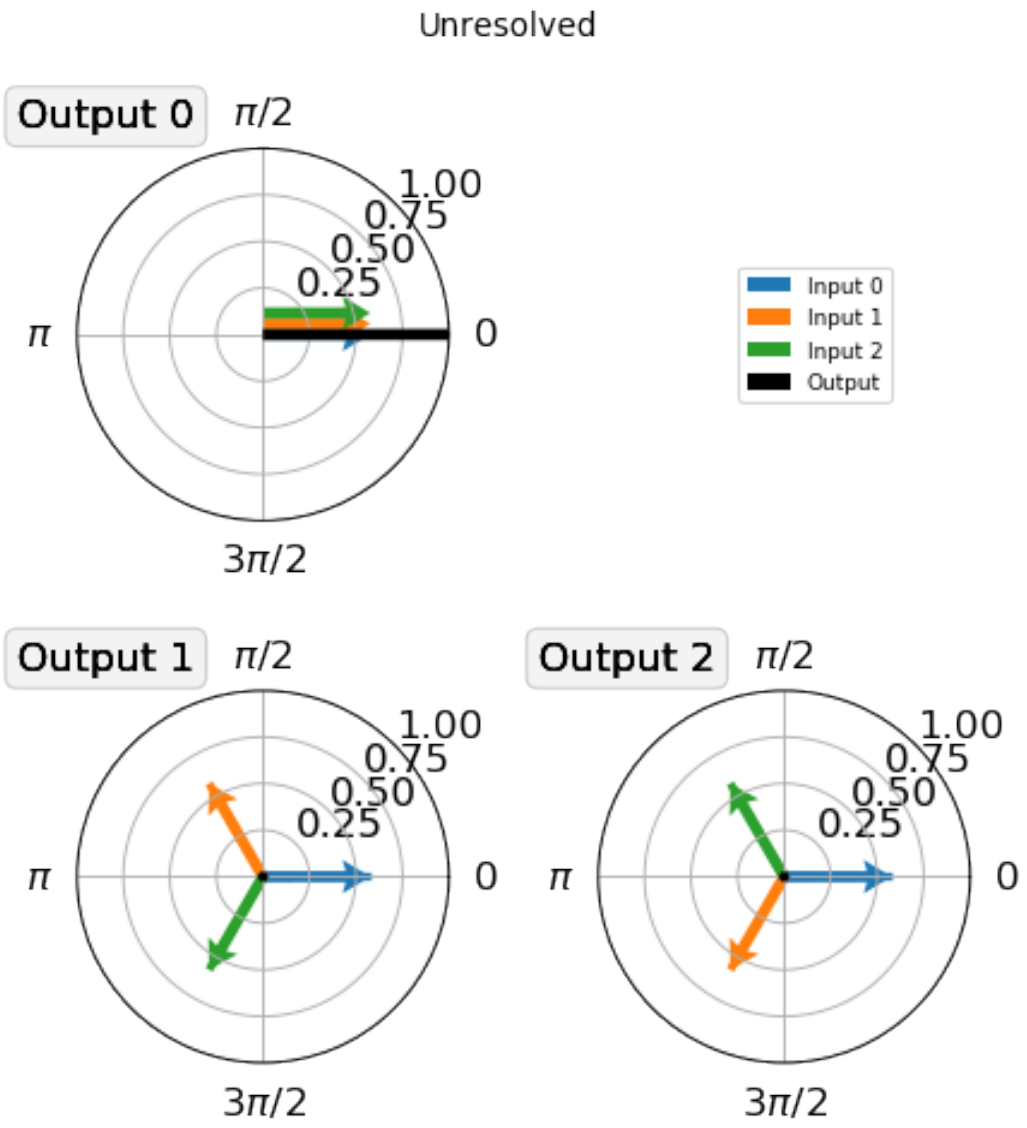
1. Each position is defined in separation ρ and position angle θ relative to the optical axis.
2. The `rmax` parameter is defined to adjust manually the radial extent of the plot (often needed when plotting the output)
3. A signal of unit amplitude is created with the method `kernuller.incoming_light()` and the signal is normalized (the `binary=` parameter will create both the light from an on-axis source and one from a companion. Here, we take only the companion)
4. The `plot_outputs_smart` is called to plot everything, and the light is passed in the `inputfield` parameter.

Outlabels and other stuff are reused.

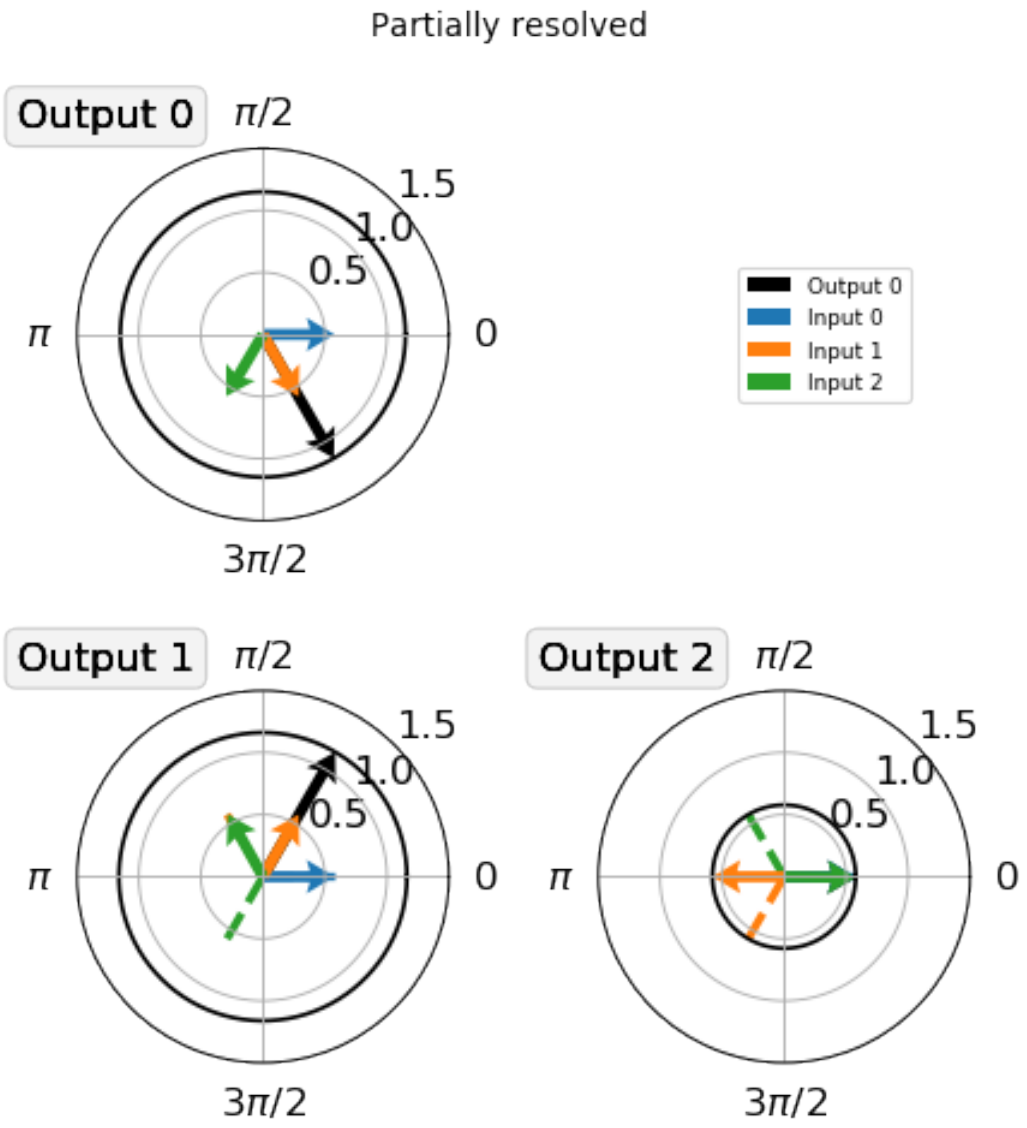
[28]:
```
titles = ["Unresolved",
          "Partially resolved",
          "On the peak"]
rhosteps = np.linspace(0,5.8,3)
thetasteps = 116.*np.ones_like(rhosteps)
rmaxes = np.array([1., 1.5, 1.8])
for i in range(rhosteps.shape[0]):
    print(rhosteps[i], thetasteps[i])
    input_binary = mykernuller.incoming_light(1,binary=np.
 ↪array([rhosteps[i],thetasteps[i] , 1.]))#110. +180
    input_binary = input_binary/ input_binary[1,0]
    fig, axs = mykernuller.plot_outputs_smart(Mn2,inputfield=input_binary[1],
                                              nx=2,legendoffset=(1.75,0.5),
                                              plotsize=3, osfrac=0.1,
                                             title=titles[i], mainlinewidth=0.03,
                                              plotspaces=(0.35,0.4),
                                              rmax=rmaxes[i],
                                              legendstring="center left",
                                              rlabelpos=45, out_label=outlabels,
                                            labels=False, onlyoneticklabel=False,
                                              outputontop=(rhosteps[i]<2.),
                                              thealpha=0.05, dpi=80)
```

```
0.0 116.0
we do plot on top
we do plot on top
we do plot on top
we do plot on top
removing the labels: [False False False False]
```

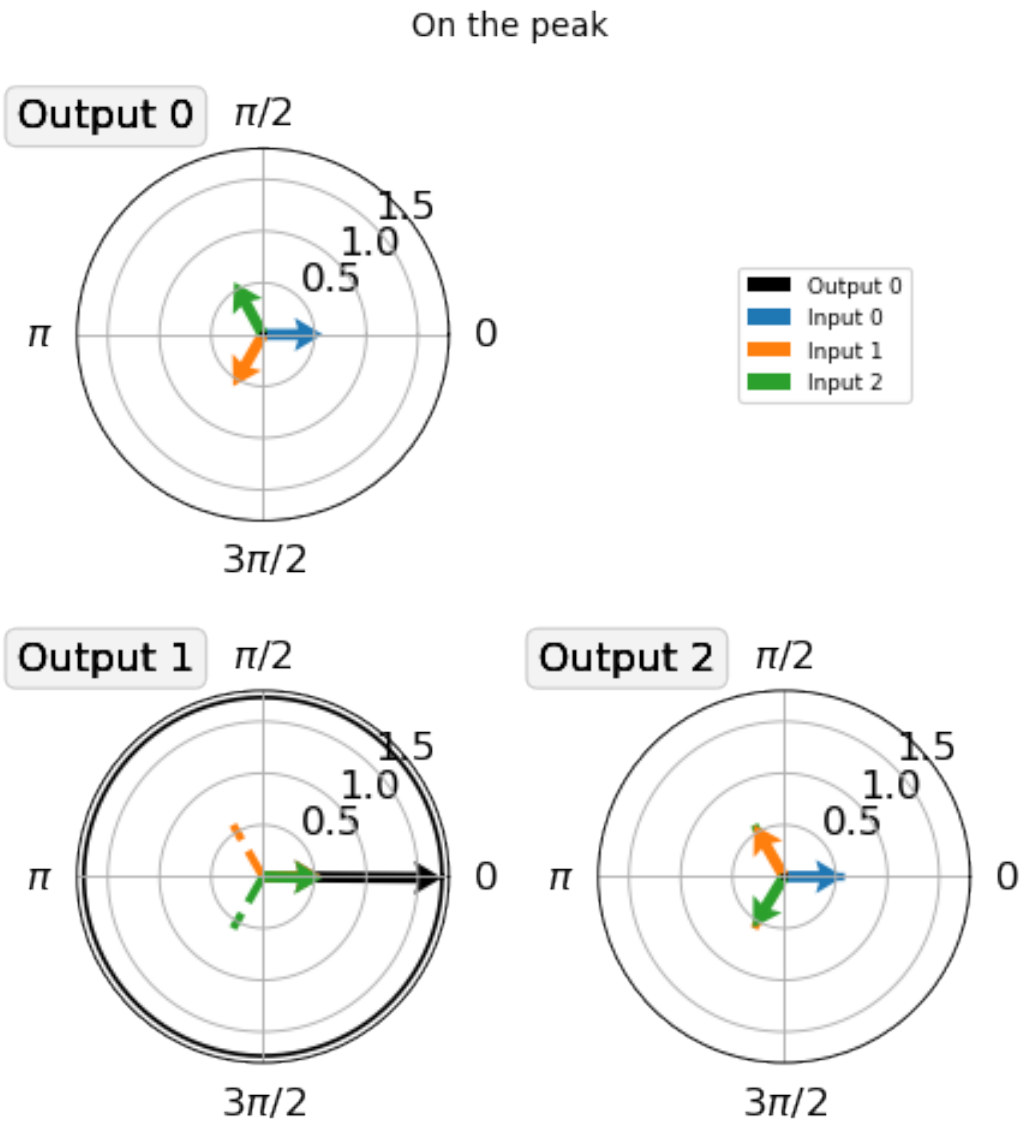
```
/home/rlaugier/Documents/kernel/kernuller/kernuller/kernuller.py:1452:
UserWarning: This figure includes Axes that are not compatible with
tight_layout, so results might be incorrect.
  fig.tight_layout()
```




```
2.9 116.0
removing the labels: [False False False False]
```




```
5.8 116.0
removing the labels: [False False False False]
```



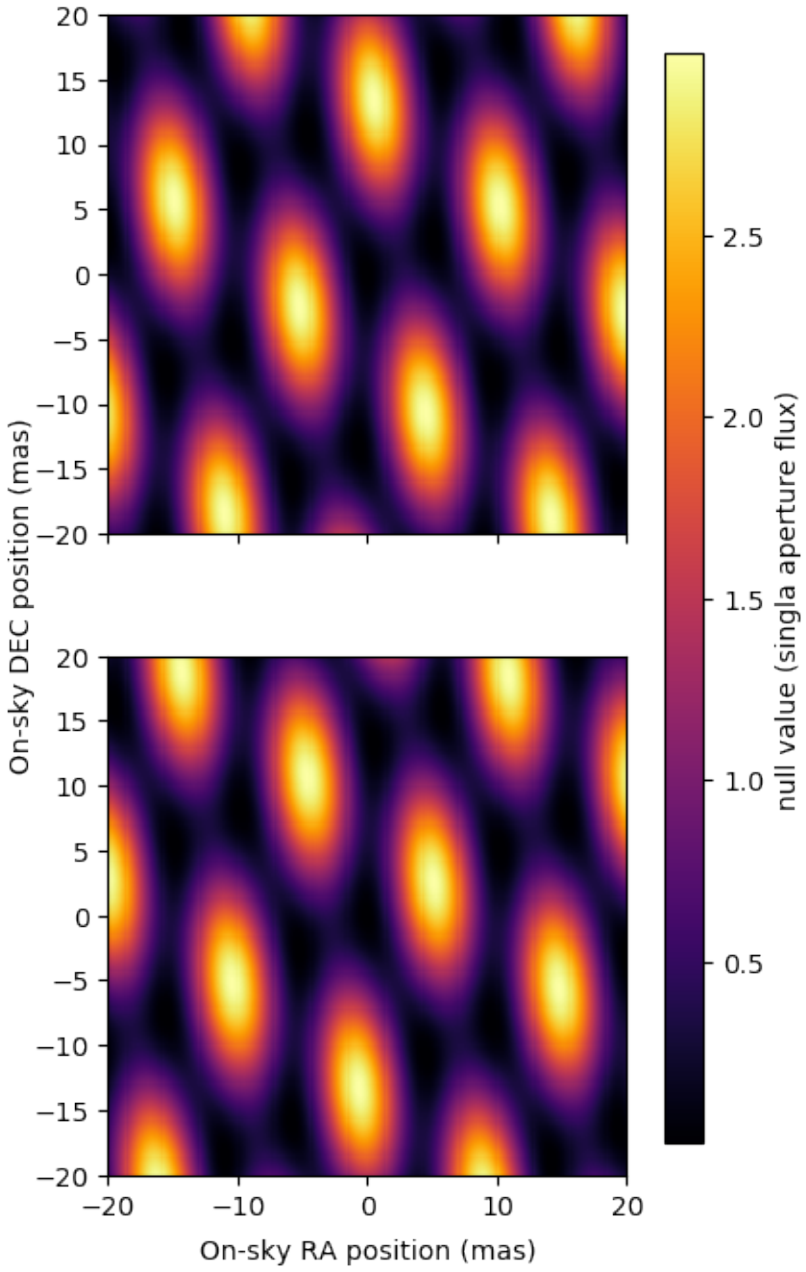
1.4 Building response maps

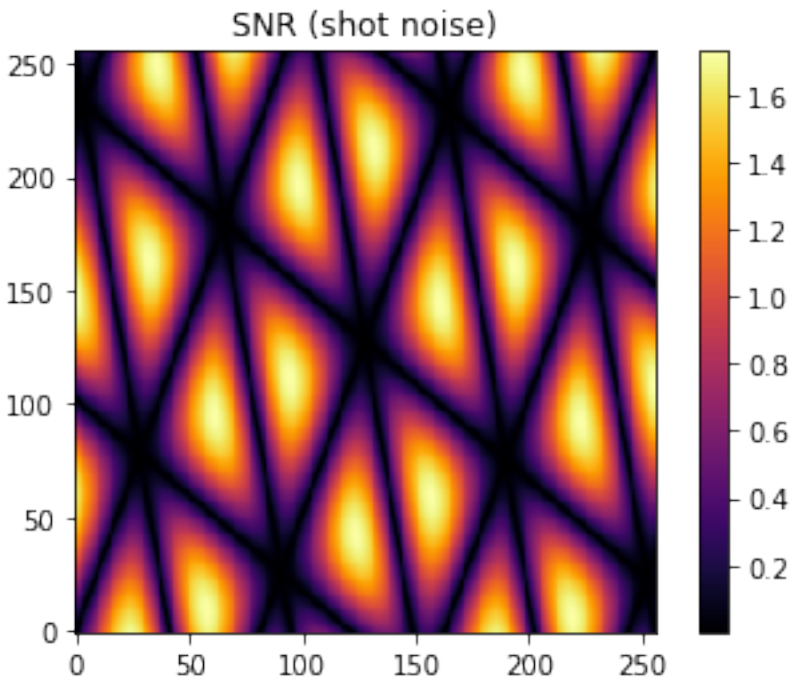
1. Construct the grid of coordinates to be used.
2. Here, i just use complex formalism to convert to polar coordinates
3. `mapparams` is a flattened map of parameters in the same format as xara binary (separation, position angle, contrast primary/secondary)
4. `outintensities` are built that correspond to the output signal for the light from each of these postions
5. `outkers` are the corresponding kernel-signals

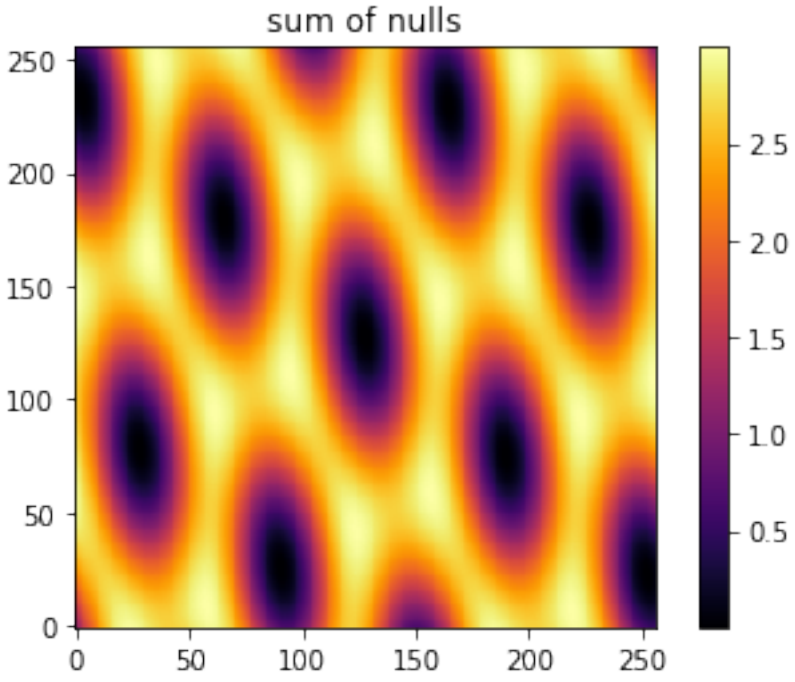
[27]:
```
xx, yy = np.meshgrid(np.linspace(-20,20,256), np.linspace(-20,20,256))
cpx = xx + yy*1j
rhos = np.abs(cpx).flatten()
thetas = (np.angle(cpx)*180/np.pi - 90).flatten()
mapparams = np.array([[rhos[i], thetas[i], 1] for i in range(rhos.shape[0])])
outintensities = np.array([mykernuller.get_I(binary=mapparams[i]) for i in␣
↪range(mapparams.shape[0])])
outkers = mykernuller.K.dot(outintensities.T).T
#nullmap = outintensities.reshape((xx.shape[0], xx.shape[1], mykernuller.K.
↪shape[1]))
nullmap = np.array([outintensities[:,i].reshape(xx.shape) for i in␣
↪range(outintensities.shape[1])])
#kermap = outkers.reshape((xx.shape[0], xx.shape[1], mykernuller.K.shape[0]))
kermap = np.array([outkers[:,i].reshape(xx.shape) for i in range(outkers.
↪shape[1])])
```

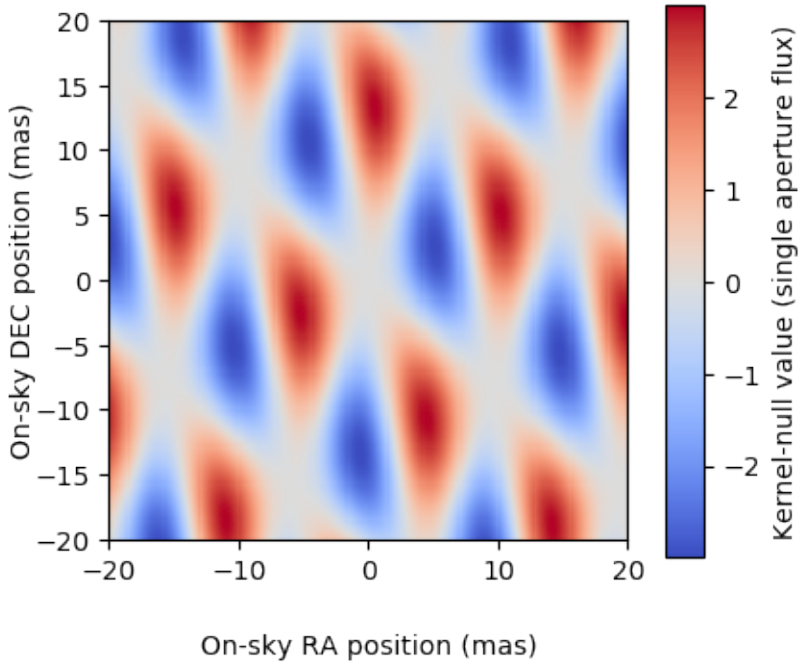
The maps can the be plotted using `plot_response_maps()`. This is another method that has a large number of parameters, including the number of columns, colorbar labels, dpi, etc...

[7]:
```
fig, axs = mykernuller.plot_response_maps(nullmap, nx=1, cmap="inferno",␣
↪title=False,
                                          plotsize=4, cbar_label="null value␣
↪(singla aperture flux)",
                                          extent=[np.min(xx),np.max(xx), np.
↪min(yy), np.max(yy)], dpi=100)
noise = np.sqrt(nullmap.sum(axis=0))
plt.figure()
plt.imshow(np.abs(kermap[0])/noise, cmap="inferno")
plt.colorbar()
plt.title("SNR (shot noise)")
plt.show()

plt.figure()
plt.imshow(np.sum(nullmap, axis=0), cmap="inferno")
plt.colorbar()
plt.title("sum of nulls")
plt.show()

fig, axs = mykernuller.plot_response_maps(kermap,␣
↪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)
null value (singla aperture flux)
20
15
10
5
0
−5
−10
−15
−20
2.5
2.0
1.5
1.0
0.5


SNR (shot noise)
250
200
150
100
50
0
0
50
100
150
200
250
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2


sum of nulls
250
200
150
100
50
0
0
50
100
150
200
250
2.5
2.0
1.5
1.0
0.5



2 The Four telescopes kernel nuller

Works exactly the same as with 3 inputs

[29]:
```
statlocs = kernuller.VLTI
mykernuller = kernuller.kernuller(statlocs, 3.6e-6)
mykernuller.build_procedural_model()
#matrices, recipes = kernuller.generative_random_pruning(mykernuller.Ms, 1)
#mymatrix = np.load("didactic_plot_matrix.npy", allow_pickle=True)
#ykernuller.build_model_from_matrix(mymatrix)
```

`Building a model from scratch`

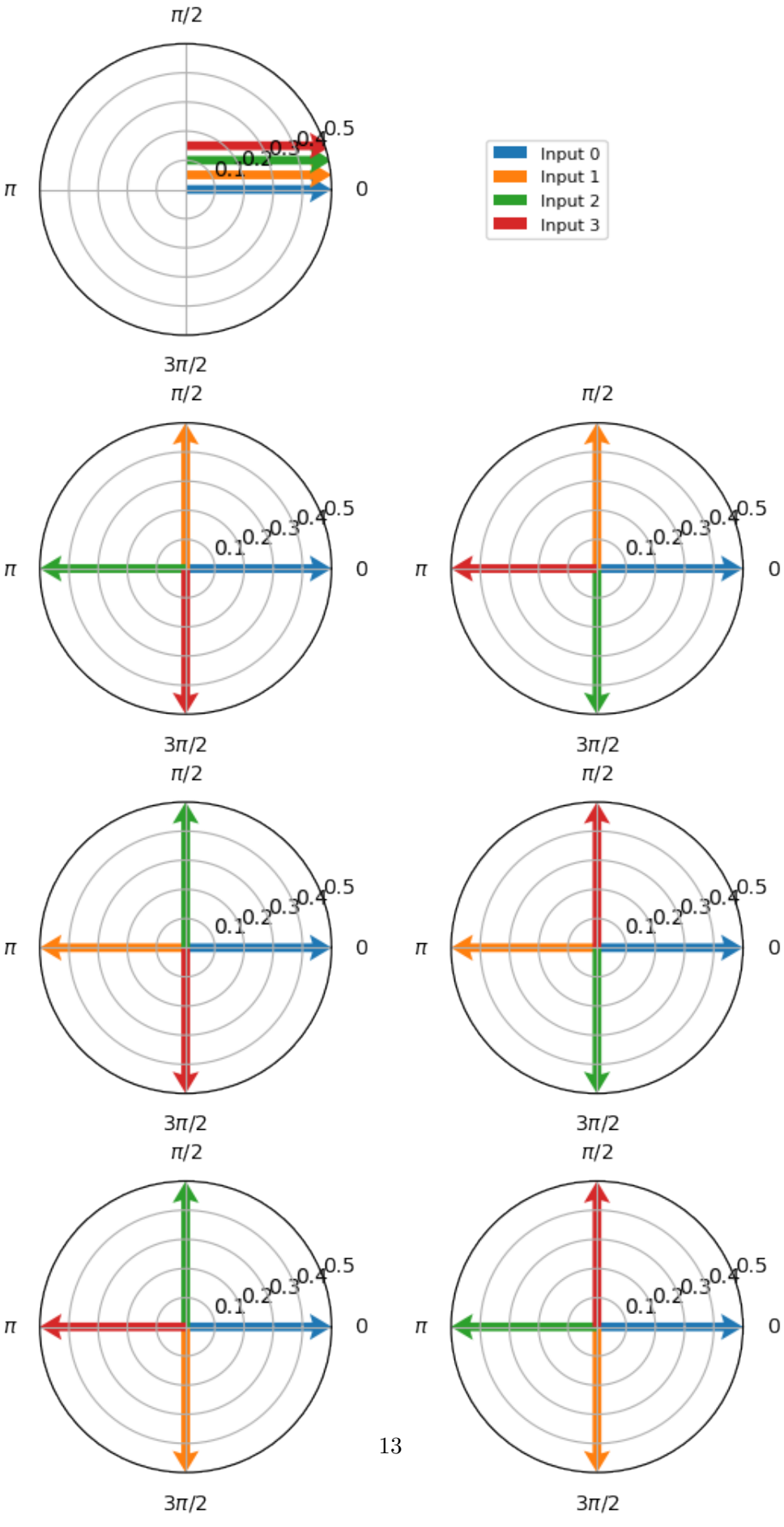
Note that this time, we let the plotting tool come up with automatic offsets for overlapping arrows.

[30]:
```
#invec = 1*np.exp(1j*np.random.normal(scale=0.1, size=4))
Mn = np.array(sp.N(mykernuller.Ms), dtype=np.complex64)
Mn2 = np.vstack((Mn[:1,:], np.zeros_like(Mn[1,:]), Mn[1:,:]))
invec = np.array([0.9944306 -0.10539342j,
                  0.99986418-0.01648085j,
                  0.99544197+0.0953692j ,
                  0.99997709-0.00676858j])
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)
kernuller.printlatex(sp.expand_power_exp(mykernuller.Ms))
```

```
/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
```

```
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

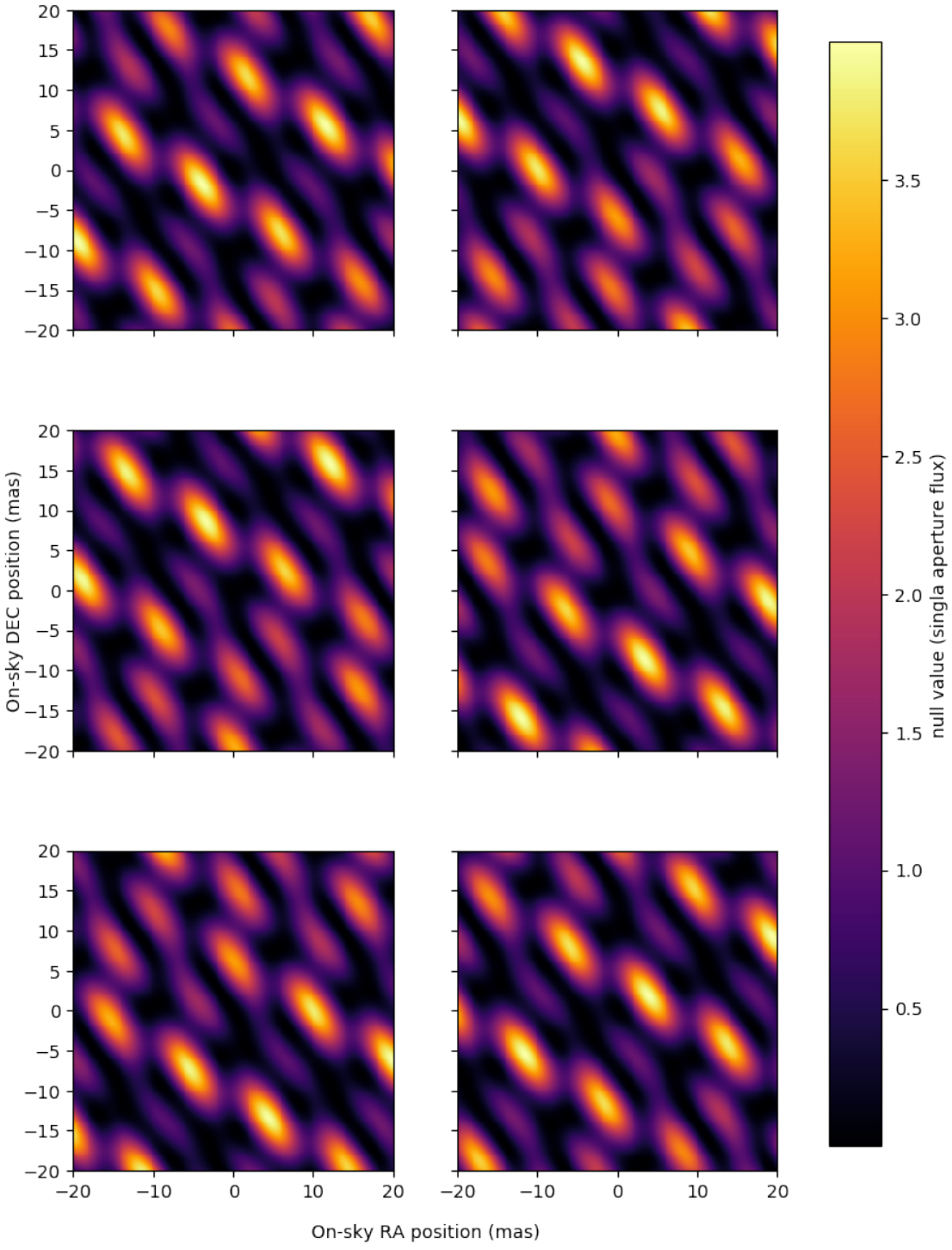
```
\begin{equation}
  \left[\begin{matrix}\frac{1}{2} & \frac{1}{2} & \frac{1}{2} &
\frac{1}{2}\\\frac{1}{2} & \frac{j}{2} & - \frac{1}{2} & -
\frac{j}{2}\\\frac{1}{2} & \frac{j}{2} & - \frac{j}{2} & -
\frac{1}{2}\\\frac{1}{2} & - \frac{1}{2} & \frac{j}{2} & -
\frac{j}{2}\\\frac{1}{2} & - \frac{1}{2} & - \frac{j}{2} &
\frac{j}{2}\\\frac{1}{2} & - \frac{j}{2} & \frac{j}{2} & -
\frac{1}{2}\\\frac{1}{2} & - \frac{j}{2} & - \frac{1}{2} &
\frac{j}{2}\end{matrix}\right]
\end{equation}
```

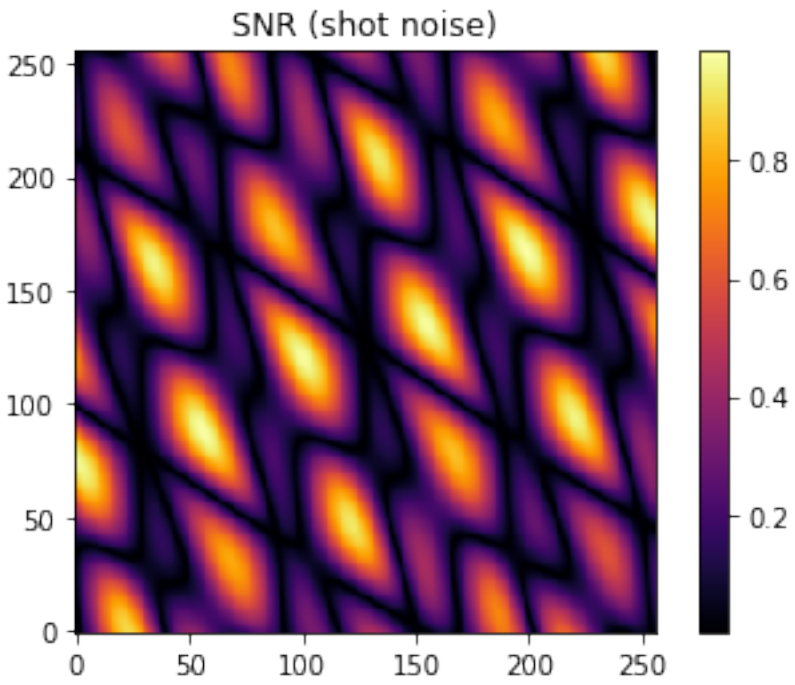
[10]:
```
xx, yy = np.meshgrid(np.linspace(-20,20,256), np.linspace(-20,20,256))
cpx = xx + yy*1j
rhos = np.abs(cpx).flatten()
thetas = (np.angle(cpx)*180/np.pi - 90).flatten()
mapparams = np.array([[rhos[i], thetas[i], 1] for i in range(rhos.shape[0])])
outintensities = np.array([mykernuller.get_I(binary=mapparams[i]) for i in␣
↪range(mapparams.shape[0])])
outkers = mykernuller.K.dot(outintensities.T).T
#nullmap = outintensities.reshape((xx.shape[0], xx.shape[1], mykernuller.K.
↪shape[1]))
nullmap = np.array([outintensities[:,i].reshape(xx.shape) for i in␣
↪range(outintensities.shape[1])])
#kermap = outkers.reshape((xx.shape[0], xx.shape[1], mykernuller.K.shape[0]))
kermap = np.array([outkers[:,i].reshape(xx.shape) for i in range(outkers.
↪shape[1])])
```

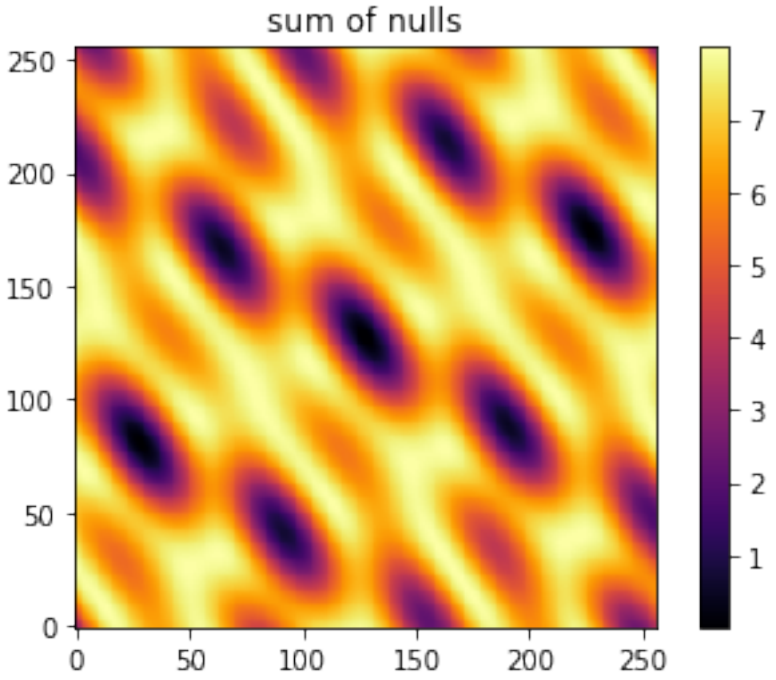
The maps can the be plotted using `plot_response_maps()`. This is another method that has a large number of parameters, including the number of columns, colorbar labels, dpi, etc...

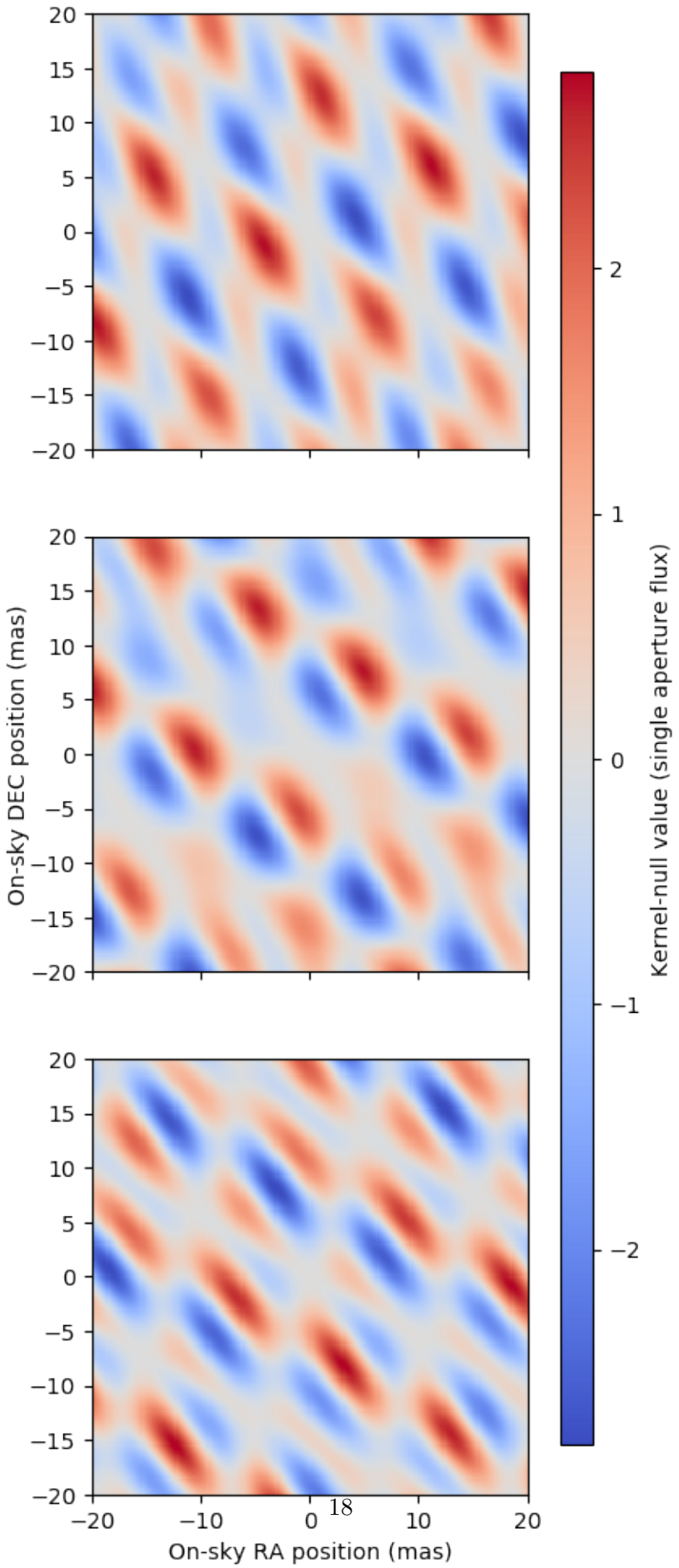
[11]:
```
fig, axs = mykernuller.plot_response_maps(nullmap, nx=2, cmap="inferno",␣
↪title=False,
                                          plotsize=4, cbar_label="null value␣
↪(singla aperture flux)",
                                          extent=[np.min(xx),np.max(xx), np.
↪min(yy), np.max(yy)], dpi=100)
noise = np.sqrt(nullmap.sum(axis=0))
plt.figure()
plt.imshow(np.abs(kermap[0])/noise, cmap="inferno")
plt.colorbar()
plt.title("SNR (shot noise)")
plt.show()

plt.figure()
plt.imshow(np.sum(nullmap, axis=0), cmap="inferno")
```

```
plt.colorbar()
plt.title("sum of nulls")
plt.show()

fig, axs = mykernuller.plot_response_maps(kermap,␣
 ↪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)
null value (singla aperture flux)
20
15
10
5
0
−5
−10
−15
−20
3.5
3.0
2.5
2.0
1.5
1.0
0.5


SNR (shot noise)
250
200
150
100
50
0
0
50
100
150
200
250
0.8
0.6
0.4
0.2


sum of nulls
250
200
150
100
50
0
0
50
100
150
200
250
7
6
5
4
3
2
1


On-sky DEC position (mas)
On-sky RA position (mas)
Kernel-null value (single aperture flux)
20
15
10
5
0
−5
−10
−15
−20
2
1
0
−1
−2

3 Building models from pre-computed matrices

Pre-computed matrices are available in `kernuller.nullers` for 3, 4, 5, and 6 telescopes. They are reduced, normalized and complete.

[12]:
```
import kernuller.nullers
```

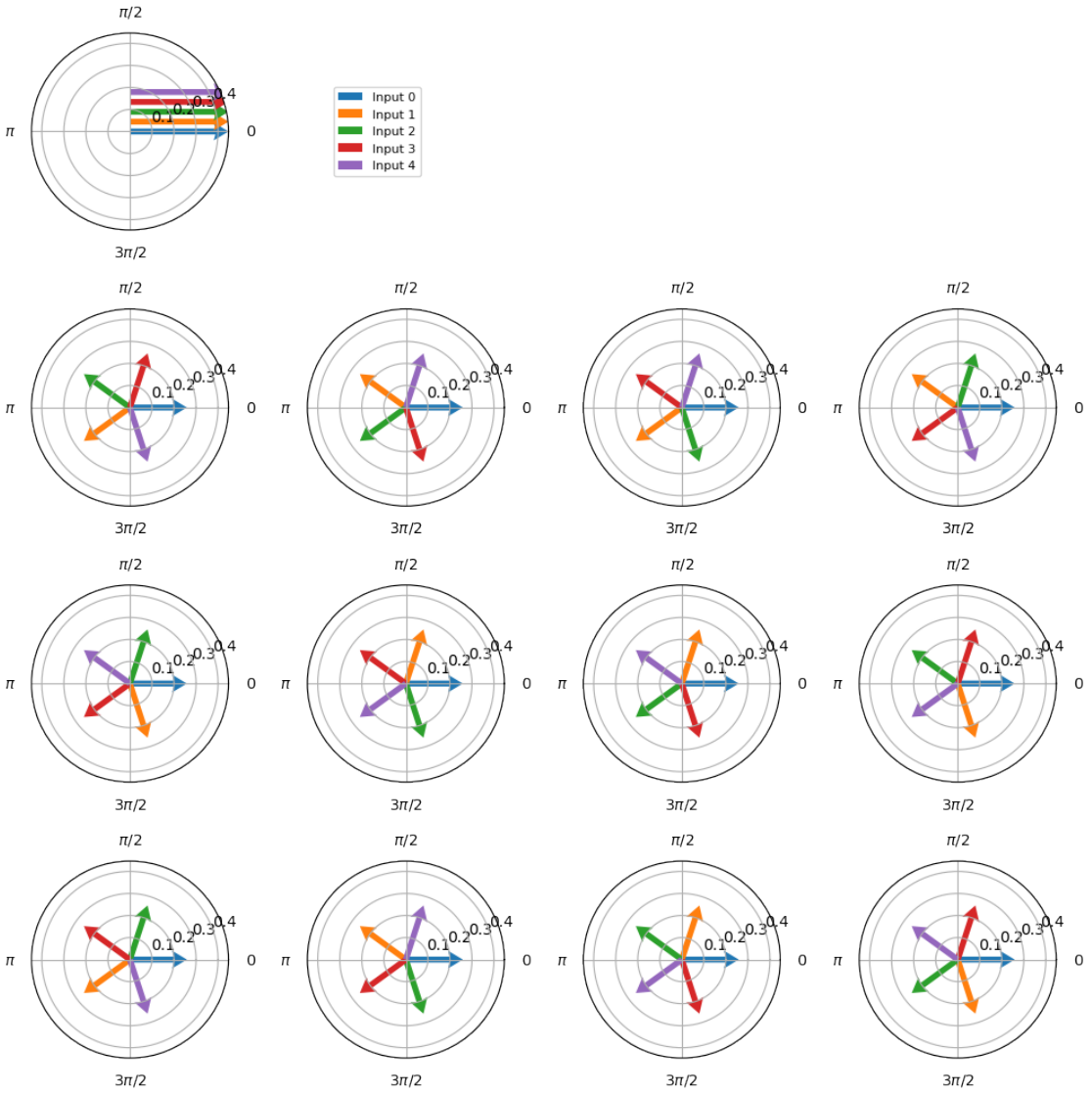
3.1 A 5 -input combiner

[13]:
```
mymatrix_5 = kernuller.nullers.matrices_5T[0]
mykernuller_5 = kernuller.kernuller(kernuller.CHARA[:5],3.6e-6)
mykernuller_5.build_model_from_matrix(mymatrix_5)
mykernuller_5.K = kernuller.pairwise_kernel(12)
```

```
Building a model from scratch
```

[14]:
```
Mn = np.array(sp.N(mykernuller_5.Ms), dtype=np.complex64)
Mn2 = np.vstack((Mn[:1,:], np.zeros_like(Mn[1:4,:]), Mn[1:,:]))
fig, axs = mykernuller_5.plot_outputs_smart(Mn2, nx=4,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)
```

```
removing the labels: [False False False False False False False False False
False False False
 False False False False]
```

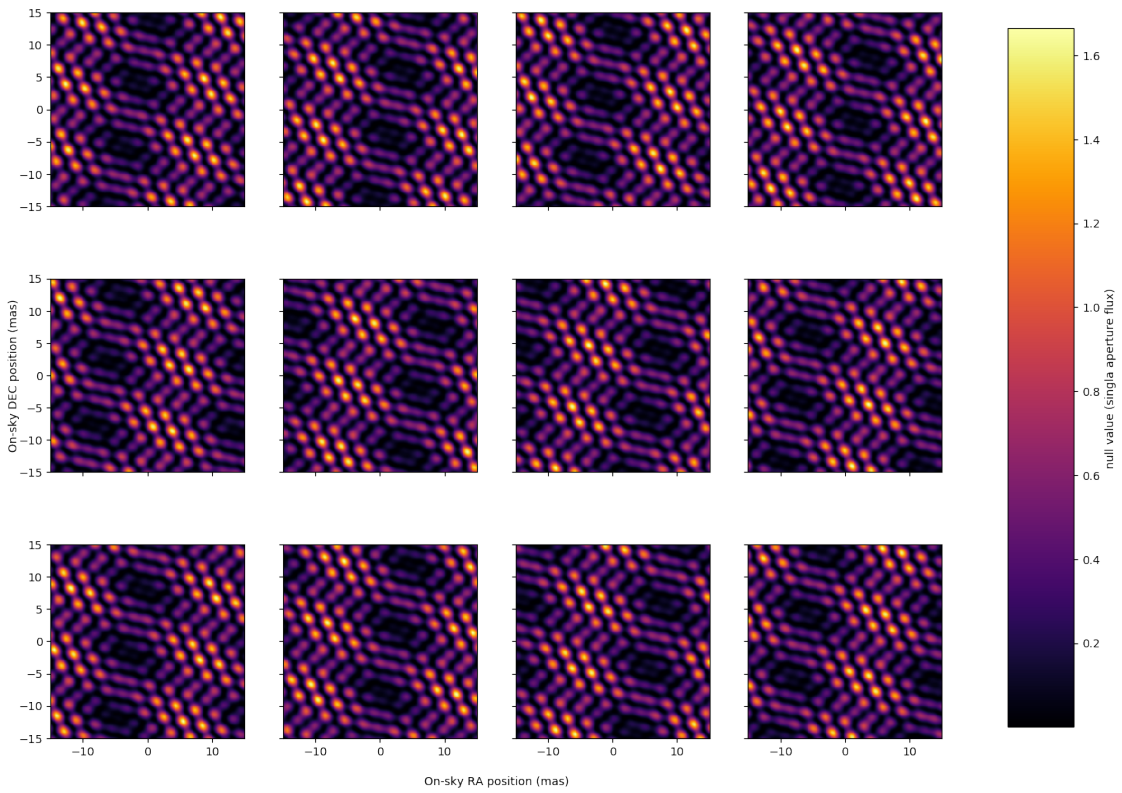


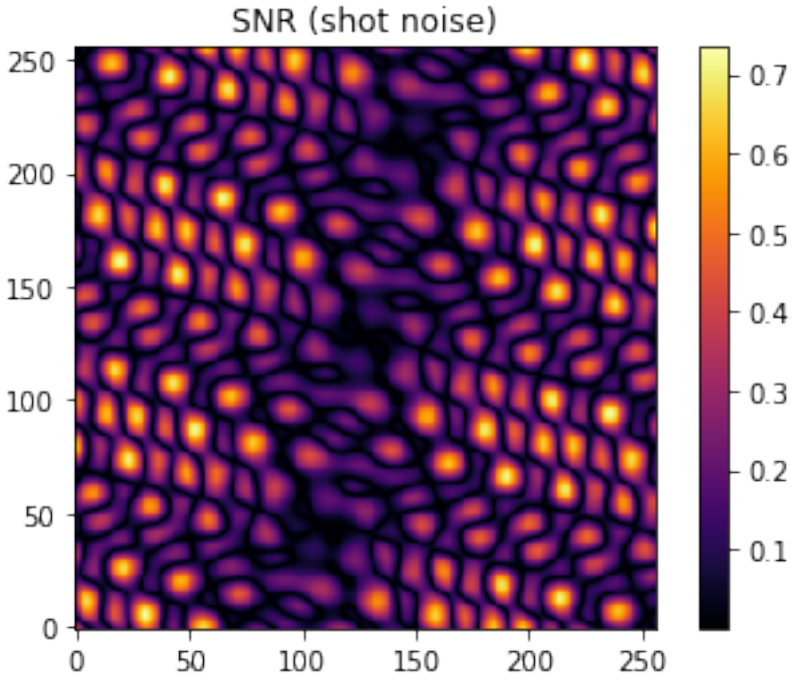
[15]:
```
xx, yy = np.meshgrid(np.linspace(-15,15,256), np.linspace(-15,15,256))
cpx = xx + yy*1j
rhos = np.abs(cpx).flatten()
thetas = (np.angle(cpx)*180/np.pi - 90).flatten()
mapparams = np.array([[rhos[i], thetas[i], 1] for i in range(rhos.shape[0])])
outintensities = np.array([mykernuller_5.get_I(binary=mapparams[i]) for i in␣
↪range(mapparams.shape[0])])
outkers = mykernuller_5.K.dot(outintensities.T).T
#nullmap = outintensities.reshape((xx.shape[0], xx.shape[1], mykernuller_5.K.
↪shape[1]))
nullmap = np.array([outintensities[:,i].reshape(xx.shape) for i in␣
↪range(outintensities.shape[1])])
```

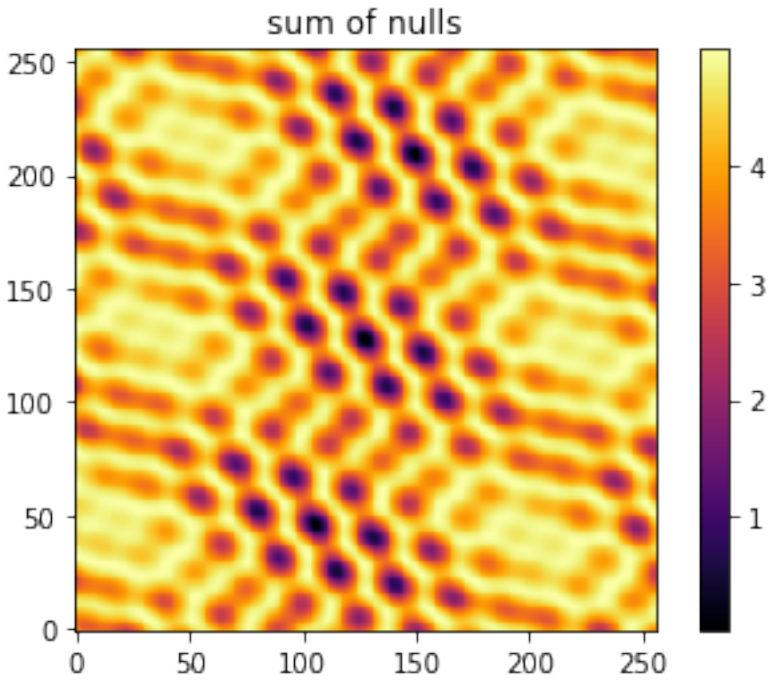
```
#kermap = outkers.reshape((xx.shape[0], xx.shape[1], mykernuller_5.K.shape[0]))
kermap = np.array([outkers[:,i].reshape(xx.shape) for i in range(outkers.
 ↪shape[1])])
```

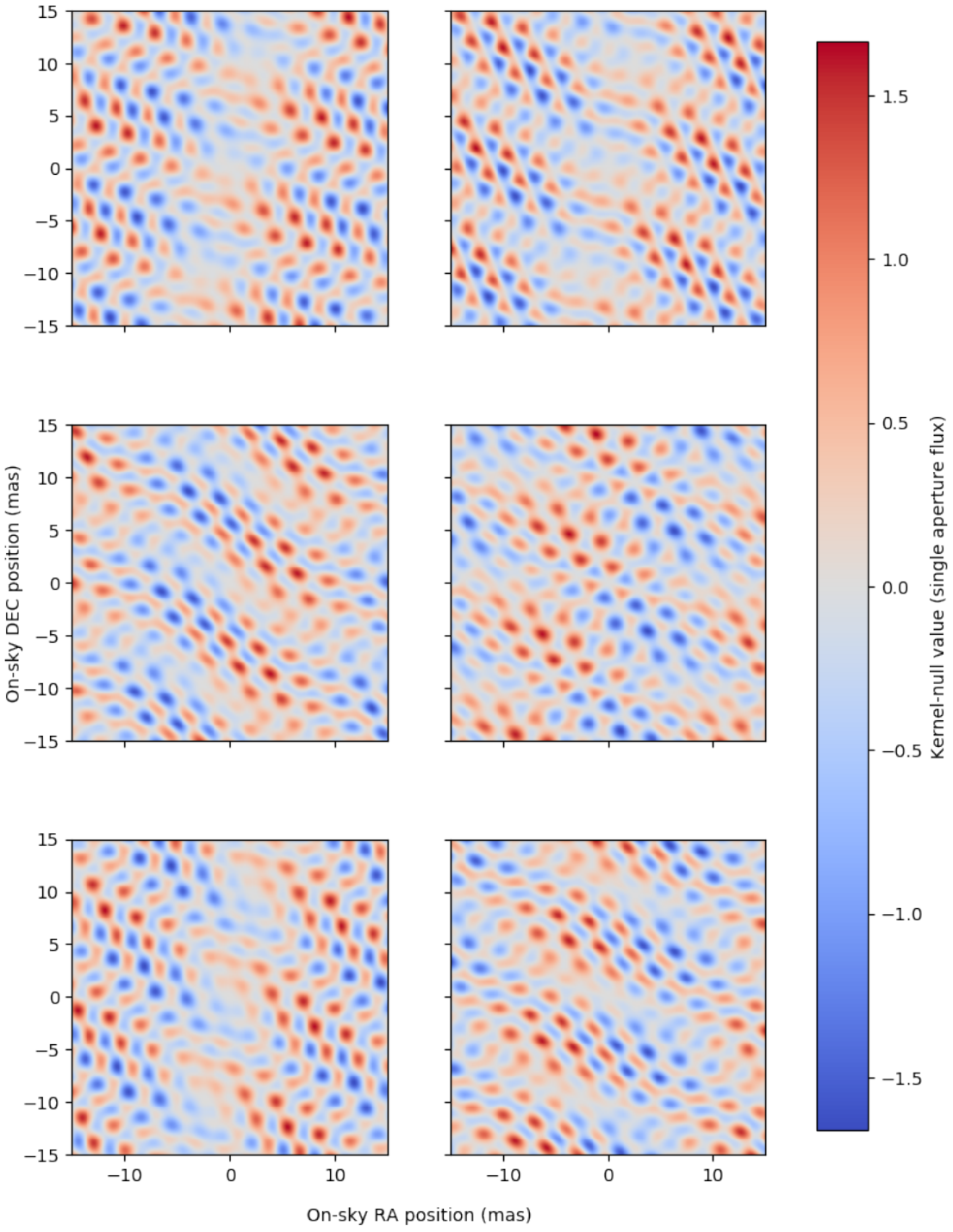
The maps can the be plotted using `plot_response_maps()`. This is another method that has a large number of parameters, including the number of columns, colorbar labels, dpi, etc…

[16]:
```
fig, axs = mykernuller_5.plot_response_maps(nullmap, nx=4, cmap="inferno",␣
 ↪title=False,
                                            plotsize=4, cbar_label="null value␣
 ↪(singla aperture flux)",
                                            extent=[np.min(xx),np.max(xx), np.
 ↪min(yy), np.max(yy)], dpi=100)
noise = np.sqrt(nullmap.sum(axis=0))
plt.figure()
plt.imshow(np.abs(kermap[0])/noise, cmap="inferno")
plt.colorbar()
plt.title("SNR (shot noise)")
plt.show()

plt.figure()
plt.imshow(np.sum(nullmap, axis=0), cmap="inferno")
plt.colorbar()
plt.title("sum of nulls")
plt.show()

fig, axs = mykernuller_5.plot_response_maps(kermap,nx=2,␣
 ↪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)
null value (singla aperture flux)


SNR (shot noise)


sum of nulls
250
200
150
100
50
0
0
50
100
150
200
250
4
3
2
1


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

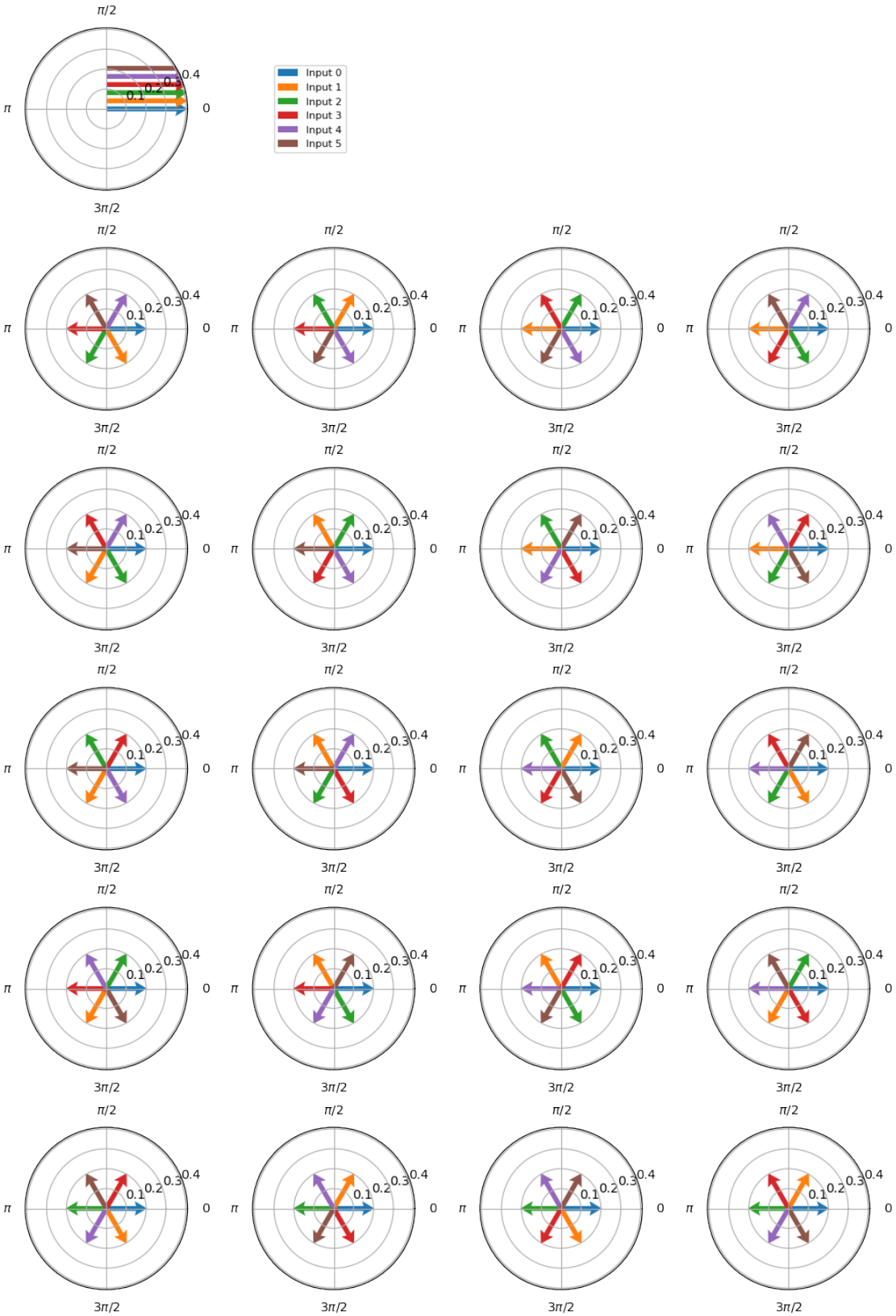
3.2 A 6 -input combiner

[17]:
```
mymatrix_6 = kernuller.nullers.matrices_6T[0]
mykernuller_6 = kernuller.kernuller(kernuller.CHARA[:],3.6e-6)
mykernuller_6.build_model_from_matrix(mymatrix_6)
mykernuller_6.K = kernuller.pairwise_kernel(20)
```

```
Building a model from scratch
```

[18]:
```
Mn = np.array(sp.N(mykernuller_6.Ms), dtype=np.complex64)
Mn2 = np.vstack((Mn[:1,:], np.zeros_like(Mn[1:4,:]), Mn[1:,:]))
fig, axs = mykernuller_6.plot_outputs_smart(Mn2, nx=4,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)
```

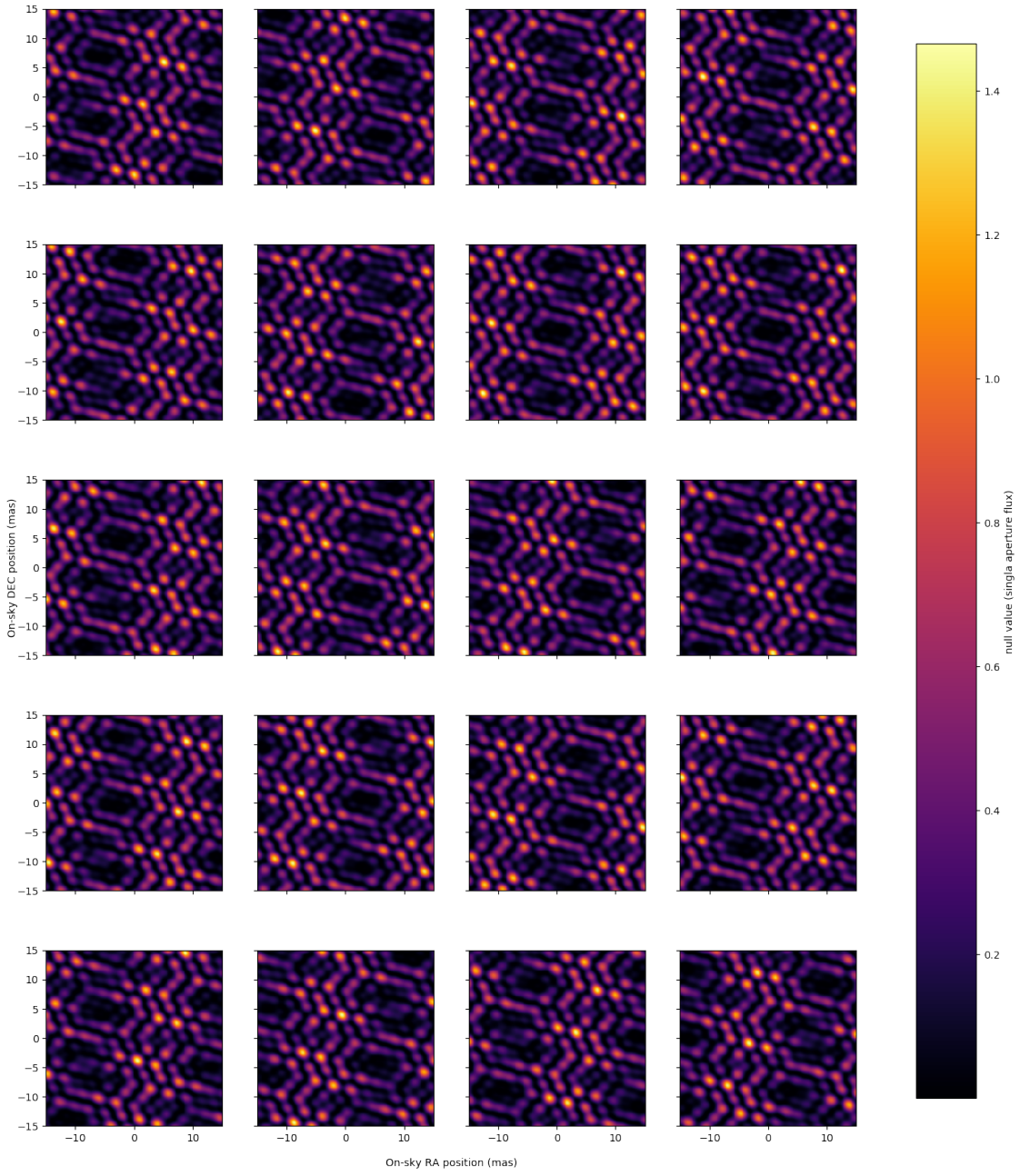
```
removing the labels: [False False False False False False False False False
False False False
 False False False False False False False False False False False False]
```

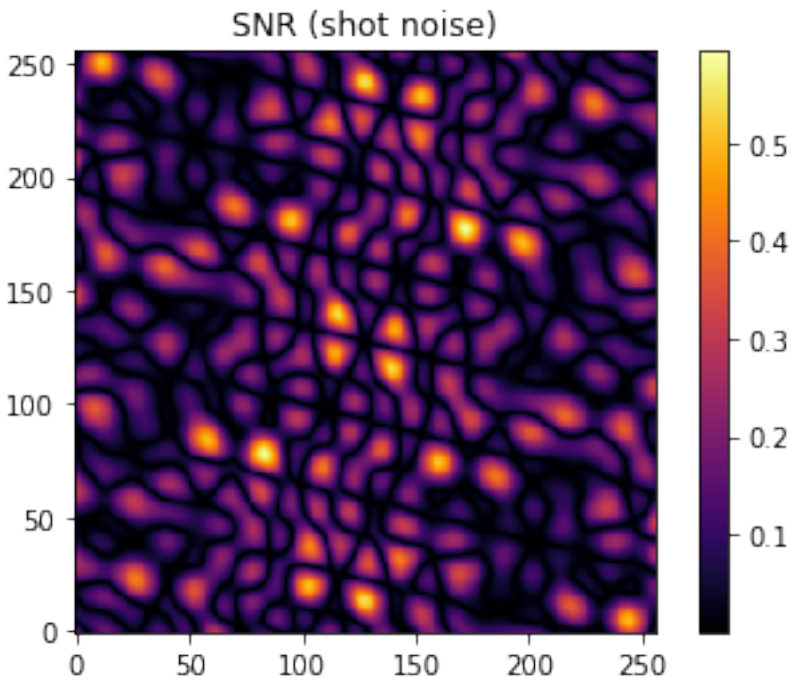

Input 0
Input 1
Input 2
Input 3
Input 4
Input 5
π/2
π
0
3π/2
0.1
0.2
0.3
0.4

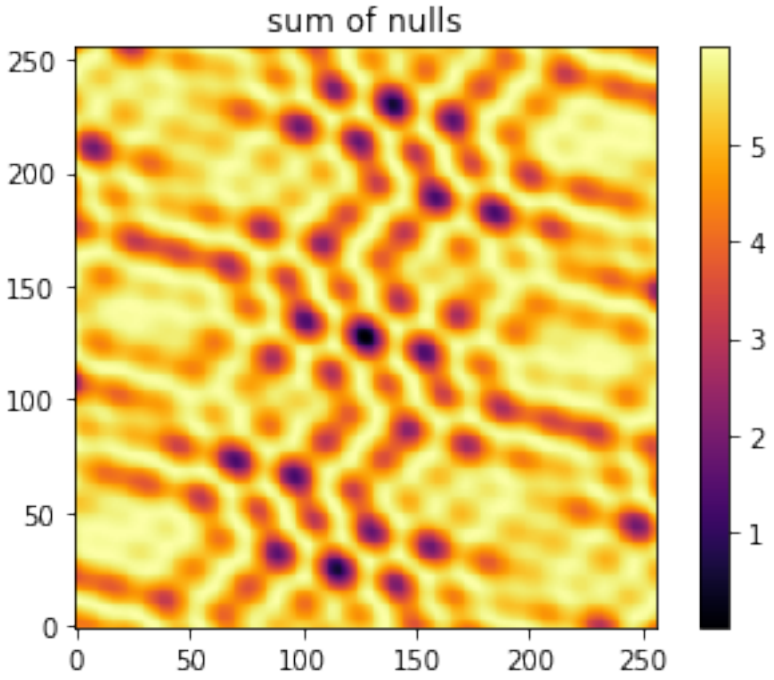
[19]:
```
xx, yy = np.meshgrid(np.linspace(-15,15,256), np.linspace(-15,15,256))
cpx = xx + yy*1j
rhos = np.abs(cpx).flatten()
thetas = (np.angle(cpx)*180/np.pi - 90).flatten()
mapparams = np.array([[rhos[i], thetas[i], 1] for i in range(rhos.shape[0])])
outintensities = np.array([mykernuller_6.get_I(binary=mapparams[i]) for i in␣
↪range(mapparams.shape[0])])
outkers = mykernuller_6.K.dot(outintensities.T).T
#nullmap = outintensities.reshape((xx.shape[0], xx.shape[1], mykernuller_6.K.
↪shape[1]))
nullmap = np.array([outintensities[:,i].reshape(xx.shape) for i in␣
↪range(outintensities.shape[1])])
#kermap = outkers.reshape((xx.shape[0], xx.shape[1], mykernuller_6.K.shape[0]))
kermap = np.array([outkers[:,i].reshape(xx.shape) for i in range(outkers.
↪shape[1])])
```

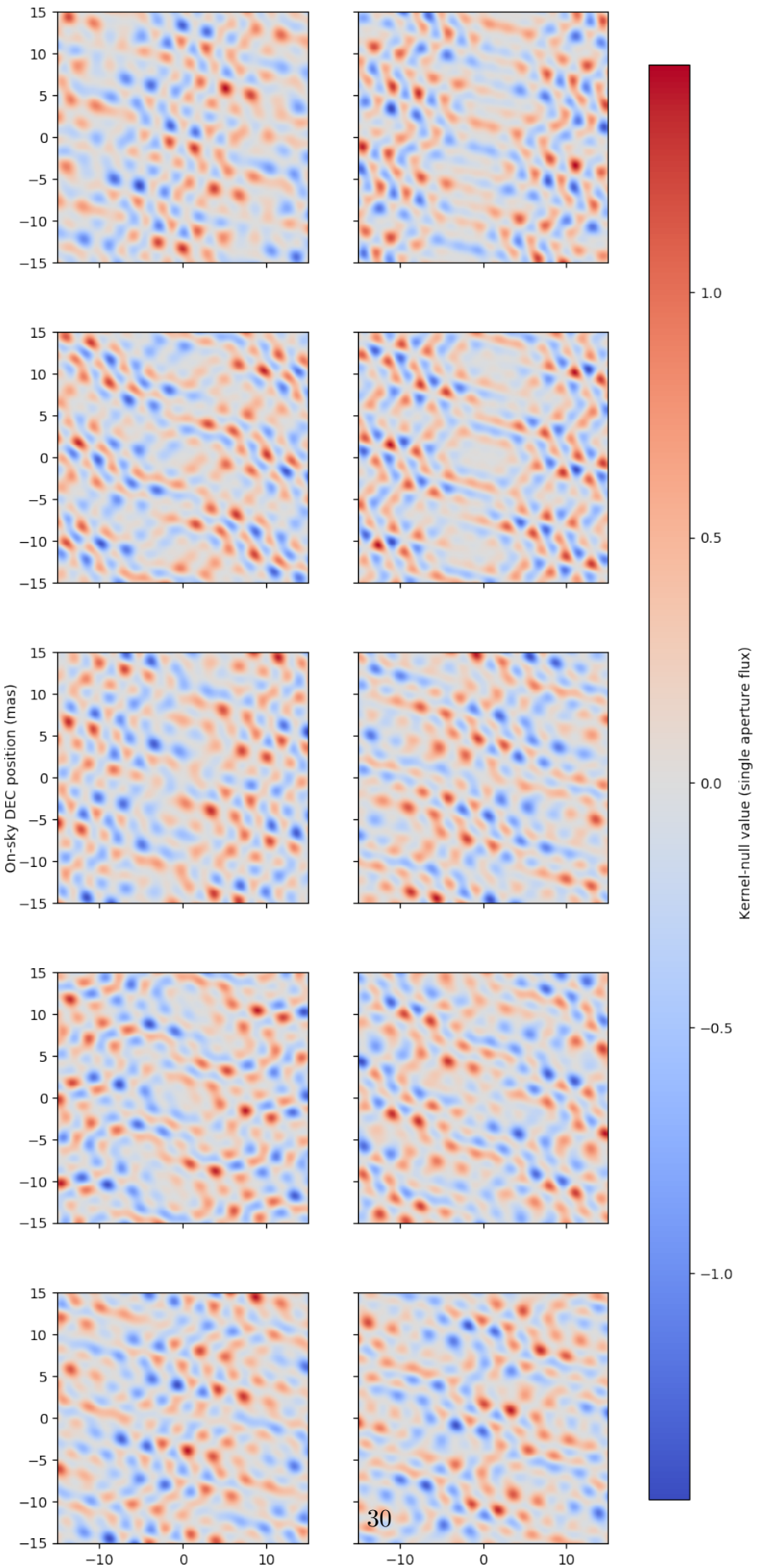
The maps can the be plotted using `plot_response_maps()`. This is another method that has a large number of parameters, including the number of columns, colorbar labels, dpi, etc...

[20]:
```
fig, axs = mykernuller_6.plot_response_maps(nullmap, nx=4, cmap="inferno",␣
↪title=False,
                                            plotsize=4, cbar_label="null value␣
↪(singla aperture flux)",
                                            extent=[np.min(xx),np.max(xx), np.
↪min(yy), np.max(yy)], dpi=100)
noise = np.sqrt(nullmap.sum(axis=0))
plt.figure()
plt.imshow(np.abs(kermap[0])/noise, cmap="inferno")
plt.colorbar()
plt.title("SNR (shot noise)")
plt.show()

plt.figure()
plt.imshow(np.sum(nullmap, axis=0), cmap="inferno")
plt.colorbar()
plt.title("sum of nulls")
plt.show()

fig, axs = mykernuller_6.plot_response_maps(kermap,nx=2,␣
↪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)
null value (singla aperture flux)


SNR (shot noise)
250
200
150
100
50
0
0
50
100
150
200
250
0.5
0.4
0.3
0.2
0.1


sum of nulls
250
200
150
100
50
0
0
50
100
150
200
250
5
4
3
2
1


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

4 A few legacy functionalities

4.1 Building the model from the paper

4.1.1 Building from hard-coded matrices

`kernuller.build_classical_model()` helps you build the model from Martinache & Ireland 2018 from hard-coded matrices.

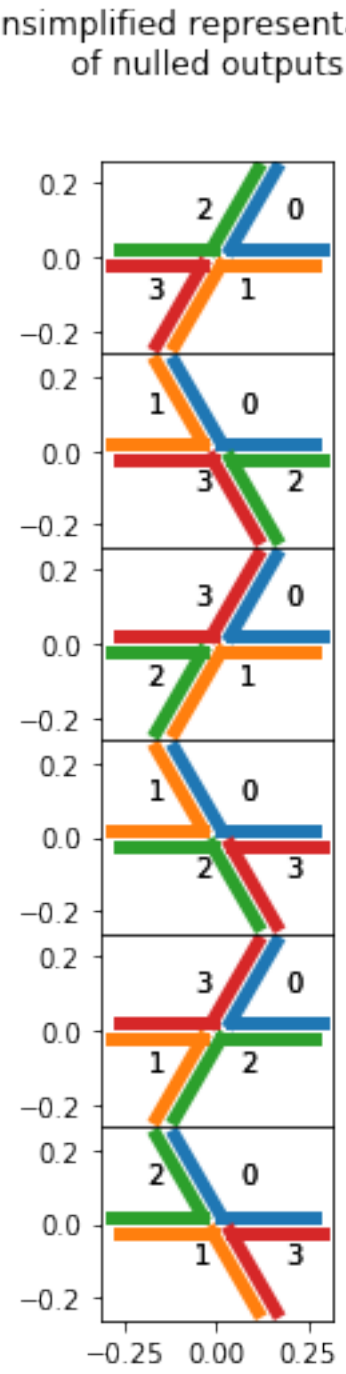
[21]:
```
mykernuller = kernuller.kernuller(kernuller.VLTI,5.e-6)
mykernuller.build_classical_model(verbose=False)
```

```
Building a model from scratch
```

4.2 Building a symbolic model

[22]:
```
mykernuller = kernuller.kernuller(kernuller.VLTI,5.e-6)
mykernuller.build_symbolic_model(verbose=False)
fig, axs = mykernuller.legacy_nulled_output(valtheta=sp.pi/3, simplified=False)
```

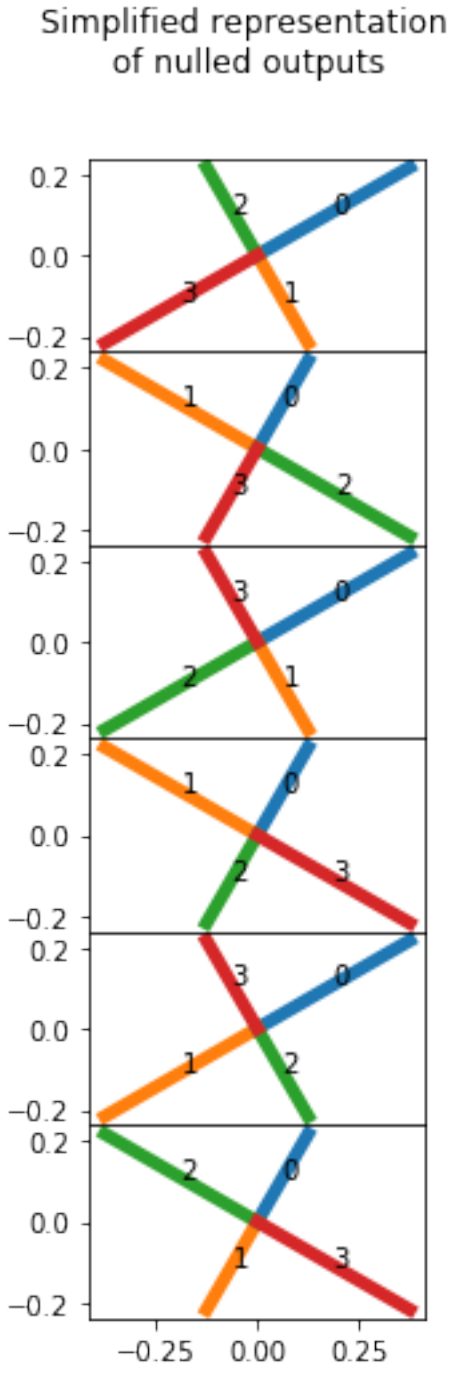
```
Building a model from scratch
[False False False  True  True  True]
```

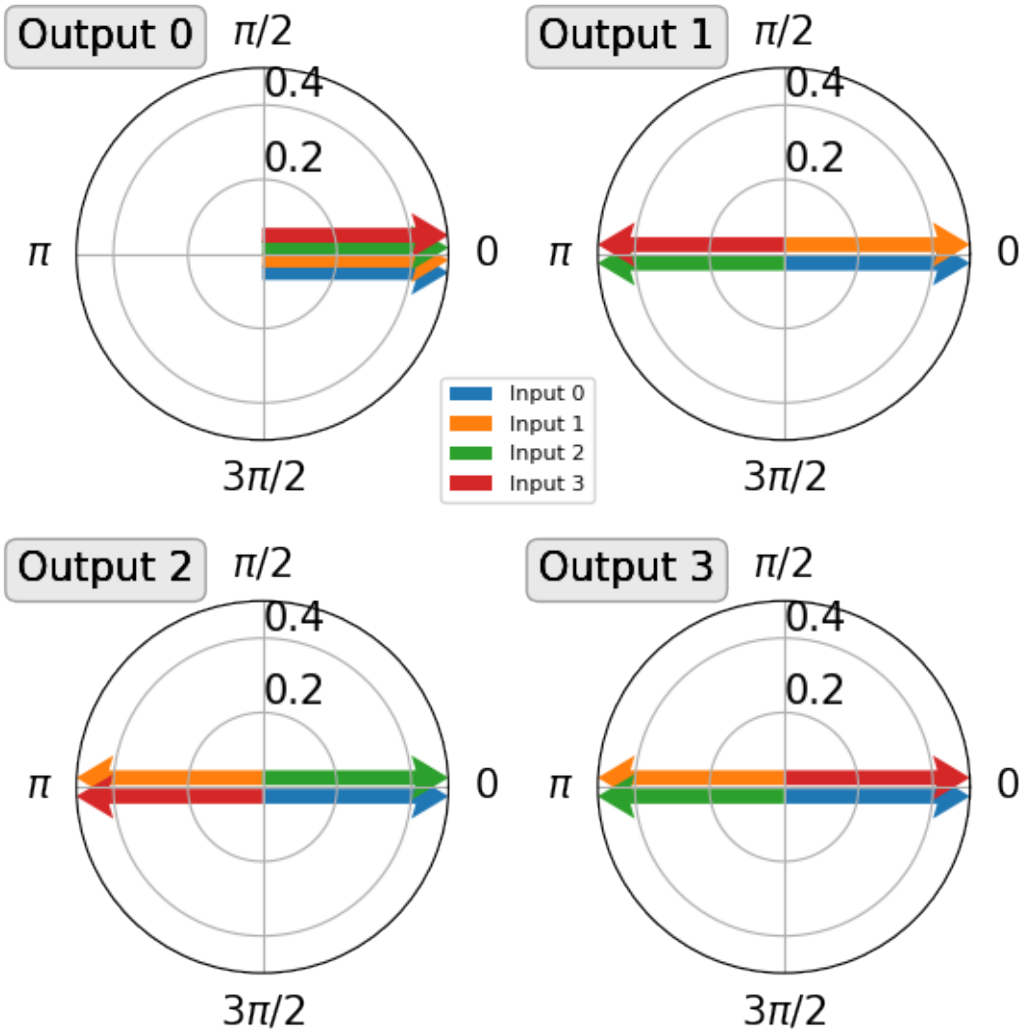


[23]:
```
fig, axs = mykernuller.legacy_nulled_output(valtheta=sp.pi/3, simplified=True)
theM = np.vstack([1/2*np.ones(4),mykernuller.Np])
base_preoffset = np.zeros_like(theM)
base_preoffset[0,:] = np.linspace(0-0.05j,0+0.05j,base_preoffset.shape[1])
semioffset = np.linspace(0-0.025j,0+0.025j,base_preoffset.shape[1]//2)
```

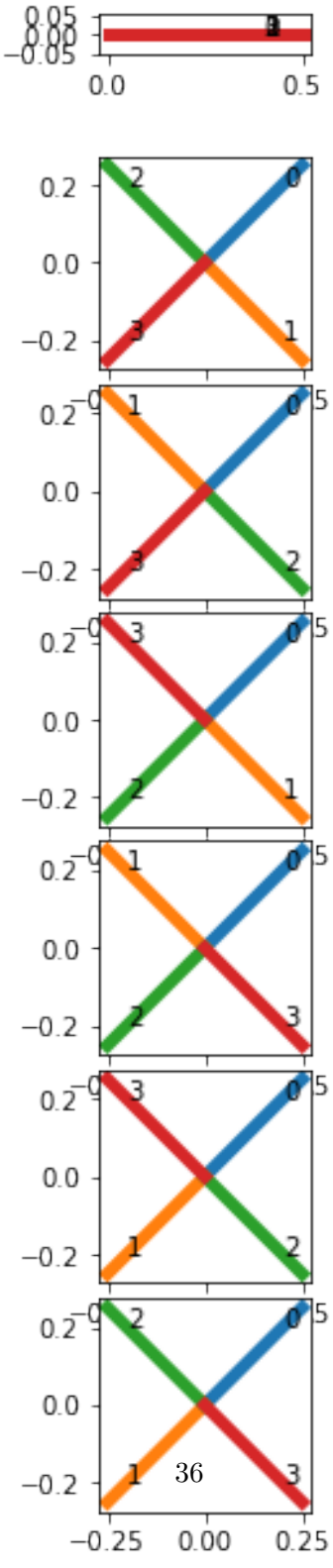
```
base_preoffset[1,:] = np.concatenate((semioffset,semioffset))
base_preoffset[2,:] = np.concatenate((semioffset,semioffset))
base_preoffset[3,:] = np.concatenate((semioffset,semioffset))

outlabels = ["Output %d"%(i) for i in range(4)]
fig, axs = mykernuller.plot_outputs_smart(matrix=theM,␣
 ↪base_preoffset=base_preoffset,
                                          nx=2,legend=True, plotsize=3,␣
 ↪onlyonelegend=True,legendstring="center left",
                                          title=False, legendoffset=(0.95,-0.),␣
 ↪labels=False, rlabelpos=90,
                                          onlyoneticklabel=False,␣
 ↪out_label=outlabels, thealpha=0.1,dpi=100,
                                          )

#fig, axs = mykernuller.legacy_nulled_output(valtheta=sp.pi/2, simplified=True)

fig, axs = mykernuller.plot_nulled_outputs(matrix=np.vstack([1/2*np.
 ↪ones(4),mykernuller.S.dot(mykernuller.Np)]),nx=1)

theM = np.vstack([1/2*np.ones(4), np.zeros(4),mykernuller.S.dot(mykernuller.
 ↪Np)])
base_preoffset = np.zeros_like(theM)
base_preoffset[0,:] = np.linspace(0-0.05j,0+0.05j,base_preoffset.shape[1])
outlabels = ["Output %d"%(i) for i in range(7)]
outlabels.insert(1, "")
fig, axs = mykernuller.plot_outputs_smart(matrix=theM,␣
 ↪base_preoffset=base_preoffset,labels=False,
                                          nx=2,␣
 ↪plotsize=3,onlyonelegend=True,legendoffset=(1.5,0.5), rlabelpos=90,
                                         title=False,legendstring="center left",␣
 ↪onlyoneticklabel=False,
                                       out_label=outlabels, thealpha=0.1, dpi=100)




Mn = np.vstack([1/2*np.ones(4),mykernuller.S.dot(mykernuller.Np)])
u, s, vh = np.linalg.svd(Mn)

print(s)
```

```
removing the labels: [False False False False]
```

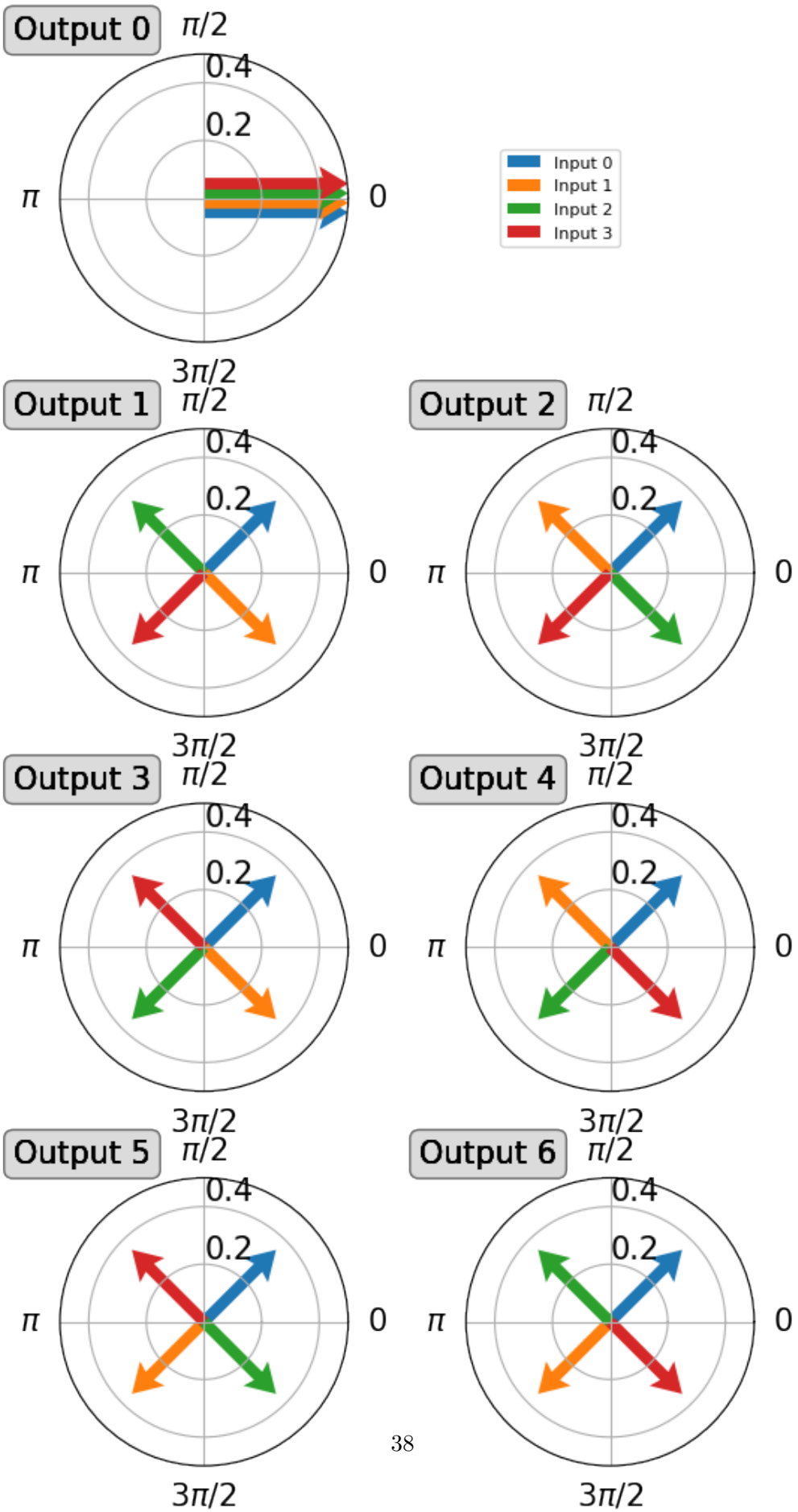

Simplified representation
of nulled outputs
0.2
0.0
−0.2
−0.25
0.00
0.25
0
1
2
3


Output 0
Output 1
Output 2
Output 3
π/2
0.4
0.2
π
0
3π/2
Input 0
Input 1
Input 2
Input 3

The null configurations
of all the 7 outputs



```
removing the labels: [False False False False False False False False]
```


Output 0
Output 1
Output 2
Output 3
Output 4
Output 5
Output 6
π/2
0.4
0.2
π
0
3π/2
Input 0
Input 1
Input 2
Input 3

```
[1. 1. 1. 1.]
```

[ ]:

[ ]: