The code to the 1-group, fixed-source, steady-state, finite difference diffusion equation will work in the following way. We will order our unknowns ijk by spanning the i coordinates first, then the j coordinates, and finally the k coordinates. As an illustration we start at 111, 211, 311, … $I11$, then go to 121, 221, …, $I21$, and so on until we get to $1J1, 2J1, \ldots IJ1$, and then go to $112, 212, 312, I12$. This means that we first span a single xy plane at a time before increasing the z coordinate. It will be useful to have a function to translate between ijk coordinates and a single coordinate l:

In [468]:
```
def coordLookup_l(i, j, k, I, J):
    """get the position in a 1-D vector
    for the (i,j,k) index
    """
    return i + j*I + k*J*I

def coordLookup_ijk(l, I, J):
    """get the position in a (i,j,k)  coordinates
    for the index l in a 1-D vector
    """
    k = (l // (I*J)) + 1
    j = (l - k*J*I) // I + 1
    i = l - (j*I + k*J*I)-1
    return i,j,k
```

We will also need to define arrays that contain the diffusion coefficient, D, the macroscopic absorption cross-section, Σ_a , and the source, Q. We will define these with ijk coordinates.

The there are six boundary conditions that we need to define for the problem: the two x-faces that we call the left and right faces, the two y-faces that we call front and back, and the two z-faces that we call top and bottom. For each of these we need to define 3 values, $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$.

The next code snippet will build the linear system and solve it using SciPy. We use sparse matrices for this problem.

In [269]:

```
import numpy as np
import scipy as sp
import scipy.sparse as sparse
import scipy.sparse.linalg as splinalg

def diffusion_steady_fixed_source(Dims,Lengths,BCs,D,Sigma,Q, toleranc
e=1.0e-12, LOUD=False):
    """Solve a steady state, single group diffusion problem with a fixed
source
    Inputs:
        Dims:            number of zones (I,J,K)
        Lengths:         size in each dimension (Nx,Ny,Nz)
        BCs:             A, B, and C for each boundary, there are 8 of t
hese
        D,Sigma,Q:       Each is an array of size (I,J,K) containing the
quantity
    Outputs:
        x,y,z:           Vectors containing the cell centers in each dim
ension
        phi:             A vector containing the solution
    """
    I = Dims[0]
    J = Dims[1]
    K = Dims[2]
    L = I*J*K
    Nx = Lengths[0]
    Ny = Lengths[1]
    Nz = Lengths[2]

    hx,hy,hz = np.array(Lengths)/np.array(Dims)
    ihx2,ihy2,ihz2 = (1.0/hx**2,1.0/hy**2,1.0/hz**2)

    #allocate the A matrix, and b vector
    A = sparse.lil_matrix((L,L))
    b = np.zeros(L)

    temp_term = 0
    for k in range(K):
        for j in range(J):
            for i in range(I):
                temp_term = Sigma[i,j,k]
                row = coordLookup_l(i,j,k,I,J)
                b[row] = Q[i,j,k]
                #do x-term left
                if (i>0):
                    Dhat = 2* D[i,j,k]*D[i-1,j,k] / (D[i,j,k] + D[i-
1,j,k])
                    temp_term += Dhat*ihx2
                    A[row, coordLookup_l(i-1,j,k,I,J)]  = -Dhat*ihx2
                else:
                    bA,bB,bC = BCs[0,:]
                    if (np.abs(bB) > 1.0e-8):
                        if (i<I-1):
                            temp_term += -1.5*D[i,j,k]*bA/bB/hx
                            b[row] += -D[i,j,k]/bB*bC/hx
```

```
                    A[row,  coordLookup_l(i+1,j,k,I,J)]  +=
0.5*D[i,j,k]*bA/bB/hx
                else:
                    temp_term += -0.5*D[i,j,k]*bA/bB/hx
                    b[row] += -D[i,j,k]/bB*bC/hx
            else:
                temp_term += D[i,j,k]*ihx2*2.0
                b[row] += D[i,j,k]*bC/bA*ihx2*2.0
        #do x-term right
        if (i < I-1):
            Dhat = 2* D[i,j,k]*D[i+1,j,k] / (D[i,j,k] +
D[i+1,j,k])
            temp_term += Dhat*ihx2
            A[row, coordLookup_l(i+1,j,k,I,J)]  += -Dhat*ihx2
        else:
            bA,bB,bC = BCs[1,:]
            if (np.abs(bB) > 1.0e-8):
                if (i>0):
                    temp_term += 1.5*D[i,j,k]*bA/bB/hx
                    b[row] += D[i,j,k]/bB*bC/hx
                    A[row,  coordLookup_l(i-1,j,k,I,J)]  +=
-0.5*D[i,j,k]*bA/bB/hx
                else:
                    temp_term += -0.5*D[i,j,k]*bA/bB/hx
                    b[row] += -D[i,j,k]/bB*bC/hx

            else:
                temp_term += D[i,j,k]*ihx2*2.0
                b[row] += D[i,j,k]*bC/bA*ihx2*2.0
        #do y-term
        if (j>0):
            Dhat = 2* D[i,j,k]*D[i,j-1,k] / (D[i,j,k] + D[i,j-
1,k])
            temp_term += Dhat*ihy2
            A[row, coordLookup_l(i,j-1,k,I,J)]  += -Dhat*ihy2
        else:
            bA,bB,bC = BCs[2,:]
            if (np.abs(bB) > 1.0e-8):
                if (j<J-1):
                    temp_term += -1.5*D[i,j,k]*bA/bB/hy
                    b[row] += -D[i,j,k]/bB*bC/hy
                    A[row,  coordLookup_l(i,j+1,k,I,J)]  +=
0.5*D[i,j,k]*bA/bB/hy
                else:
                    temp_term += -0.5*D[i,j,k]*bA/bB/hy
                    b[row] += -D[i,j,k]/bB*bC/hy
            else:
                temp_term += D[i,j,k]*ihy2*2.0
                b[row] += D[i,j,k]*bC/bA*ihy2*2.0
        if (j < J-1):
            Dhat = 2* D[i,j,k]*D[i,j+1,k] / (D[i,j,k] +
D[i,j+1,k])
            temp_term += Dhat*ihy2
            A[row, coordLookup_l(i,j+1,k,I,J)]  += -Dhat*ihy2
        else:
            bA,bB,bC = BCs[3,:]
```

```
                    if (np.abs(bB) > 1.0e-8):
                        if (j>0):
                            temp_term += 1.5*D[i,j,k]*bA/bB/hy
                            b[row] += D[i,j,k]/bB*bC/hy
                            A[row,  coordLookup_l(i,j-1,k,I,J)]  += 
-0.5*D[i,j,k]*bA/bB/hy
                        else:
                            temp_term += 0.5*D[i,j,k]*bA/bB/hy
                            b[row] += D[i,j,k]/bB*bC/hy

                    else:
                        temp_term += D[i,j,k]*ihy2*2.0
                        b[row] += D[i,j,k]*bC/bA*ihy2*2.0
                #do z-term
                if (k>0):
                    Dhat = 2* D[i,j,k]*D[i,j,k-1] / (D[i,j,k] + D[i,j,k-
1])
                    temp_term += Dhat*ihz2
                    A[row, coordLookup_l(i,j,k-1,I,J)]  += -Dhat*ihz2
                else:
                    bA,bB,bC = BCs[4,:]
                    if (np.abs(bB) > 1.0e-8):
                        if (k<K-1):
                            temp_term += -1.5*D[i,j,k]*bA/bB/hz
                            b[row] += -D[i,j,k]/bB*bC/hz
                            A[row,  coordLookup_l(i,j,k+1,I,J)]  += 
0.5*D[i,j,k]*bA/bB/hz
                        else:
                            temp_term += -0.5*D[i,j,k]*bA/bB/hz
                            b[row] += -D[i,j,k]/bB*bC/hz
                    else:
                        temp_term += D[i,j,k]*ihz2*2.0
                        b[row] += D[i,j,k]*bC/bA*ihz2*2.0
                if (k < K-1):
                    Dhat = 2* D[i,j,k]*D[i,j,k+1] / (D[i,j,k] + 
D[i,j,k+1])
                    temp_term += Dhat*ihz2
                    A[row, coordLookup_l(i,j,k+1,I,J)]  += -Dhat*ihz2
                else:
                    bA,bB,bC = BCs[5,:]
                    if (np.abs(bB) > 1.0e-8):
                        if (k>0):
                            temp_term += 1.5*D[i,j,k]*bA/bB/hz
                            b[row] += D[i,j,k]/bB*bC/hz
                            A[row,  coordLookup_l(i,j,k-1,I,J)]  += 
-0.5*D[i,j,k]*bA/bB/hz
                        else:
                            temp_term += 0.5*D[i,j,k]*bA/bB/hz
                            b[row] += D[i,j,k]/bB*bC/hz

                    else:
                        temp_term += D[i,j,k]*ihz2*2.0
                        b[row] += D[i,j,k]*bC/bA*ihz2*2.0
                A[row,row] += temp_term
    phi,code = splinalg.cg(A,b, tol=tolerance)
    if (LOUD):
```

```
        print("The CG solve exited with code",code)
    phi_block = np.zeros((I,J,K))
    for k in range(K):
        for j in range(J):
            for i in range(I):
                phi_block[i,j,k] = phi[coordLookup_l(i,j,k,I,J)]
    x = np.linspace(hx*.5,Nx-hx*.5,I)
    y = np.linspace(hy*.5,Ny-hy*.5,J)
    z = np.linspace(hz*.5,Nz-hz*.5,K)
    if (I*J*K <= 10):
        print(A.toarray())
    return x,y,z,phi_block
```

To test this code we will solve a simple infinite medium problem. We will have reflective boundary conditions everywhere, $\mathcal{A} = \mathcal{C} = 0$, and $\mathcal{B} = 1$. We will set the diffusion coefficient to 3 and $\Sigma_a = 2$ and $Q = 1$. The solution to this problem is $\phi = 0.5$ everywhere.

In [270]:
```
I = 3
J = 2
K = 1
Sigma = np.ones((I,J,K))*2
D = Sigma*3/2
Q = Sigma*0.5
BCs = np.ones((6,3))
BCs[:,0] = 0
BCs[:,2] = 0

x,y,z,phi_infinite_medium = diffusion_steady_fixed_source((I,J,K),
(3,2,1),BCs,D,Sigma,Q)
print("Solution is")
print(phi_infinite_medium)
```

```
[[  8.  -3.   0.  -3.   0.   0.]
 [ -3.  11.  -3.   0.  -3.   0.]
 [  0.  -3.   8.   0.   0.  -3.]
 [ -3.   0.   0.   8.  -3.   0.]
 [  0.  -3.   0.  -3.  11.  -3.]
 [  0.   0.  -3.   0.  -3.   8.]]
Solution is
[[[ 0.5]
  [ 0.5]]

 [[ 0.5]
  [ 0.5]]

 [[ 0.5]
  [ 0.5]]]
```

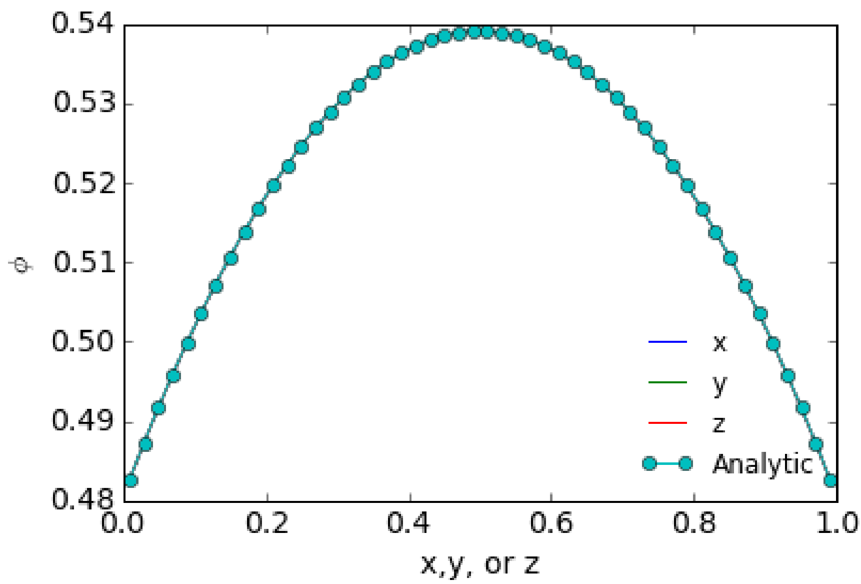
That solution looks good. Now let's try a different problem. It will have vacuum Marshak conditions, and $Q = D = \Sigma_\mathrm{a} = 1$. The solution to this problem is

$$\phi(x) = \frac{e^{1-x} + e^x + 1 - 3e}{1 - 3e}.$$

In [271]:
```
import matplotlib.pyplot as plt
%matplotlib inline
#solve in x direction
print("Solving Problem in X direction")
I = 50
J = 1
K = 1
Nx = 1
Sigma = np.ones((I,J,K))
D = Sigma.copy()
Q = Sigma.copy()
BCs = np.ones((6,3))
BCs[:,0] = 0
BCs[:,2] = 0
BCs[0,:] = [0.25,-D[0,0,0]/2,0]
BCs[1,:] = [0.25,D[I-1,0,0]/2,0]

x,y,z,phi_x = diffusion_steady_fixed_source((I,J,K),(Nx,Nx*1,Nx),BCs,D,S
igma,Q)
plt.plot(x,phi_x[:,0,0],label='x')
solution = (np.exp(1-x) + np.exp(x) + 1 - 3*np.exp(1))/(1-3*np.exp(1))
#solve in y direction
print("Solving Problem in Y direction")
I = 1
J = 50
K = 1
Nx = 1
Sigma = np.ones((I,J,K))
D = Sigma.copy()
Q = Sigma.copy()
BCs = np.ones((6,3))
BCs[:,0] = 0
BCs[:,2] = 0
BCs[2,:] = [0.25,-D[0,0,0]/2,0]
BCs[3,:] = [0.25,D[0,J-1,0]/2,0]
x,y,z,phi_y = diffusion_steady_fixed_source((I,J,K),(Nx,Nx*1,Nx),BCs,D,S
igma,Q)
plt.plot(y,phi_y[0,:,0],label='y')

#solve in z direction
print("Solving Problem in Z direction")
I = 1
J = 1
K = 50
Nx = 1
Sigma = np.ones((I,J,K))
D = Sigma.copy()
Q = Sigma.copy()
BCs = np.ones((6,3))
BCs[:,0] = 0
BCs[:,2] = 0
BCs[4,:] = [0.25,-D[0,0,0]/2,0]
BCs[5,:] = [0.25,D[0,J-1,0]/2,0]
x,y,z,phi_z = diffusion_steady_fixed_source((I,J,K),(Nx,Nx*1,Nx),BCs,D,S
igma,Q)
```

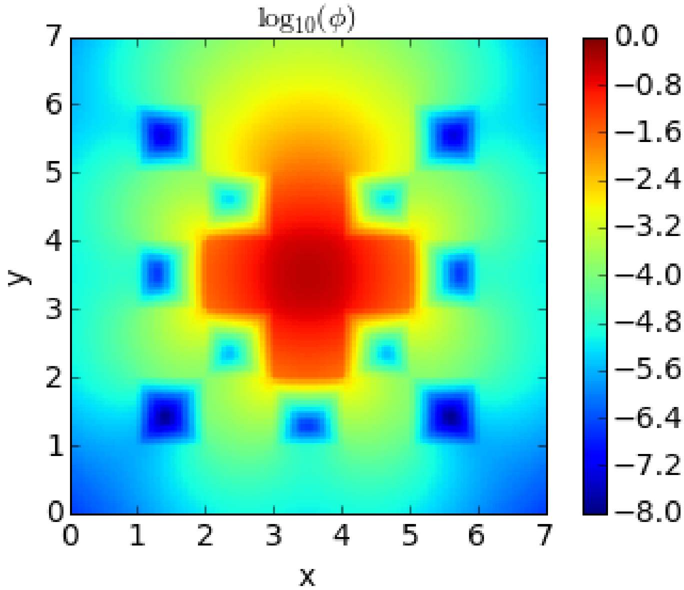
```
plt.plot(z,phi_z[0,0,:],label='z')

plt.plot(z,solution,'o-',label='Analytic')
plt.xlabel("x,y, or z")
plt.ylabel("$\phi$")
plt.legend(loc=4)
plt.show()
```

```
Solving Problem in X direction
Solving Problem in Y direction
Solving Problem in Z direction
```



In [272]:
```
def lattice(Lengths,Dims):
    I = Dims[0]
    J = Dims[1]
    K = Dims[2]
    L = I*J*K
    Nx = Lengths[0]
    Ny = Lengths[1]
    Nz = Lengths[2]
    hx,hy,hz = np.array(Lengths)/np.array(Dims)

    Sigma = np.ones((I,J,K))*1
    Q = np.zeros((I,J,K))
    for k in range(K):
        for j in range(J):
            for i in range(I):
                x = (i+0.5)*hx
                y = (j+0.5)*hy
                z = (k+0.5)*hz

                if (x>=3.0) and (x<=4.0):
                    if (y>=3.0) and (y<=4.0):
                        Q[i,j,k] = 1.0
                    if (y>=1.0) and (y<=2.0):
                        Sigma[i,j,k] = 10.0
                if ( ((x>=1.0) and (x<=2.0)) or ((x>=5.0) and
(x<=6.0))):
                    if ( ((y>=1.0) and (y<=2.0)) or
                        ((y>=3.0) and (y<=4.0)) or
                        ((y>=5.0) and (y<=6.0))):
                        Sigma[i,j,k] = 10.0
                if ( ((x>=2.0) and (x<=3.0)) or ((x>=4.0) and
(x<=5.0))):
                    if ( ((y>=2.0) and (y<=3.0)) or
                        ((y>=4.0) and (y<=5.0))):
                        Sigma[i,j,k] = 10.0

    D = 1.0/(3.0*Sigma)
    return D,Q,Sigma


I = 150
J = 150
K = 1
Nx = 7
D,Q,Sigma = lattice((Nx,Nx,1),(I,J,K))
BCs = np.ones((6,3))
BCs[:,0] = 0
BCs[:,2] = 0
BCs[(0,2,4),:] = [0.25,-D[0,0,0]/2,0]
BCs[(1,3,5),:] = [0.25,D[I-1,0,0]/2,0]

x,y,z,phi = diffusion_steady_fixed_source((I,J,K),(Nx,Nx*1,Nx),BCs,D,Sig
ma,Q)
plt.pcolor(x,y,np.transpose(np.log10(np.abs(phi[:,:,0]))))
plt.colorbar()
```

```
plt.clim([-8,0])
plt.axes().set_aspect('equal',adjustable='box')
plt.xlabel('x')
plt.ylabel('y')
plt.title("$\log_{10} (\phi)$")
plt.show()
```



In [273]:
```
np.min(phi)
```

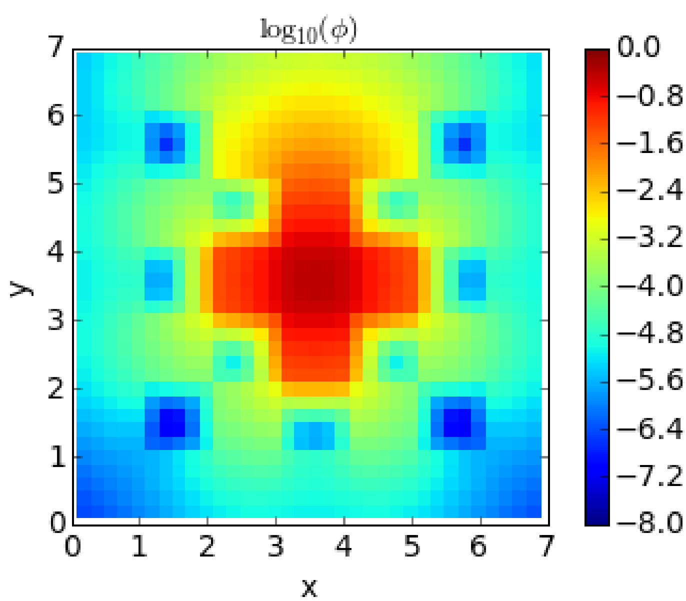
Out[273]: 1.461780509074057e-08

Now we will make this solve a fixed source, time-dependent problem

In [274]:
```
def time_dependent_diffusion(phi0, v, dt, T, Dims,Lengths,BCs,D,Sigma,Q,
tolerance=1.0e-12):
    """Solve a time dependent diffusion problem
    Inputs:
    phi0:       initial condition
    v:          particle speed
    dt:         time step size
    T:          final time
    The remaining inputs are the same as for diffusion_steady_fixed_sour
ce
    Outputs:
        x,y,z:           Vectors containing the cell centers in each dim
ension
        phi:             A vector containing the solution at time T
    """
    numsteps = int(T/dt)
    for step in range(numsteps):
        Qhat = Q + phi0/dt/v
        Sighat = Sigma + 1.0/dt/v
        print("Solving for time",(step+1)*dt)
        x,y,z,phi0 = diffusion_steady_fixed_source(Dims,Lengths,BCs,D,
                                                    Sigma,Q, tolerance)
    return x,y,z,phi0
```

In [275]:
```
I = 35
J = 35
K = 1
Nx = 7
D,Q,Sigma = lattice((Nx,Nx,1),(I,J,K))
BCs = np.ones((6,3))
BCs[:,0] = 0
BCs[:,2] = 0
BCs[(0,2,4),:] = [0.25,-D[0,0,0]/2,0]
BCs[(1,3,5),:] = [0.25,D[I-1,0,0]/2,0]

phi = np.zeros((I,J,K))
x,y,z,phi = time_dependent_diffusion(phi,1,0.1,1,(I,J,K),(Nx,Nx*1,Nx),BC
s,D,Sigma,Q)
plt.pcolor(x,y,np.transpose(np.log10(np.abs(phi[:,:,0]))))
plt.colorbar()
plt.clim([-8,0])
plt.axes().set_aspect('equal',adjustable='box')
plt.xlabel('x')
plt.ylabel('y')
plt.title("$\log_{10} (\phi)$")
plt.show()
```

```
Solving for time 0.1
Solving for time 0.2
Solving for time 0.30000000000000004
Solving for time 0.4
Solving for time 0.5
Solving for time 0.6000000000000001
Solving for time 0.7000000000000001
Solving for time 0.8
Solving for time 0.9
Solving for time 1.0
```



In [276]:
```
def lattice2G(Lengths,Dims):
    I = Dims[0]
    J = Dims[1]
    K = Dims[2]
    L = I*J*K
    Nx = Lengths[0]
    Ny = Lengths[1]
    Nz = Lengths[2]
    hx,hy,hz = np.array(Lengths)/np.array(Dims)

    Sigmaag = np.ones((I,J,K,2))*1
    Sigmasgg = np.zeros((I,J,K,2,2))
    nuSigmafg = np.zeros((I,J,K,2))
    nug = np.ones((I,J,K,2))*2.3
    chig = np.zeros((I,J,K,2))
    D = np.zeros((I,J,K,2))
    Q = np.zeros((I,J,K,2))

    Sigmasgg[:,:,:,0,0] = 0.05
    Sigmasgg[:,:,:,0,1] = 0.1
    Sigmasgg[:,:,:,1,1] = 0.25




    for k in range(K):
        for j in range(J):
            for i in range(I):
                x = (i+0.5)*hx
                y = (j+0.5)*hy
                z = (k+0.5)*hz


                if (x>=3.0) and (x<=4.0):
                    if (y>=3.0) and (y<=4.0):
                        Q[i,j,k,0] = 1.0
                    if (y>=1.0) and (y<=2.0):
                        Sigmaag[i,j,k,(0,1)] = [10.0,15.0]
                        nuSigmafg[i,j,k,(0,1)] = [5.0,7.5]
                        chig[i,j,k,(0,1)] = [1.0,0.0]
                elif ( ((x>=1.0) and (x<=2.0)) or ((x>=5.0) and
(x<=6.0))):
                    if ( ((y>=1.0) and (y<=2.0)) or
                        ((y>=3.0) and (y<=4.0)) or
                        ((y>=5.0) and (y<=6.0))):
                        Sigmaag[i,j,k,(0,1)] = [10.0,15.0]
                        nuSigmafg[i,j,k,(0,1)] = [5.0,7.5]
                        chig[i,j,k,(0,1)] = [1.0,0.0]
                elif  ( ((x>=2.0) and (x<=3.0)) or ((x>=4.0) and
(x<=5.0))):
                    if ( ((y>=2.0) and (y<=3.0)) or
                        ((y>=4.0) and (y<=5.0))):
                        Sigmaag[i,j,k,(0,1)] = [10.0,15.0]
                        nuSigmafg[i,j,k,(0,1)] = [5.0,7.5]
                        chig[i,j,k,(0,1)] = [1.0,0.0]
```

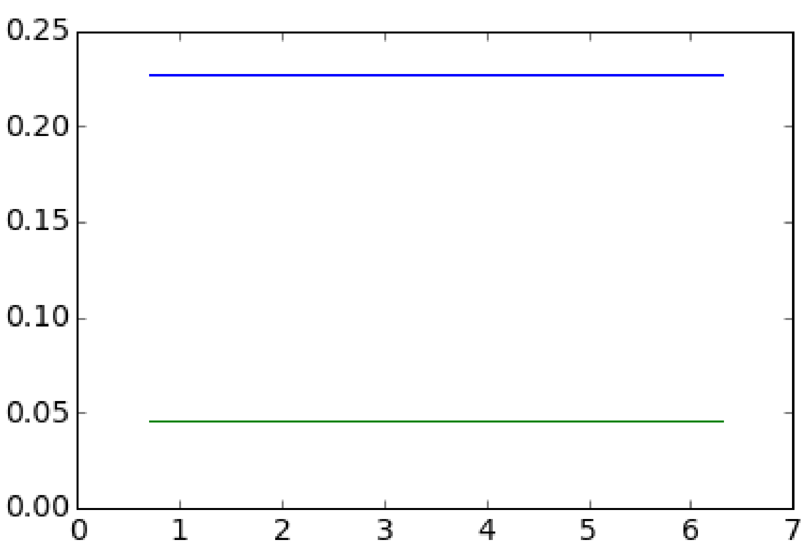
```
    D[:,:,:,0] = 1.0/(3.0*(Sigmaag[:,:,:,0]))
    D[:,:,:,1] = 1.0/(3.0*(Sigmaag[:,:,:,1]))
    return   Sigmaag,Sigmasgg,nuSigmafg,nug,chig,D,Q
```

In [277]:

```
def steady_multigroup_diffusion(G,Dims,Lengths,BCGs,
                                Sigmatg,Sigmasgg,nuSigmafg,
                                nug,chig,D,Q,
                                lintol=1.0e-8,grouptol=1.0e-6,maxits = 1
2,
                                LOUD=False):
    I = Dims[0]
    J = Dims[1]
    K = Dims[2]
    iteration = 1
    converged = False
    phig = np.zeros((I,J,K,G))
    while not(converged):
        phiold = phig.copy()
        for g in range(G):
            #compute Qhat and Sigmar
            Qhat = Q[:,:,:,g].copy()
            Sigmar = Sigmatg[:,:,:,g] - Sigmasgg[:,:,:,g,g] - chi
g[:,:,:,g]*nuSigmafg[:,:,:,g]
            for gprime in range(0,G):
                if (g != gprime):
                    Qhat += (chig[:,:,:,g]*nuSigmafg[:,:,:,gprime] + Sig
masgg[:,:,:,gprime,g])*phig[:,:,:,gprime]
            x,y,z,phi0 = diffusion_steady_fixed_source(Dims,Lengths,BCG
s[:,:,g],D[:,:,:,g],
                                                       Sigmar,Qhat, lint
ol)
            phig[:,:,:,g] = phi0.copy()
        change = np.linalg.norm(np.reshape(phig - phiol
d,I*J*K*G)/(I*J*K*G))
        if LOUD:
            print("Iteration",iteration,"Change =",change)
        iteration += 1
        converged = (change < grouptol) or iteration > maxits
    return x,y,z,phig
```

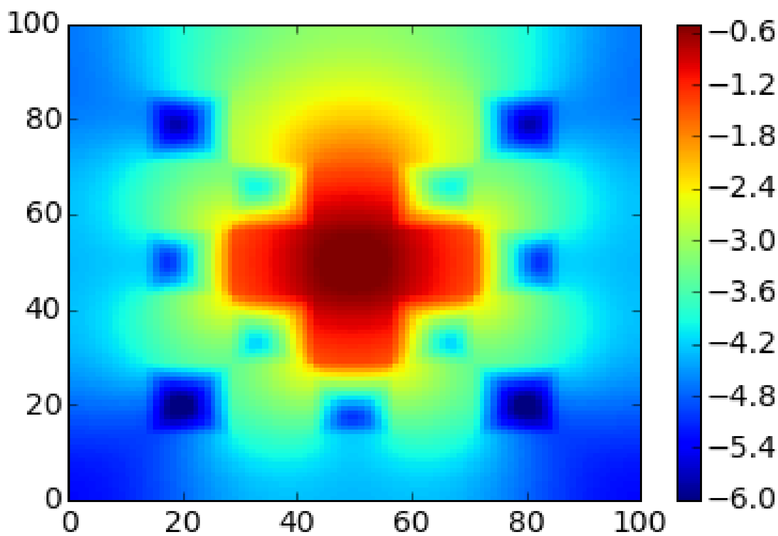
In [278]:
```
I = 5
J = 5
K = 1
Nx = 7
G = 2
BCGs = np.ones((6,3,G))
BCGs[:,0,:] = 0
BCGs[:,2,:] = 0

Sigmatg = np.ones((I,J,K,G))
Sigmasgg = np.zeros((I,J,K,G,G))
Sigmasgg[:,:,:,0,1] = 1
Sigmasgg[:,:,:,0,0] = 0.5
Sigmasgg[:,:,:,1,1] = 0.25
Sigmatg[:,:,:,0] = 5.5
Sigmatg[:,:,:,1] = 5.25
nuSigmafg = np.ones((I,J,K,G))*0.5
nug = np.ones((I,J,K,G))
chig = np.ones((I,J,K,G))
chig[:,:,:,1] = 0
D = np.ones((I,J,K,G))
Q = np.ones((I,J,K,G))
Q[:,:,:,1] = 0

x,y,z,phig = steady_multigroup_diffusion(G,(I,J,K),(Nx,Nx,1),BCGs,Sigmat
g,Sigmasgg,nuSigmafg,nug,chig,D,Q,maxits=2)
plt.plot(x,phig[:,0,0,0], label="group 1")
plt.plot(x,phig[:,0,0,1], label="group 2")
```

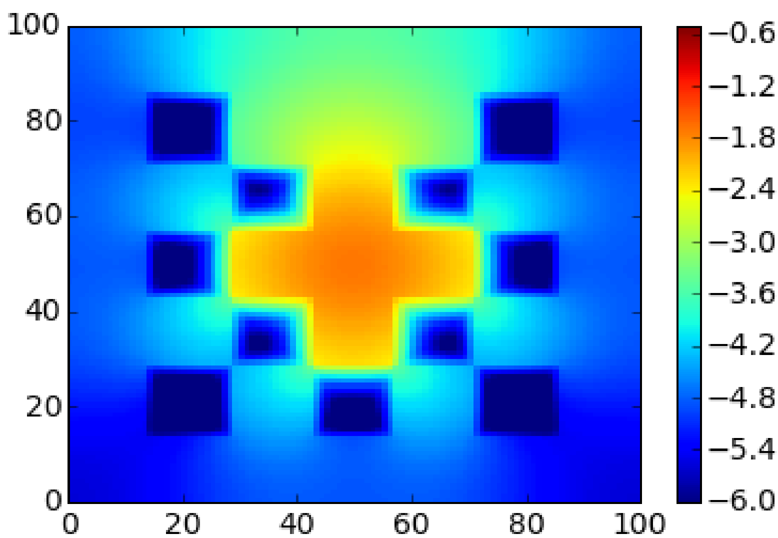
Out[278]: [<matplotlib.lines.Line2D at 0x11d7c3b38>]



In [279]:
```
I = 100
J = 100
K = 1
Nx = 7
BCs = np.ones((6,3))
BCs[:,0] = 0
BCs[:,2] = 0
BCs[(0,2,4),:] = [0.25,-1/2,0]
BCs[(1,3,5),:] = [0.25,1/2,0]
G = 2
Sigmaag,Sigmasgg,nuSigmafg,nug,chig,D,Q = lattice2G((Nx,Nx,1),(I,J,K))
x,y,z,phig = steady_multigroup_diffusion(2,(I,J,K),(Nx,Nx,1),BCGs,Sigmaa
g,Sigmasgg,nuSigmafg,nug,chig,D,Q)
```

In [280]:
```
plt.pcolor(np.transpose(np.log10(phig[:,:,0,0]))); plt.colorbar();
plt.clim(-6,-0.5)
plt.show()
plt.pcolor(np.transpose(np.log10(phig[:,:,0,1]))); plt.colorbar();
plt.clim(-6,-0.5)
plt.show()
```



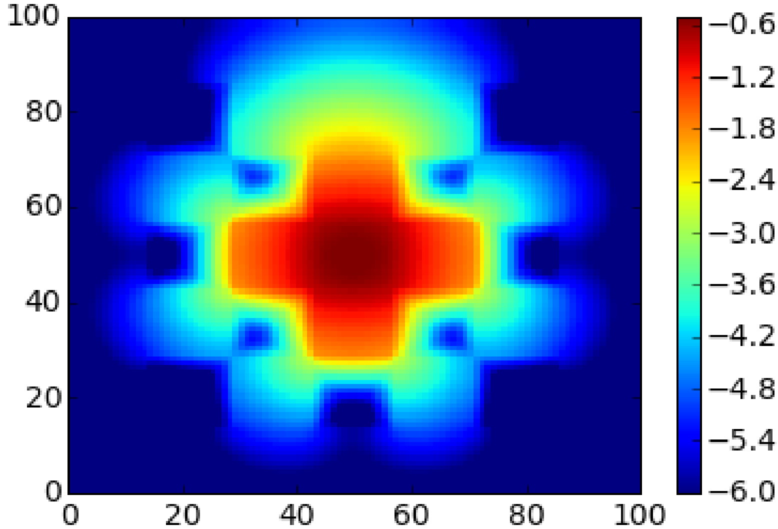


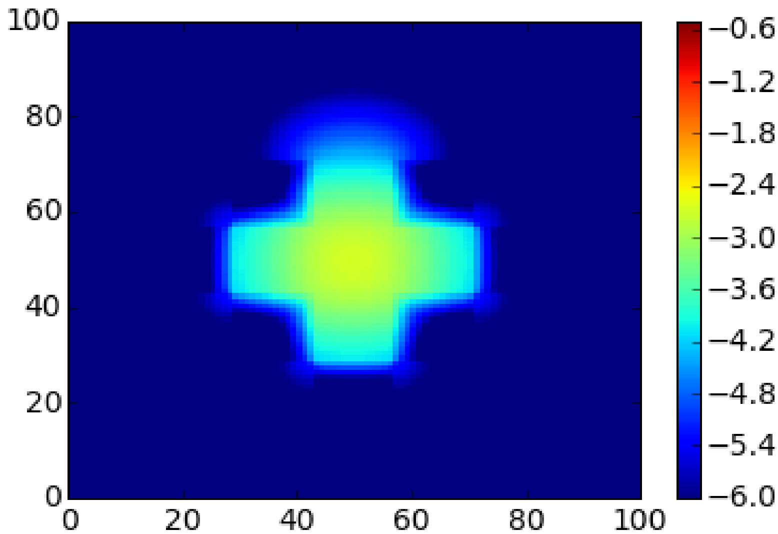
In [283]:
```
def time_dependent_mg_diffusion(phi0, v, dt, T, G,Dims,Lengths,BCGs,
                                Sigmatg,Sigmasgg,nuSigmafg,nug,chi
g,D,Q,lintol=1.0e-8,grouptol=1.0e-6,maxits = 12):
    """Solve a time dependent diffusion problem
    Inputs:
    phi0:       initial condition
    v:          particle speed
    dt:         time step size
    T:          final time
    The remaining inputs are the same as for diffusion_steady_fixed_sour
ce
    Outputs:
        x,y,z:          Vectors containing the cell centers in each dim
ension
        phi:            A vector containing the solution at time T
    """
    numsteps = int(T/dt)
    for step in range(numsteps):
        Qhat = Q + phi0/dt/v
        Sighat = Sigmatg + 1.0/dt/v
        print("=================================")
        print("Solving for time",(step+1)*dt)
        x,y,z,phi0 = steady_multigroup_diffusion(G,Dims,Lengths,BCGs,Sig
hat,Sigmasgg,nuSigmafg,nug,chig,D,Qhat)
    return x,y,z,phi0
```

In [284]:
```
I = 100
J = 100
K = 1
Nx = 7
BCs = np.ones((6,3))
BCs[:,0] = 0
BCs[:,2] = 0
BCs[(0,2,4),:] = [0.25,-1/2,0]
BCs[(1,3,5),:] = [0.25,1/2,0]
G = 2
Sigmatg,Sigmasgg,nuSigmafg,nug,chig,D,Q = lattice2G((Nx,Nx,1),(I,J,K))
v = Sigmatg.copy()*0 + 1
v[:,:,:,1] = 0.1
phi0 = v*0
x,y,z,phig = time_dependent_mg_diffusion(phi0, v, 0.1, 1.0, G,(I,J,K),(N
x,Nx,1),BCGs,
                                Sigmatg,Sigmasgg,nuSigmafg,nug,chig,D,Q)
```

```
================================
Solving for time 0.1
================================
Solving for time 0.2
================================
Solving for time 0.30000000000000004
================================
Solving for time 0.4
================================
Solving for time 0.5
================================
Solving for time 0.6000000000000001
================================
Solving for time 0.7000000000000001
================================
Solving for time 0.8
================================
Solving for time 0.9
================================
Solving for time 1.0
```

In [285]:
```
plt.pcolor(np.transpose(np.log10(phig[:,:,0,0]))); plt.colorbar();
plt.clim(-6,-0.5)
plt.show()
plt.pcolor(np.transpose(np.log10(phig[:,:,0,1]))); plt.colorbar();
plt.clim(-6,-0.5)
plt.show()
```





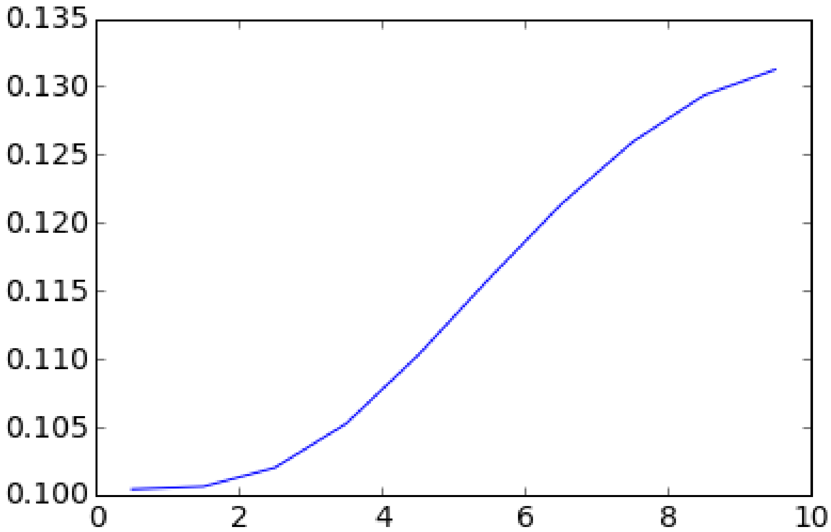
In [417]:

```
def inner_iteration(G,Dims,Lengths,BCGs,Sigmar,Sigmasgg,
                    D,Q,lintol=1.0e-8,grouptol=1.0e-6,maxits = 20,LOUD=F
alse):
    I = Dims[0]
    J = Dims[1]
    K = Dims[2]
    iteration = 1
    converged = False
    phig = np.zeros((I,J,K,G))
    while not(converged):
        phiold = phig.copy()
        for g in range(G):
            #compute Qhat
            Qhat = Q[:,:,:,g].copy()
            for gprime in range(0,G):
                if (g != gprime):
                    Qhat +=   Sigmasgg[:,:,:,gprime,g]*phig[:,:,:,gprim
e]
            x,y,z,phi0 = diffusion_steady_fixed_source(Dims,Lengths,BCG
s[:,:,g],D[:,:,:,g],
                                                                   Sigmar[:,:,:,g],Q
hat, lintol)
            phig[:,:,:,g] = phi0.copy()
        change = np.linalg.norm(np.reshape(phig - phiol
d,I*J*K*G)/(I*J*K*G))
        if LOUD:
            print("Iteration",iteration,"Change =",change)
        iteration += 1
        converged = (change < grouptol) or iteration > maxits
    return x,y,z,phig

def kproblem_mg_diffusion(G,Dims,Lengths,BCGs,Sigmarg,Sigmasgg,nuSigmaf
g,
                          chig,D,lintol=1.0e-8,grouptol=1.0e-6,tol=1.0e-
8,maxits = 12, k = 1, LOUD=False):
    I = Dims[0]
    J = Dims[1]
    K = Dims[2]
    phi0 = np.random.rand(I,J,K,G)
    phi0 = phi0 / np.linalg.norm(np.reshape(phi0,I*J*K*G))
    phiold = phi0.copy()
    converged = False
    iteration = 1
    while not(converged):
        Qhat = chig*0
        for g in range(G):
            for gprime in range(G):
                Qhat[:,:,:,g] += chig[:,:,:,g]*nuSigmafg[:,:,:,gprime]*p
hi0[:,:,:,gprime]
        x,y,z,phi0 = inner_iteration(G,Dims,Lengths,BCGs,Sigmarg,Sigmasg
g,D,Qhat)
        knew = np.linalg.norm(np.reshape(phi0,I*J*K*G))/np.linalg.norm(n
p.reshape(phiold,I*J*K*G))
        solnorm = np.linalg.norm(np.reshape(phiold,I*J*K*G))
        converged = (((np.abs(knew-k) < tol) and
```

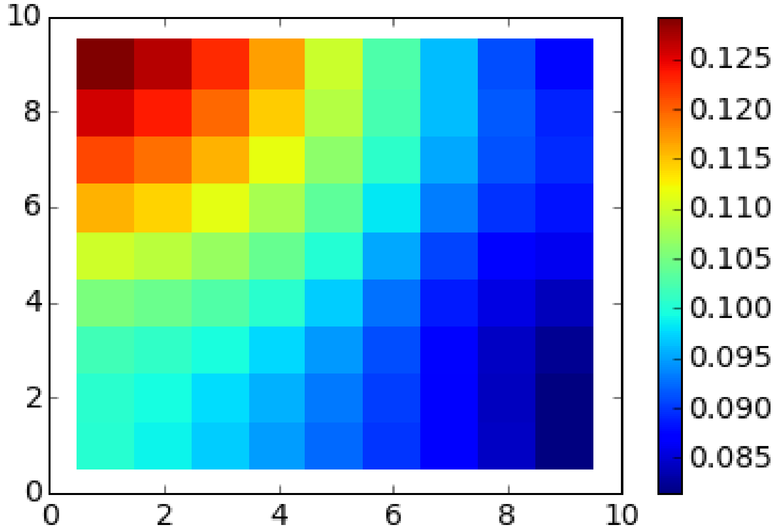
```
                    (np.linalg.norm(np.reshape(phi0,I*J*K*G))-solnor
m)/solnorm < tol)
                   or (iteration > maxits))
        k = knew
        phi0 /= k
        phiold = phi0.copy()
        if (LOUD):
            print("=================================")
            print("Iteration",iteration,": k =",k)
        iteration += 1
    return x,y,z,k,iteration-1,phi0
```

In [463]:
```
I = 10
J = 10
K = 1
Nx = 10
G = 1
BCGs = np.ones((6,3,G))
BCGs[:,0,:] = 0
BCGs[:,2,:] = 0

Sigmarg = np.ones((I,J,K,G))*1.5
Sigmasgg = np.zeros((I,J,K,G,G))
Sigmasgg[:,:,:,0,0] = 0.5
nuSigmafg = np.ones((I,J,K,G))*1.5
nug = np.ones((I,J,K,G))
chig = np.ones((I,J,K,G))
D = np.ones((I,J,K,G))

x,y,z,k,iterations,phig = kproblem_mg_diffusion(G,(I,J,K),(Nx,Nx,Nx),BCG
s,Sigmarg,Sigmasgg,nuSigmafg,
                          chig,D,lintol=1.0e-13,grouptol=1.0e-13,tol=1.0
e-12,maxits = 5, k = 1, LOUD=False)
print("k =",k,"Number of iterations =",iterations)
plt.plot(x,phig[:,0,0,0], label="group 1")
plt.show()
plt.pcolor(x,y,phig[:,:,0,0])
plt.colorbar()
```

```
k = 0.998707618308 Number of iterations = 6
```



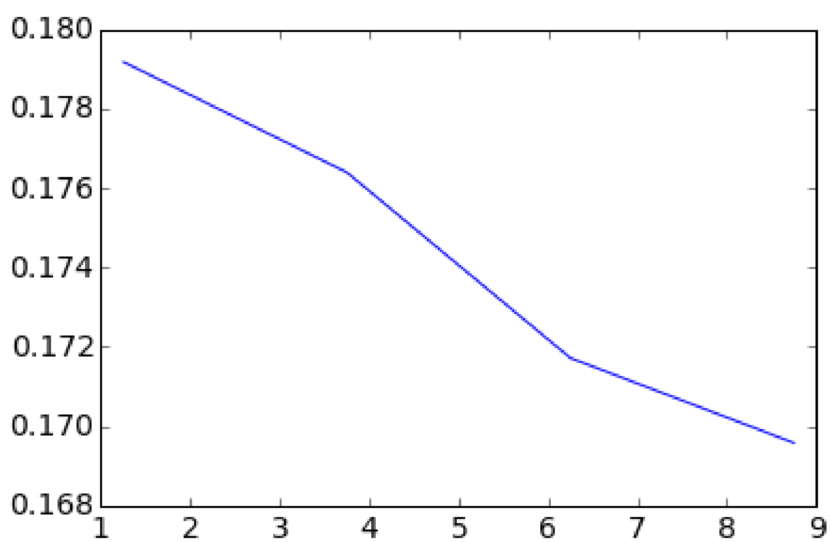

```
Out[463]: <matplotlib.colorbar.Colorbar at 0x1269b90b8>
```



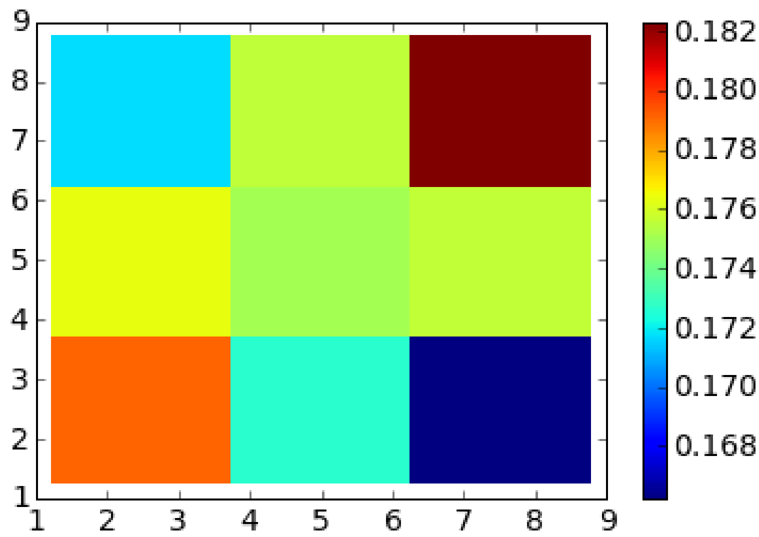
In [464]:

```
I = 4
J = 4
K = 1
Nx = 10
G = 2
BCGs = np.ones((6,3,G))
BCGs[:,0,:] = 0
BCGs[:,2,:] = 0

Sigmarg = np.ones((I,J,K,G))*3
Sigmasgg = np.ones((I,J,K,G,G))
nuSigmafg = np.ones((I,J,K,G))*1.5
chig = np.ones((I,J,K,G))*0.5
D = np.ones((I,J,K,G))

x,y,z,k,iterations,phig = kproblem_mg_diffusion(G,(I,J,K),(Nx,Nx,Nx),BCG
s,Sigmarg,Sigmasgg,nuSigmafg,
                          chig,D,lintol=1.0e-8,grouptol=1.0e-6,tol=1.0e-
8,maxits = 15, k = 1, LOUD=True)
print("k =",k,"Number of iterations =",iterations)
plt.plot(x,phig[:,0,0,0], label="group 1")
plt.show()
plt.pcolor(x,y,phig[:,:,0,0])
plt.colorbar()
```

```
================================
Iteration 1 : k = 0.706835671548
================================
Iteration 2 : k = 0.744227091408
================================
Iteration 3 : k = 0.746384629247
================================
Iteration 4 : k = 0.747610618904
================================
Iteration 5 : k = 0.748346968991
================================
Iteration 6 : k = 0.748813867371
================================
Iteration 7 : k = 0.74912434137
================================
Iteration 8 : k = 0.749339005741
================================
Iteration 9 : k = 0.749492074982
================================
Iteration 10 : k = 0.749603901814
================================
Iteration 11 : k = 0.749687196265
================================
Iteration 12 : k = 0.749750233982
================================
Iteration 13 : k = 0.749798590122
================================
Iteration 14 : k = 0.749836123917
================================
Iteration 15 : k = 0.749865565186
================================
Iteration 16 : k = 0.749888879078
k = 0.749888879078 Number of iterations = 16
```



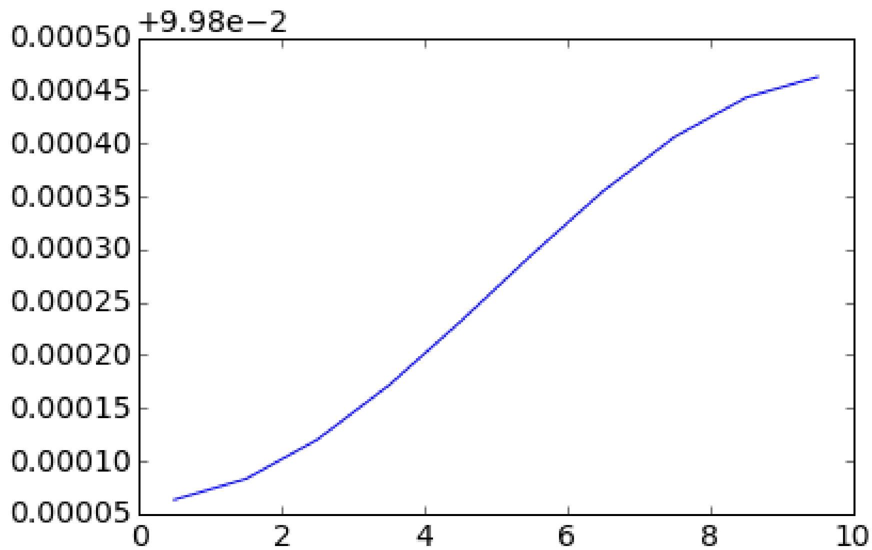

```
Out[464]: <matplotlib.colorbar.Colorbar at 0x1267c7f98>
```



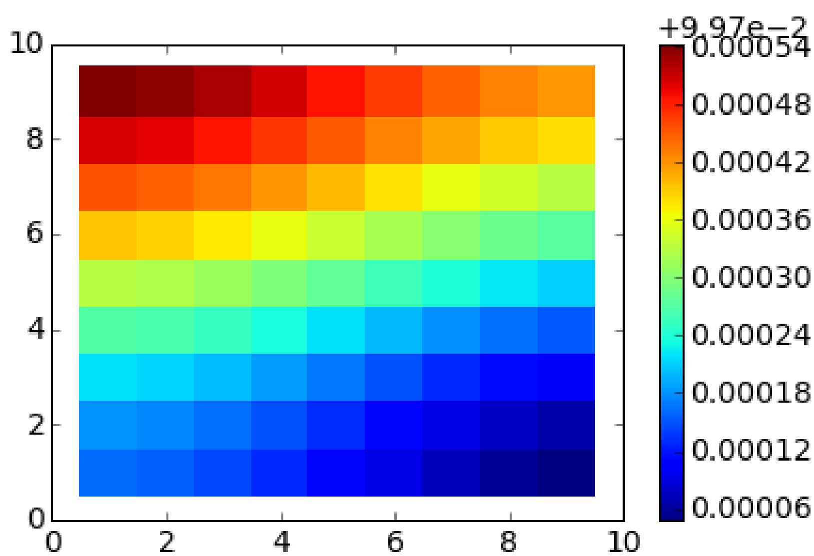
In [459]:
```
def alpha_mg_diffusion(G,Dims,Lengths,BCGs,Sigmarg,Sigmasgg,nuSigmafg,
                          chig,D,v,lintol=1.0e-8,grouptol=1.0e-6,tol=1.0
e-9,maxits = 20, alpha = 1000, LOUD=False):
    iteration = 0
    converged = False
    while not(converged):
        Sigmarg_alpha = Sigmarg + alpha/v
        x,y,z,k,iterations,phig = kproblem_mg_diffusion(G,(I,J,K),(Nx,N
x,Nx),BCGs,Sigmarg_alpha,Sigmasgg,nuSigmafg,
                                                        chig,D,linto
l=1.0e-8,grouptol=1.0e-6,tol=1.0e-12,
                                                         maxits = 100, k
= 1, LOUD=False)
        if (iteration > 0):
            #update via secant method
            slope = (k-kold)/(alpha-alpha_old)
        else:
            Sigmarg_alpha = Sigmarg + (alpha+tol)/v
            x,y,z,kperturb,iterations,phig = kproblem_mg_diffusion(G,
(I,J,K),(Nx,Nx,Nx),BCGs,Sigmarg_alpha,Sigmasgg,nuSigmafg,
                                                        chig,D,linto
l=1.0e-8,grouptol=1.0e-6,tol=1.0e-12,
                                                         maxits = 100, k
= 1, LOUD=False)
            slope = (kperturb - k)/tol
        alpha_old = alpha
        kold = k
        alpha = (1-k)/slope + alpha
        converged = (np.abs(k - 1) < tol) or (iteration >= maxits)
        iteration += 1
        if (LOUD):
            print("==================================")
            print("Iteration",iteration,": k =",k, "alpha =",alpha)
    return x,y,z,alpha,iteration,phig
```

In [460]:
```
#This one should have alpha be 2.5
I = 10
J = 10
K = 1
Nx = 10
G = 1
BCGs = np.ones((6,3,G))
BCGs[:,0,:] = 0
BCGs[:,2,:] = 0

Sigmarg = np.ones((I,J,K,G))*1.25
Sigmasgg = np.zeros((I,J,K,G,G))
Sigmasgg[:,:,:,0,0] = 0.5
nuSigmafg = np.ones((I,J,K,G))*2.5
nug = np.ones((I,J,K,G))
chig = np.ones((I,J,K,G))
D = np.ones((I,J,K,G))
v = D.copy()*2.0

x,y,z,alpha,iteration,phig = alpha_mg_diffusion(G,(I,J,K),(Nx,Nx,Nx),BCG
s,Sigmarg,Sigmasgg,nuSigmafg,
                                                chig,D,v,
                                                lintol=1.0e-10,grouptol=1.0
e-10,tol=1.0e-6,maxits = 20, alpha = 1.5, LOUD=True)
print("alpha =",alpha,"Number of iterations =",iteration)
plt.plot(x,phig[:,0,0,0], label="group 1")
plt.show()
plt.pcolor(x,y,phig[:,:,0,0])
plt.colorbar()
```

```
==================================
Iteration 1 : k = 1.24999998747 alpha = 2.32373277469
==================================
Iteration 2 : k = 1.03654163457 alpha = 2.46474642737
==================================
Iteration 3 : k = 1.00710076441 alpha = 2.49875713156
==================================
Iteration 4 : k = 1.00024846374 alpha = 2.49999035633
==================================
Iteration 5 : k = 1.00000145306 alpha = 2.49999761088
==================================
Iteration 6 : k = 1.00000039029 alpha = 2.50000027503
alpha = 2.50000027503 Number of iterations = 6
```



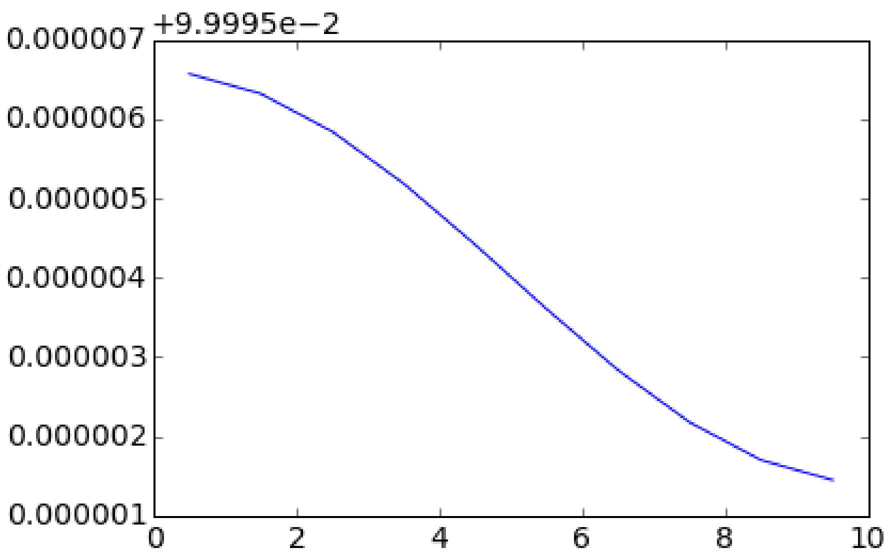

```
Out[460]: <matplotlib.colorbar.Colorbar at 0x12a140b38>
```



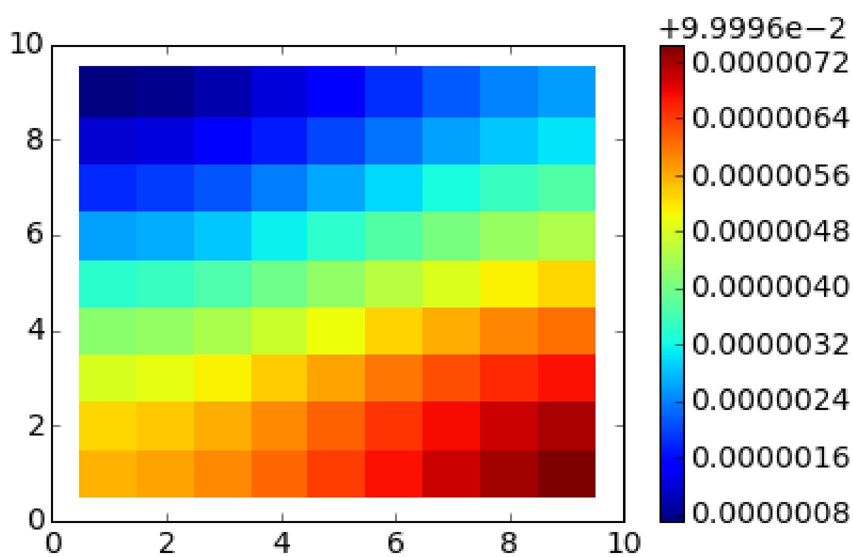
In [461]:

```
#This one should have alpha be -1
I = 10
J = 10
K = 1
Nx = 10
G = 1
BCGs = np.ones((6,3,G))
BCGs[:,0,:] = 0
BCGs[:,2,:] = 0

Sigmarg = np.ones((I,J,K,G))*(1.25+1.0/3.0)
Sigmasgg = np.zeros((I,J,K,G,G))
Sigmasgg[:,:,:,0,0] = 0.5
nuSigmafg = np.ones((I,J,K,G))*1.25
nug = np.ones((I,J,K,G))
chig = np.ones((I,J,K,G))
D = np.ones((I,J,K,G))
v = D.copy()*3.0

x,y,z,alpha,iteration,phig = alpha_mg_diffusion(G,(I,J,K),(Nx,Nx,Nx),BCG
s,Sigmarg,Sigmasgg,nuSigmafg,
                                            chig,D,v,
                                            lintol=1.0e-10,grouptol=1.0
e-10,tol=1.0e-6,maxits = 20, alpha = 1.5, LOUD=True)
print("alpha =",alpha,"Number of iterations =",iteration)
plt.plot(x,phig[:,0,0,0], label="group 1")
plt.show()
plt.pcolor(x,y,phig[:,:,0,0])
plt.colorbar()
```

```
=================================
Iteration 1 : k = 0.599999998332 alpha = -2.72494425083
=================================
Iteration 2 : k = 1.85180087093 alpha = 0.149962830053
=================================
Iteration 3 : k = 0.765311927329 alpha = -0.471034195176
=================================
Iteration 4 : k = 0.876379987687 alpha = -1.16221093543
=================================
Iteration 5 : k = 1.04521194857 alpha = -0.977118924015
=================================
Iteration 6 : k = 0.993935383382 alpha = -0.999010251646
=================================
Iteration 7 : k = 0.999736136666 alpha = -1.00000603933
=================================
Iteration 8 : k = 1.00000160998 alpha = -1.00000000033
=================================
Iteration 9 : k = 1.00000000006 alpha = -1.00000000012
alpha = -1.00000000012 Number of iterations = 9
```



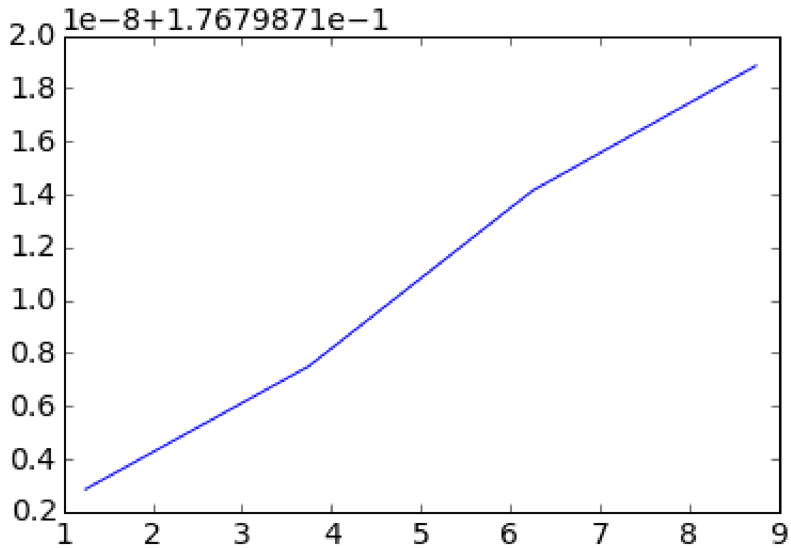

```
Out[461]: <matplotlib.colorbar.Colorbar at 0x1292b35f8>
```



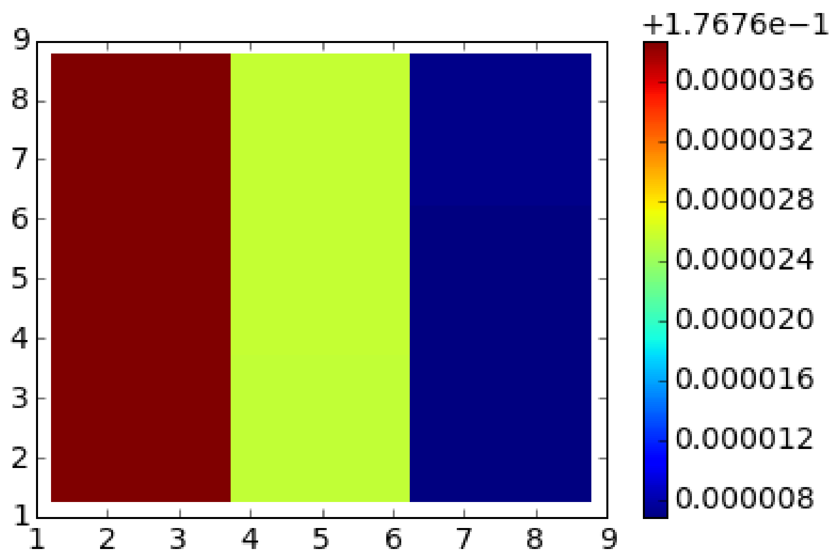
In [467]:

```
#This one should have alpha be 1.25
I = 4
J = 4
K = 1
Nx = 10
G = 2
BCGs = np.ones((6,3,G))
BCGs[:,0,:] = 0
BCGs[:,2,:] = 0

Sigmarg = np.ones((I,J,K,G))*1.25
Sigmasgg = np.ones((I,J,K,G,G))
nuSigmafg = np.ones((I,J,K,G))*1.5
chig = np.ones((I,J,K,G))*0.5
D = np.ones((I,J,K,G))
v = D.copy()*1.0

x,y,z,alpha,iteration,phig = alpha_mg_diffusion(G,(I,J,K),(Nx,Nx,Nx),BCG
s,Sigmarg,Sigmasgg,nuSigmafg,
                                            chig,D,v,
                                            lintol=1.0e-10,grouptol=1.0
e-10,tol=1.0e-6,maxits = 20, alpha = 1.251, LOUD=True)
print("alpha =",alpha,"Number of iterations =",iteration)
plt.plot(x,phig[:,0,0,0], label="group 1")
plt.show()
plt.pcolor(x,y,phig[:,:,0,0])
plt.colorbar()
```

```
================================
Iteration 1 : k = 0.999329103147 alpha = 1.24999586478
================================
Iteration 2 : k = 0.999998056155 alpha = 1.24999294696
================================
Iteration 3 : k = 1.00000000362 alpha = 1.24999295239
alpha = 1.24999295239 Number of iterations = 3
```




```
Out[467]: <matplotlib.colorbar.Colorbar at 0x12b5112e8>
```




```
In [ ]:
```