plot_DSA2000_sensitivity

March 18, 2024

1 DSA-2000 Sensitivity Plotting

This notebook generates plots for Byrne et al. 2024 “21 cm Intensity Mapping with the DSA-2000”. Author: Ruby Byrne, Caltech First published March 2024. Contents: 1. Basic Sensitivity Analysis 2. BAO Analysis 3. Generate PSF Image

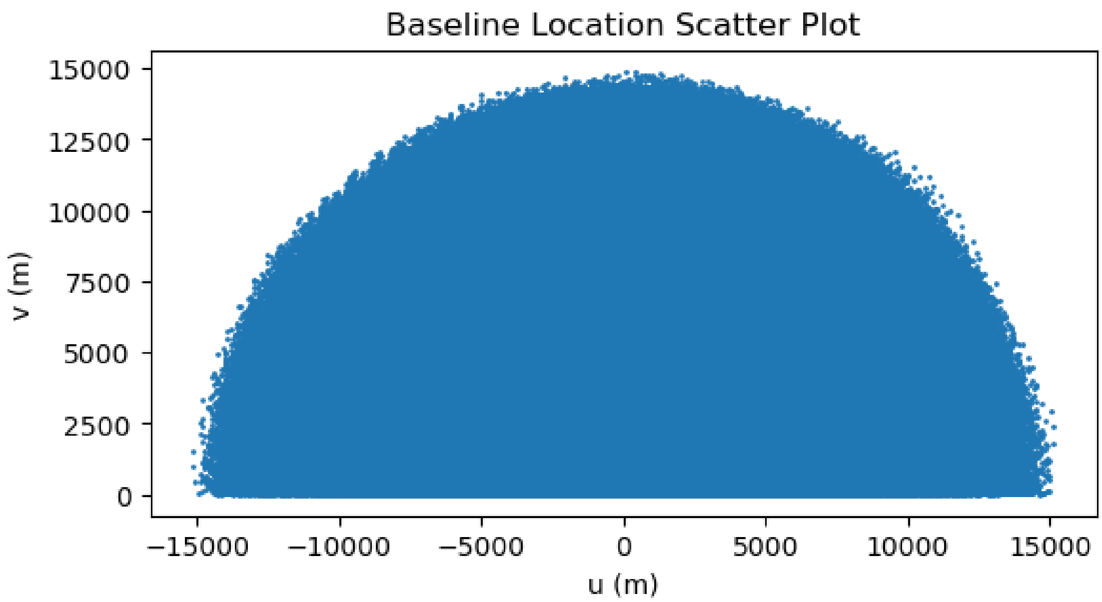
1.1 1. Basic Sensitivity Analysis

[1]:
```
import numpy as np
import pyuvdata
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.lines import Line2D
import array_sensitivity
import importlib
importlib.reload(array_sensitivity)
from matplotlib.patches import Rectangle
import scipy
import seaborn
from scipy import interpolate
```

[2]:
```
# Set instrument parameters
antpos_filepath = "W2-17.cfg"
c = 3e8
min_freq_hz = 0.7e9
max_freq_hz = c / 0.21
freq_hz = np.mean([min_freq_hz, max_freq_hz])
tsys_k = 25
aperture_efficiency = 0.7
field_of_view_deg2 = 30.0
antenna_diameter_m = 5
freq_resolution_hz = 130.2e3
int_time_s = 15.0 * 60  # 15 minutes in each survey field
max_bl_m = 1000
bao_scales_k = np.array([.03, .2]) / 0.71
```

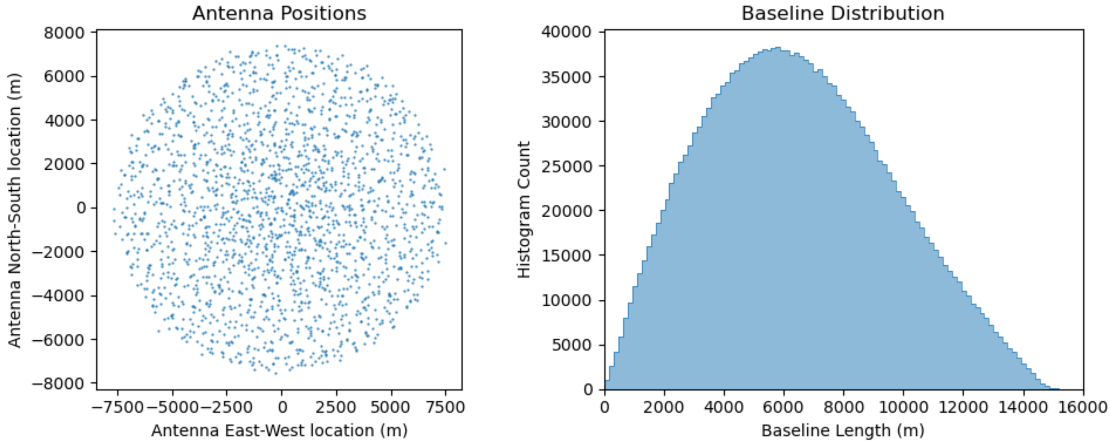
[3]:
```
antpos = array_sensitivity.get_antpos(antpos_filepath)
```

[4]:
```
# Get baselines
baselines_m = array_sensitivity.get_baselines(antpos)
fig, ax = plt.subplots()
ax.scatter(baselines_m[:,0], baselines_m[:,1], s=1)
ax.set_aspect(1)
ax.set_xlabel("u (m)")
ax.set_ylabel("v (m)")
ax.set_title("Baseline Location Scatter Plot")
plt.show()
```



[5]:
```
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))

ax[0].plot(antpos[:,0], antpos[:,1], marker="o", markersize=.5, linewidth=0,␣
 ↪color="tab:blue")
ax[0].set_aspect(1)
ax[0].set_xlabel("Antenna East-West location (m)")
ax[0].set_ylabel("Antenna North-South location (m)")
ax[0].set_title("Antenna Positions")
ax[0].set_aspect("equal")

ax[1].hist(
    np.sqrt(np.sum(baselines_m**2., axis=1)),
    bins=100, color="tab:blue",
    alpha=0.5,
)
ax[1].hist(
```

```python
    np.sqrt(np.sum(baselines_m**2., axis=1)),
    bins=100,
    linewidth=.5,
    edgecolor="tab:blue",
    histtype="step"
)
ax[1].set_xlabel("Baseline Length (m)")
ax[1].set_ylabel("Histogram Count")
ax[1].set_title("Baseline Distribution")
ax[1].set_xlim([0,16000])

plt.tight_layout()
plt.savefig("plots/antlocs.png", dpi=600)
plt.show()
```



[6]:
```python
# Define bin edges:
freq_array_hz = np.arange(min_freq_hz, max_freq_hz, freq_resolution_hz)
delay_array_s = np.fft.fftshift(
    np.fft.fftfreq(len(freq_array_hz), d=freq_resolution_hz)
)
kpar_conv_factor = array_sensitivity.get_kpar_conversion_factor(freq_hz)
max_kpar = kpar_conv_factor * np.max(delay_array_s)

kperp_conv_factor = array_sensitivity.get_kperp_conversion_factor(freq_hz)
max_baseline_wl = np.max(np.sqrt(np.sum(baselines_m**2.0, axis=1))) *␣
 ↪max_freq_hz / c
max_kperp = kperp_conv_factor * max_baseline_wl
max_k = np.sqrt(max_kpar**2.0 + max_kperp**2.0)

k_bin_size = 0.1
min_k = 0.02
```

```
bin_edges = np.arange(min_k, max_k, k_bin_size)
kpar_bin_edges = np.arange(0, max_kpar, k_bin_size)
kperp_bin_edges = np.arange(0, max_kperp, k_bin_size)
```

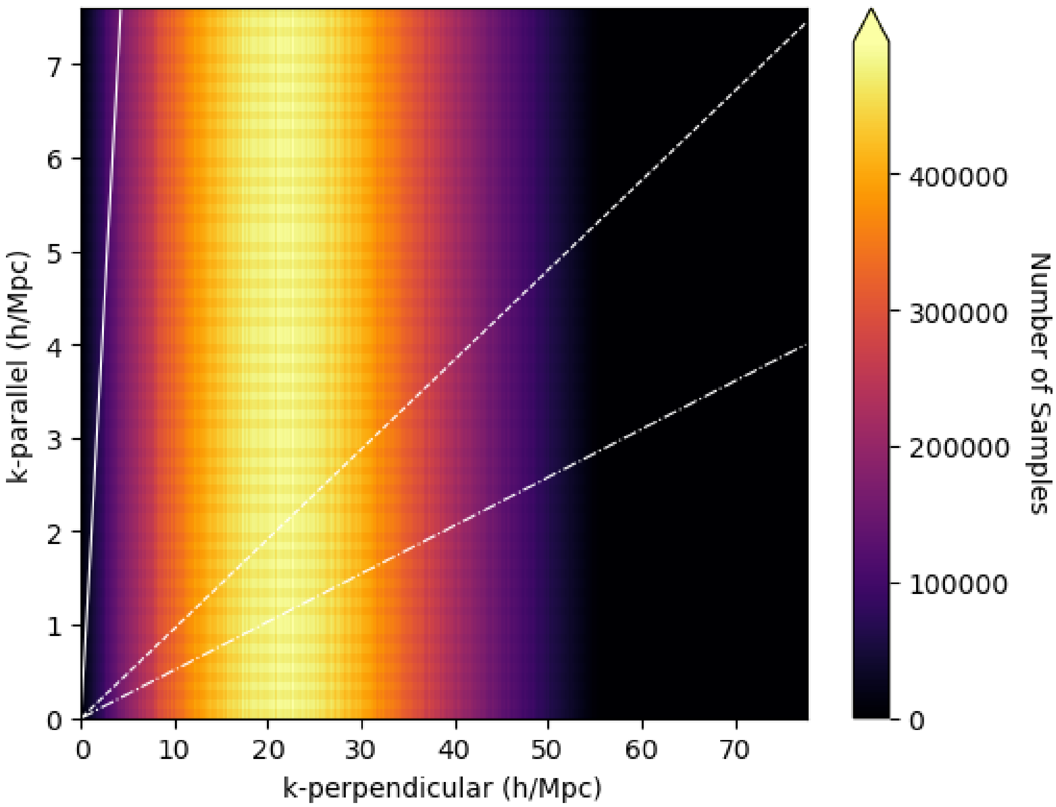
[7]:
```
# Restore simulation with horizon wedge cut
with open("simulation_outputs/thermal_noise_no_core_wedge_cut_horizon_za_0.0.
 ↪npy", "rb") as f:
    nsamples_horizon_cut = np.load(f)
    binned_ps_variance_horizon_cut = np.load(f)
    true_bin_edges_horizon_cut = np.load(f)
    true_bin_centers_horizon_cut = np.load(f)
    nsamples_2d_restored = np.load(f)
    binned_ps_variance_2d_restored = np.load(f)
f.close()
```

[8]:
```
# Restore simulation with FoV wedge cut
with open("simulation_outputs/thermal_noise_no_core_wedge_cut_fov_za_0.0.npy",␣
 ↪"rb") as f:
    nsamples_fov_cut = np.load(f)
    binned_ps_variance_fov_cut = np.load(f)
    true_bin_edges_fov_cut = np.load(f)
    true_bin_centers_fov_cut = np.load(f)
f.close()
```

[9]:
```
# Account for 2 polarizations
binned_ps_variance_horizon_cut /= 4
binned_ps_variance_fov_cut /= 4
```

[10]:
```
# Make 2D Nsamples plot from restored data

# Get wedge slope
kperp_conv_factor = array_sensitivity.get_kperp_conversion_factor(freq_hz)
kpar_conv_factor = array_sensitivity.get_kpar_conversion_factor(freq_hz)
wedge_slope = kpar_conv_factor / (kperp_conv_factor * freq_hz)

field_of_view_radius = 3.09
fov_wedge_slope = wedge_slope * np.sin(np.radians(field_of_view_radius))
half_max_radius = 1.66
half_max_slope = wedge_slope * np.sin(np.radians(half_max_radius))

fig, ax = plt.subplots()
use_cmap = cm.get_cmap("inferno").copy()
cax = ax.imshow(
    nsamples_2d_restored.T,
    origin="lower",
    interpolation="none",
    extent=[
```

```
        np.min(kperp_bin_edges),
        np.max(kperp_bin_edges),
        np.min(kpar_bin_edges),
        np.max(kpar_bin_edges)
    ],
    vmin=0,
    vmax=np.max(nsamples_2d_restored),
    cmap=use_cmap,
    #norm="log",
    aspect=10.,
)
ax.set_xlabel("k-perpendicular (h/Mpc)")
ax.set_ylabel("k-parallel (h/Mpc)")
ax.set_xlim([0, np.max(kperp_bin_edges)])
ax.set_ylim([0, np.max(kpar_bin_edges)])
plt.plot([0, np.max(kperp_bin_edges)], [0, wedge_slope*np.
 ↪max(kperp_bin_edges)], c="white", linewidth=.8)
plt.plot([0, np.max(kperp_bin_edges)], [0, fov_wedge_slope*np.
 ↪max(kperp_bin_edges)], "--", c="white", linewidth=.8)
plt.plot([0, np.max(kperp_bin_edges)], [0, half_max_slope*np.
 ↪max(kperp_bin_edges)], "-.", c="white", linewidth=.8)
#ax.grid(linewidth=.5)
cbar = fig.colorbar(cax, extend="max")
cbar.set_label("Number of Samples", rotation=270, labelpad=15)
plt.savefig("plots/2d_nsampes.png", dpi=300)
plt.show()
```



[11]:

```
# Run horizon cut simulation in notebook
"""
(
    nsamples_horizon_cut,
    binned_ps_variance_horizon_cut,
    true_bin_edges_horizon_cut,
    true_bin_centers_horizon_cut,
    nsamples_2d_horizon_cut,
    binned_ps_variance_2d_horizon_cut,
) = array_sensitivity.delay_ps_sensitivity_analysis(
    antpos_filepath=antpos_filepath,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    tsys_k=tsys_k,
    aperture_efficiency=aperture_efficiency,
    antenna_diameter_m=antenna_diameter_m,
    freq_resolution_hz=freq_resolution_hz,
    int_time_s=int_time_s,
    max_bl_m=max_bl_m,
    k_bin_edges_1d=bin_edges,
    kpar_bin_edges=kpar_bin_edges,
```

```
    kperp_bin_edges=kperp_bin_edges,
    wedge_extent_deg=90.0,
    zenith_angle=0.0,
)
"""
```

[11]: '\n(\n   nsamples_horizon_cut,\n   binned_ps_variance_horizon_cut,\n   true_bin_edges_horizon_cut,\n   true_bin_centers_horizon_cut,\n   nsamples_2d_horizon_cut,\n   binned_ps_variance_2d_horizon_cut,\n) = array_sensitivity.delay_ps_sensitivity_analysis(\n   antpos_filepath=antpos_filepath,\n   min_freq_hz=min_freq_hz,\n   max_freq_hz=max_freq_hz,\n   tsys_k=tsys_k,\n   aperture_efficiency=aperture_efficiency,\n   antenna_diameter_m=antenna_diameter_m,\n   freq_resolution_hz=freq_resolution_hz,\n   int_time_s=int_time_s,\n   max_bl_m=max_bl_m,\n   k_bin_edges_1d=bin_edges,\n   kpar_bin_edges=kpar_bin_edges,\n   kperp_bin_edges=kperp_bin_edges,\n   wedge_extent_deg=90.0,\n   zenith_angle=0.0,\n)\n'

[12]:
```
# Run FoV cut simulation in notebook
"""
(
    nsamples_fov_cut,
    binned_ps_variance_fov_cut,
    true_bin_edges_fov_cut,
    true_bin_centers_fov_cut,
    nsamples_2d_fov_cut,
    binned_ps_variance_2d_fov_cut,
) = array_sensitivity.delay_ps_sensitivity_analysis(
    antpos_filepath=antpos_filepath,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    tsys_k=tsys_k,
    aperture_efficiency=aperture_efficiency,
    antenna_diameter_m=antenna_diameter_m,
    freq_resolution_hz=freq_resolution_hz,
    int_time_s=int_time_s,
    max_bl_m=max_bl_m,
    k_bin_edges_1d=bin_edges,
    kpar_bin_edges=kpar_bin_edges,
    kperp_bin_edges=kperp_bin_edges,
    wedge_extent_deg=1.84,
    zenith_angle=0.0,
)
"""
```

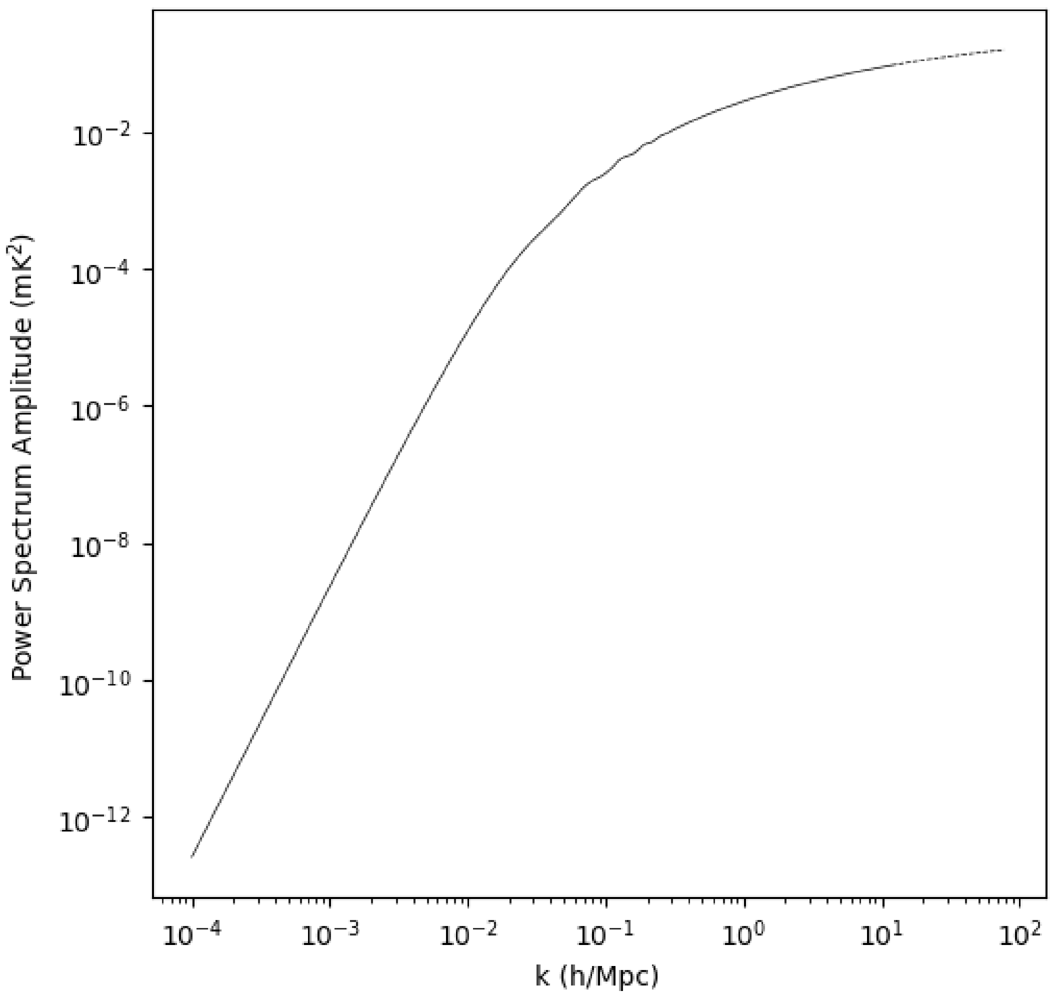
```
[12]: '\n(\n    nsamples_fov_cut,\n    binned_ps_variance_fov_cut,\n
true_bin_edges_fov_cut,\n    true_bin_centers_fov_cut,\n
nsamples_2d_fov_cut,\n    binned_ps_variance_2d_fov_cut,\n) =
array_sensitivity.delay_ps_sensitivity_analysis(\n
antpos_filepath=antpos_filepath,\n    min_freq_hz=min_freq_hz,\n
max_freq_hz=max_freq_hz,\n    tsys_k=tsys_k,\n
aperture_efficiency=aperture_efficiency,\n
antenna_diameter_m=antenna_diameter_m,\n
freq_resolution_hz=freq_resolution_hz,\n    int_time_s=int_time_s,\n
max_bl_m=max_bl_m,\n    k_bin_edges_1d=bin_edges,\n
kpar_bin_edges=kpar_bin_edges,\n    kperp_bin_edges=kperp_bin_edges,\n
wedge_extent_deg=1.84,\n    zenith_angle=0.0,\n)\n'
```

[13]:
```
# Load CAMB power spectrum data
f = open("camb_49591724_matterpower_z0.5.dat", "r")
file_data = f.readlines()
f.close()
model_k_axis = []
ps_model_unnorm = []
for line in file_data:
    model_k_axis.append(float(line.split()[0]))
    ps_model_unnorm.append(float(line.split()[1]))
ps_model = array_sensitivity.matter_ps_to_21cm_ps_conversion(
    np.array(model_k_axis),
    np.array(ps_model_unnorm),
    0.5
)
print(f"Maximum theory value: {np.max(model_k_axis)}")
```

```
Maximum theory value: 11.818
```

[14]:
```
# Extrapolate  CAMB power spectrum to higher k values
model_k_axis_extrapolate = kperp_bin_edges[np.where(kperp_bin_edges>np.
 ↪max(model_k_axis))]
ps_model_extrapolate_fn = interpolate.interp1d(np.log(model_k_axis), ps_model,␣
 ↪fill_value = "extrapolate")
ps_model_extrapolate = ps_model_extrapolate_fn(np.log(model_k_axis_extrapolate))

model_k_axis_extended = np.concatenate((model_k_axis, model_k_axis_extrapolate))
ps_model_extended = np.concatenate((ps_model, ps_model_extrapolate))

# Plot theory line
fig, ax = plt.subplots(figsize=(6,6))
ax.plot(model_k_axis, ps_model, color="black", marker="none", linewidth=0.5)
ax.plot(model_k_axis_extrapolate, ps_model_extrapolate, "--", color="black",␣
 ↪marker="none", linewidth=0.5)
ax.set_yscale("log")
```

```
ax.set_xscale("log")
ax.set_xlabel("k (h/Mpc)")
ax.set_ylabel("Power Spectrum Amplitude (mK$^2$)")
plt.show()
```

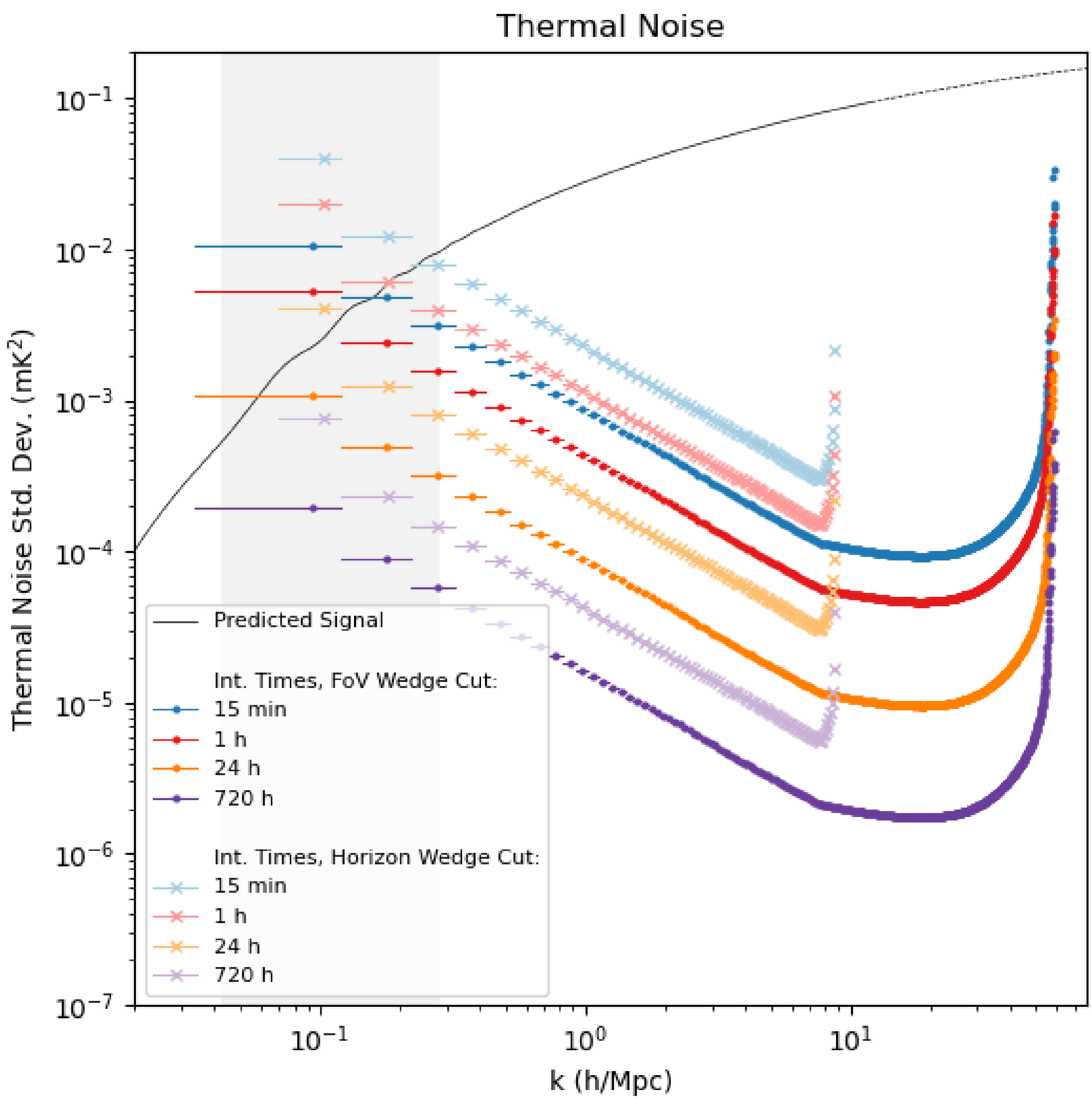


[15]:
```
plot_integration_times_h = [.25, 1, 24, 720]
colors = np.array([
    [seaborn.color_palette("Paired")[1], seaborn.color_palette("Paired")[5],␣
  ↪seaborn.color_palette("Paired")[7], seaborn.color_palette("Paired")[9]],
    [seaborn.color_palette("Paired")[0], seaborn.color_palette("Paired")[4],␣
  ↪seaborn.color_palette("Paired")[6], seaborn.color_palette("Paired")[8]],
])
markers = ["o", "x"]
marker_sizes = [2, 5]
time_int_labels = [
```

```
    "15 min",
    "1 h",
    "24 h",
    "720 h",
]
legend_headings = ["Int. Times, FoV Wedge Cut:", "Int. Times, Horizon Wedge Cut:
↪"]
#xrange = [2e-2, 10]
xrange = [2e-2, np.max(kperp_bin_edges)]
yrange = [1e-7, 2e-1]

fig, ax = plt.subplots(figsize=(6,6))

# Plot theory line
ax.plot(model_k_axis, ps_model, color="black", marker="none", linewidth=0.5)
ax.plot(model_k_axis_extrapolate, ps_model_extrapolate, linestyle="dashed",␣
↪color="black", marker="none", linewidth=0.5)
legend_lines = [Line2D([0], [0], linewidth=0.5, marker="none", color="black")]
legend_labels = ["Predicted Signal"]

# Plot BAO scales
ax.fill_between(bao_scales_k, [yrange[0], yrange[0]], [yrange[1], yrange[1]],␣
↪color="grey", alpha=.1, linewidth=0)

for wedge_cut_ind in range(2):
    legend_lines = legend_lines + [Line2D([0], [0], linewidth=0), Line2D([0],␣
↪[0], linewidth=0)]
    legend_labels = legend_labels + ["", legend_headings[wedge_cut_ind]]
    if wedge_cut_ind == 0:
        use_binned_ps_variance = binned_ps_variance_fov_cut
        use_true_bin_centers = true_bin_centers_fov_cut
        use_true_bin_edges = true_bin_edges_fov_cut
    else:
        use_binned_ps_variance = binned_ps_variance_horizon_cut
        use_true_bin_centers = true_bin_centers_horizon_cut
        use_true_bin_edges = true_bin_edges_horizon_cut

    for int_time_ind, int_time in enumerate(plot_integration_times_h):
        legend_lines = legend_lines + [Line2D([0], [0], linewidth=0.8,␣
↪marker=markers[wedge_cut_ind], markersize=marker_sizes[wedge_cut_ind],␣
↪color=colors[wedge_cut_ind, int_time_ind, :])]
        legend_labels = legend_labels + [time_int_labels[int_time_ind]]

        plot_vals = np.sqrt(use_binned_ps_variance * .25 / int_time)
        ax.plot(
            use_true_bin_centers,
            plot_vals,
```

```
            marker=markers[wedge_cut_ind],␣
↪markersize=marker_sizes[wedge_cut_ind], linewidth=0,
            color=colors[wedge_cut_ind, int_time_ind, :],
        )
        for bin_ind in range(len(plot_vals)):
            ax.plot(
                use_true_bin_edges[bin_ind, :],
                [plot_vals[bin_ind], plot_vals[bin_ind]],
                marker="none",
                linewidth=0.8,
                color=colors[wedge_cut_ind, int_time_ind, :],
            )

ax.set_yscale("log")
ax.set_xscale("log")
ax.set_xlim(xrange)
ax.set_ylim(yrange)
ax.set_xlabel("k (h/Mpc)")
ax.set_ylabel("Thermal Noise Std. Dev. (mK$^2$)")
ax.legend(legend_lines, legend_labels, prop={'size': 8})
ax.set_title("Thermal Noise")

plt.tight_layout()
plt.savefig("plots/thermal_noise_stddev.png", dpi=300)
plt.show()
```

```
/var/folders/x0/sh3xmymj56x6t_0516hx76zw0000gn/T/ipykernel_15118/2049844408.py:7
1: UserWarning: Creating legend with loc="best" can be slow with large amounts
of data.
  plt.tight_layout()
/var/folders/x0/sh3xmymj56x6t_0516hx76zw0000gn/T/ipykernel_15118/2049844408.py:7
2: UserWarning: Creating legend with loc="best" can be slow with large amounts
of data.
  plt.savefig("plots/thermal_noise_stddev.png", dpi=300)
/Users/ruby/opt/anaconda3/lib/python3.8/site-
packages/IPython/core/pylabtools.py:152: UserWarning: Creating legend with
loc="best" can be slow with large amounts of data.
  fig.canvas.print_figure(bytes_io, **kw)
```



[16]:
```
binned_ps_sample_variance_horizon_cut = array_sensitivity.get_sample_variance(
    ps_model_extended,  # Units mK^2
    model_k_axis_extended,  # Units h/Mpc
    field_of_view_deg2=field_of_view_deg2,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    freq_resolution_hz=freq_resolution_hz,
    k_bin_edges=bin_edges,
    wedge_extent_deg=90.0,
)
binned_ps_sample_variance_fov_cut = array_sensitivity.get_sample_variance(
    ps_model_extended,  # Units mK^2
    model_k_axis_extended,  # Units h/Mpc
```

```
    field_of_view_deg2=field_of_view_deg2,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    freq_resolution_hz=freq_resolution_hz,
    k_bin_edges=bin_edges,
    wedge_extent_deg=3.09,
)
```

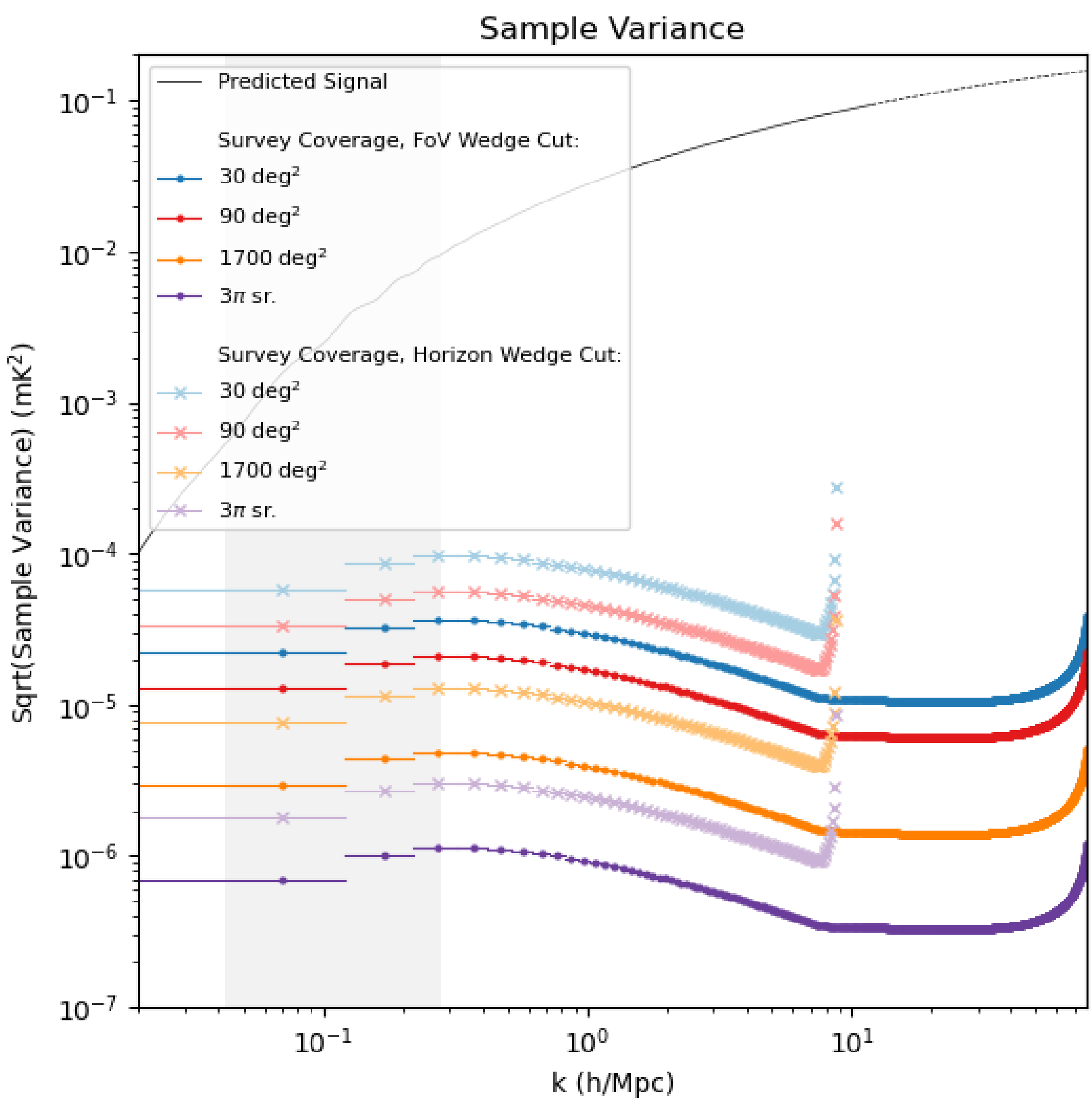
```
Kpar correlation length: 0.002736362210334458
Kperp correlation length: 0.015331664324187
Correlation volume: 6.432091122141715e-07
Kpar correlation length: 0.002736362210334458
Kperp correlation length: 0.015331664324187
Correlation volume: 6.432091122141715e-07
```

[17]:
```
plot_fov_deg = [30.0, 90.0, 1700.0, 3*np.pi*(180/np.pi)**2.]

colors = np.array([
    [seaborn.color_palette("Paired")[1], seaborn.color_palette("Paired")[5],␣
 ↪seaborn.color_palette("Paired")[7], seaborn.color_palette("Paired")[9]],
    [seaborn.color_palette("Paired")[0], seaborn.color_palette("Paired")[4],␣
 ↪seaborn.color_palette("Paired")[6], seaborn.color_palette("Paired")[8]],
])
markers = ["o", "x"]
marker_sizes = [2, 5]
fov_int_labels = [
    "30 deg$^2$",
    "90 deg$^2$",
    "1700 deg$^2$",
    "3$\pi$ sr.",
]
legend_headings = ["Survey Coverage, FoV Wedge Cut:", "Survey Coverage, Horizon␣
 ↪Wedge Cut:"]
xrange = [2e-2, np.max(kperp_bin_edges)]
yrange = [1e-7, 2e-1]

fig, ax = plt.subplots(figsize=(6,6))

# Plot theory line
ax.plot(model_k_axis, ps_model, color="black", marker="none", linewidth=0.5)
ax.plot(model_k_axis_extrapolate, ps_model_extrapolate, linestyle="dashed",␣
 ↪color="black", marker="none", linewidth=0.5)
legend_lines = [Line2D([0], [0], linewidth=0.5, marker="none", color="black")]
legend_labels = ["Predicted Signal"]

# Plot BAO scales
```

```python
ax.fill_between(bao_scales_k, [yrange[0], yrange[0]], [yrange[1], yrange[1]], color="grey", alpha=.1, linewidth=0)

for wedge_cut_ind in range(2):
    if wedge_cut_ind == 0:
        use_binned_ps_sample_variance = binned_ps_sample_variance_fov_cut
    else:
        use_binned_ps_sample_variance = binned_ps_sample_variance_horizon_cut

    legend_lines = legend_lines + [Line2D([0], [0], linewidth=0), Line2D([0], [0], linewidth=0)]
    legend_labels = legend_labels + ["", legend_headings[wedge_cut_ind]]

    for fov_ind, use_fov in enumerate(plot_fov_deg):
        legend_lines = legend_lines + [Line2D([0], [0], linewidth=0.8, marker=markers[wedge_cut_ind], markersize=marker_sizes[wedge_cut_ind], color=colors[wedge_cut_ind, fov_ind, :])]
        legend_labels = legend_labels + [fov_int_labels[fov_ind]]

        plot_vals = np.sqrt(use_binned_ps_sample_variance * field_of_view_deg2 / use_fov)
        ax.plot(
            (bin_edges[:-1] + bin_edges[1:]) / 2,
            plot_vals,
            marker=markers[wedge_cut_ind], markersize=marker_sizes[wedge_cut_ind], linewidth=0,
            color=colors[wedge_cut_ind, fov_ind, :],
        )
        for bin_ind in range(len(plot_vals)):
            ax.plot(
                [bin_edges[bin_ind], bin_edges[bin_ind + 1]],
                [plot_vals[bin_ind], plot_vals[bin_ind]],
                marker="none",
                linewidth=0.8,
                color=colors[wedge_cut_ind, fov_ind, :],
            )

ax.set_yscale("log")
ax.set_xscale("log")
ax.set_xlim(xrange)
ax.set_ylim(yrange)
ax.set_xlabel("k (h/Mpc)")
ax.set_ylabel("Sqrt(Sample Variance) (mK$^2$)")
ax.legend(legend_lines, legend_labels, prop={'size': 8})
ax.set_title("Sample Variance")

plt.tight_layout()
```

```
plt.savefig("plots/sample_stddev.png", dpi=300)
plt.show()
```

```
/var/folders/x0/sh3xmymj56x6t_0516hx76zw0000gn/T/ipykernel_15118/3566822657.py:4
3: RuntimeWarning: invalid value encountered in sqrt
  plot_vals = np.sqrt(use_binned_ps_sample_variance * field_of_view_deg2 /
use_fov)
/var/folders/x0/sh3xmymj56x6t_0516hx76zw0000gn/T/ipykernel_15118/3566822657.py:6
8: UserWarning: Creating legend with loc="best" can be slow with large amounts
of data.
  plt.tight_layout()
/var/folders/x0/sh3xmymj56x6t_0516hx76zw0000gn/T/ipykernel_15118/3566822657.py:6
9: UserWarning: Creating legend with loc="best" can be slow with large amounts
of data.
  plt.savefig("plots/sample_stddev.png", dpi=300)
```



[18]:
```
binned_ps_shot_noise_horizon_cut = array_sensitivity.get_shot_noise(
    field_of_view_deg2=field_of_view_deg2,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    freq_resolution_hz=freq_resolution_hz,
    k_bin_edges=bin_edges,
    wedge_extent_deg=90.0,
)
binned_ps_shot_noise_fov_cut = array_sensitivity.get_shot_noise(
    field_of_view_deg2=field_of_view_deg2,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    freq_resolution_hz=freq_resolution_hz,
    k_bin_edges=bin_edges,
    wedge_extent_deg=3.09,
)
```

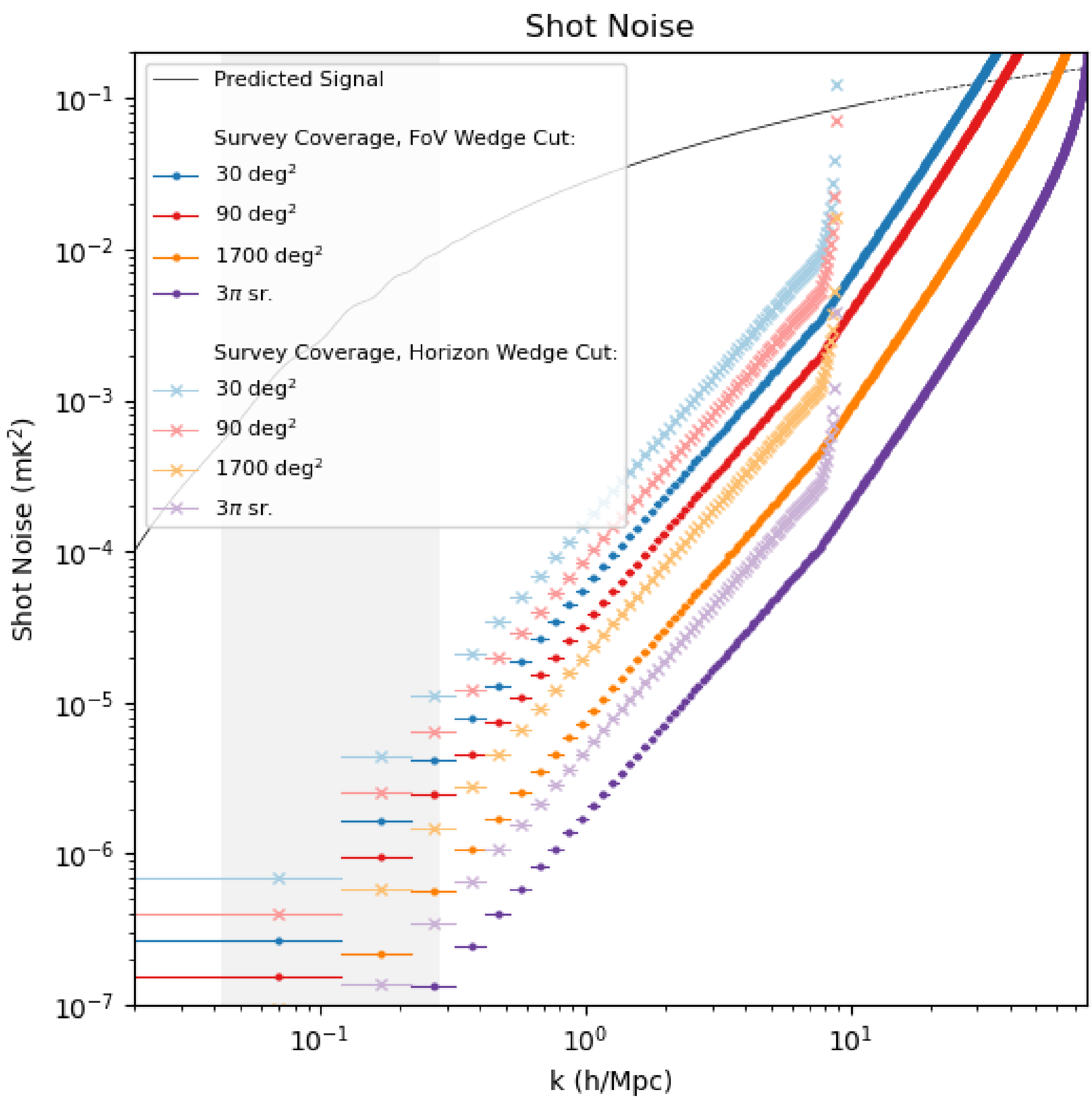
```
Kpar correlation length: 0.002736362210334458
Kperp correlation length: 0.015331664324187
Correlation volume: 6.432091122141715e-07
Kpar correlation length: 0.002736362210334458
Kperp correlation length: 0.015331664324187
Correlation volume: 6.432091122141715e-07
```

[19]:
```
plot_fov_deg = [30.0, 90.0, 1700.0, 3*np.pi*(180/np.pi)**2.]

colors = np.array([
    [seaborn.color_palette("Paired")[1], seaborn.color_palette("Paired")[5],␣
↪seaborn.color_palette("Paired")[7], seaborn.color_palette("Paired")[9]],
    [seaborn.color_palette("Paired")[0], seaborn.color_palette("Paired")[4],␣
↪seaborn.color_palette("Paired")[6], seaborn.color_palette("Paired")[8]],
])
markers = ["o", "x"]
marker_sizes = [2, 5]
fov_int_labels = [
    "30 deg$^2$",
    "90 deg$^2$",
    "1700 deg$^2$",
    "3$\pi$ sr.",
]
legend_headings = ["Survey Coverage, FoV Wedge Cut:", "Survey Coverage, Horizon␣
↪Wedge Cut:"]
xrange = [2e-2, np.max(kperp_bin_edges)]
yrange = [1e-7, 2e-1]
```

```python
fig, ax = plt.subplots(figsize=(6,6))

# Plot theory line
ax.plot(model_k_axis, ps_model, color="black", marker="none", linewidth=0.5)
ax.plot(model_k_axis_extrapolate, ps_model_extrapolate, linestyle="dashed", color="black", marker="none", linewidth=0.5)
legend_lines = [Line2D([0], [0], linewidth=0.5, marker="none", color="black")]
legend_labels = ["Predicted Signal"]

# Plot BAO scales
ax.fill_between(bao_scales_k, [yrange[0], yrange[0]], [yrange[1], yrange[1]], color="grey", alpha=.1, linewidth=0)

for wedge_cut_ind in range(2):
    if wedge_cut_ind == 0:
        use_binned_ps_shot_noise = binned_ps_shot_noise_fov_cut
    else:
        use_binned_ps_shot_noise = binned_ps_shot_noise_horizon_cut

    legend_lines = legend_lines + [Line2D([0], [0], linewidth=0), Line2D([0], [0], linewidth=0)]
    legend_labels = legend_labels + ["", legend_headings[wedge_cut_ind]]

    for fov_ind, use_fov in enumerate(plot_fov_deg):
        legend_lines = legend_lines + [Line2D([0], [0], linewidth=0.8, marker=markers[wedge_cut_ind], markersize=marker_sizes[wedge_cut_ind], color=colors[wedge_cut_ind, fov_ind, :])]
        legend_labels = legend_labels + [fov_int_labels[fov_ind]]

        plot_vals = np.sqrt(use_binned_ps_shot_noise * field_of_view_deg2 / use_fov)
        ax.plot(
            (bin_edges[:-1] + bin_edges[1:]) / 2,
            plot_vals,
            marker=markers[wedge_cut_ind], markersize=marker_sizes[wedge_cut_ind], linewidth=0,
            color=colors[wedge_cut_ind, fov_ind, :],
        )
        for bin_ind in range(len(plot_vals)):
            ax.plot(
                [bin_edges[bin_ind], bin_edges[bin_ind + 1]],
                [plot_vals[bin_ind], plot_vals[bin_ind]],
                marker="none",
                linewidth=0.8,
                color=colors[wedge_cut_ind, fov_ind, :],
            )
```

```
ax.set_yscale("log")
ax.set_xscale("log")
ax.set_xlim(xrange)
ax.set_ylim(yrange)
ax.set_xlabel("k (h/Mpc)")
ax.set_ylabel("Shot Noise (mK$^2$)")
ax.legend(legend_lines, legend_labels, prop={'size': 8}, loc="upper left")
ax.set_title("Shot Noise")

plt.tight_layout()
plt.savefig("plots/shot_noise.png", dpi=300)
plt.show()
```

```
/var/folders/x0/sh3xmymj56x6t_05l6hx76zw0000gn/T/ipykernel_15118/3575698639.py:4
3: RuntimeWarning: invalid value encountered in sqrt
  plot_vals = np.sqrt(use_binned_ps_shot_noise * field_of_view_deg2 / use_fov)
```



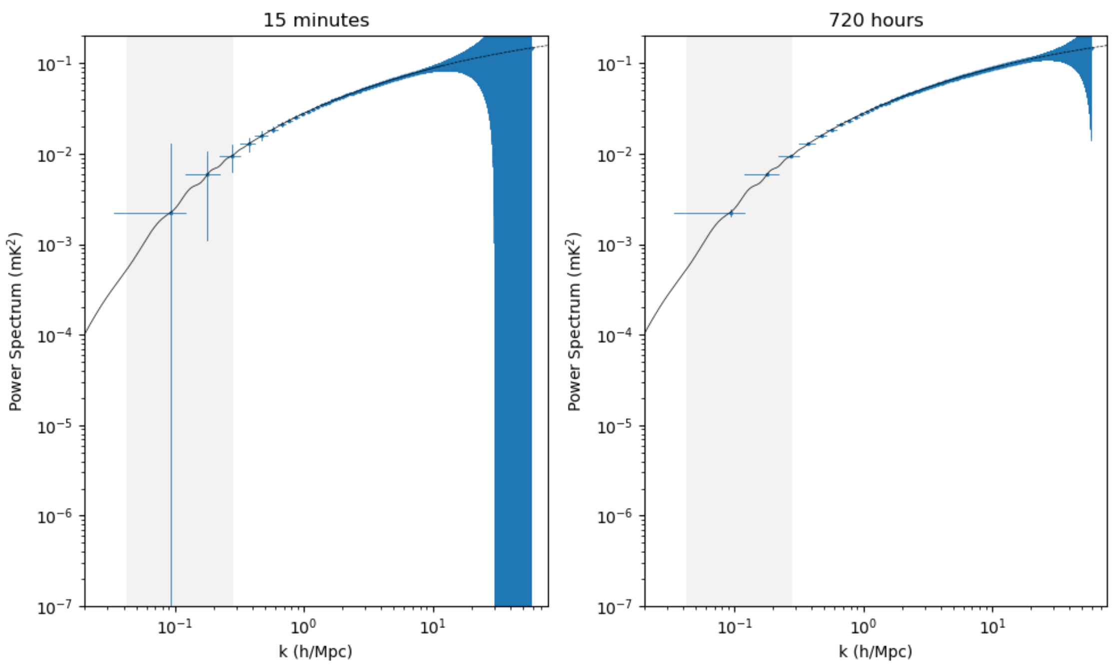
[20]:
```
combined_variance_fov_cut = (
    (binned_ps_variance_fov_cut * .25 / 720)
    + ((binned_ps_sample_variance_fov_cut + binned_ps_shot_noise_fov_cut) *␣
 ↪field_of_view_deg2 / 1700.0)
)
combined_variance_horizon_cut = (
    (binned_ps_variance_horizon_cut * .25 / 720)
    + ((binned_ps_sample_variance_horizon_cut +␣
 ↪binned_ps_shot_noise_horizon_cut) * field_of_view_deg2 / 1700.0)
)
combined_variance_fov_cut_15_min = binned_ps_variance_fov_cut +␣
 ↪binned_ps_sample_variance_fov_cut + binned_ps_shot_noise_fov_cut
```

```
combined_variance_horizon_cut_15_min = binned_ps_variance_horizon_cut +␣
↪binned_ps_sample_variance_horizon_cut + binned_ps_shot_noise_horizon_cut
```

[21]:
```
ps_model_interp = np.interp(true_bin_centers_fov_cut, model_k_axis_extended,␣
↪ps_model_extended)
```

[22]:
```
xrange = [2e-2, np.max(kperp_bin_edges)]
yrange = [1e-7, 2e-1]

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,6))

for int_time_ind in range(2):
    if int_time_ind == 0:
        var_use = combined_variance_fov_cut_15_min
        title = "15 minutes"
    else:
        var_use = combined_variance_fov_cut
        title = "720 hours"

    # Plot BAO scales
    ax[int_time_ind].fill_between(bao_scales_k, [yrange[0], yrange[0]],␣
↪[yrange[1], yrange[1]], color="grey", alpha=.1, linewidth=0)

    ax[int_time_ind].plot(true_bin_centers_fov_cut, ps_model_interp,␣
↪marker="o", color="tab:blue", linewidth=0, markersize=1.5)
    for ind in range(len(ps_model_interp)):
        xvals = [true_bin_centers_fov_cut[ind], true_bin_centers_fov_cut[ind]]
        yvals = [
            ps_model_interp[ind] - np.sqrt(var_use[ind]),
            ps_model_interp[ind] + np.sqrt(var_use[ind])
        ]
        ax[int_time_ind].plot(
            xvals,
            yvals,
            color="tab:blue",
            linewidth=0.6,
            marker="none"
        )
        ax[int_time_ind].plot(
            true_bin_edges_fov_cut[ind, :],
            [ps_model_interp[ind], ps_model_interp[ind]],
            marker="none",
            linewidth=0.6,
            color="tab:blue",
        )

    # Plot theory line
```

```
    ax[int_time_ind].plot(model_k_axis, ps_model, color="black", marker="none",␣
↪linewidth=0.5)
    ax[int_time_ind].plot(model_k_axis_extrapolate, ps_model_extrapolate,␣
↪linestyle="dashed", color="black", marker="none", linewidth=0.5)

    ax[int_time_ind].set_yscale("log")
    ax[int_time_ind].set_xscale("log")
    ax[int_time_ind].set_xlim(xrange)
    ax[int_time_ind].set_ylim(yrange)
    ax[int_time_ind].set_xlabel("k (h/Mpc)")
    ax[int_time_ind].set_ylabel("Power Spectrum (mK$^2$)")
    ax[int_time_ind].set_title(title)
plt.tight_layout()
plt.savefig("plots/error_bars.png", dpi=300)
plt.show()
```



[23]:
```
detected_inds = np.where(ps_model_interp > 5*np.
↪sqrt(combined_variance_fov_cut_15_min))
print(f"Min mode: {np.min(true_bin_edges_fov_cut[detected_inds, 0])}")
print(f"Max mode: {np.max(true_bin_edges_fov_cut[detected_inds, 1])}")
#print(ps_model_interp - 5*np.sqrt(combined_variance_fov_cut))
```

```
Min mode: 0.32000004289525574
Max mode: 15.31999999158581
```

[24]:
```
detected_inds = np.where(ps_model_interp > 5*np.sqrt(combined_variance_fov_cut))
print(f"Min mode: {np.min(true_bin_edges_fov_cut[detected_inds, 0])}")
print(f"Max mode: {np.max(true_bin_edges_fov_cut[detected_inds, 1])}")
```

```
Min mode: 0.033749221465163834
Max mode: 35.119999984964025
```

1.2 2. BAO Analysis

[25]:
```
convolution_kernel_fwhm = 0.06
convolution_kernel_stddev = convolution_kernel_fwhm / 2.355
ps_model_fit_values = np.zeros(len(ps_model))
for k_ind, k_val in enumerate(model_k_axis):
    taper_function = np.exp(-(np.array(model_k_axis) - k_val)**2./
 ↪(convolution_kernel_stddev)**2.)
    ps_model_fit_values[k_ind] = np.sum(taper_function*ps_model)/np.
 ↪sum(taper_function)
    #ps_model_fit_values[k_ind] = np.nanmean(ps_model[
    #    np.where((model_k_axis > k_val - boxcar_size/2.0) & (model_k_axis <␣
 ↪k_val + boxcar_size/2.0))
    #])
```

[26]:
```
print(ps_model_fit_values)
```

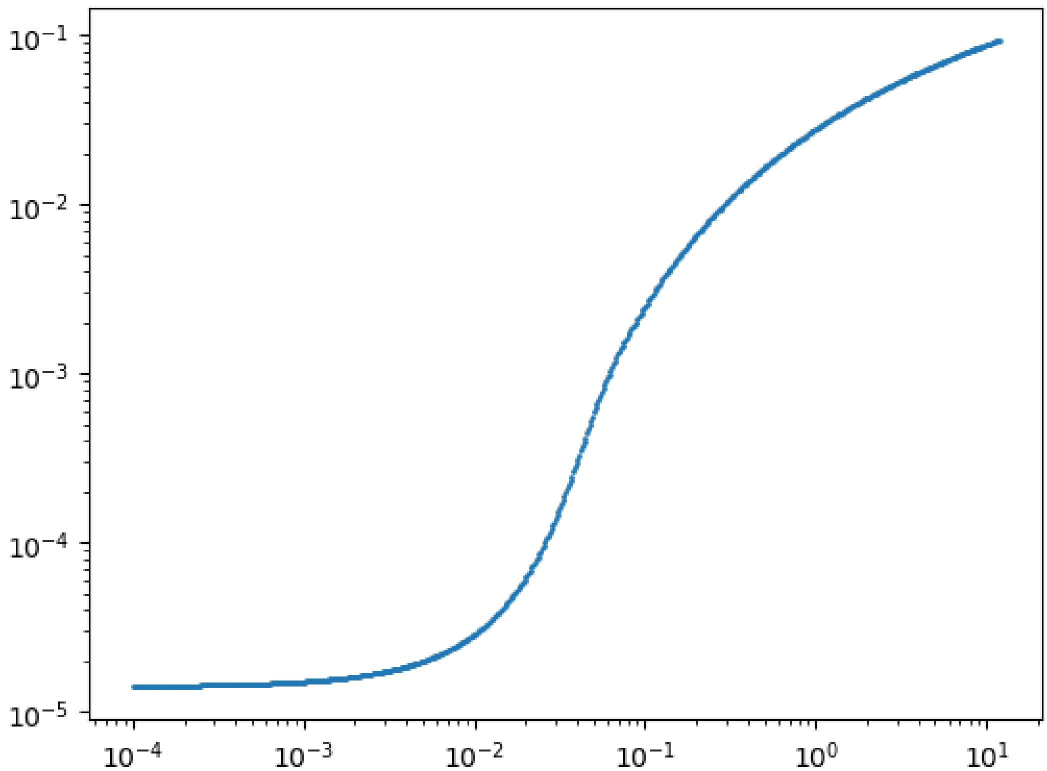
```
[1.42121241e-05 1.42140533e-05 1.42160209e-05 1.42180271e-05
 1.42200815e-05 1.42221744e-05 1.42243058e-05 1.42264855e-05
 1.42287038e-05 1.42309703e-05 1.42332851e-05 1.42356482e-05
 1.42380499e-05 1.42405096e-05 1.42430177e-05 1.42455837e-05
 1.42481885e-05 1.42508514e-05 1.42535724e-05 1.42563515e-05
 1.42591791e-05 1.42620745e-05 1.42650185e-05 1.42680304e-05
 1.42711006e-05 1.42742292e-05 1.42774257e-05 1.42806903e-05
 1.42840231e-05 1.42874143e-05 1.42908834e-05 1.42944208e-05
 1.42980361e-05 1.43017198e-05 1.43054816e-05 1.43093215e-05
 1.43132301e-05 1.43172265e-05 1.43213110e-05 1.43254739e-05
 1.43297154e-05 1.43340547e-05 1.43384824e-05 1.43429986e-05
 1.43476033e-05 1.43523063e-05 1.43571078e-05 1.43620078e-05
 1.43670064e-05 1.43721134e-05 1.43773194e-05 1.43826340e-05
 1.43880575e-05 1.43935996e-05 1.43992509e-05 1.44050211e-05
 1.44109105e-05 1.44169192e-05 1.44230473e-05 1.44293146e-05
 1.44357017e-05 1.44422283e-05 1.44488848e-05 1.44556813e-05
 1.44626179e-05 1.44697046e-05 1.44769320e-05 1.44843099e-05
 1.44918486e-05 1.44995383e-05 1.45073893e-05 1.45154018e-05
 1.45235859e-05 1.45319420e-05 1.45404605e-05 1.45491712e-05
 1.45580550e-05 1.45671317e-05 1.45763920e-05 1.45858560e-05
 1.45955044e-05 1.46053672e-05 1.46154350e-05 1.46257082e-05
 1.46362073e-05 1.46469129e-05 1.46578552e-05 1.46690249e-05
 1.46804325e-05 1.46920886e-05 1.47039740e-05 1.47161291e-05
 1.47285247e-05 1.47411915e-05 1.47541302e-05 1.47673417e-05
```

1.47808367e-05	1.47946161e-05	1.48086808e-05	1.48230519e-05
1.48377402e-05	1.48527268e-05	1.48680430e-05	1.48836897e-05
1.48996682e-05	1.49159898e-05	1.49326558e-05	1.49496876e-05
1.49670867e-05	1.49848646e-05	1.50030229e-05	1.50215733e-05
1.50405276e-05	1.50598976e-05	1.50796853e-05	1.50999026e-05
1.51205925e-05	1.51416441e-05	1.51632444e-05	1.51852928e-05
1.52077915e-05	1.52308462e-05	1.52543560e-05	1.52784270e-05
1.53029582e-05	1.53280562e-05	1.53538284e-05	1.53800694e-05
1.54068867e-05	1.54342834e-05	1.54623683e-05	1.54910399e-05
1.55203019e-05	1.55502637e-05	1.55809298e-05	1.56123048e-05
1.56442865e-05	1.56770929e-05	1.57106224e-05	1.57448801e-05
1.57799787e-05	1.58158163e-05	1.58525067e-05	1.58900562e-05
1.59283623e-05	1.59676495e-05	1.60078159e-05	1.60489786e-05
1.60910354e-05	1.61341044e-05	1.61781942e-05	1.62232024e-05
1.62693603e-05	1.63166783e-05	1.63649424e-05	1.64144995e-05
1.64651358e-05	1.65170889e-05	1.65702578e-05	1.66246547e-05
1.66804071e-05	1.67374138e-05	1.67959197e-05	1.68558250e-05
1.69171449e-05	1.69800127e-05	1.70443284e-05	1.71103452e-05
1.71779648e-05	1.72473259e-05	1.73183301e-05	1.73911184e-05
1.74658347e-05	1.75423827e-05	1.76209088e-05	1.77013167e-05
1.77838796e-05	1.78686277e-05	1.79554672e-05	1.80446800e-05
1.81360482e-05	1.82299857e-05	1.83262763e-05	1.84250850e-05
1.85265816e-05	1.86308095e-05	1.87378138e-05	1.88477731e-05
1.89606050e-05	1.90766269e-05	1.91957607e-05	1.93181981e-05
1.94439996e-05	1.95733658e-05	1.97063645e-05	1.98430658e-05
1.99836837e-05	2.01284391e-05	2.02771291e-05	2.04302692e-05
2.05878075e-05	2.07499856e-05	2.09169048e-05	2.10888205e-05
2.12658455e-05	2.14482504e-05	2.16361612e-05	2.18298651e-05
2.20295027e-05	2.22352195e-05	2.24473287e-05	2.26661560e-05
2.28920408e-05	2.31250030e-05	2.33654002e-05	2.36137765e-05
2.38701816e-05	2.41350162e-05	2.44085218e-05	2.46913098e-05
2.49836566e-05	2.52858502e-05	2.55985662e-05	2.59221339e-05
2.62568974e-05	2.66034115e-05	2.69624581e-05	2.73342547e-05
2.77194360e-05	2.81188779e-05	2.85330813e-05	2.89619240e-05
2.94076763e-05	2.98723099e-05	3.03520168e-05	3.08519501e-05
3.13705593e-05	3.19085587e-05	3.24715984e-05	3.30532591e-05
3.36618755e-05	3.42960326e-05	3.49541616e-05	3.56399574e-05
3.63574046e-05	3.71051216e-05	3.78844564e-05	3.86998090e-05
3.95498445e-05	4.04392798e-05	4.13700295e-05	4.23441222e-05
4.33637067e-05	4.44310603e-05	4.55485956e-05	4.67225318e-05
4.79558937e-05	4.92480210e-05	5.06019893e-05	5.20292901e-05
5.35299019e-05	5.51079278e-05	5.67722547e-05	5.85233043e-05
6.03708689e-05	6.23207261e-05	6.43738717e-05	6.65470411e-05
6.88422584e-05	7.12727658e-05	7.38471978e-05	7.65685836e-05
7.94525097e-05	8.25161815e-05	8.57646519e-05	8.92099312e-05
9.28799590e-05	9.67745282e-05	1.00916494e-04	1.05331072e-04
1.10037113e-04	1.15045348e-04	1.20385707e-04	1.26091133e-04
1.32176392e-04	1.38678274e-04	1.45613585e-04	1.53035471e-04

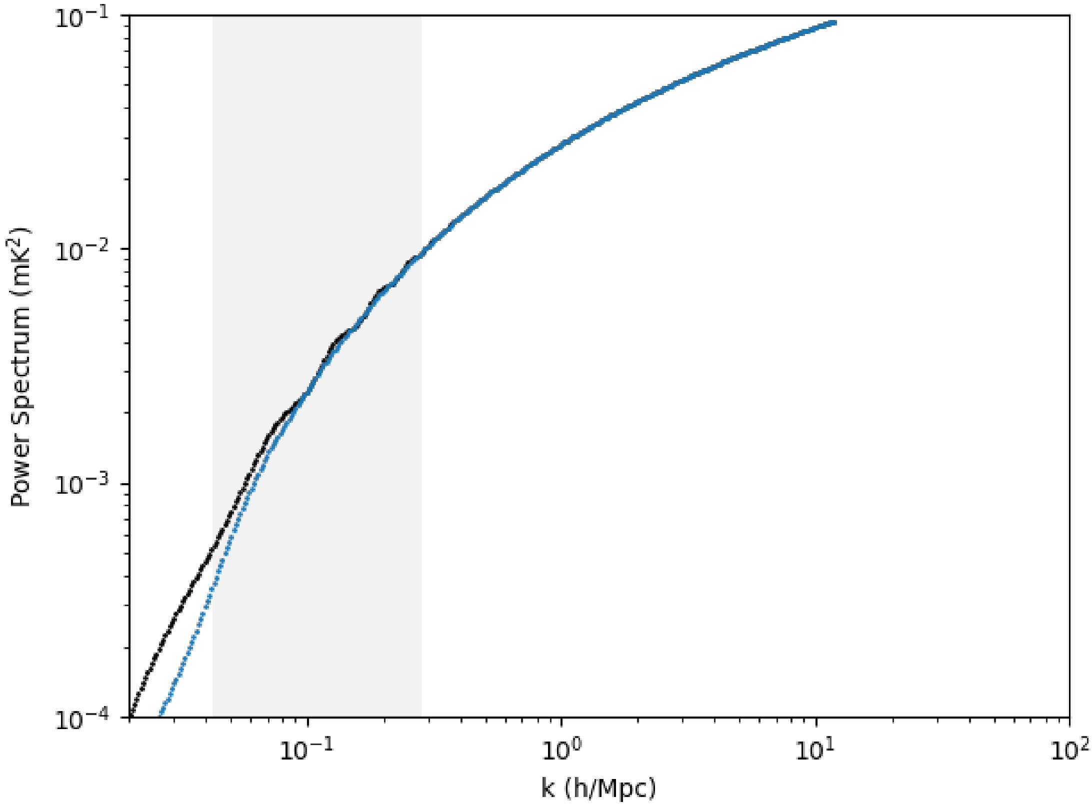
1.60977371e-04	1.69460907e-04	1.78535156e-04	1.88238443e-04
1.98626151e-04	2.09725207e-04	2.21612100e-04	2.34316686e-04
2.47885023e-04	2.62383048e-04	2.77880015e-04	2.94405042e-04
3.12027217e-04	3.30793915e-04	3.50749716e-04	3.71986597e-04
3.94494031e-04	4.18358214e-04	4.43581612e-04	4.70244252e-04
4.98365327e-04	5.27924250e-04	5.58984885e-04	5.91509827e-04
6.25551057e-04	6.61054055e-04	6.98023611e-04	7.36423603e-04
7.76247548e-04	8.17485205e-04	8.60048407e-04	9.03919058e-04
9.49039885e-04	9.95428636e-04	1.04302721e-03	1.09173888e-03
1.14158345e-03	1.19250565e-03	1.24449119e-03	1.29749018e-03
1.35145664e-03	1.40642662e-03	1.46236496e-03	1.51920603e-03
1.57704689e-03	1.63588094e-03	1.69571578e-03	1.75661345e-03
1.81861694e-03	1.88182810e-03	1.94637143e-03	2.01239522e-03
2.08003337e-03	2.14944106e-03	2.22090710e-03	2.29457986e-03
2.37068949e-03	2.44942148e-03	2.53098354e-03	2.61584543e-03
2.70339049e-03	2.79448761e-03	2.88866919e-03	2.98623879e-03
3.08652914e-03	3.18920862e-03	3.29475238e-03	3.40176815e-03
3.51014979e-03	3.61888952e-03	3.72831160e-03	3.83750117e-03
3.94611402e-03	4.05438324e-03	4.16196026e-03	4.26954526e-03
4.37733221e-03	4.48651850e-03	4.59811550e-03	4.71324714e-03
4.83303195e-03	4.95884514e-03	5.09099350e-03	5.22973596e-03
5.37567896e-03	5.52670172e-03	5.68274835e-03	5.84157715e-03
6.00158204e-03	6.15989273e-03	6.31572442e-03	6.46806337e-03
6.61680316e-03	6.76286907e-03	6.90781195e-03	7.05530150e-03
7.20718073e-03	7.36678028e-03	7.53628061e-03	7.71539367e-03
7.90339105e-03	8.09811187e-03	8.29555895e-03	8.49230456e-03
8.68518383e-03	8.87257310e-03	9.05526614e-03	9.23542315e-03
9.41736005e-03	9.60539279e-03	9.80284667e-03	1.00109163e-02
1.02289003e-02	1.04523445e-02	1.06768151e-02	1.08984753e-02
1.11149572e-02	1.13279001e-02	1.15402309e-02	1.17572778e-02
1.19827674e-02	1.22177511e-02	1.24597216e-02	1.27052522e-02
1.29489760e-02	1.31886349e-02	1.34238959e-02	1.36576069e-02
1.38934640e-02	1.41347725e-02	1.43838171e-02	1.46411636e-02
1.49066070e-02	1.51787100e-02	1.54566394e-02	1.57385633e-02
1.60230100e-02	1.63095460e-02	1.65965464e-02	1.68834789e-02
1.71711371e-02	1.74591419e-02	1.77482436e-02	1.80384279e-02
1.83303598e-02	1.86247817e-02	1.89221438e-02	1.92230037e-02
1.95274149e-02	1.98352758e-02	2.01467129e-02	2.04617811e-02
2.07800298e-02	2.11019759e-02	2.14267742e-02	2.17545254e-02
2.20849428e-02	2.24179893e-02	2.27534399e-02	2.30913860e-02
2.34318712e-02	2.37745589e-02	2.41201873e-02	2.44683829e-02
2.48190882e-02	2.51723537e-02	2.55289789e-02	2.58881391e-02
2.62500867e-02	2.66151316e-02	2.69828777e-02	2.73529276e-02
2.77263053e-02	2.81026062e-02	2.84816256e-02	2.88622743e-02
2.92454418e-02	2.96347943e-02	3.00251191e-02	3.04154819e-02
3.08107541e-02	3.12106106e-02	3.16104125e-02	3.20175845e-02
3.24235159e-02	3.28313198e-02	3.32415172e-02	3.36582593e-02
3.40760406e-02	3.44934646e-02	3.49162823e-02	3.53384456e-02

```
3.57687824e-02 3.61978908e-02 3.66299427e-02 3.70688533e-02
3.75029200e-02 3.79474061e-02 3.83884184e-02 3.88372873e-02
3.92856857e-02 3.97378926e-02 4.01914078e-02 4.06459973e-02
4.11071905e-02 4.15679897e-02 4.20337858e-02 4.24967867e-02
4.29687756e-02 4.34423789e-02 4.39171681e-02 4.43909939e-02
4.48699263e-02 4.53525229e-02 4.58384459e-02 4.63248887e-02
4.68188400e-02 4.73107350e-02 4.78073704e-02 4.83016445e-02
4.88025660e-02 4.93046899e-02 4.98116507e-02 5.03221786e-02
5.08292227e-02 5.13437664e-02 5.18582504e-02 5.23775551e-02
5.28955818e-02 5.34176253e-02 5.39433143e-02 5.44716974e-02
5.50028392e-02 5.55319050e-02 5.60681193e-02 5.66068643e-02
5.71415726e-02 5.76889693e-02 5.82298344e-02 5.87804313e-02
5.93252701e-02 5.98750929e-02 6.04316152e-02 6.09867233e-02
6.15465669e-02 6.21058895e-02 6.26708439e-02 6.32348353e-02
6.38020751e-02 6.43716626e-02 6.49450350e-02 6.55178704e-02
6.60963513e-02 6.66746825e-02 6.72547031e-02 6.78395993e-02
6.84271896e-02 6.90132814e-02 6.96050564e-02 7.01960663e-02
7.07909904e-02 7.13912444e-02 7.19882690e-02 7.25913222e-02
7.31964174e-02 7.38027131e-02 7.44086650e-02 7.50178818e-02
7.56315046e-02 7.62465562e-02 7.68618511e-02 7.74790949e-02
7.80993652e-02 7.87260792e-02 7.93496455e-02 7.99786915e-02
8.06055049e-02 8.12412145e-02 8.18736461e-02 8.25084213e-02
8.31476597e-02 8.37884166e-02 8.44354096e-02 8.50775194e-02
8.57242267e-02 8.63674736e-02 8.70201079e-02 8.76685702e-02
8.83227897e-02 8.89630730e-02 8.96277482e-02 9.02911830e-02
9.09511580e-02 9.16046698e-02 9.22753850e-02 9.29360075e-02
9.36108521e-02]
```

[27]:
```
plt.plot(model_k_axis, ps_model_fit_values,"o", markersize=1)
plt.yscale("log")
plt.xscale("log")
```

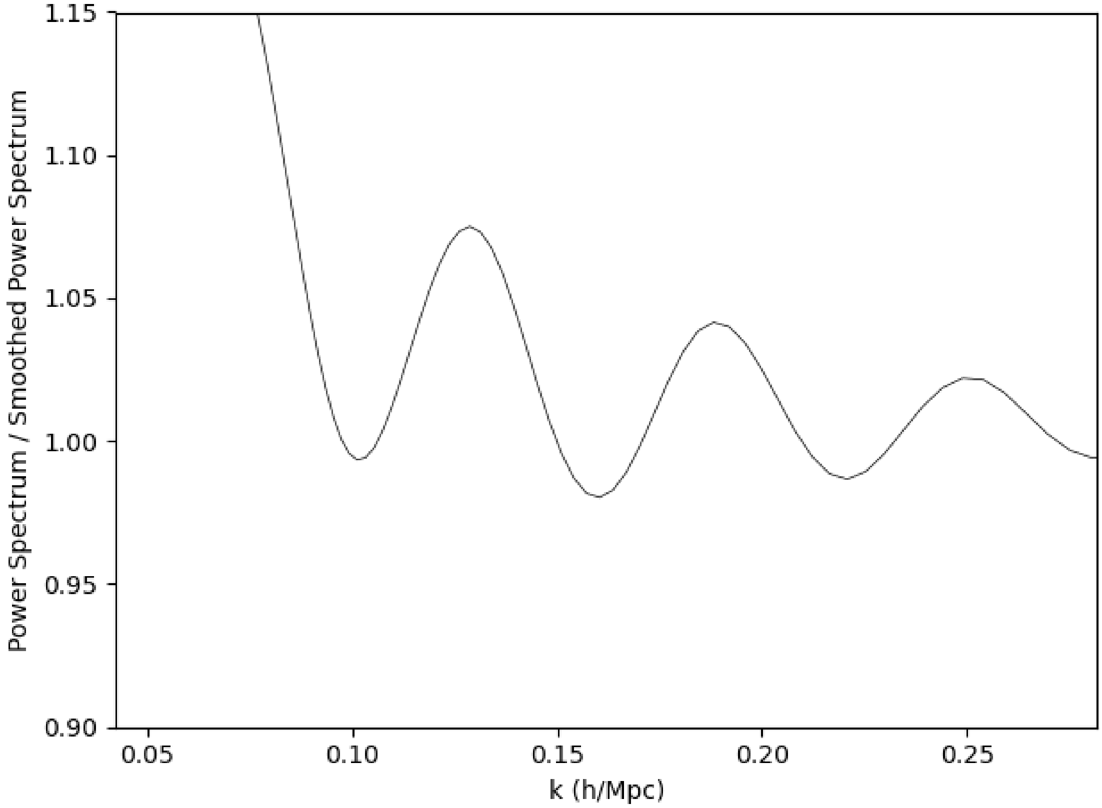


[28]:
```
xrange = [2e-2, 100]
yrange = [1e-4, 1e-1]

# Plot theory line
plt.plot(model_k_axis, ps_model, "o", color="black", linewidth=0, markersize=1)
plt.plot(model_k_axis, ps_model_fit_values,"o", markersize=1)

# Plot BAO scales
plt.fill_between(bao_scales_k, [yrange[0], yrange[0]], [yrange[1], yrange[1]],␣
 ↪color="grey", alpha=.1, linewidth=0)

plt.yscale("log")
plt.xscale("log")
plt.xlim(xrange)
plt.ylim(yrange)
plt.xlabel("k (h/Mpc)")
plt.ylabel("Power Spectrum (mK$^2$)")
plt.tight_layout()
plt.show()
```



[29]:
```
xrange = bao_scales_k
yrange = [.9,1.15]

# Plot theory line
plt.plot(model_k_axis, ps_model/ps_model_fit_values, "-", color="black",␣
 ↪linewidth=0.5, markersize=0)
#plt.plot(bin_edges_bao, [1]*len(bin_edges_bao), "o", color="black",␣
 ↪linewidth=0, markersize=1)

# Plot BAO scales
#plt.fill_between(bao_scales_k, [yrange[0], yrange[0]], [yrange[1], yrange[1]],␣
 ↪color="grey", alpha=.1, linewidth=0)

#plt.yscale("log")
#plt.xscale("log")
plt.xlim(xrange)
plt.ylim(yrange)
plt.xlabel("k (h/Mpc)")
plt.ylabel("Power Spectrum / Smoothed Power Spectrum")
plt.tight_layout()
```

```
plt.show()
```



[30]:
```
# Find peaks
model_k_axis = np.array(model_k_axis)
ps_ratio = ps_model/ps_model_fit_values
use_inds = np.where((model_k_axis > 0.05) & (model_k_axis < 0.1))
print(model_k_axis[list(ps_ratio).index(np.max(ps_ratio[use_inds]))])
use_inds = np.where((model_k_axis > 0.1) & (model_k_axis < 0.15))
print(model_k_axis[list(ps_ratio).index(np.max(ps_ratio[use_inds]))])
use_inds = np.where((model_k_axis > 0.15) & (model_k_axis < 0.23))
print(model_k_axis[list(ps_ratio).index(np.max(ps_ratio[use_inds]))])
use_inds = np.where((model_k_axis > 0.23) & (model_k_axis < 0.27))
print(model_k_axis[list(ps_ratio).index(np.max(ps_ratio[use_inds]))])
```

```
0.05027
0.12869
0.18818
0.24899
```

[31]:
```
print(0.25402-0.18818)
print(0.18818-0.12869)
```

```
print(0.12869-0.07351)
print(np.mean([0.25402-0.18818,0.18818-0.12869,0.12869-0.07351])/2)
```

```
0.06584000000000004
0.05948999999999999
0.05517999999999999
0.030085
```

[32]:
```
k_bin_size_bao = 0.03
min_k_bao = 0.12869 - 2.5*k_bin_size_bao
max_k_bao = .35
bin_edges_bao = np.arange(min_k_bao, max_k_bao, k_bin_size_bao)
print(bin_edges_bao)
```

```
[0.05369 0.08369 0.11369 0.14369 0.17369 0.20369 0.23369 0.26369 0.29369
 0.32369]
```

[33]:
```
(
    null,
    binned_ps_variance_bao,
    true_bin_edges_bao,
    true_bin_centers_bao,
    null,
    null,
) = array_sensitivity.delay_ps_sensitivity_analysis(
    antpos_filepath=antpos_filepath,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    tsys_k=tsys_k,
    aperture_efficiency=aperture_efficiency,
    antenna_diameter_m=antenna_diameter_m,
    freq_resolution_hz=freq_resolution_hz,
    int_time_s=int_time_s,
    max_bl_m=1000,
    k_bin_edges_1d=bin_edges_bao,
    kpar_bin_edges=bin_edges_bao,
    kperp_bin_edges=bin_edges_bao,
    wedge_extent_deg=3.09,
    zenith_angle=0.0,
)
(
    null,
    binned_ps_variance_bao_offaxis,
    true_bin_edges_bao_offaxis,
    true_bin_centers_bao_offaxis,
    null,
    null,
) = array_sensitivity.delay_ps_sensitivity_analysis(
```

```
    antpos_filepath=antpos_filepath,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    tsys_k=tsys_k,
    aperture_efficiency=aperture_efficiency,
    antenna_diameter_m=antenna_diameter_m,
    freq_resolution_hz=freq_resolution_hz,
    int_time_s=int_time_s,
    max_bl_m=1000,
    k_bin_edges_1d=bin_edges_bao,
    kpar_bin_edges=bin_edges_bao,
    kperp_bin_edges=bin_edges_bao,
    wedge_extent_deg=3.09,
    zenith_angle=60.0,
)
(
    null,
    binned_ps_variance_bao_core,
    true_bin_edges_bao_core,
    true_bin_centers_bao_core,
    null,
    null,
) = array_sensitivity.delay_ps_sensitivity_analysis(
    antpos_filepath="W2-17_core.cfg",
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    tsys_k=tsys_k,
    aperture_efficiency=aperture_efficiency,
    antenna_diameter_m=antenna_diameter_m,
    freq_resolution_hz=freq_resolution_hz,
    int_time_s=int_time_s,
    max_bl_m=1000,
    k_bin_edges_1d=bin_edges_bao,
    kpar_bin_edges=bin_edges_bao,
    kperp_bin_edges=bin_edges_bao,
    wedge_extent_deg=3.09,
    zenith_angle=0.0,
)

# Account for 2 polarizations
binned_ps_variance_bao /= 4
binned_ps_variance_bao_offaxis /= 4
binned_ps_variance_bao_core /= 4

binned_ps_sample_variance_bao = array_sensitivity.get_sample_variance(
    ps_model,  # Units mK^2
    model_k_axis,  # Units h/Mpc
```

```
    field_of_view_deg2=field_of_view_deg2,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    freq_resolution_hz=freq_resolution_hz,
    k_bin_edges=bin_edges_bao,
    wedge_extent_deg=3.09,
)
binned_ps_shot_noise_bao = array_sensitivity.get_shot_noise(
    field_of_view_deg2=field_of_view_deg2,
    min_freq_hz=min_freq_hz,
    max_freq_hz=max_freq_hz,
    freq_resolution_hz=freq_resolution_hz,
    k_bin_edges=bin_edges_bao,
    wedge_extent_deg=3.09,
)
combined_variance_bao = (
    (binned_ps_variance_bao * .25 / 720)
    + ((binned_ps_sample_variance_bao + binned_ps_shot_noise_bao) *␣
 ↪field_of_view_deg2 / (3*np.pi*(180/np.pi)**2.))
)
combined_variance_bao_offaxis = (
    (binned_ps_variance_bao_offaxis * .25 / 720)
    + ((binned_ps_sample_variance_bao + binned_ps_shot_noise_bao) *␣
 ↪field_of_view_deg2 / (3*np.pi*(180/np.pi)**2.))
)
combined_variance_bao_core = (
    (binned_ps_variance_bao_core * .25 / 720)
    + ((binned_ps_sample_variance_bao + binned_ps_shot_noise_bao) *␣
 ↪field_of_view_deg2 / (3*np.pi*(180/np.pi)**2.))
)
```

```
Kpar correlation length: 0.002736362210334458
Kperp correlation length: 0.015331664324187
Correlation volume: 6.432091122141715e-07
Kpar correlation length: 0.002736362210334458
Kperp correlation length: 0.015331664324187
Correlation volume: 6.432091122141715e-07
```

[34]:
```
ps_ratio_interp = np.interp(true_bin_centers_bao, model_k_axis, ps_ratio)
ps_model_fit_values_interp = np.interp(true_bin_centers_bao, model_k_axis,␣
 ↪ps_model_fit_values)

combined_variance_bao_ratio = combined_variance_bao /␣
 ↪ps_model_fit_values_interp**2.
combined_variance_bao_ratio_offaxis = combined_variance_bao_offaxis /␣
 ↪ps_model_fit_values_interp**2.
```

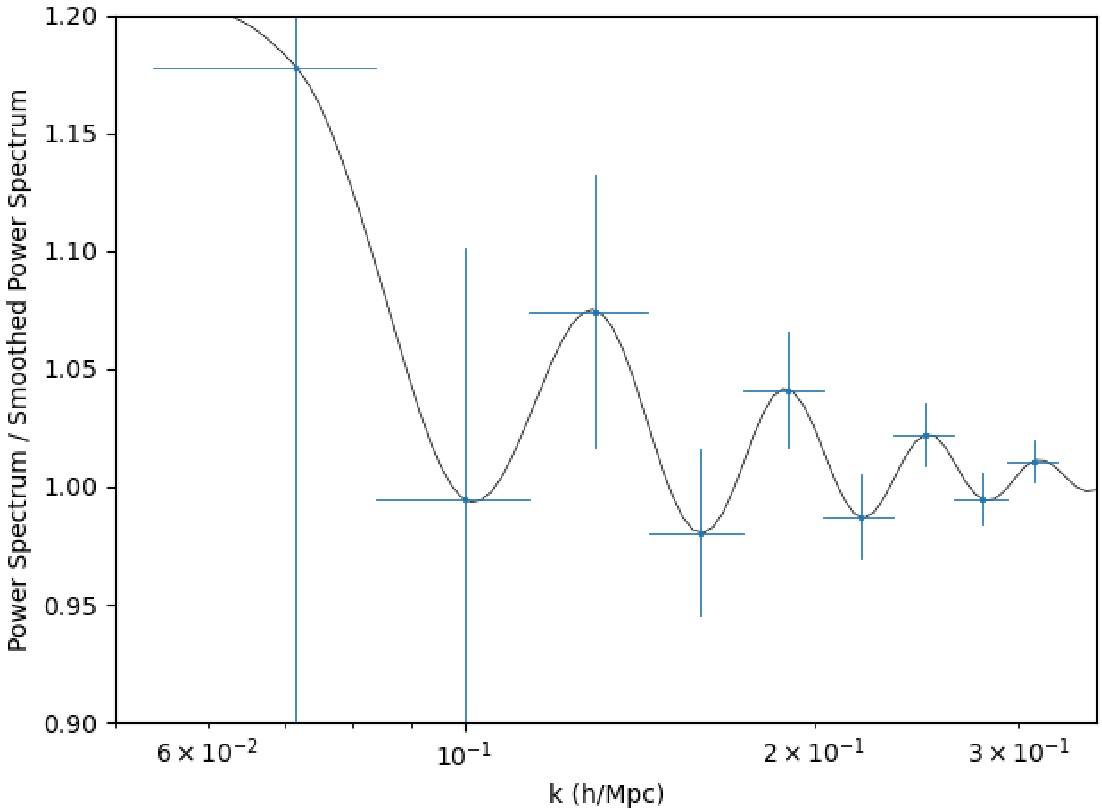
```
combined_variance_bao_ratio_core = combined_variance_bao_core /␣
↪ps_model_fit_values_interp**2.
```

[35]:
```
print(bao_scales_k)
```

```
[0.04225352 0.28169014]
```

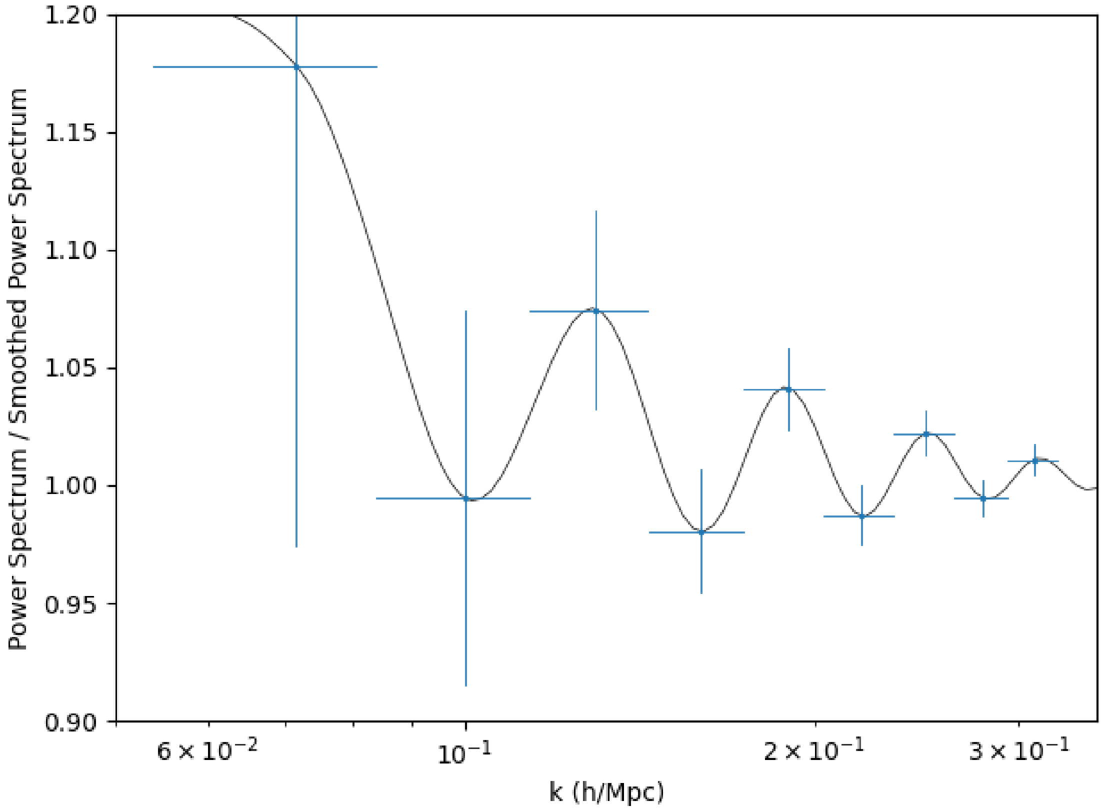
[36]:
```
xrange = [0.05,0.35]
yrange = [.9,1.2]

# Plot theory line
plt.plot(model_k_axis, ps_model/ps_model_fit_values, "-", color="black",␣
↪linewidth=0.5, markersize=0)

plt.plot(true_bin_centers_bao, ps_ratio_interp, marker="o", color="tab:blue",␣
↪linewidth=0, markersize=1.5)
for ind in range(len(ps_ratio_interp)):
    xvals = [true_bin_centers_bao[ind], true_bin_centers_bao[ind]]
    yvals = [
        ps_ratio_interp[ind] - np.sqrt(combined_variance_bao_ratio[ind]),
        ps_ratio_interp[ind] + np.sqrt(combined_variance_bao_ratio[ind])
    ]
    plt.plot(
        xvals,
        yvals,
        color="tab:blue",
        linewidth=0.6,
        marker="none"
    )
    plt.plot(
        true_bin_edges_bao[ind, :],
        [ps_ratio_interp[ind], ps_ratio_interp[ind]],
        marker="none",
        linewidth=0.6,
        color="tab:blue",
    )


# Plot BAO scales
#plt.fill_between(bao_scales_k, [yrange[0], yrange[0]], [yrange[1], yrange[1]],␣
↪color="grey", alpha=.1, linewidth=0)

#plt.yscale("log")
plt.xscale("log")
plt.xlim(xrange)
plt.ylim(yrange)
plt.xlabel("k (h/Mpc)")
```

```
plt.ylabel("Power Spectrum / Smoothed Power Spectrum")
plt.tight_layout()
plt.savefig("plots/bao_error_bars.png", dpi=300)
plt.show()
```



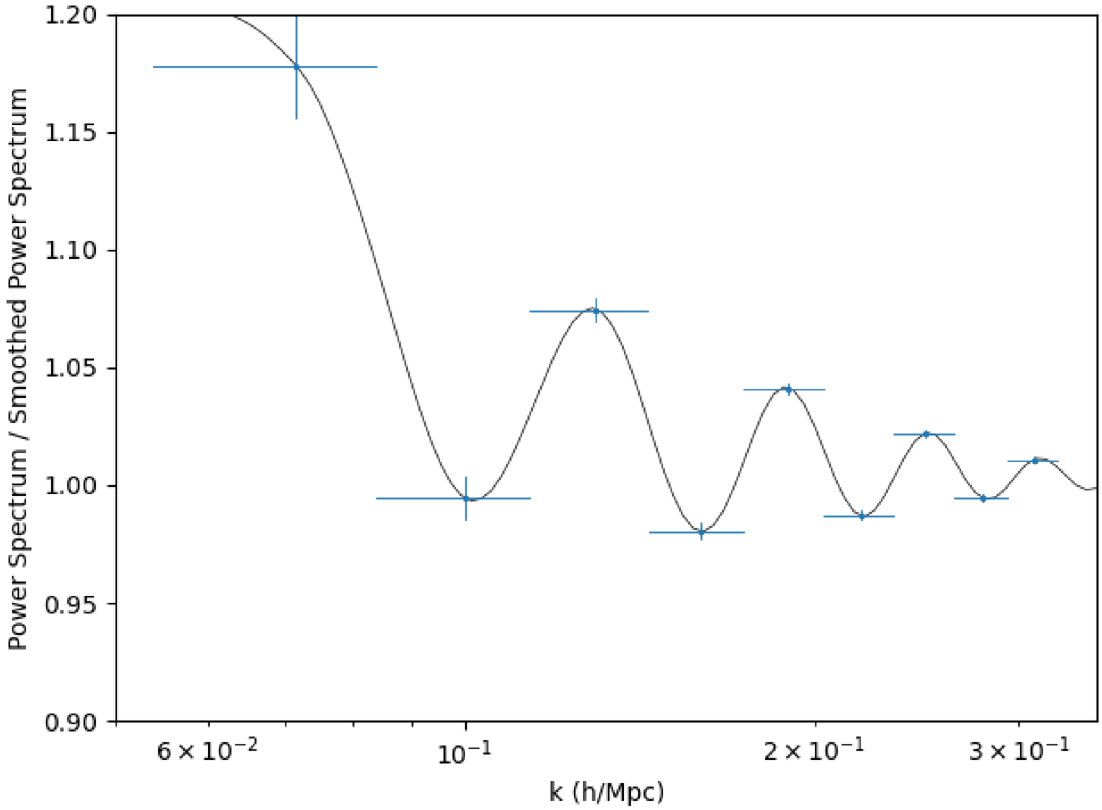
[37]:
```
xrange = [0.05,0.35]
yrange = [.9,1.2]

# Plot theory line
plt.plot(model_k_axis, ps_model/ps_model_fit_values, "-", color="black",␣
 ↪linewidth=0.5, markersize=0)

plt.plot(true_bin_centers_bao, ps_ratio_interp, marker="o", color="tab:blue",␣
 ↪linewidth=0, markersize=1.5)
for ind in range(len(ps_ratio_interp)):
    xvals = [true_bin_centers_bao[ind], true_bin_centers_bao[ind]]
    yvals = [
        ps_ratio_interp[ind] - np.
 ↪sqrt(combined_variance_bao_ratio_offaxis[ind]),
        ps_ratio_interp[ind] + np.sqrt(combined_variance_bao_ratio_offaxis[ind])
```

```
    ]
    plt.plot(
        xvals,
        yvals,
        color="tab:blue",
        linewidth=0.6,
        marker="none"
    )
    plt.plot(
        true_bin_edges_bao[ind, :],
        [ps_ratio_interp[ind], ps_ratio_interp[ind]],
        marker="none",
        linewidth=0.6,
        color="tab:blue",
    )


# Plot BAO scales
#plt.fill_between(bao_scales_k, [yrange[0], yrange[0]], [yrange[1], yrange[1]],␣
 ↪color="grey", alpha=.1, linewidth=0)

#plt.yscale("log")
plt.xscale("log")
plt.xlim(xrange)
plt.ylim(yrange)
plt.xlabel("k (h/Mpc)")
plt.ylabel("Power Spectrum / Smoothed Power Spectrum")
plt.tight_layout()
plt.savefig("plots/bao_error_bars_offaxis.png", dpi=300)
plt.show()
```



[38]:
```
xrange = [0.05,0.35]
yrange = [.9,1.2]

# Plot theory line
plt.plot(model_k_axis, ps_model/ps_model_fit_values, "-", color="black",␣
 ↪linewidth=0.5, markersize=0)

plt.plot(true_bin_centers_bao, ps_ratio_interp, marker="o", color="tab:blue",␣
 ↪linewidth=0, markersize=1.5)
for ind in range(len(ps_ratio_interp)):
    xvals = [true_bin_centers_bao[ind], true_bin_centers_bao[ind]]
    yvals = [
        ps_ratio_interp[ind] - np.sqrt(combined_variance_bao_ratio_core[ind]),
        ps_ratio_interp[ind] + np.sqrt(combined_variance_bao_ratio_core[ind])
    ]
    plt.plot(
        xvals,
        yvals,
        color="tab:blue",
        linewidth=0.6,
```

```
        marker="none"
    )
    plt.plot(
        true_bin_edges_bao[ind, :],
        [ps_ratio_interp[ind], ps_ratio_interp[ind]],
        marker="none",
        linewidth=0.6,
        color="tab:blue",
    )


# Plot BAO scales
#plt.fill_between(bao_scales_k, [yrange[0], yrange[0]], [yrange[1], yrange[1]],␣
 ↪color="grey", alpha=.1, linewidth=0)

#plt.yscale("log")
plt.xscale("log")
plt.xlim(xrange)
plt.ylim(yrange)
plt.xlabel("k (h/Mpc)")
plt.ylabel("Power Spectrum / Smoothed Power Spectrum")
plt.tight_layout()
plt.savefig("plots/bao_error_bars_core.png", dpi=300)
plt.show()
```



[39]:
```
print([ps_ratio_interp[0],ps_ratio_interp[2],ps_ratio_interp[4]])
print([ps_ratio_interp[1],ps_ratio_interp[3],ps_ratio_interp[5]])
print(np.sqrt(combined_variance_bao_ratio_core[0:6]))
```

```
[1.178039263266831, 1.0743534202585894, 1.0409701774378044]
[0.9948022097751336, 0.9806173432100984, 0.9873332161537907]
[0.02265916 0.00924209 0.00493102 0.00323605 0.00230732 0.0018194 ]
```

[40]:
```
print(1.0530108835298397-0.9558258520464863)
print(5*(np.sqrt(combined_variance_bao_ratio_core[0])+np.
  ↪sqrt(combined_variance_bao_ratio_core[1])))
```

```
0.09718503148335345
0.15950627032159986
```

[41]:
```
print(1.078080348903745-0.9678294999064591)
print(5*(np.sqrt(combined_variance_bao_ratio_core[2])+np.
  ↪sqrt(combined_variance_bao_ratio_core[3])))
```

```
0.11025084899728588
0.04083534726380202
```

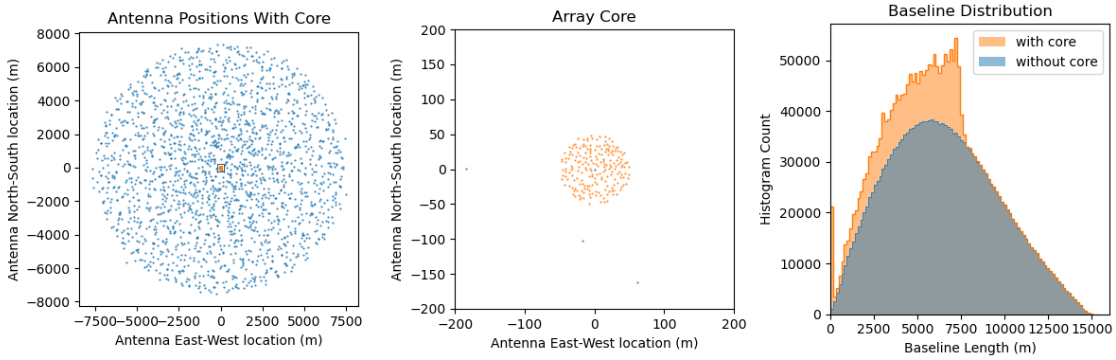
[42]:
```
print(1.0379603923150817-0.9696283722555493)
print(5*(np.sqrt(combined_variance_bao_ratio_core[4])+np.
 ↪sqrt(combined_variance_bao_ratio_core[5])))
```

```
0.06833202005953243
0.020633584040523525
```

[43]:
```
antlocs_core = array_sensitivity.get_antpos("W2-17_core.cfg")
baselines_core = array_sensitivity.get_baselines(antlocs_core)
antlocs_core = antlocs_core[-200:, :]
core_plot_size_m = 200
```

[44]:
```
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(12,4))

ax[0].plot(antpos[:,0], antpos[:,1], marker="o", markersize=.5, linewidth=0,␣
 ↪color="tab:blue")
ax[0].plot(antlocs_core[:,0], antlocs_core[:,1], marker="o", markersize=.5,␣
 ↪linewidth=0, color="tab:orange")
ax[0].set_aspect(1)
ax[0].set_xlabel("Antenna East-West location (m)")
ax[0].set_ylabel("Antenna North-South location (m)")
ax[0].set_title("Antenna Positions With Core")
ax[0].add_patch(
    Rectangle((-core_plot_size_m, -core_plot_size_m), 2*core_plot_size_m,␣
 ↪2*core_plot_size_m, edgecolor = 'black', fill=False, lw=0.5)
)
ax[0].set_aspect("equal")

ax[1].plot(antpos[:,0], antpos[:,1], marker="o", markersize=.5, linewidth=0,␣
 ↪color="tab:blue")
ax[1].plot(antlocs_core[:,0], antlocs_core[:,1], marker="o", markersize=.5,␣
 ↪linewidth=0, color="tab:orange")
ax[1].set_aspect(1)
ax[1].set_xlabel("Antenna East-West location (m)")
ax[1].set_ylabel("Antenna North-South location (m)")
ax[1].set_title("Array Core")
ax[1].set_aspect("equal")
ax[1].set_xlim([-core_plot_size_m, core_plot_size_m])
ax[1].set_ylim([-core_plot_size_m, core_plot_size_m])

ax[2].hist(
    np.sqrt(np.sum(baselines_core**2., axis=1)),
    bins=100,
    color="tab:orange",
    label="with core",
    alpha=0.5
)
```

```
ax[2].hist(
    np.sqrt(np.sum(baselines_core**2., axis=1)),
    bins=100,
    linewidth=1,
    color="tab:orange",
    histtype="step"
)
ax[2].hist(
    np.sqrt(np.sum(baselines_m**2., axis=1)),
    bins=100,
    color="tab:blue",
    label="without core",
    alpha=0.5,
)

ax[2].hist(
    np.sqrt(np.sum(baselines_m**2., axis=1)),
    bins=100,
    linewidth=.5,
    color="tab:blue",
    histtype="step"
)
ax[2].hist(
    np.sqrt(np.sum(baselines_core**2., axis=1)),
    bins=100,
    linewidth=.5,
    color="tab:orange",
    histtype="step"
)
ax[2].set_xlabel("Baseline Length (m)")
ax[2].set_ylabel("Histogram Count")
ax[2].set_title("Baseline Distribution")
ax[2].set_xlim([0,16000])
plt.legend()

plt.tight_layout()
plt.savefig("plots/antlocs_core.png", dpi=600)
plt.show()
```



1.3 3. Generate PSF Image

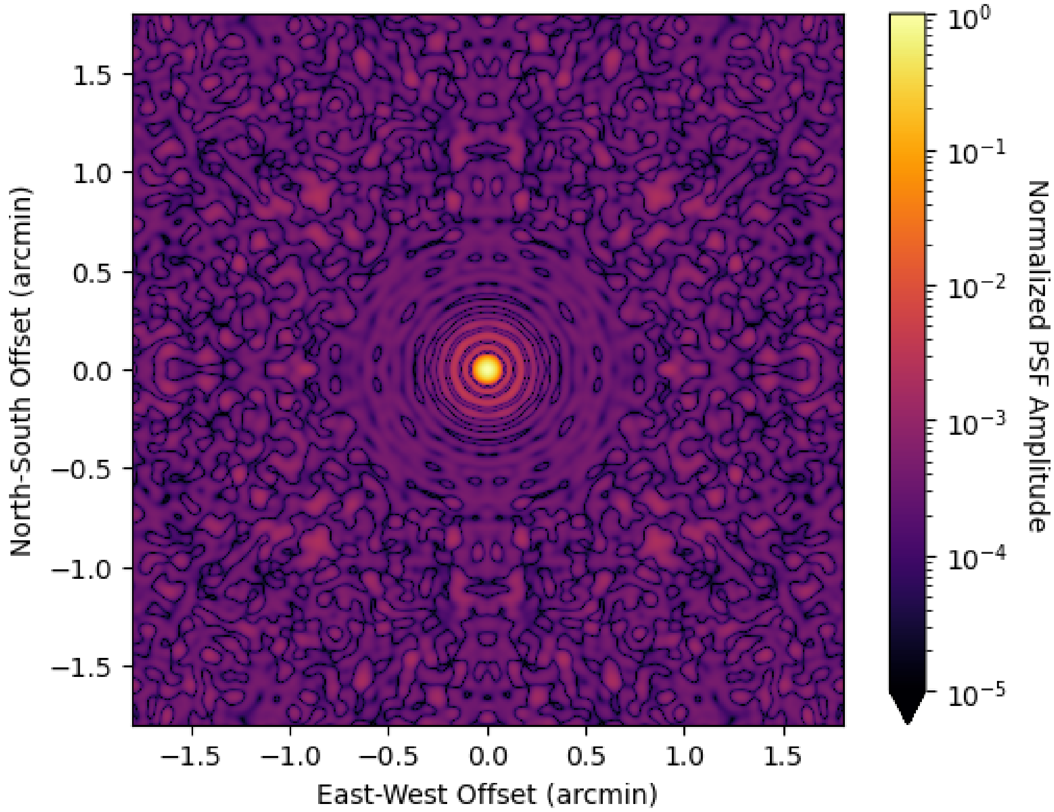
[45]:
```
# Restore PSF simulation
with open("simulation_outputs/psf.npy", "rb") as f:
    psf = np.load(f)
    frequencies = np.load(f)
    ew_axis = np.load(f)
    ns_axis = np.load(f)
f.close()
```

[46]:
```
np.nanmax(psf)
```

[46]: 2068064.9395488538

[47]:
```
image_extent_arcmin = .03*60.0
fig, ax = plt.subplots()
use_cmap = cm.get_cmap("inferno").copy()
cax = ax.imshow(
    np.abs(psf[0,:,:])/np.nanmax(np.abs(psf[0,:,:])),
    origin="lower",
    interpolation="none",
    extent=[
        np.min(ew_axis*60),
        np.max(ew_axis*60),
        np.min(ns_axis*60),
        np.max(ns_axis*60)
    ],
    vmin=1e-5,
    vmax=1,
    cmap=use_cmap,
    norm="log",
    aspect=1.,
)
ax.set_xlabel("East-West Offset (arcmin)")
```

```
ax.set_ylabel("North-South Offset (arcmin)")
ax.set_xlim([-image_extent_arcmin, image_extent_arcmin])
ax.set_ylim([-image_extent_arcmin, image_extent_arcmin])
cbar = fig.colorbar(cax, extend="min")
cbar.set_label("Normalized PSF Amplitude", rotation=270, labelpad=15)
plt.savefig("plots/psf.png", dpi=600)
plt.show()
```



[ ]: