Progress thus far

- Delay & Sum is done, with decent results. The sound localization is good, albeit a bit fuzzy.
- After playing around with β_{PM} for a little bit, I got very good results using Pressure Matching. The dark bands are bigger and there are 4 of them now, and the bright bands are more directed. However, I needed to multiply the entire $\mathbf{q}$ vector by scalar g to get the same magnitudes I got with D&S.
- Neither Amplitude Contrast Maximization (ACM) nor Energy Difference Maximization (EDM) have been developed yet. I have been having trouble calculation the regularization parameters (β_{ACM} and α) for both methods. This is likely due to an issue with the eigenvalue routine, which should hopefully get fixed soon. Talking with Dr. Osborne on this would also really help.
- The overall plan for the next few days is to get all four methods working here, in a Python Jupyter notebook, a purely synthetic environment. After that, I will start building the actual beamformer and make my final decision regarding what should be doing the DSP (an FPGA, processor, or my laptop). That choice will likely come down to time, I think. Or how much lab service I end up doing.

In [426]:
```
import numpy as np
import matplotlib.pyplot as plt
from math import exp, pi, e
```

Formulas for Delay & Sum

$$Z(\mathbf{x}_m, \mathbf{y}_\ell, \omega) = \frac{e^{-j\frac{\omega}{c}||\mathbf{x}_m - \mathbf{y}_\ell||}}{4\pi \; || \; \mathbf{x}_m - \mathbf{y}_\ell \; ||}$$

$$\mathbf{p}(\omega) = \mathbf{Z}(\omega)\mathbf{q}(\omega)s(\omega)$$

$$\mathbf{Z}_{M\times L}(\omega) = \begin{bmatrix} Z(\mathbf{x}_1, \mathbf{y}_1, \omega) & \dots & Z(\mathbf{x}_1, \mathbf{y}_L, \omega) \\ \vdots & \ddots & \vdots \\ Z(\mathbf{x}_M, \mathbf{y}_1, \omega) & \dots & Z(\mathbf{x}_M, \mathbf{y}_L, \omega) \end{bmatrix}$$

$$\mathbf{q}_{DAS} = \mathbf{\Gamma}\mathbf{z}_B^*$$

$$\mathbf{\Gamma} = \begin{bmatrix} \gamma_1 & 0 & \dots & 0 \\ 0 & \gamma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \gamma_L \end{bmatrix}$$

$$\gamma_\ell = \frac{16\pi^2 \; || \; \mathbf{x}_B - \mathbf{y}_\ell \; ||^2}{L}$$

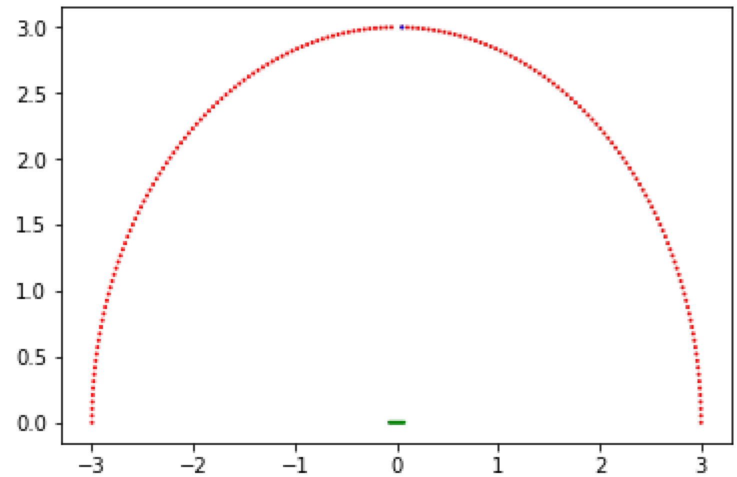
In [474]:
```
c = 343 #speed of sound
l2 = lambda arr: np.sqrt(np.sum(arr**2))


Zf = lambda x_m, y_l, omega: e**(-1j * omega / c * l2(x_m - y_l)) / (4 * pi * l2(
```

In [475]:
```
delta, delta_theta = 0.02, 1 * pi / 180 #sources 20 mm apart, validation points 1
M, L = 181, 8 #181 validation points, 8 sources
R = 3 # 3 meters
x = np.zeros((M, 3))
for i in range(M):
    x[i] = np.array([R*np.cos(delta_theta*(i)), R*np.sin(delta_theta*(i)), 0])
print('\n'.join(['{:d}> {}'.format(i, x[i]) for i in range(M)]))
x[1: M//2+1], x[0] = x[:M//2], x[M//2]
y = np.zeros((L, 3))
for l in range(L):
    y[l] = np.array([(l-3.5)*delta, 0, 0])
print(x, y)
```

```
 [ -2.78155156e+00   1.12381978e+00   0.00000000e+00]
 [ -2.80074128e+00   1.07510385e+00   0.00000000e+00]
 [ -2.81907786e+00   1.02606043e+00   0.00000000e+00]
 [ -2.83655573e+00   9.76704463e-01   0.00000000e+00]
 [ -2.85316955e+00   9.27050983e-01   0.00000000e+00]
 [ -2.86891427e+00   8.77115114e-01   0.00000000e+00]
 [ -2.88378509e+00   8.26912067e-01   0.00000000e+00]
 [ -2.89777748e+00   7.76457135e-01   0.00000000e+00]
 [ -2.91088718e+00   7.25765687e-01   0.00000000e+00]
 [ -2.92311019e+00   6.74853163e-01   0.00000000e+00]
 [ -2.93444280e+00   6.23735072e-01   0.00000000e+00]
 [ -2.94488155e+00   5.72426986e-01   0.00000000e+00]
 [ -2.95442326e+00   5.20944533e-01   0.00000000e+00]
 [ -2.96306502e+00   4.69303395e-01   0.00000000e+00]
 [ -2.97080421e+00   4.17519303e-01   0.00000000e+00]
 [ -2.97763845e+00   3.65608030e-01   0.00000000e+00]
 [ -2.98356569e+00   3.13585390e-01   0.00000000e+00]
 [ -2.98858409e+00   2.61467228e-01   0.00000000e+00]
 [ -2.99269215e+00   2.09269421e-01   0.00000000e+00]
 [ -2.99588860e+00   1.57007869e-01   0.00000000e+00]
```

In [476]:
```
print(np.asarray(np.concatenate((x, y))[:, 0]).shape)
plt.scatter(np.asarray(np.concatenate((x, y))[:, 0]), np.concatenate((x, y))[:, 1
plt.show()
```

```
(189,)
```



In [486]:
```
f = 3000 #3 kHz signal
omega = 2 * pi * f #angular frequency
b = M//2

print(omega, b)

z_b = np.asmatrix(np.vectorize(complex)(np.zeros((L, 1))))
Z_d = np.asmatrix(np.vectorize(complex)(np.zeros((M-1, L))))
print(z_b.shape, Z_d.shape)
for l in range(L):
    print(x[b] - y[l], l2(x[b] - y[l]), Zf(x[b], y[l], omega))
    z_b[l, 0] = Z(x[b], y[l], omega)
n = 0
for m in range(0, M):
    if m == M//2: continue
    for l in range(L):
        print(n, m, l)
        Z_d[n, l] = Z(x[n], y[l], omega)
    n += 1
gamma = lambda x_b, y, l: 16 * (pi ** 2) * l2(x_b - y[l])**2 / L
Gamma = np.asmatrix(np.zeros((L, L)))
for l in range(L):
    Gamma[l, l] = gamma(x[b], y, l)
print(z_b, Gamma)
```

```
50 50 2
50 50 3
50 50 4
50 50 5
50 50 6
50 50 7
51 51 0
51 51 1
51 51 2
51 51 3

51 51 4
51 51 5
51 51 6
51 51 7
52 52 0
52 52 1
52 52 2
52 52 3
52 52 4
52 52 5
```

In [499]:
```
q_DAS = Gamma * np.conjugate(z_b)
```

In [500]:
```
print(q_DAS)
```

```
[[-0.20405013+4.71117287j]
 [-0.01010674+4.71440292j]
 [ 0.14930517+4.71108159j]
 [ 0.27406334+4.70471356j]
 [ 0.36421000+4.6980447j ]
 [ 0.41985128+4.69305916j]
 [ 0.44108448+4.69098116j]
 [ 0.42795313+4.69227772j]]
```

In [501]:
```
Z = np.asmatrix(np.vectorize(complex)(np.zeros((M, L))))
print(Z)
for m in range(M):
    for l in range(L):
        Z[m, l] = Zf(x[m], y[l], omega)
print(Z)
```

```
[[ 0.+0.j  0.+0.j  0.+0.j ...,  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j ...,  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j ...,  0.+0.j  0.+0.j  0.+0.j]
 ...,
 [ 0.+0.j  0.+0.j  0.+0.j ...,  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j ...,  0.+0.j  0.+0.j  0.+0.j]
 [ 0.+0.j  0.+0.j  0.+0.j ...,  0.+0.j  0.+0.j  0.+0.j]]
[[ -1.14703023e-03-0.02648299j  -5.68415159e-05-0.02651437j
    8.40054803e-04-0.02650656j ...,   2.36391255e-03-0.0264236j
    2.48359946e-03-0.02641335j   2.40957934e-03-0.02641975j]
 [  1.54083275e-02+0.02084422j  -1.16423297e-02+0.02334941j
   -2.62620903e-02+0.00024055j ...,   2.65065659e-02+0.0039125j
    8.61732776e-03+0.02556198j  -1.89835720e-02+0.01942331j]
 [  1.53964431e-02+0.02085311j  -1.16519688e-02+0.02334468j
   -2.62621885e-02+0.00023402j ...,   2.65075162e-02+0.00390577j
    8.62818320e-03+0.02555824j  -1.89718461e-02+0.01943462j]
 ...,
 [ -1.89366317e-02+0.01946851j   8.66073632e-03+0.025547j
    2.65103565e-02+0.00388558j ...,  -2.62624734e-02+0.00021443j
   -1.16808717e-02+0.02333044j   1.53607628e-02+0.02087974j]
 [ -1.89718461e-02+0.01943462j   8.62818320e-03+0.02555824j
    2.65075162e-02+0.00390577j ...,  -2.62621885e-02+0.00023402j
   -1.16519688e-02+0.02334468j   1.53964431e-02+0.02085311j]
 [ -1.89835720e-02+0.01942331j   8.61732776e-03+0.02556198j
    2.65065659e-02+0.0039125j  ...,  -2.62620903e-02+0.00024055j
   -1.16423297e-02+0.02334941j   1.54083275e-02+0.02084422j]]
```

In [502]:
```
p = Z * q_DAS
print(p)
```

```
[[  1.00000000e+00 +4.77618352e-18j]
 [ -2.24530648e-01 -1.28799003e-02j]
 [ -2.24499938e-01 -1.28739379e-02j]
 [ -2.24407168e-01 -1.28560273e-02j]
 [ -2.24250422e-01 -1.28260989e-02j]
 [ -2.24026507e-01 -1.27840374e-02j]
 [ -2.23730950e-01 -1.27296838e-02j]
 [ -2.23358002e-01 -1.26628372e-02j]
 [ -2.22900633e-01 -1.25832575e-02j]
 [ -2.22350537e-01 -1.24906686e-02j]
 [ -2.21698130e-01 -1.23847629e-02j]
 [ -2.20932556e-01 -1.22652049e-02j]
 [ -2.20041686e-01 -1.21316369e-02j]
 [ -2.19012131e-01 -1.19836842e-02j]
 [ -2.17829245e-01 -1.18209615e-02j]
 [ -2.16477139e-01 -1.16430796e-02j]
 [ -2.14938698e-01 -1.14496521e-02j]
 [ -2.13195602e-01 -1.12403035e-02j]
 [ -2.11228347e-01 -1.10146772e-02j]
 [ -2.09016281e-01 -1.07724436e-02j]
```

In [503]:
```
M_test = 800 * 400
x_test = np.ones((M_test, 3))
n = 0
for i in range(-400, 400):
    for j in range(0, 400):
        x_test[n] = np.array([i / 100, j / 100, 0])
        n += 1
print(x_test)
Z_test = np.asmatrix(np.vectorize(complex)(np.zeros((M_test, L))))
for m in range(M_test):
    for l in range(L):
        try:
            Z_test[m, l] = Zf(x_test[m], y[l], omega)
        except:
            Z_test[m, l] = 0
print(Z_test)
p_test = Z_test * q_DAS
print('>>', p_test)
print('\n\n>>', x_test, y)
```

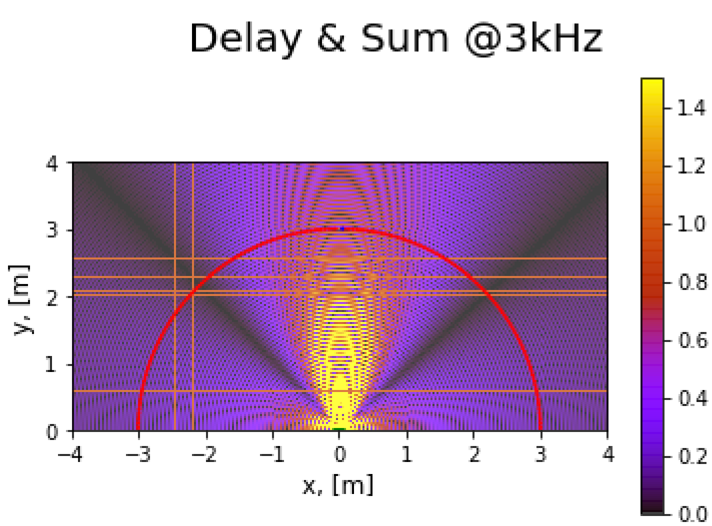
```
[[-4.    0.    0.  ]
 [-4.    0.01  0.  ]
 [-4.    0.02  0.  ]
 ...,
 [ 3.99  3.97  0.  ]
 [ 3.99  3.98  0.  ]
 [ 3.99  3.99  0.  ]]
[[-0.01415315-0.01448099j -0.01923294+0.00599694j -0.00338024+0.01975763j
  ...,  0.00027128-0.01974441j -0.01737877-0.00916799j
  -0.01598478+0.01125946j]
 [-0.01416322-0.01447105j -0.01922870+0.0060103j  -0.00336655+0.01975991j
  ...,  0.00025782-0.01974453j -0.01738494-0.00915617j
  -0.01597713+0.01127022j]
 [-0.01419341-0.01444117j -0.01921593+0.00605036j -0.00332548+0.01976667j
  ...,  0.00021743-0.01974483j -0.01740338-0.00912068j
  -0.01595412+0.01130245j]
 ...,
 [-0.00709451+0.01208554j -0.01359476+0.00354495j -0.01216731-0.00709489j
  ...,  0.01365331-0.00387155j  0.01249317+0.00680733j
   0.00419675+0.01363189j]
 [-0.00204135+0.0138471j  -0.01125075+0.00838551j -0.01392665-0.00198404j
  ...,  0.01114799-0.00875306j  0.01412564+0.00154013j
   0.00906690+0.010987j  ]
 [ 0.00330375+0.01358355j -0.00725792+0.01198883j -0.01362893+0.00341299j
  ...,  0.00697658-0.01231724j  0.01363005-0.00395169j
   0.01255904+0.00668404j]]
>> [[ 0.01385280-0.17130394j]
 [ 0.01373503-0.17131278j]
 [ 0.01338170-0.17133881j]
 ...,
 [ 0.01837335-0.00133758j]
 [ 0.01716767-0.0083957j ]
 [ 0.01324572-0.01471528j]]
```

```
>> [[-4.    0.    0.  ]
 [-4.    0.01  0.  ]
 [-4.    0.02  0.  ]
 ...,
 [ 3.99  3.97  0.  ]
 [ 3.99  3.98  0.  ]
 [ 3.99  3.99  0.  ]] [[-0.07  0.    0.  ]
 [-0.05  0.    0.  ]
 [-0.03  0.    0.  ]
 [-0.01  0.    0.  ]
 [ 0.01  0.    0.  ]
 [ 0.03  0.    0.  ]
 [ 0.05  0.    0.  ]
 [ 0.07  0.    0.  ]]
```

In [504]:

```
output = np.ones((400,800))
for n in range(x_test.shape[0]):
    output[399 - int(x_test[n, 1]*100), int(x_test[n, 0]*100+400)] = p_test[n, 0]
fig = plt.figure()
plt.scatter(np.concatenate((x, y))[:, 0], np.concatenate((x, y))[:, 1], c=['r']*(
implot = plt.imshow(np.abs(output), interpolation='nearest', cmap='gnuplot', exte
plt.colorbar()
fig.suptitle("Delay & Sum @3kHz", fontsize=20)
plt.xlabel("x, [m]", fontsize=12)
plt.ylabel("y, [m]", fontsize=12)
plt.show()
```

```
C:\Users\sarki\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: ComplexWarn
ing: Casting complex values to real discards the imaginary part
  This is separate from the ipykernel package so we can avoid doing imports unt
il
```



Formulas for Pressure matching

$$\min_{\mathbf{q}} J_{PM} = \min_{\mathbf{q}} (\mathbf{e}_{PM}^H \mathbf{e}_{PM} + \beta_{PM} E_q)$$

$$\mathbf{e}_{PM} = \hat{\mathbf{p}} - \mathbf{Z}\mathbf{q}$$

$$E_{\mathbf{q}} = \mathbf{q}^H \mathbf{q}$$

$$\mathbf{q}_{PM} = (\mathbf{Z}^H \mathbf{Z} + \beta_{PM} \mathbf{I})^{-1} \mathbf{Z}^H \hat{\mathbf{p}}$$

In [527]:
```
E_ref = (4 * pi * R)**2 / L
sigma = 25
epsilon_beta, beta_min, E_max = 10e-6, 10e-20, sigma * E_ref
p_b_hat = 1
p_hat = np.asmatrix(np.array([p_b_hat] + [0]*(M-1))).T
g = 30 # weakness compensation

W = lambda q: (p_b_hat / np.dot(z_b.T, q))[0, 0]
E = lambda q: np.dot(np.conjugate(q).T, q)[0, 0]
```

In [528]:
```
### Calculate Tikhonov Regularization Parameter ###

beta = beta_min
print('a')
print(np.conjugate(Z).T.shape, Z.shape, np.asmatrix(np.eye(L)).shape, np.linalg.i
q_temp = np.linalg.inv(np.conjugate(Z).T * Z + beta * np.asmatrix(np.eye(L))) * n

# np.dot(np.dot(np.linalg.inv(np.dot(np.conjugate(ZZ).T, ZZ)
#                                 + beta * np.eye(L)), np.conjugate(ZZ).T), p_hat)

print('b')
q_hat = W(q_temp) * q_temp
n = 0
while E(q_hat) > E_max:
    if n % 1000 == 0: print(beta, E(q_hat), E_max, E(q_hat) > E_max)
    beta += epsilon_beta
    q_temp = np.linalg.inv(np.conjugate(Z).T * Z + beta * np.asmatrix(np.eye(L)))
    q_hat = W(q_temp) * q_temp
    n += 1

print(beta, E(q_hat), E_max, E(q_hat) > E_max)
beta_PM = beta
```

```
a
(8, 181) (181, 8) (8, 8) (8, 8) (181, 1)
b
1e-19 (4578904.81551+0j) 4441.321980490211 True
0.00011000000000000001 (4194.65314199+0j) 4441.321980490211 False
```

In [529]:
```
q_PM = g * np.linalg.inv(np.conjugate(Z).T * Z + beta_PM * np.asmatrix(np.eye(L))

            #g * np.dot(np.dot(np.linalg.inv(np.dot(np.conjugate(ZZ).T, ZZ) + bet

p = Z * q_PM
print(p)
```

```
[[ 0.96503735 -4.52415883e-15j]
 [ 0.15481745 +8.00011972e-03j]
 [ 0.15441202 +7.97633575e-03j]
 [ 0.15319672 +7.90512154e-03j]
 [ 0.15117459 +7.78688975e-03j]
 [ 0.14835065 +7.62232700e-03j]
 [ 0.14473205 +7.41239205e-03j]
 [ 0.14032805 +7.15831318e-03j]
 [ 0.13515014 +6.86158480e-03j]
 [ 0.12921211 +6.52396304e-03j]
 [ 0.12253017 +6.14746045e-03j]
 [ 0.11512304 +5.73433944e-03j]
 [ 0.10701209 +5.28710465e-03j]
 [ 0.09822149 +4.80849386e-03j]
 [ 0.08877831 +4.30146754e-03j]
 [ 0.07871270 +3.76919678e-03j]
 [ 0.06805801 +3.21504958e-03j]
 [ 0.05685098 +2.64257534e-03j]
 [ 0.04513179 +2.05548749e-03j]
```

In [530]:
```
M_test = 800 * 400
x_test = np.ones((M_test, 3))
n = 0
for i in range(-400, 400):
    for j in range(0, 400):
        x_test[n] = np.array([i / 100, j / 100, 0])
        n += 1
print(x_test)
Z_test = np.asmatrix(np.vectorize(complex)(np.zeros((M_test, L))))
for m in range(M_test):
    for l in range(L):
        try:
            Z_test[m, l] = Zf(x_test[m], y[l], omega)
        except:
            Z_test[m, l] = 0
print(Z_test)
p_test2 = Z_test * q_PM
print('>>', p_test2, p_test2.shape)
print('\n\n>>', x_test, y)
```

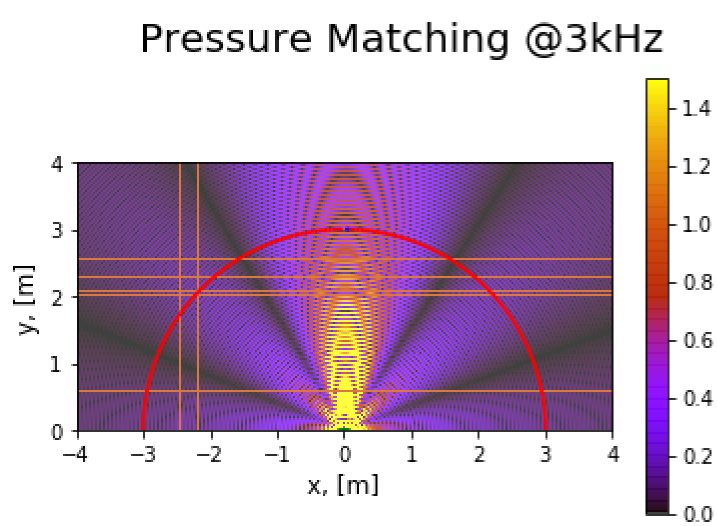
```
[[-4.    0.    0.  ]
 [-4.    0.01  0.  ]
 [-4.    0.02  0.  ]
 ...,
 [ 3.99  3.97  0.  ]
 [ 3.99  3.98  0.  ]
 [ 3.99  3.99  0.  ]]
[[-0.01415315-0.01448099j -0.01923294+0.00599694j -0.00338024+0.01975763j
  ...,  0.00027128-0.01974441j -0.01737877-0.00916799j
  -0.01598478+0.01125946j]
 [-0.01416322-0.01447105j -0.01922870+0.0060103j  -0.00336655+0.01975991j
  ...,  0.00025782-0.01974453j -0.01738494-0.00915617j
  -0.01597713+0.01127022j]
 [-0.01419341-0.01444117j -0.01921593+0.00605036j -0.00332548+0.01976667j
  ...,  0.00021743-0.01974483j -0.01740338-0.00912068j
  -0.01595412+0.01130245j]
 ...,
 [-0.00709451+0.01208554j -0.01359476+0.00354495j -0.01216731-0.00709489j
  ...,  0.01365331-0.00387155j  0.01249317+0.00680733j
   0.00419675+0.01363189j]
 [-0.00204135+0.0138471j  -0.01125075+0.00838551j -0.01392665-0.00198404j
  ...,  0.01114799-0.00875306j  0.01412564+0.00154013j
   0.00906690+0.010987j  ]
 [ 0.00330375+0.01358355j -0.00725792+0.01198883j -0.01362893+0.00341299j
  ...,  0.00697658-0.01231724j  0.01363005-0.00395169j
   0.01255904+0.00668404j]]
>> [[-0.01002116+0.10207439j]
 [-0.00995062+0.10207621j]
 [-0.00973900+0.10208139j]
 ...,
 [-0.11297123-0.01446874j]
 [-0.10973761+0.0292232j ]
 [-0.09024771+0.06837764j]] (320000, 1)
```

```
>> [[-4.    0.    0.  ]
 [-4.    0.01  0.  ]
 [-4.    0.02  0.  ]
 ...,
 [ 3.99  3.97  0.  ]
 [ 3.99  3.98  0.  ]
 [ 3.99  3.99  0.  ]] [[-0.07  0.    0.  ]
 [-0.05  0.    0.  ]
 [-0.03  0.    0.  ]
 [-0.01  0.    0.  ]
 [ 0.01  0.    0.  ]
 [ 0.03  0.    0.  ]
 [ 0.05  0.    0.  ]
 [ 0.07  0.    0.  ]]
```

In [531]:

```
output = np.ones((400,800))
for n in range(x_test.shape[0]):
    output[399 - int(x_test[n, 1]*100), int(x_test[n, 0]*100+400)] = p_test2[n, 0
fig = plt.figure()
plt.scatter(np.concatenate((x, y))[:, 0], np.concatenate((x, y))[:, 1], c=['r']*(l
implot = plt.imshow(np.abs(output), interpolation='nearest', cmap='gnuplot', exte
plt.colorbar()
fig.suptitle("Pressure Matching @3kHz", fontsize=20)
plt.xlabel("x, [m]", fontsize=12)
plt.ylabel("y, [m]", fontsize=12)
plt.show()
```

```
C:\Users\sarki\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: ComplexWarn
ing: Casting complex values to real discards the imaginary part
  This is separate from the ipykernel package so we can avoid doing imports unt
il
```



In [532]:
```
print(q_DAS, q_PM, np.average(q_DAS / q_PM))
```

```
[[-0.20405013+4.71117287j]
 [-0.01010674+4.71440292j]
 [ 0.14930517+4.71108159j]
 [ 0.27406334+4.70471356j]
 [ 0.36421000+4.6980447j ]
 [ 0.41985128+4.69305916j]
 [ 0.44108448+4.69098116j]
 [ 0.42795313+4.69227772j]] [[-4.24528812+22.55133075j]
 [ 8.15669008-26.35215524j]
 [-5.11258791+25.04097817j]
 [-2.15854695 -3.25158063j]
 [ 4.61473470 -2.94830762j]
 [ 1.05705848+25.38077933j]
 [-4.01363225-26.99611305j]
 [ 2.96923682+22.83613293j]] (-0.126521345014+0.0148289148266j)
```

Acoustic Contract Maximization Formulas

$$E_B = \frac{1}{\rho_0 c^2} \mid p_B \mid^2 = \frac{1}{\rho_0 c^2} \mathbf{q}^H \mathbf{R}_B \mathbf{q}$$

$$E_D = \frac{1}{\rho_0 c^2} \frac{1}{M_D} \mid\mid \mathbf{p}_D \mid\mid^2 = \frac{1}{\rho_0 c^2} \mathbf{q}^H \mathbf{R}_D \mathbf{q}$$

$$\mathbf{R}_B = \mathbf{z}_B^H \mathbf{z}_B$$

$$\mathbf{R}_D = \mathbf{Z}_D^H \mathbf{Z}_D / M_D$$

$$AC = \frac{E_B}{E_D} = \frac{\mathbf{q}^H \mathbf{R}_B \mathbf{q}}{\mathbf{q}^H \mathbf{R}_D \mathbf{q}}$$

$$\max_{\mathbf{q}} J_{ACM} = \max_{\mathbf{q}} \frac{\mathbf{q}^H \mathbf{R}_B \mathbf{q}}{\mathbf{q}^H \mathbf{R}_D \mathbf{q}}$$

$$\mathbf{q}_{ACM} = \Phi(\mathbf{R}_D^{-1} \mathbf{R}_B)$$

However, $\mathbf{R}_D^{-1}$ is rather unsavory and often ill-conditioned to inversion. Rather, we would use

$$\mathbf{q}_{ACM} = \Phi\Big((\mathbf{R}_D + \beta_{ACM} \mathbf{I})^{-1} \mathbf{R}_B \Big)$$

where $\Phi(\mathbf{A})$ is the eigenvector corresponding to the maximum eigenvalue of matrix $\mathbf{A}$.

In [559]:
```
E_ref = (4 * pi * R)**2 / L
sigma = 25
epsilon_beta, beta_min, E_max = 10e-9, 10e-20, sigma * E_ref
p_b_hat = 1
p_hat = np.asmatrix(np.array([p_b_hat] + [0]*(M-1))).T

W = lambda q: (p_b_hat / np.dot(z_b.T, q))[0, 0]
#W = lambda q: 1 #temporary....
E = lambda q: np.dot(np.conjugate(q).T, q)[0, 0]
def PHI(A):
    B = A.real
    w, v = np.linalg.eig(B)
    print(w)
    i = np.argmax(w)
    print(i)
    print(v[:, i])
    print(B * v[:, i] / v[:, i])
    return v[:, i]

M_D = M - 1
R_D = np.dot(np.conjugate(Z_d).T, Z_d) / (M-1)
R_B = np.dot(np.conjugate(z_b).T, z_b)

print(np.conjugate(z_b).T, z_b)

print(R_B)

print(type(R_D), type(R_B))

print(R_D.shape, R_B.shape)
```

```
[[ -1.14703023e-03+0.02648299j  -5.68415159e-05+0.02651437j
    8.40054803e-04+0.02650656j   1.54249351e-03+0.02647924j
    2.05033734e-03+0.02644786j   2.36391255e-03+0.0264236j
    2.48359946e-03+0.02641335j   2.40957934e-03+0.02641975j]] [[ -1.14703023e-0
3-0.02648299j]
 [ -5.68415159e-05-0.02651437j]
 [  8.40054803e-04-0.02650656j]
 [  1.54249351e-03-0.02647924j]
 [  2.05033734e-03-0.02644786j]
 [  2.36391255e-03-0.0264236j ]
 [  2.48359946e-03-0.02641335j]
 [  2.40957934e-03-0.02641975j]]
[[ 0.00562764+0.j]]
<class 'numpy.matrixlib.defmatrix.matrix'> <class 'numpy.matrixlib.defmatrix.ma
trix'>
(8, 8) (1, 1)
```

In [562]:
```
### Calculate ACM Regularization Parameter ###


beta = beta_min
print('a')
q_temp = PHI(np.linalg.inv(R_D + beta * np.asmatrix(np.eye(L))) * R_B[0, 0])
# q_temp = PHI(np.dot(np.linalg.inv(R_D + beta * np.eye(L)), R_B))
print('b')
q_hat = W(q_temp) * q_temp
n = 0
print(z_b.T, q_temp, z_b.T * q_temp)

print(W(q_temp), q_hat)
print("E =", E(q_hat))
return
while E(q_hat) > E_max:
    if n % 1000 == 0:
        print(beta, E(q_hat), E_max, E(q_hat) > E_max)
    beta += epsilon_beta
    q_temp = PHI(np.linalg.inv(R_D + beta * np.asmatrix(np.eye(L))) * R_B[0, 0])
    q_hat = W(q_temp) * q_temp
    n += 1

print(beta, E(q_hat), E_max, E(q_hat) > E_max)
beta_ACM = beta
```

```
a
[  3.85007763e+07   2.33990395e+05   3.61908532e+03   1.15959324e+02
   9.88286292e+00   4.29325815e+00   2.99844711e+00   3.10027200e+00]
0
[[ 0.03028093]
 [-0.15464089]
 [ 0.38014602]
 [-0.57506416]
 [ 0.5750379 ]
 [-0.38009401]
 [ 0.15460617]
 [-0.03027141]]
[[ 38500776.30020554]
 [ 38500776.30020554]
 [ 38500776.30020548]
 [ 38500776.30020548]
 [ 38500776.30020548]
 [ 38500776.30020548]
 [ 38500776.30020547]
 [ 38500776.30020548]]
b
[[ -1.14703023e-03-0.02648299j  -5.68415159e-05-0.02651437j
    8.40054803e-04-0.02650656j   1.54249351e-03-0.02647924j
    2.05033734e-03-0.02644786j   2.36391255e-03-0.0264236j
    2.48359946e-03-0.02641335j   2.40957934e-03-0.02641975j]] [[ 0.03028093]
 [-0.15464089]
 [ 0.38014602]
 [-0.57506416]
 [ 0.5750379 ]
```

```
 [-0.38009401]
 [ 0.15460617]
 [-0.03027141]] [[ -2.08125453e-06 +2.00105028e-07j]]
1 [[ 0.03028093]
 [-0.15464089]
 [ 0.38014602]
 [-0.57506416]
 [ 0.5750379 ]
 [-0.38009401]
 [ 0.15460617]
 [-0.03027141]]
E = 1.0

  File "<ipython-input-562-fd5cb8b912e5>", line 15
    return
          ^
SyntaxError: 'return' outside function
```

In [410]:

```
q_ACM = PHI(np.dot(np.linalg.inv(R_D), R_B))
print(q_ACM, q_DAS, q_DAS.real / q_ACM.real)
```

```
[-0.03056138 +3.46882534e-05j  0.15525338 +6.72881479e-06j
 -0.38045526 -1.33860403e-04j  0.57467685 +0.00000000e+00j
 -0.57465195 +5.41708986e-04j  0.38040639 -9.41969248e-04j
 -0.15522097 +7.24904088e-04j  0.03055282 -2.36279763e-04j] [[-4.42774546+1.611
11273j -4.42310489+1.62357719j -4.43059043+1.60341753j
  -4.44975458+1.55046996j -4.47941986+1.46432173j -4.51766439+1.34435495j
  -4.56180116+1.18981816j -4.60835469+0.99992665j]] [[ 144.88042254  -28.489587
71   11.6454965    -7.74305515    7.79501369
   -11.87588991   29.3890781  -150.8323745 ]]
```

In [413]:
```
M_test = 800 * 400
x_test = np.ones((M_test, 3))
n = 0
for i in range(-400, 400):
    for j in range(0, 400):
        x_test[n] = np.array([i / 100, j / 100, 0])
        n += 1
print(x_test)
ZZ_test = np.vectorize(complex)(np.zeros((M_test, L)))
for m in range(M_test):
    for l in range(L):
        try:
            ZZ_test[m, l] = Z(x_test[m], y[l], omega)
        except:
            ZZ_test[m, l] = 0
print(ZZ_test)
p_test = np.dot(ZZ_test, q_ACM.T)
print('>>', p_test, p_test.shape)
print('\n\n>>', x_test, y)
```

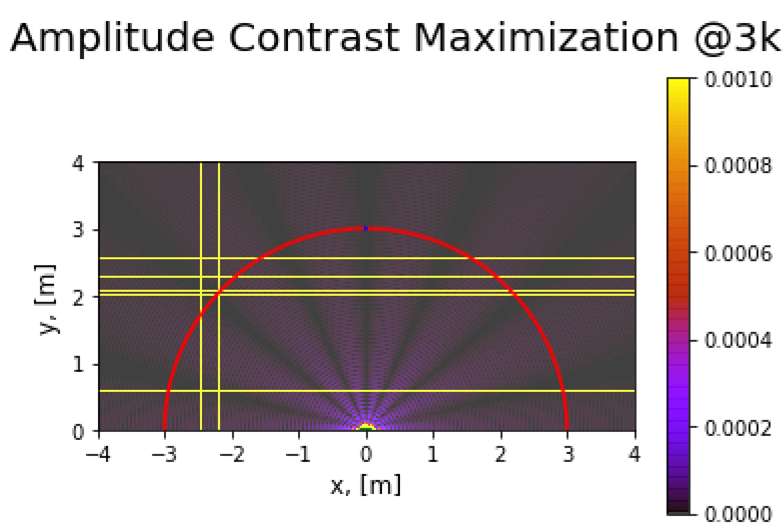
```
[[-4.    0.    0.  ]
 [-4.    0.01  0.  ]
 [-4.    0.02  0.  ]
 ...,
 [ 3.99  3.97  0.  ]
 [ 3.99  3.98  0.  ]
 [ 3.99  3.99  0.  ]]
[[ -1.22167916e-02 +1.61480861e-02j   8.94701020e-03 +1.80504902e-02j
    2.00446263e-02 +5.55163640e-05j ...,  -1.94300831e-02 +3.51952905e-03j
   -5.50079608e-03 +1.88630578e-02j   1.43501319e-02 +1.32801509e-02j]
 [ -1.22053688e-02 +1.61566395e-02j   8.95963577e-03 +1.80441545e-02j
    2.00445965e-02 +4.15323070e-05j ...,  -1.94275999e-02 +3.53287071e-03j
   -5.48787841e-03 +1.88667576e-02j   1.43591223e-02 +1.32703426e-02j]
 [ -1.21710649e-02 +1.61822508e-02j   8.99748531e-03 +1.80250947e-02j
    2.00444488e-02 -4.19159442e-07j ...,  -1.94200953e-02 +3.57288492e-03j
   -5.44911071e-03 +1.88778040e-02j   1.43860533e-02 +1.32408811e-02j]
 ...,
 [  1.29958148e-02 -5.24415624e-03j   1.28983411e-02 +5.56927750e-03j
    5.14539576e-03 +1.31112826e-02j ...,  -1.30084970e-02 -5.67280682e-03j
   -5.25325409e-03 -1.32220353e-02j   5.57023837e-03 -1.31306361e-02j]
 [  1.00380666e-02 -9.75431055e-03j   1.40294079e-02 +2.68074806e-04j
    9.72632718e-03 +1.01629990e-02j ...,  -1.41711966e-02 -2.66411240e-04j
   -9.90793098e-03 -1.01852206e-02j   9.69116900e-05 -1.42447708e-02j]
 [  5.59438307e-03 -1.28113377e-02j   1.30667884e-02 -5.06640627e-03j
    1.28436598e-02 +5.69532832e-03j ...,  -1.31766618e-02 +5.17326929e-03j
   -1.30416426e-02 -5.59549850e-03j  -5.38376418e-03 -1.31689423e-02j]]
>> [  1.47576456e-05 -2.61536238e-06j   1.47538823e-05 -2.62518670e-06j
   1.47425539e-05 -2.65463586e-06j ...,   3.76992311e-06 +3.07837368e-06j
   4.72108447e-06 +1.43089564e-06j   4.97513263e-06 -4.82560764e-07j] (320000,)


>> [[-4.    0.    0.  ]
 [-4.    0.01  0.  ]
 [-4.    0.02  0.  ]
```

```
 ...,
 [ 3.99  3.97  0.  ]
 [ 3.99  3.98  0.  ]
 [ 3.99  3.99  0.  ]] [[-0.07  0.    0.  ]
 [-0.05  0.    0.  ]
 [-0.03  0.    0.  ]
 [-0.01  0.    0.  ]
 [ 0.01  0.    0.  ]
 [ 0.03  0.    0.  ]
 [ 0.05  0.    0.  ]
 [ 0.07  0.    0.  ]]
```

In [418]:

```python
output = np.ones((400,800))
for n in range(x_test.shape[0]):
    output[399 - int(x_test[n, 1]*100), int(x_test[n, 0]*100+400)] = p_test[n]
print(np.max(output))
fig = plt.figure()
plt.scatter(np.concatenate((x, y))[:, 0], np.concatenate((x, y))[:, 1], c=['r']*(
implot = plt.imshow(np.abs(output), interpolation='nearest', cmap='gnuplot', vmin
plt.colorbar()
fig.suptitle("Amplitude Contrast Maximization @3kHz", fontsize=20)
plt.xlabel("x, [m]", fontsize=12)
plt.ylabel("y, [m]", fontsize=12)
plt.show()
```

```
C:\Users\sarki\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: ComplexWarn
ing: Casting complex values to real discards the imaginary part
  This is separate from the ipykernel package so we can avoid doing imports unt
il

2.33394760509
```



The above plot makes no sense, likely because it omits the regularization parameter β_{ACM}. This cements the importance of regularization

Equations for Energy Difference

Maximization

$$\max_{\mathbf{q}} \frac{E_B - \alpha E_D}{E_{\mathbf{q}}} = \max_{\mathbf{q}} \frac{\mathbf{q}^H(\mathbf{R}_B - \alpha \mathbf{R}_D)\mathbf{q}}{\mathbf{q}^H\mathbf{q}}$$

$$\mathbf{q}_{EDM} = \Phi(\mathbf{R}_B - \alpha \mathbf{R}_D)$$

In [384]:

```
E_ref = (4 * pi * R)**2 / L
sigma = 1.05
epsilon_alpha, alpha_max, E_max = 1e2, 1e6, sigma * E_ref
p_b_hat = 1
p_hat = np.array([0]*(M//2) + [1] + [0]*(M//2))

W = lambda q: p_b_hat / np.dot(z_b.T, q)
E = lambda q: np.dot(np.conjugate(q).T, q)
def PHI(A):
    w, v = np.linalg.eig(A)
    i = np.argmax(w)
    return v[:, i]

M_D = M - 1
R_D = np.dot(np.conjugate(Z_d).T, Z_d).real / (M_D)
R_B = np.dot(np.conjugate(z_b).T, z_b).real

print(M_D, R_D, R_B)
```

```
180 [[  7.03820003e-04   5.03863000e-04   7.16140042e-05  -2.45788624e-04
   -2.36252144e-04   4.55922445e-06   1.96757577e-04   1.57841092e-04]
 [  5.03863000e-04   7.03684463e-04   5.03774310e-04   7.15699207e-05
   -2.45807129e-04  -2.36245596e-04   4.59739389e-06   1.96806324e-04]
 [  7.16140042e-05   5.03774310e-04   7.03611504e-04   5.03730421e-04
    7.15322220e-05  -2.45847472e-04  -2.36260054e-04   4.63597547e-06]
 [ -2.45788624e-04   7.15699207e-05   5.03730421e-04   7.03601088e-04
    5.03731310e-04   7.15008879e-05  -2.45909677e-04  -2.36295526e-04]
 [ -2.36252144e-04  -2.45807129e-04   7.15322220e-05   5.03731310e-04
    7.03653209e-04   5.03776978e-04   7.14759017e-05  -2.45993776e-04]
 [  4.55922445e-06  -2.36245596e-04  -2.45847472e-04   7.15008879e-05
    5.03776978e-04   7.03767895e-04   5.03867449e-04   7.14572501e-05]
 [  1.96757577e-04   4.59739389e-06  -2.36260054e-04  -2.45909677e-04
    7.14759017e-05   5.03867449e-04   7.03945207e-04   5.04002770e-04]
 [  1.57841092e-04   1.96806324e-04   4.63597547e-06  -2.36295526e-04
   -2.45993776e-04   7.14572501e-05   5.04002770e-04   7.04185240e-04]] 0.00562
764336198
```

In [385]:

```
### Calculate alpha ###

alpha = alpha_max
q_temp = PHI(R_B - alpha * R_D)
q_hat = W(q_temp) * q_temp
n = 0
while E(q_hat) > E_max and alpha >= 0:
    if n % 1000:
        print(alpha, E(q_hat), E_max, E(q_hat) > E_max)
    alpha = alpha - epsilon_alpha
    q_temp = PHI(R_B - alpha * R_D)
    q_hat = W(q_temp) * q_temp
    n += 1

print(alpha, E(q_hat), E_max, E(q_hat) > E_max)
alpha_EDM = alpha
```

```
80300.0 (202233112841+0j) 186.53552318058888 True
80200.0 (202233112834+0j) 186.53552318058888 True
80100.0 (202233112813+0j) 186.53552318058888 True
80000.0 (202233112797+0j) 186.53552318058888 True
79900.0 (202233112779+0j) 186.53552318058888 True
79800.0 (202233112773+0j) 186.53552318058888 True
79700.0 (202233112747+0j) 186.53552318058888 True
79600.0 (202233112737+0j) 186.53552318058888 True
79500.0 (202233112719+0j) 186.53552318058888 True
79400.0 (202233112698+0j) 186.53552318058888 True
79300.0 (202233112692+0j) 186.53552318058888 True
79200.0 (202233112673+0j) 186.53552318058888 True
79100.0 (202233112651+0j) 186.53552318058888 True
79000.0 (202233112635+0j) 186.53552318058888 True
78900.0 (202233112631+0j) 186.53552318058888 True
78800.0 (202233112603+0j) 186.53552318058888 True
78700.0 (202233112596+0j) 186.53552318058888 True
78600.0 (202233112566+0j) 186.53552318058888 True
78500.0 (202233112561+0j) 186.53552318058888 True
78400.0 (202233112542+0j) 186.53552318058888 True
```

In [453]:
```
q_EDM = PHI(R_D - alpha * R_B)
print(q_EDM, q_DAS, np.average(q_DAS / q_EDM))
p = np.dot(ZZ, q_EDM.T)
print(p)
print(np.max(p))
```

```
[-0.41100165-0.01978789j -0.28569511+0.03771158j  0.11410313+0.08442244j
  0.46737839+0.07169087j  0.47335254+0.j          0.12701134-0.07695935j
 -0.27611268-0.09927769j -0.41072655-0.05482335j] [[ 0.42795313+4.69227772j  0.
44108448+4.69098116j  0.41985128+4.69305916j
   0.36421000+4.6980447j   0.27406334+4.70471356j  0.14930517+4.71108159j
  -0.01010674+4.71440292j -0.20405013+4.71117287j]] (0.239559987472+2.246323529
83j)
[  9.46335080e-03-0.05666284j   9.46404126e-03-0.05667323j
   9.46610277e-03-0.05670436j   9.46950572e-03-0.05675607j
   9.47420084e-03-0.05682807j   9.48011927e-03-0.05692001j
   9.48717269e-03-0.05703137j   9.49525349e-03-0.05716158j
   9.50423500e-03-0.05730991j   9.51397171e-03-0.05747554j
   9.52429964e-03-0.05765754j   9.53503664e-03-0.05785486j
   9.54598288e-03-0.05806633j   9.55692129e-03-0.05829066j
   9.56761816e-03-0.05852647j   9.57782377e-03-0.05877221j
   9.58727311e-03-0.05902627j   9.59568667e-03-0.05928687j
   9.60277138e-03-0.05955215j   9.60822158e-03-0.0598201j
   9.61172011e-03-0.06008861j   9.61293954e-03-0.06035546j
   9.61154345e-03-0.06061829j   9.60718786e-03-0.06087465j
   9.59952276e-03-0.06112197j   9.58819373e-03-0.06135761j
   9.57284371e-03-0.06157878j   9.55311479e-03-0.06178265j
   9.52865022e-03-0.06196627j   9.49909643e-03-0.06212664j
   9.46410512e-03-0.06226069j   9.42333552e-03-0.0623653j
   9.37645663e-03-0.0624373j    9.32314955e-03-0.0624735j
   9.26310984e-03-0.06247071j   9.19604994e-03-0.06242572j
   9.12170154e-03-0.06233538j   9.03981798e-03-0.06219655j
   8.95017661e-03-0.06200617j   8.85258114e-03-0.06176124j
   8.74686383e-03-0.0614589j    8.63288767e-03-0.06109638j
   8.51054842e-03-0.0606711j    8.37977654e-03-0.06018062j
   8.24053888e-03-0.05962273j   8.09284033e-03-0.05899541j
   7.93672514e-03-0.05829694j   7.77227812e-03-0.05752582j
   7.59962553e-03-0.05668088j   7.41893582e-03-0.05576128j
   7.23041999e-03-0.05476649j   7.03433177e-03-0.05369639j
   6.83096750e-03-0.05255121j   6.62066565e-03-0.0513316j
   6.40380618e-03-0.05003863j   6.18080948e-03-0.0486738j
   5.95213511e-03-0.04723906j   5.71828020e-03-0.04573681j
   5.47977762e-03-0.0441699j    5.23719386e-03-0.04254166j
   4.99112668e-03-0.04085586j   4.74220252e-03-0.03911672j
   4.49107370e-03-0.03732892j   4.23841542e-03-0.03549757j
   3.98492263e-03-0.03362816j   3.73130674e-03-0.03172662j
   3.47829216e-03-0.02979921j   3.22661288e-03-0.02785255j
   2.97700886e-03-0.02589355j   2.73022246e-03-0.02392939j
   2.48699482e-03-0.02196749j   2.24806231e-03-0.02001544j
   2.01415299e-03-0.018081j     1.78598310e-03-0.01617197j
   1.56425373e-03-0.01429626j   1.34964753e-03-0.0124617j
   1.14282554e-03-0.01067611j   9.44424216e-04-0.00894718j
   7.55052574e-04-0.00728241j   5.75289495e-04-0.00568909j
   4.05681238e-04-0.00417425j   2.46739113e-04-0.00274456j
   9.89373526e-05-0.00140634j  -3.72888181e-05-0.00016547j
  -1.61544923e-04+0.00097264j  -2.73478774e-04+0.00200309j
```

```
  -3.72781809e-04+0.0029215j   -4.59190239e-04+0.00372411j
  -5.32485994e-04+0.00440773j  -5.92497473e-04+0.00496983j
  -6.39100105e-04+0.00540851j  -6.72216711e-04+0.00572254j
  -6.91817689e-04+0.00591135j  -6.97920991e-04+0.00597505j
  -6.90591928e-04+0.00591442j  -6.69942780e-04+0.00573089j
  -6.36132220e-04+0.00542653j  -5.89364548e-04+0.00500405j
  -5.29888745e-04+0.00446674j  -4.57997330e-04+0.00381849j
  -3.74025038e-04+0.00306368j  -2.78347308e-04+0.00220725j
  -1.71378586e-04+0.00125454j  -5.35704455e-05+0.00021135j
   7.45904714e-05-0.00091617j   2.12584688e-04-0.00212155j
   3.59862297e-04-0.00339805j   5.15845525e-04-0.0047387j
   6.79931451e-04-0.00613638j   8.51494864e-04-0.00758385j
   1.02989125e-03-0.00907379j   1.21445990e-03-0.01059888j
   1.40452708e-03-0.01215183j   1.59940935e-03-0.01372545j
   1.79841684e-03-0.01531264j   2.00085667e-03-0.0169065j
   2.20603629e-03-0.01850033j   2.41326688e-03-0.02008767j
   2.62186666e-03-0.02166235j   2.83116419e-03-0.02321849j
   3.04050153e-03-0.02475056j   3.24923737e-03-0.02625339j
   3.45674993e-03-0.02772216j   3.66243978e-03-0.02915246j
   3.86573247e-03-0.03054028j   4.06608092e-03-0.03188201j
   4.26296770e-03-0.03317446j   4.45590691e-03-0.03441485j
   4.64444605e-03-0.03560082j   4.82816739e-03-0.03673039j
   5.00668928e-03-0.037802j     5.17966712e-03-0.03881446j
   5.34679399e-03-0.03976699j   5.50780119e-03-0.04065911j
   5.66245832e-03-0.04149073j   5.81057326e-03-0.04226207j
   5.95199183e-03-0.04297365j   6.08659717e-03-0.04362627j
   6.21430902e-03-0.04422099j   6.33508267e-03-0.04475914j
   6.44890776e-03-0.04524223j   6.55580692e-03-0.04567198j
   6.65583422e-03-0.04605031j   6.74907346e-03-0.04637926j
   6.83563643e-03-0.04666101j   6.91566096e-03-0.04689786j
   6.98930892e-03-0.04709219j   7.05676421e-03-0.04724647j
   7.11823065e-03-0.04736321j   7.17392981e-03-0.04744496j
   7.22409897e-03-0.04749429j   7.26898890e-03-0.04751377j
   7.30886180e-03-0.04750598j   7.34398924e-03-0.04747346j
   7.37465011e-03-0.04741873j   7.40112865e-03-0.04734425j
   7.42371258e-03-0.04725245j   7.44269127e-03-0.04714567j
   7.45835399e-03-0.04702622j   7.47098825e-03-0.04689629j
   7.48087826e-03-0.04675801j   7.48830347e-03-0.04661343j
   7.49353715e-03-0.0464645j    7.49684517e-03-0.04631308j
   7.49848479e-03-0.04616093j   7.49870358e-03-0.04600971j
   7.49773844e-03-0.04586098j   7.49581465e-03-0.04571621j
   7.49314512e-03-0.04557674j   7.48992958e-03-0.04544385j
   7.48635397e-03-0.04531868j   7.48258984e-03-0.04520228j
   7.47879382e-03-0.0450956j    7.47510716e-03-0.04499949j
   7.47165538e-03-0.0449147j    7.46854790e-03-0.04484186j
   7.46587775e-03-0.04478153j   7.46372136e-03-0.04473414j
   7.46213836e-03-0.04470005j   7.46117146e-03-0.04467949j
   7.46084631e-03-0.04467262j]
(0.0096129395363-0.0603554565737j)
```

I gave up on EDM after seeing this maximum value. I need to get the regularization parameters figured out.

In [ ]: