cmb-map-fit

November 30, 2020

1 Week 13 Problem Set attempt

Ethan Marx, Haochen Wang, Geoffrey Mo 8.942 Fall 2020 Commit: https://github.com/EthanMarx/cmb-map-making/commit/1766d0bfa895f725cf5f7522d0669de2177ce820

[1]:
```
import cmb_utils
import numpy as np
import pyfftw
import pyfftw.interfaces.numpy_fft as fft
from scipy.interpolate import interp1d
from scipy import linalg as LA
from scipy.integrate import trapz
from scipy.special import jv
import matplotlib.pyplot as plt

import camb
import fiducial_parameters as fp
import importlib
```

[2]:
```
pix_width = 0.0015707 # radians
cmb_data = np.load('cmb_analysis_pset_data.npz')
```

[3]:
```
for key in cmb_data.keys():
    print(key)
```

```
test_signal
test_white_noise
test_red_noise
test_x
test_y
data_small
x_small
y_small
data_large
x_large
y_large
```

[4]:
```
cmb_data['test_red_noise'].shape
```

[4]: (65536,)

2 1. Map making

2.1 1.1 Operators

All steps in this section are already provided in the starter code (cmb_utils). Section 1.1 in the pset is rather an explanation for what the starter code (more speficially the NoisePointingModel class) does.

2.2 1.2 Estimating the time stream noise power spectrum

First, we choose a data set to be processed.

[5]:
```
dt_test = cmb_data['test_signal'] + cmb_data['test_white_noise'] +␣
↪cmb_data['test_red_noise'] # test data
x_test = np.round(cmb_data['test_x']/pix_width).astype(int) # x coordinates of␣
↪the test data, starting with 0
y_test = np.round(cmb_data['test_y']/pix_width).astype(int) # y coordinates of␣
↪the test data, starting with 0
nx_test = np.amax(x_test) + 1 # Map size in x direction (plus 1 counting the␣
↪0th index)
ny_test = np.amax(y_test) + 1 # Map size in y direction (plus 1 counting the␣
↪0th index)
nt_test = len(x_test) # total number of data points
```

[6]:
```
print(nx_test, ny_test, nt_test)
```

32 32 65536

1. FFT d_t and divide by $\sqrt{n_t}$ to obtain d_ω.

[7]:
```
dt = dt_test
x = x_test
y = y_test
nx = nx_test
ny = ny_test
nt = nt_test
```

[8]:
```
d_omega = fft.rfft(dt)/np.sqrt(nt) # fft with real data as input
omega = fft.rfftfreq(nt)*2*np.pi # angular frequencies
```

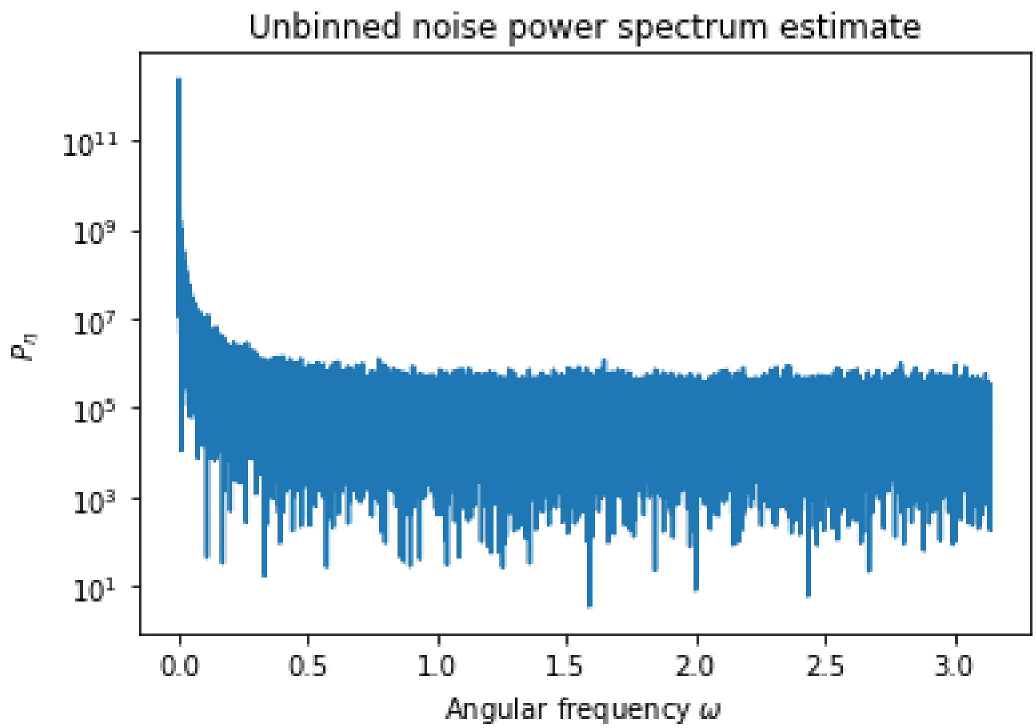
2. The quantity $d_\omega d_\omega^*$ is then a noisy estimate for $P_\eta(\omega)$.

[9]:
```
P_eta_unbinned = d_omega*np.conjugate(d_omega)
```

[10]:
```
plt.plot(omega[:], P_eta_unbinned[:])
plt.title("Unbinned noise power spectrum estimate")
plt.yscale('log')
plt.xlabel(r'Angular frequency $\omega$')
plt.ylabel(r'$P_\eta$')
```

```
/Users/gmo/miniconda3/envs/cosmo-perturb/lib/python3.7/site-
packages/numpy/core/_asarray.py:83: ComplexWarning: Casting complex values to
real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

[10]:
```
Text(0, 0.5, '$P_\\eta$')
```



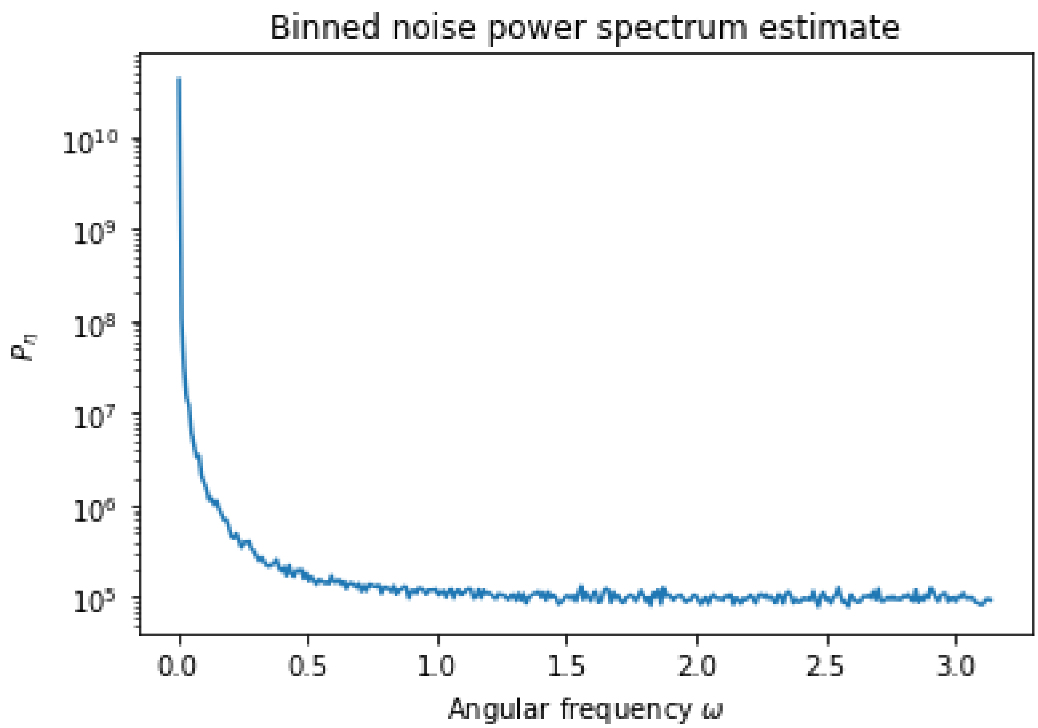
3. Accumulate the estimate over bins in ω to reduce uncertainty

[11]:
```
n_omega = len(omega) # number of frequencies
bins = np.linspace(omega[0], omega[-1], num = int(n_omega/100), endpoint =␣
↪True) # bin edges, each bin contains about 100 points
P_eta_binned = np.histogram(omega, bins, weights=P_eta_unbinned)[0]/np.
↪histogram(omega, bins)[0]
bin_centers = np.array([(bins[i] + bins[i+1])/2 for i in range(len(bins) -1)])
```

[12]:
```
plt.plot(bin_centers[:], P_eta_binned[:])
plt.title("Binned noise power spectrum estimate")
plt.yscale('log')
plt.xlabel(r'Angular frequency $\omega$')
plt.ylabel(r'$P_\eta$')
```

```
/Users/gmo/miniconda3/envs/cosmo-perturb/lib/python3.7/site-
packages/numpy/core/_asarray.py:83: ComplexWarning: Casting complex values to
real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

[12]:
```
Text(0, 0.5, '$P_\\eta$')
```



4. Interpolate/extrapolate the result to any ω.

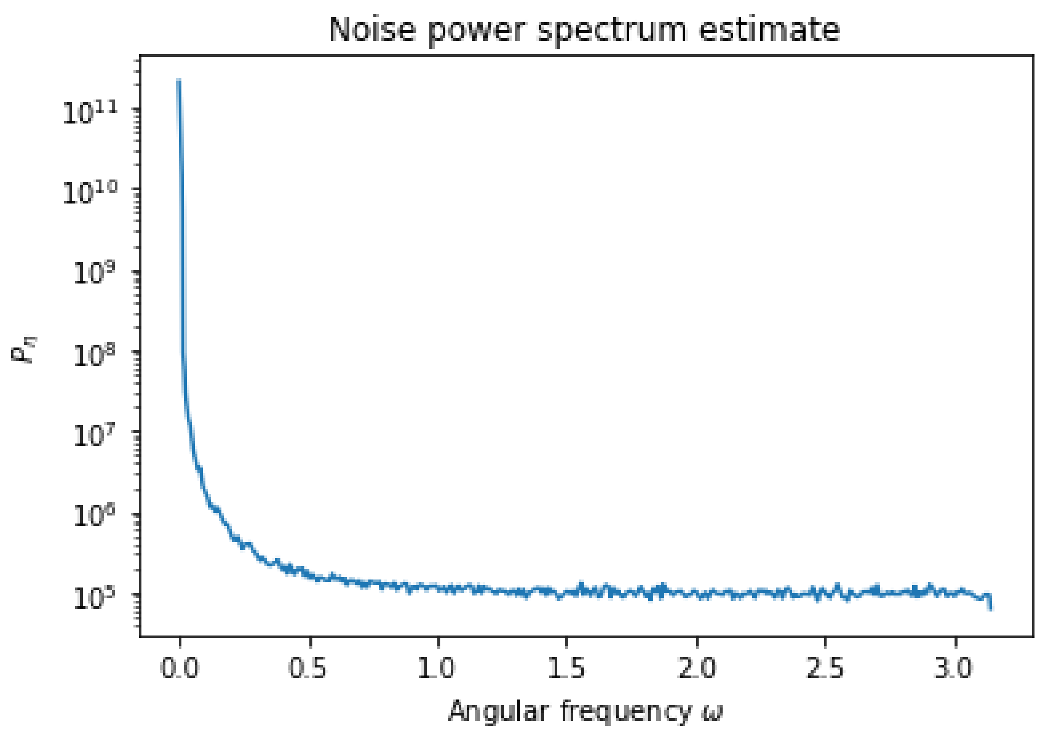
[13]:
```
# Before we interpolate, we have to add the first and last data point.␣
↪Otherwise interpolator will return "out of range"
first = np.sum(P_eta_unbinned[0:20])/20 # estimate for P_eta at omega[0], 20 is␣
↪arbitrarily chosen
last = np.sum(P_eta_unbinned[-20:])/20 # estimate for P_eta at omega[-1]
omega_for_interp = np.insert(np.append(bin_centers, omega[-1]), 0, omega[0])
P_eta_for_interp = np.insert(np.append(P_eta_binned, last), 0, first)
```

[14]:
```
P_eta_interp = interp1d(omega_for_interp, P_eta_for_interp)
P_eta_final = P_eta_interp(omega) # use interpolator to get P_eta at orignial␣
↪omegas but now it is much smoother
```

[15]:
```
plt.plot(omega, P_eta_final)
plt.title("Noise power spectrum estimate")
plt.yscale('log')
plt.xlabel(r'Angular frequency $\omega$')
plt.ylabel(r'$P_\eta$')
```

```
/Users/gmo/miniconda3/envs/cosmo-perturb/lib/python3.7/site-
packages/numpy/core/_asarray.py:83: ComplexWarning: Casting complex values to
real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

[15]:
```
Text(0, 0.5, '$P_\\eta$')
```



2.3 1.3 Noise covariance inverse

Obtain C_N^{-1}

```
[16]: model_test = cmb_utils.NoisePointingModel(x_test, y_test, nx_test, ny_test,␣
       ↪P_eta_final)
      CN_inv_test = model_test.map_noise_inv()
      CN_inv_test.shape
```

```
x-index 0
x-index 1
x-index 2
x-index 3
x-index 4
x-index 5
x-index 6
x-index 7
x-index 8
x-index 9
x-index 10
x-index 11
x-index 12
x-index 13
x-index 14
x-index 15
x-index 16
x-index 17
x-index 18
x-index 19
x-index 20
x-index 21
x-index 22
x-index 23
x-index 24
x-index 25
x-index 26
x-index 27
x-index 28
x-index 29
x-index 30
x-index 31
```

```
[16]: (32, 32, 32, 32)
```

Reshape C_N^{-1} into a 2D matrix.

```
[17]: CN_inv_test_reshape = np.reshape(CN_inv_test, (nx*ny, nx*ny))
      CN_inv_test_reshape.shape
```

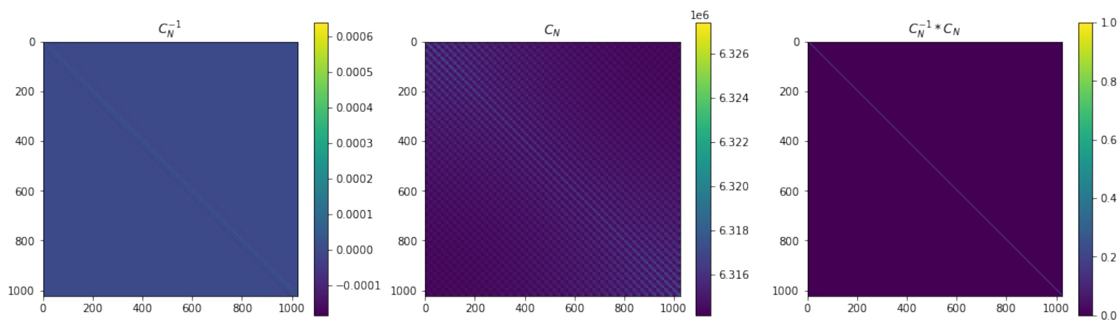
```
[17]: (1024, 1024)
```

Invert to obtain C_N.

[18]:
```
CN_test = LA.inv(CN_inv_test_reshape)
```

[19]:
```
plt.figure(figsize=(18,5))
plt.subplot(131)
plt.imshow(CN_inv_test_reshape)
plt.colorbar()
plt.title(r'$C_N^{-1}$')
plt.subplot(132)
plt.imshow(CN_test)
plt.colorbar()
plt.title(r'$C_N$')
plt.subplot(133)
plt.imshow(np.dot(CN_test,CN_inv_test_reshape))
plt.colorbar()
plt.title(r'$C_N^{-1}*C_N$')
```

[19]:
```
Text(0.5, 1.0, '$C_N^{-1}*C_N$')
```



Get the signal estimate with D&S (14.30)

[20]:
```
N_inv_d_test = model_test.apply_noise_weights(dt)
out = np.zeros((nx, ny), dtype=float)
P_T_N_inv_d_test = model_test.grid_data(N_inv_d_test, out)
```

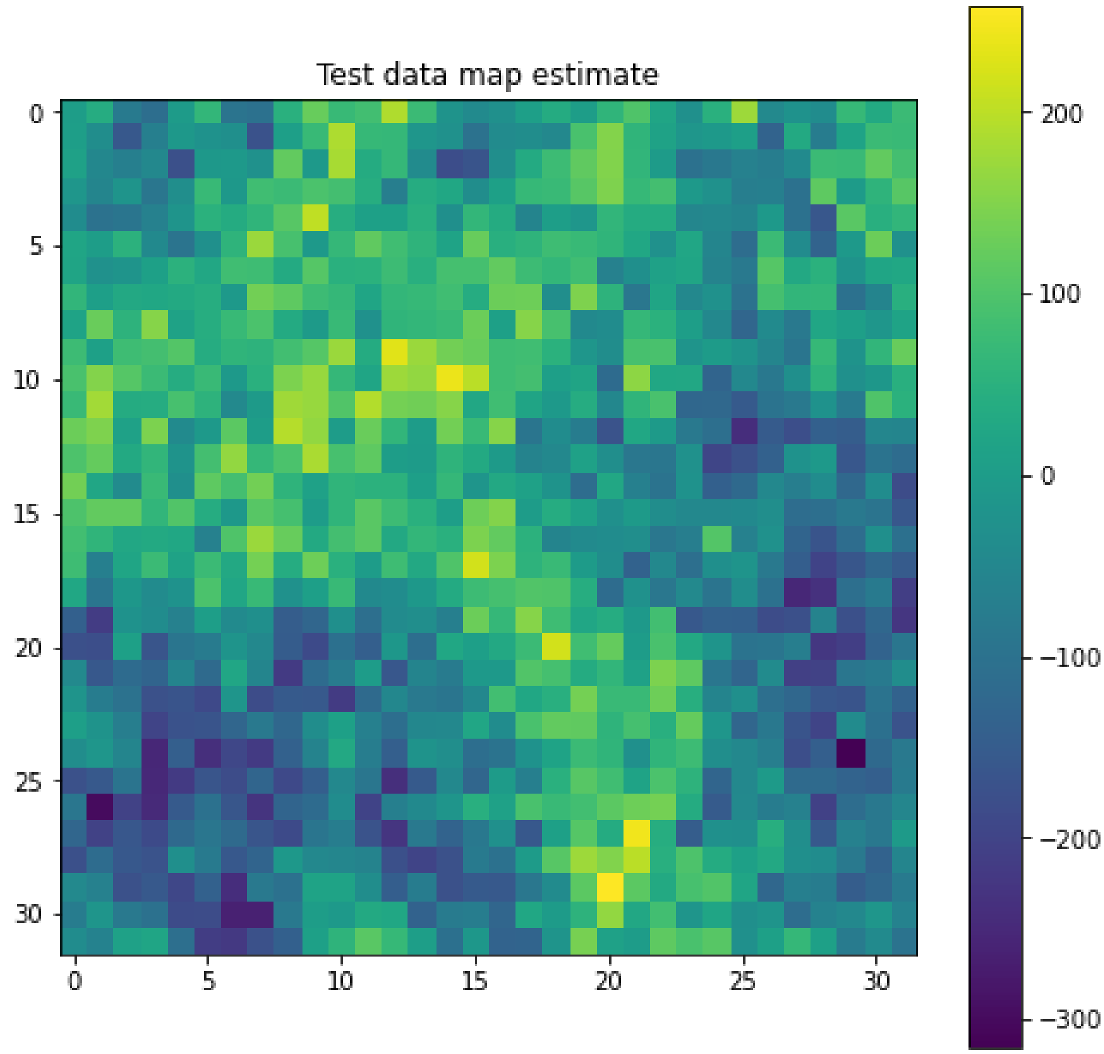
[21]:
```
P_T_N_inv_d_test.shape
```

[21]:
```
(32, 32)
```

[22]:
```
s_hat_test = np.dot(CN_test, P_T_N_inv_d_test.flatten())
s_hat_test = np.reshape(s_hat_test, (nx, ny))
```

[23]:
```
plt.figure(figsize=(8,8))
plt.imshow(s_hat_test)
plt.colorbar()
plt.title('Test data map estimate')
```

[23]: Text(0.5, 1.0, 'Test data map estimate')



3 1.4 Testing

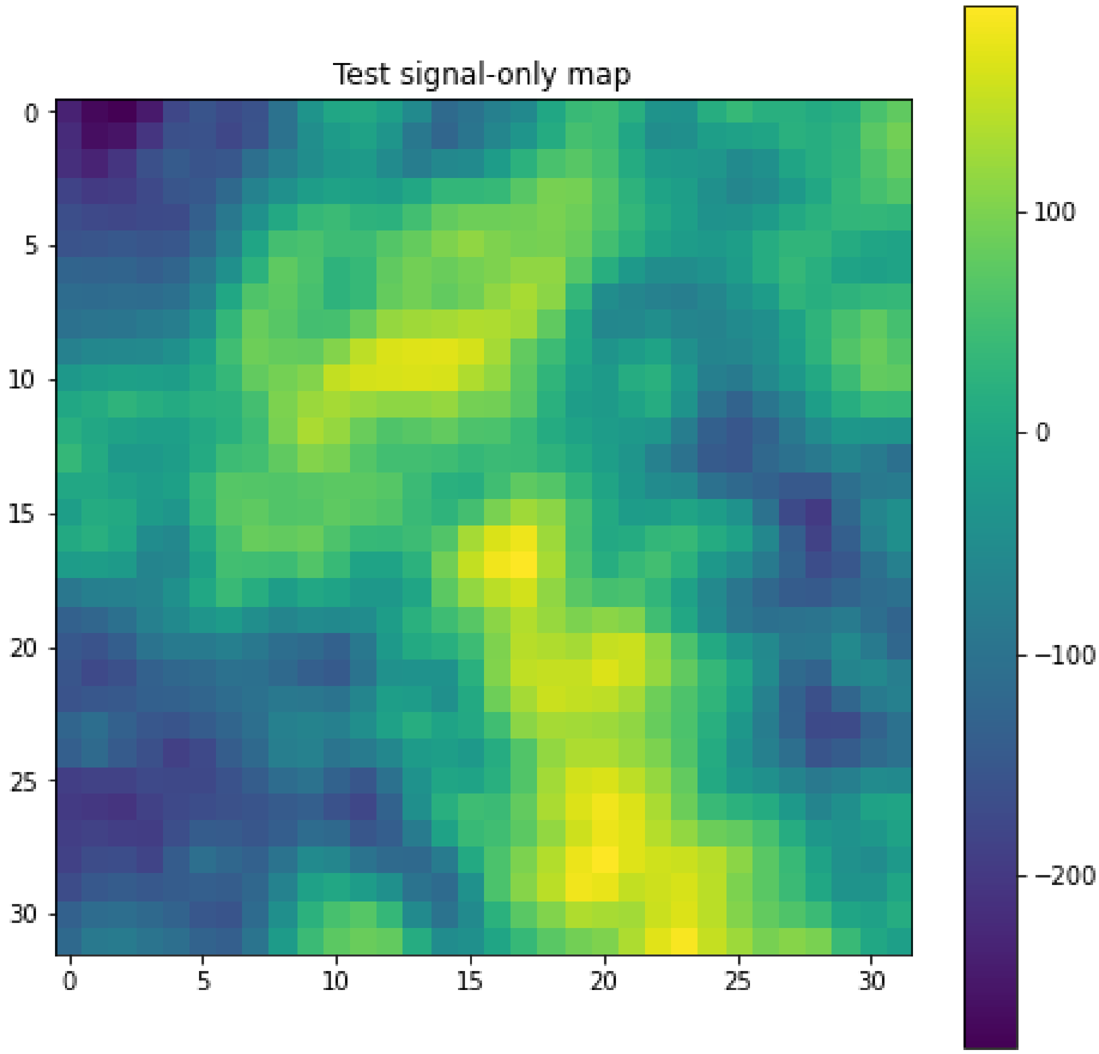
Check the result with test signal only and construct the map with D&S (14.33)

[24]:
```
dt_signal = cmb_data['test_signal']
```

[25]:
```
s_hat_test_signal = np.zeros((nx, ny), dtype=float)
m = np.zeros((nx, ny), dtype=float)
np.add.at(s_hat_test_signal, (x, y), dt_signal)
np.add.at(m, (x, y), 1)
s_hat_test_signal = s_hat_test_signal/m
```

[26]:
```
plt.figure(figsize=(8,8))
plt.imshow(s_hat_test_signal)
plt.colorbar()
plt.title('Test signal-only map')
```

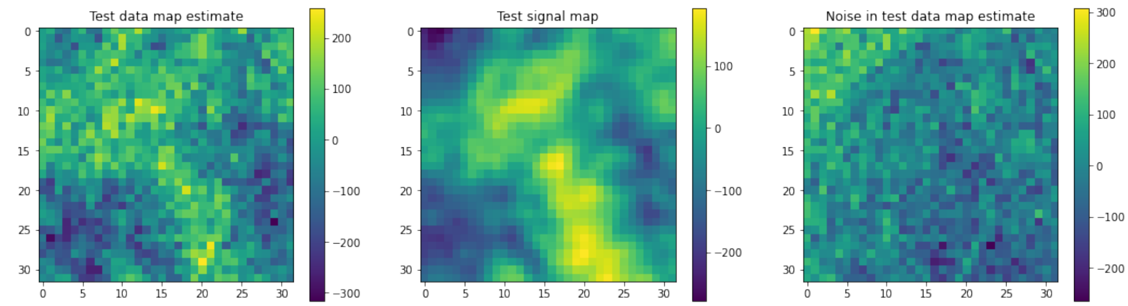
[26]: Text(0.5, 1.0, 'Test signal-only map')



[27]:
```
plt.figure(figsize=(18,5))
plt.subplot(131)
plt.imshow(s_hat_test)
plt.colorbar()
plt.title('Test data map estimate')
plt.subplot(132)
plt.imshow(s_hat_test_signal)
```

```
plt.colorbar()
plt.title('Test signal map')
plt.subplot(133)
plt.imshow(s_hat_test - s_hat_test_signal)
plt.colorbar()
plt.title('Noise in test data map estimate')
```

[27]: Text(0.5, 1.0, 'Noise in test data map estimate')



4 2. Power Spectrum estimation

In this section, we are looking to estimate the band powers, c^α

Applying Dodelson 1st edition eqn 11.94 in this context gives:

$$\hat{c^\alpha} = c_0^\alpha + F_{\alpha\beta}^{-1}\frac{sC^{-1}C_{,\beta}C^{-1}s - Tr(C^{-1}C_{,\beta})}{2} \tag{1}$$

where

$$F_{\alpha\beta} = \frac{Tr(C_{,\alpha}C^{-1}C_{,\beta}C^{-1})}{2} \tag{2}$$

$$C = C_S + C_N \tag{3}$$

s is our signal map created in the previous part, and c_0^α is calculated from a fiducial model, in our case, CAMB

We have already calculated C_N in the previous part, we just need C_S, which enters into the equation for $\hat{c^\alpha}$ in two parts:

1. C^{-1}
2. $C_{,\alpha}$

4.1 2.1 Signal Covariance Matrix

To account for C_S in the calculation of C^{-1}, we will use our fiducial cosmology to fix C_l and apply:

$$C_S = \bar{T}^2 \omega(|\theta_i - \theta_j|, \lambda_\alpha) \tag{4}$$

with

$$\omega(\theta, \lambda_\alpha) = \int_0^\infty \frac{dl}{2\pi} C_l(\lambda_\alpha) J_0(l\theta) \tag{5}$$

[40]:
```
# Get C_l from CAMB
pars = cmb_utils.fast_camb_settings() # Use this from what Kiyo gave us
results = camb.get_results(pars)

#get dictionary of CAMB power spectra
powers = results.get_cmb_power_spectra(pars, CMB_unit='muK')
totCL = powers['total']
ls = np.arange(totCL.shape[0])
Cls = totCL[:,0] # this goes out to l = 1500 per the settings
```

[47]:
```
# calculate angular correlation function

# calculate theta values between all pixels
thetas = np.zeros((nx,ny,nx,ny), dtype=float) # initalize array
# FIXME: get the thetas
thetas = thetas.flatten()

Tbar = 2.725e6 # need to calc this, maybe? for now, this is the known value in␣
↪microK
theta_beam = 0.000667 # radians, as specified in pdf
# Eq. 4, 5, and 6 from the pset
C_S = np.array([Tbar**2 *  1/(2*np.pi) * trapz(jv(0, ls * theta) * Cls * np.
↪exp(-ls**2 * theta_beam**2)) for theta in thetas])
#C_S = np.reshape(C_S, (nx*ny,nx*ny))
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-47-308f7f74fc73> in <module>()
      9 theta_beam = 0.000667 # radians
     10 #C_S = [Tbar**2 *  1/(2*np.pi) * trapz(jv(0,ls*theta) * Cls * np.exp(-,␣
↪ls) for theta in thetas]
---> 11 C_S = np.array([Tbar**2 *  1/(2*np.pi) * trapz(jv(0, ls * theta) * Cls *␣
↪np.exp(-ls**2 * theta_beam**2)) for theta in thetas])
     12 #C_S = np.reshape(C_S, (nx*ny,nx*ny))
```

```
<ipython-input-47-308f7f74fc73> in <listcomp>(.0)
      9 theta_beam = 0.000667 # radians
     10 #C_S = [Tbar**2 *  1/(2*np.pi) * trapz(jv(0,ls*theta) * Cls * np.exp(-␣
↪ls) for theta in thetas]
---> 11 C_S = np.array([Tbar**2 *  1/(2*np.pi) * trapz(jv(0, ls * theta) * Cls * ␣
↪np.exp(-ls**2 * theta_beam**2)) for theta in thetas])
     12 #C_S = np.reshape(C_S, (nx*ny,nx*ny))

KeyboardInterrupt:
```

[48]:
```
thetas
```

[48]:
```
array([0., 0., 0., …, 0., 0., 0.])
```

4.2 2.2 Band Powers

To account for C_S in the calculation of $C_{,\alpha}$ we will use band powers c^α as our parameters:

$$C_l^{obs} \approx \sum_\alpha c^\alpha E_\alpha(l) \tag{6}$$

where

$$E_\alpha(l) = \begin{cases} 1 & l_\alpha^{low} \leq l < l_{\alpha+1}^{low} \\ 0 & otherwise \end{cases} \tag{7}$$

It can then be shown that

$$C_{,\alpha} = \int_0^\infty \frac{dl}{2\pi} E_\alpha(l) J_0(l\theta) \tag{8}$$

[55]:
```
# create logarithmically spaced l bins to use for bandpowers
lmin = 0
lmax = pars.max_l # get from CAMB parameters
l_bins = np.logspace(lmin, np.log10(lmax), num=30) # 30 bins as suggested by␣
↪Kiyo
```

[56]:
```
l_bins
```

[56]:
```
array([1.00000000e+00, 1.28682766e+00, 1.65592543e+00, 2.13089065e+00,
       2.74208904e+00, 3.52859603e+00, 4.54069498e+00, 5.84309191e+00,
       7.51905231e+00, 9.67572450e+00, 1.24509899e+01, 1.60222783e+01,
       2.06179109e+01, 2.65316981e+01, 3.41417231e+01, 4.39345137e+01,
       5.65361476e+01, 7.27522786e+01, 9.36196447e+01, 1.20472349e+02,
       1.55027151e+02, 1.99493226e+02, 2.56713402e+02, 3.30345907e+02,
       4.25098251e+02, 5.47028188e+02, 7.03931005e+02, 9.05837890e+02,
       1.16565725e+03, 1.50000000e+03])
```

```
[ ]:
```