In [233]:
```
import numpy as np
import matplotlib.pyplot as plt
from math import exp, pi, e
```

Formulas for Delay & Sum

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

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

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

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

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

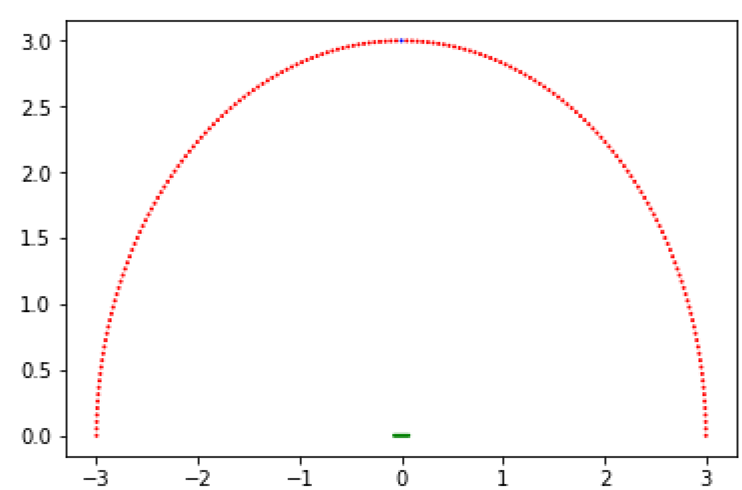
$$\gamma_\ell = \frac{16\pi^2 \ || \ \mathbf{x}_B - \mathbf{y}_\ell \ ||^2}{L}$$

In [234]:
```
c = 340.29 #speed of sound
l2 = lambda arr: np.sqrt(np.sum(arr**2))

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

In [235]:
```
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])
y = np.zeros((L, 3))
for l in range(L):
    y[l] = np.array([(l-3.5)*delta, 0, 0])
print(x, y)
```

```
[[  3.00000000e+00   0.00000000e+00   0.00000000e+00]
 [  2.99954309e+00   5.23572193e-02   0.00000000e+00]
 [  2.99817248e+00   1.04698490e-01   0.00000000e+00]
 [  2.99588860e+00   1.57007869e-01   0.00000000e+00]
 [  2.99269215e+00   2.09269421e-01   0.00000000e+00]
 [  2.98858409e+00   2.61467228e-01   0.00000000e+00]
 [  2.98356569e+00   3.13585390e-01   0.00000000e+00]
 [  2.97763845e+00   3.65608030e-01   0.00000000e+00]
 [  2.97080421e+00   4.17519303e-01   0.00000000e+00]
 [  2.96306502e+00   4.69303395e-01   0.00000000e+00]
 [  2.95442326e+00   5.20944533e-01   0.00000000e+00]
 [  2.94488155e+00   5.72426986e-01   0.00000000e+00]
 [  2.93444280e+00   6.23735072e-01   0.00000000e+00]
 [  2.92311019e+00   6.74853163e-01   0.00000000e+00]
 [  2.91088718e+00   7.25765687e-01   0.00000000e+00]
 [  2.89777748e+00   7.76457135e-01   0.00000000e+00]
 [  2.88378509e+00   8.26912067e-01   0.00000000e+00]
 [  2.86891427e+00   8.77115114e-01   0.00000000e+00]
 [  2.85316955e+00   9.27050983e-01   0.00000000e+00]
```

In [236]:
```
plt.scatter(np.concatenate((x, y))[:, 0], np.concatenate((x, y))[:, 1], c=['r']*(
plt.show()
```



In [343]:
```
f = 3000 #3 kHz signal
omega = 2 * pi * f #angular frequency
b = M//2 + 1

print(omega, b)

z_b = np.vectorize(complex)(np.zeros((L,)))
Z_d = np.vectorize(complex)(np.zeros((M-1, L)))
for l in range(L):
    print(x[b] - y[l], l2(x[b] - y[l]), Z(x[b], y[l], omega))
    z_b[l] = Z(x[b], y[l], omega)
n = 0
for m in range(0, M):
    if m == M//2+1: 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)
```

```
113 114 2
113 114 3
113 114 4
113 114 5
113 114 6
113 114 7
114 115 0
114 115 1
114 115 2
114 115 3
114 115 4
114 115 5
114 115 6
114 115 7
115 116 0
115 116 1
115 116 2
115 116 3
115 116 4
115 116 5
```

In [238]:
```
q_DAS = np.dot(Gamma, np.conjugate(z_b))
```

In [239]:
```
print(q)
```

```
[-0.09112509 -0.05408986 -0.06236384 -0.08392807 -0.08392807 -0.06236384
 -0.05408986 -0.09112509]
```

In [240]:
```
ZZ = np.vectorize(complex)(np.zeros((M, L)))
print(ZZ)
for m in range(M):
    for l in range(L):
        ZZ[m, l] = Z(x[m], y[l], omega)
print(ZZ)
```

```
[[ 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]]
[[ 0.02377938-0.01031693j  0.01998065+0.01677833j -0.00612943+0.02553792j
  ...,  0.01086478-0.02449209j  0.02694759-0.00122485j
   0.01321992+0.02372498j]
 [ 0.02378542-0.01030324j  0.01997374+0.01678666j -0.00613584+0.02553642j
  ...,  0.01085850-0.02449483j  0.02694699-0.00123641j
   0.01323421+0.02371689j]
 [ 0.02380346-0.01026216j  0.01995298+0.01681164j -0.00615506+0.02553192j
  ...,  0.01083966-0.02450303j  0.02694517-0.00127108j
   0.01327706+0.02369259j]
 ...,
 [ 0.01327706+0.02369259j  0.02694517-0.00127108j  0.01083966-0.02450303j
  ..., -0.00615506+0.02553192j  0.01995298+0.01681164j
   0.02380346-0.01026216j]
 [ 0.01323421+0.02371689j  0.02694699-0.00123641j  0.01085850-0.02449483j
  ..., -0.00613584+0.02553642j  0.01997374+0.01678666j
   0.02378542-0.01030324j]
 [ 0.01321992+0.02372498j  0.02694759-0.00122485j  0.01086478-0.02449209j
  ..., -0.00612943+0.02553792j  0.01998065+0.01677833j
   0.02377938-0.01031693j]]
```

In [241]:

```
p = np.dot(ZZ, q_DAS.T)
print(p)
```

```
[[-0.22901233 -1.33190174e-02j]
 [-0.22901123 -1.33143032e-02j]
 [-0.22900729 -1.33001317e-02j]
 [-0.22899866 -1.32764160e-02j]
 [-0.22898225 -1.32430120e-02j]
 [-0.22895370 -1.31997205e-02j]
 [-0.22890741 -1.31462884e-02j]
 [-0.22883655 -1.30824120e-02j]
 [-0.22873299 -1.30077396e-02j]
 [-0.22858736 -1.29218759e-02j]
 [-0.22838901 -1.28243857e-02j]
 [-0.22812602 -1.27147993e-02j]
 [-0.22778517 -1.25926180e-02j]
 [-0.22735198 -1.24573198e-02j]
 [-0.22681065 -1.23083666e-02j]
 [-0.22614413 -1.21452109e-02j]
 [-0.22533407 -1.19673039e-02j]
 [-0.22436086 -1.17741034e-02j]
 [-0.22320364 -1.15650827e-02j]
```

In [300]:

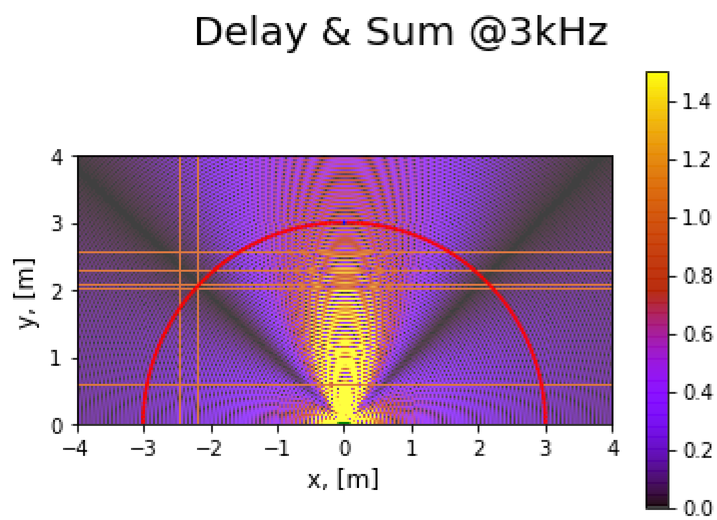
```
M_test = 800 * 400
x_test = np.ones((M_test, 3))
n = 0
for i in range(-400, 400):
    for j in range(0, 400):
        x_test[n] = np.array([i / 100, j / 100, 0])
        n += 1
print(x_test)
ZZ_test = np.vectorize(complex)(np.zeros((M_test, L)))
for m in range(M_test):
    for l in range(L):
   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

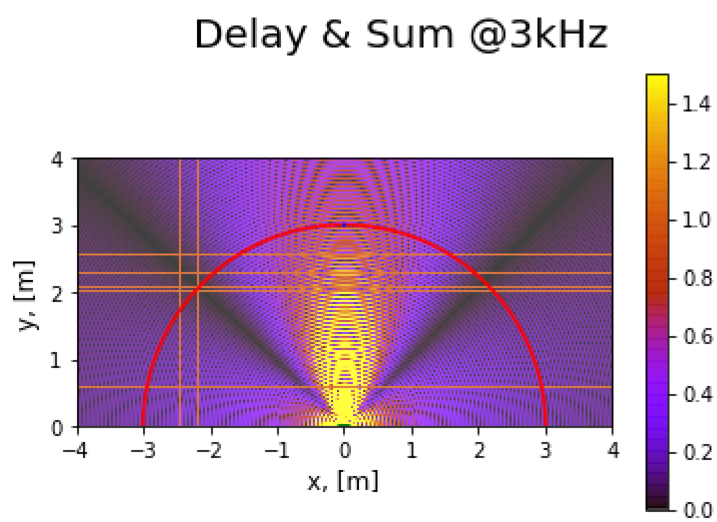
```
[[-4.    0.    0.  ]
 [-4.    0.01  0.  ]
 [-4.    0.02  0.  ]
 ...,
 [ 3.99  3.97  0.  ]
 [ 3.99  3.98  0.  ]
 [ 3.99  3.99  0.  ]]
[[ -1.22167916e-02 +1.61480861e-02j   8.94701020e-03 +1.80504902e-02j
    2.00446263e-02 +5.55163640e-05j ...,  -1.94300831e-02 +3.51952905e-03j
   -5.50079608e-03 +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]]
>> [[-0.05912782-0.15910241j]
 [-0.05923768-0.15906056j]
 [-0.05956706-0.15893454j]
 ...,
 [-0.00873668+0.01020983j]
 [-0.00411028+0.01210428j]
 [ 0.00063612+0.01211422j]]
```

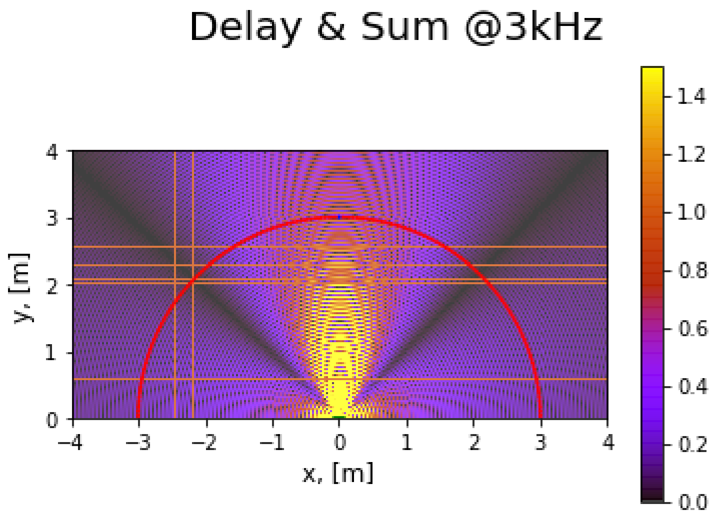
```
>> [[-4.    0.    0.  ]
 [-4.    0.01  0.  ]
 [-4.    0.02  0.  ]
 ...,
 [ 3.99  3.97  0.  ]
 [ 3.99  3.98  0.  ]
 [ 3.99  3.99  0.  ]] [[-0.07  0.    0.  ]
 [-0.05  0.    0.  ]
 [-0.03  0.    0.  ]
 [-0.01  0.    0.  ]
 [ 0.01  0.    0.  ]
 [ 0.03  0.    0.  ]
 [ 0.05  0.    0.  ]
 [ 0.07  0.  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

In [305]:
```
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']*(M
imlplot = plt.imshow(np.abs(output), interpolation='nearest', cmap='gnuplot', extent
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 [324]:
```
E_ref = (4 * pi * R)**2 / L
sigma = 1.05
epsilon_beta, beta_min, E_max = 1e-2, 0, sigma * E_ref
p_b_hat = 1
p_hat = np.array([0]*(M//2) + [1] + [0]*(M//2))
g = 60 # weakness compensation

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

In [325]:
```
### Calculate Tikhonov Regularization Parameter ###

beta = beta_min
print('a')
q_temp = 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.dot(np.dot(np.linalg.inv(np.dot(np.conjugate(ZZ).T, ZZ)
                                + beta * np.eye(L)), np.conjugate(ZZ).T), p_hat).
    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
b
0 (1204990.82048+0j) 186.53552318058888 True
0.11999999999999998 (185.826655992+0j) 186.53552318058888 False
```

In [336]:
```
q_PM = g *np.dot(np.dot(np.linalg.inv(np.dot(np.conjugate(ZZ).T, ZZ) + beta_PM * 

p = np.dot(ZZ, q_DAS.T)
print(p)
```

```
 [-0.19358861 -9.24228920e-03j]
 [-0.19755037 -9.58454572e-03j]
 [-0.20112707 -9.90898753e-03j]
 [-0.20434580 -1.02156429e-02j]
 [-0.20723281 -1.05046222e-02j]
 [-0.20981343 -1.07761083e-02j]
 [-0.21211196 -1.10303466e-02j]
 [-0.21415169 -1.12676359e-02j]
 [-0.21595475 -1.14883182e-02j]
 [-0.21754216 -1.16927701e-02j]
 [-0.21893373 -1.18813933e-02j]
 [-0.22014807 -1.20546063e-02j]
 [-0.22120254 -1.22128360e-02j]
 [-0.22211325 -1.23565099e-02j]
 [-0.22289505 -1.24860489e-02j]
 [-0.22356148 -1.26018604e-02j]
 [-0.22412483 -1.27043323e-02j]
 [-0.22459608 -1.27938267e-02j]
 [-0.22498491 -1.28706757e-02j]
 [-0.22529972 -1.29351760e-02j]
```

In [337]:
```
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_PM.T)
print('>>', p_test, p_test.shape)
print('\n\n>>', x_test, y)
```

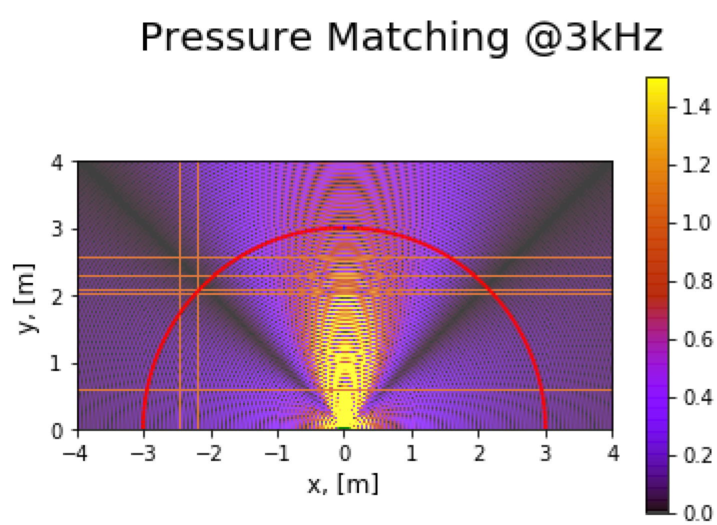
```
[[-4.    0.    0.  ]
 [-4.    0.01  0.  ]
 [-4.    0.02  0.  ]
 ...,
 [ 3.99  3.97  0.  ]
 [ 3.99  3.98  0.  ]
 [ 3.99  3.99  0.  ]]
[[ -1.22167916e-02 +1.61480861e-02j   8.94701020e-03 +1.80504902e-02j
    2.00446263e-02 +5.55163640e-05j ...,  -1.94300831e-02 +3.51952905e-03j
   -5.50079608e-03 +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]]
>> [-0.04527666-0.12127454j -0.04536064-0.12124315j -0.04561246-0.12114864j
 ..., -0.00456595+0.00226549j -0.00328904+0.00322106j
 -0.00198142+0.00363931j] (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 [338]:

```
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]
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 [329]:

```
print(q_DAS, q_PM, np.average(q_DAS / q_PM))
```

```
[[-4.42774546+1.61111273j -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]] [-5.46750534+1.52310267j -
3.24539183+1.09455475j -3.74183017+1.28210136j
 -5.03568418+1.6294137j  -5.03568418+1.6294137j  -3.74183017+1.28210136j
 -3.24539183+1.09455475j -5.46750534+1.52310267j] (1.07095055903+0.006224611498
36j)
```

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 a matrix $\mathbf{A}$.

In [340]:

```
E_ref = (4 * pi * R)**2 / L
sigma = 1.05
epsilon_beta, beta_min, E_max = 1e-2, 0, 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 =
R_B
```

In [ ]:

```
### Calculate ACM Regularization Parameter ###

beta = beta_min
print('a')
q_temp = PHI(np.linalg.inv())
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.dot(np.dot(np.linalg.inv(np.dot(np.conjugate(ZZ).T, ZZ)
                                + beta * np.eye(L)), np.conjugate(ZZ).T), p_hat).
    q_hat = W(q_temp) * q_temp
    n += 1

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