~~1.Read particle file Current Weighting and X-Y location at the highest magnitude Z-location~~

~~2.Read particle file Power Weighting and X-Y location at the highest magnitude Z-location~~

3.Plot 2-D intensity of current or power as a function of X or Y at a line intersection or region of interest (slices of the beam shape).

~~4.Plot 3-D density of current or power vs X/Y (The whole of the beam shape)~~

5.User interface: Scale the axes appropriately (what does that mean?)

6.We needs ability to adjust the scaling of each axis (X/Y/Current/Power)

7.We need the ability to use the bin “area size” to calculate density plots of power and current\n

8.Hence we need to have a good bin size automation tool based on the total number of Z-intersecting particles, the total area, and the error variable.

9.User interface: X and Y to be in user selected measure of length (inches and meters)

10.User interface: Algorithms for Gaussian and “Top Hat” shapes, and FFT analysis as a stretch goal

11.User interface: Able to modify the algorithm variables and coefficients

12.Save images

13.Save bin files

14.Document script

In [84]:
```
import physt
import numpy as np
import pandas as pd
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from scipy.stats import kde
from matplotlib import colors
from physt import h1, h2, histogramdd
```

In [2]:
```
%matplotlib inline
```

Needs to account for (1) #particles used which strike the target anode, (2) cross sectional area containing particles which strike target anode, (3) random effects on statistical error.

In [3]:
```
def resolution_factor(size, magnitude): # Function which yields a dl size, depending on
#particles and custom factor.
    dl = (1/size)*magnitude + 1
    dl *= 0.000078                      # [Inches], two microns * dl constant.
    return dl
# THIS FUNCTION IS INCOMPLETE!!!
```

In [4]:
```
x_in = []
# x-position of each particle, inches.
y_in = []
# y-position of each particle.
z_in = []
# z-position of each particle.
p_I  = []
# particle current in [Amperes].
p_KP = []
# particle power in [Watts].
```

In [5]:
```
file_name = 'quick_input.out'
```

Import some data set values

In [6]:
```
with open(file_name) as file:
    data = pd.read_csv(file,delimiter='\t',header=2,skipfooter=1,engine='python')
    # Take column headers and save separate lists for needed values.
    x_in = data['x[in]']*2.54 # now converted into centimeters.
    y_in = data['y[in]']*2.54
    z_in = data['z[in]']*2.54
    p_I  = data['pI[A]']
    p_KP = data['pKP[W]']

# Let's exclude data points which never reach the target.
z_target = stats.mode(z_in) # We find the mode of our z_in data set: assume target's z-
value.
invalids = []
for i,z in enumerate(z_in):
    if z != z_target[0]:
        invalids.append(i)  # Obtain indices of z-coords which do not equal target's.
    else:
        pass
# Remove data points for these particles which "miss".
x_in = np.delete(np.array(x_in),invalids)
y_in = np.delete(np.array(y_in),invalids)
p_I  = np.delete(np.array(p_I), invalids)
p_KP = np.delete(np.array(p_KP),invalids)
```

In [7]:
```
dl = resolution_factor(len(x_in), 1e4)  # Calculate dl for #particles and custom factor
specified.
```

Power weighting

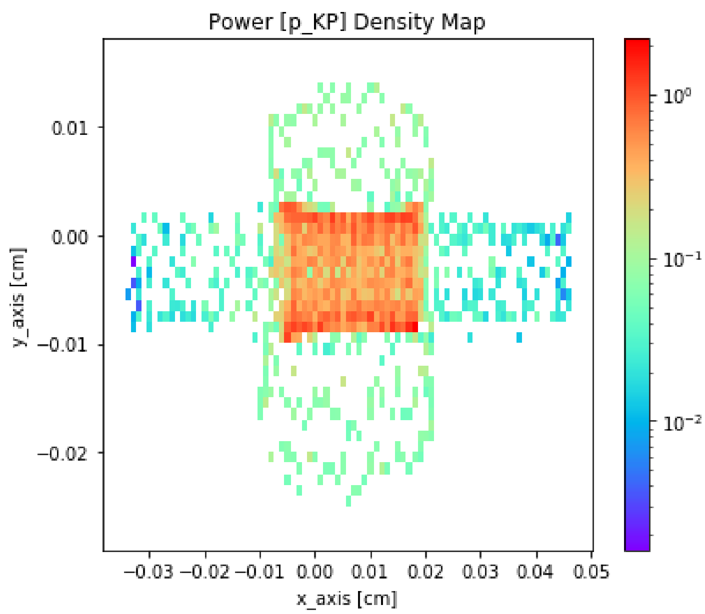
In [8]:
```
weighting = p_KP # We can easily switch global weights for either current or power.
```

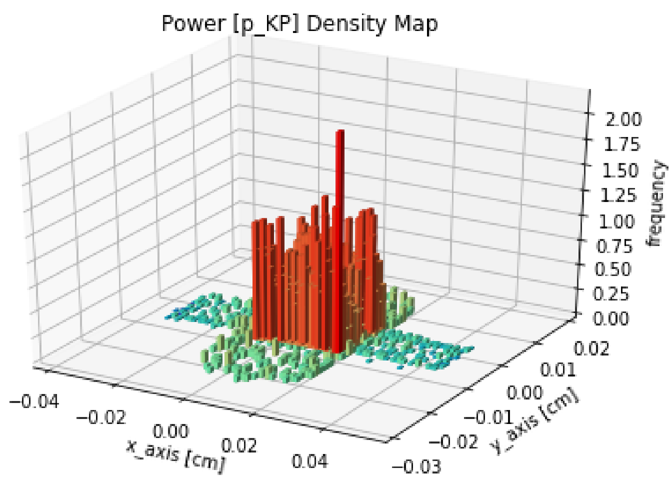
In [31]:
```
dl = 1e-3 # cm*1e-3 = decimilimeters = 1e-5 meters.
m  = 5
```

In [32]:
```
%%time
fig, ax = plt.subplots(figsize=(6, 5)) # Set some arbitrary viewing figure size.
h_2 = h2(x_in, y_in, bins=[np.arange(np.amin(x_in)-m*dl,np.amax(x_in)+m*dl,dl),
                           np.arange(np.amin(y_in)-m*dl,np.amax(y_in)+m*dl,dl)], weight
s=weighting,
                           axis_names=["x_axis [cm]", "y_axis [cm]"], name="Power [p_K
P] Density Map",aspect=1)
# Plot a 2-D histogram using x_in, y_in, and dl for binning.
# We choose to weights by the power per particle.
h_2.plot("image", cmap="rainbow", ax=ax, cmap_normalize="log")
# Plot shows image, normalized by a logarithmic scale.
h_2.plot("bar3d", cmap="rainbow", cmap_normalize="log");
```

```
Wall time: 3.2 s
```

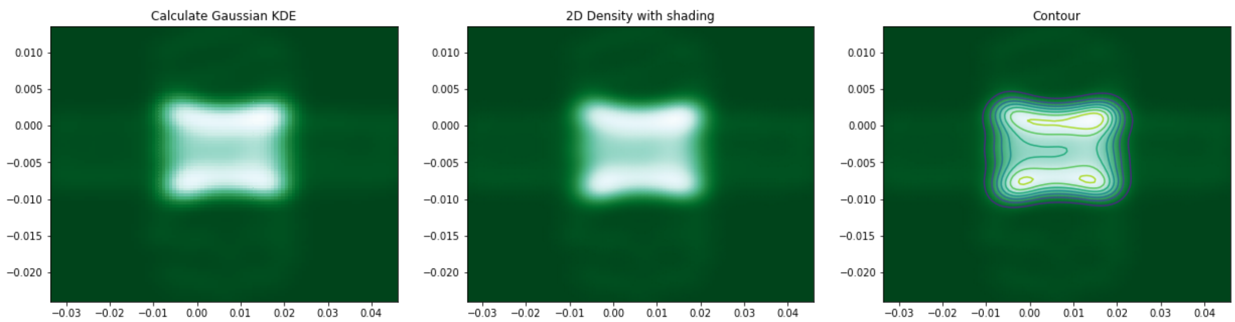
Out[32]: <matplotlib.axes._subplots.Axes3DSubplot at 0x2557add6788>



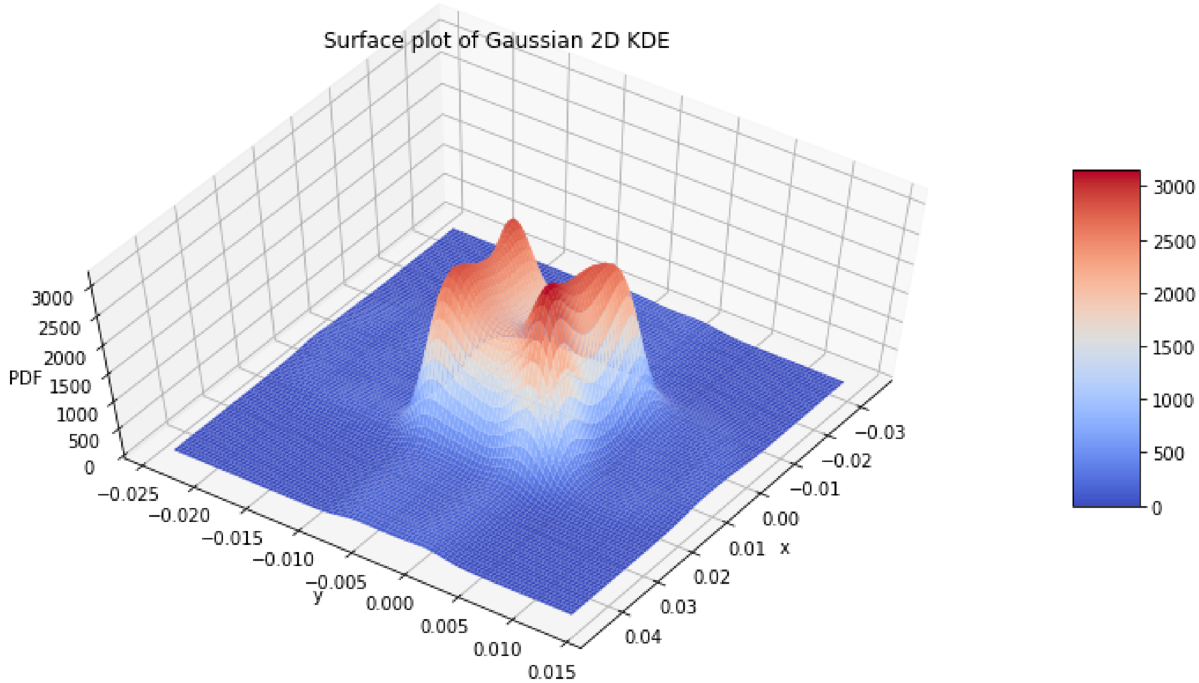


In [98]:
```
# Create a figure with 6 plot areas
fig, axes = plt.subplots(ncols=3, nrows=1, figsize=(21, 5))

# Evaluate a gaussian kde on a regular grid of nbins x nbins over data extents
nbins = 100
k = kde.gaussian_kde([x_in,y_in],bw_method='scott')
xi, yi = np.mgrid[x_in.min():x_in.max():nbins*1j, y_in.min():y_in.max():nbins*1j]
zi = k(np.vstack([xi.flatten(), yi.flatten()]))

# plot a density
axes[0].set_title('Calculate Gaussian KDE')
axes[0].pcolormesh(xi, yi, zi.reshape(xi.shape), cmap=plt.cm.BuGn_r)

# add shading
axes[1].set_title('2D Density with shading')
axes[1].pcolormesh(xi, yi, zi.reshape(xi.shape), shading='gouraud', cmap=plt.cm.BuGn_r)

# contour
axes[2].set_title('Contour')
axes[2].pcolormesh(xi, yi, zi.reshape(xi.shape), shading='gouraud', cmap=plt.cm.BuGn_r)
axes[2].contour(xi, yi, zi.reshape(xi.shape))
```

Out[98]: <matplotlib.contour.QuadContourSet at 0x255089e4988>



In [97]:
```
# need to normalize the z-component to be a true PDF
fig = plt.figure(figsize=(13, 7))
ax = plt.axes(projection='3d')
surf = ax.plot_surface(xi, yi, zi.reshape(xi.shape), rstride=1, cstride=1, cmap='coolwa
rm', edgecolor='none')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('PDF')
ax.set_title('Surface plot of Gaussian 2D KDE')
fig.colorbar(surf, shrink=0.5, aspect=5) # add color bar indicating the PDF
ax.view_init(60, 35)
```

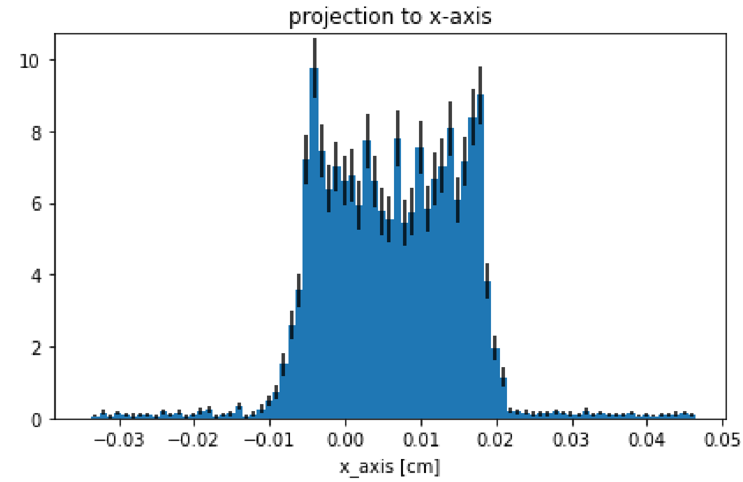


In [33]:
```
# we have access to the intensity at each bin space:
h_2.frequencies
```

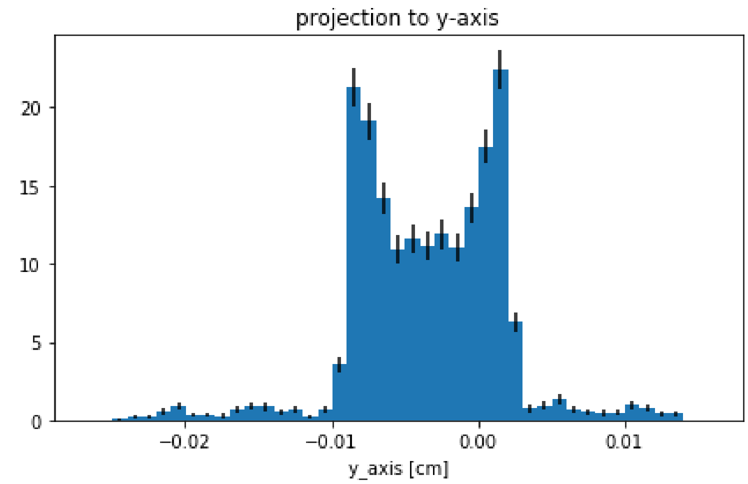
Out[33]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

X-Y Projection

In [34]:
```
xproj = h_2.projection("x_axis [cm]", name="projection to x-axis")
xproj.plot(errors=True);
```



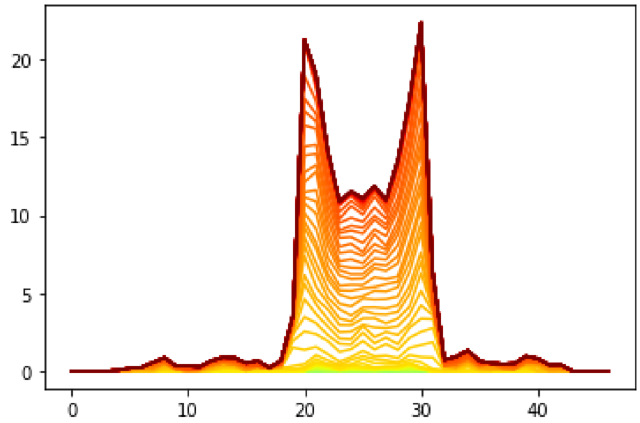
In [35]:
```
yproj = h_2.projection("y_axis [cm]", name="projection to y-axis")
yproj.plot(errors=True);
```



Variable slice for x-projection histogram?

Axes for these two graphs won't rescale to input lengths...

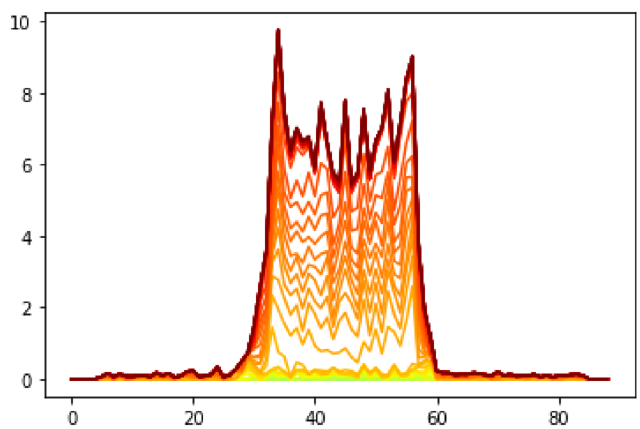
In [82]:
```
plt.gca().set_prop_cycle(plt.cycler('color', plt.cm.jet(np.linspace(0.5, 1, len(h_2.fre
quencies)))))

for f in range(len(h_2.frequencies)):
    plt.plot(sum(h_2.frequencies[:f]),'-');
#y-axis cumulative
```



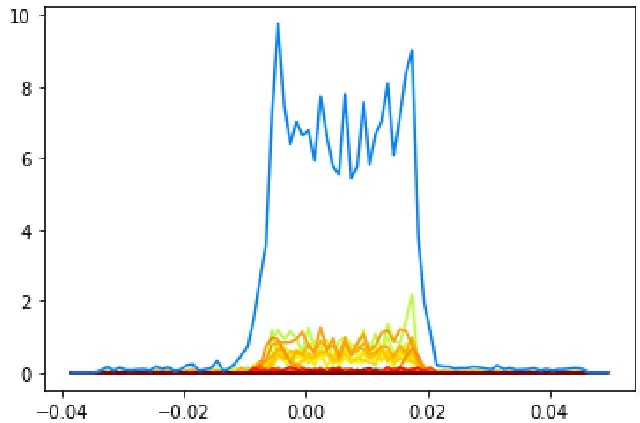
In [37]:
```
print(sum(h_2.frequencies[:20]))
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.0580929  0.27855817 0.18633795 0.05993503
 0.1334885  0.0995562  0.12032538 0.114598   0.13997605 0.2303742
 0.1407109  0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
```

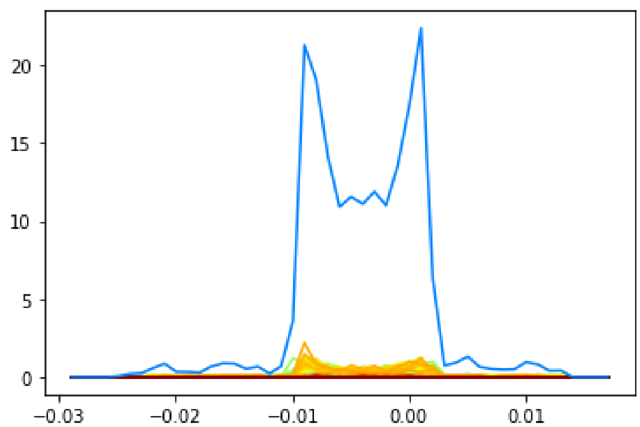
In [69]:
```
plt.gca().set_prop_cycle(plt.cycler('color', plt.cm.jet(np.linspace(0.5, 1, len(h_2.fre
quencies.T)))))
for f in range(len(h_2.frequencies.T)):
    plt.plot(sum(h_2.frequencies.T[:f]),'-');
#x-axis cumulative
```



In [68]:
```
f=1
plt.gca().set_prop_cycle(plt.cycler('color', plt.cm.jet(np.linspace(0.25, 1, len(h_2.frequencies.T)))))
for i in range(len(h_2.frequencies.T)):
    plt.plot(np.arange(np.amin(x_in)-m*dl,np.amax(x_in)+m*dl,dl)[:-1],sum(h_2.frequencies.T[i:f]),'-');
    f += 1

plt.plot(np.arange(np.amin(x_in)-m*dl,np.amax(x_in)+m*dl,dl)[:-1],
         sum(h_2.frequencies.T[:len(h_2.frequencies.T)]),'-');
```



In [56]:
```
f=1
plt.gca().set_prop_cycle(plt.cycler('color', plt.cm.jet(np.linspace(0.25, 1, len(h_2.frequencies)))))
for i in range(len(h_2.frequencies)):
    plt.plot(np.arange(np.amin(y_in)-m*dl,np.amax(y_in)+m*dl,dl)[:-1],sum(h_2.frequencies[i:f]),'-');
    f += 1

plt.plot(np.arange(np.amin(y_in)-m*dl,np.amax(y_in)+m*dl,dl)[:-1],
         sum(h_2.frequencies[:len(h_2.frequencies)]),'-');
```



In [ ]: