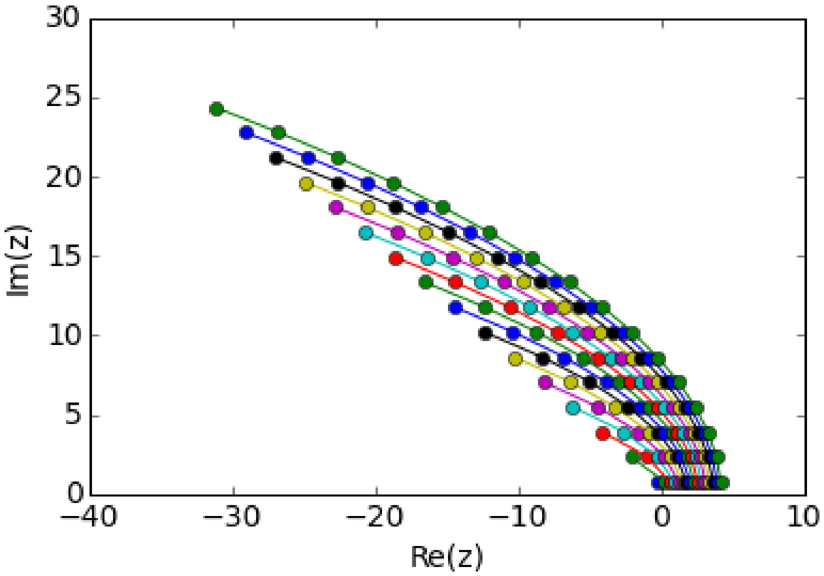
Approximation to Exponential Using Parabolic Contour

In [93]:

```
import numpy as np
import scipy as sp
import scipy.sparse as sparse
import scipy.sparse.linalg as splinalg
import matplotlib.pyplot as plt
%matplotlib inline
#compute exp
Npoints = (2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32)
errors = np.zeros(len(Npoints))
pos = 0
for N in Npoints:
    theta = np.pi*np.arange(1,N,2)/N
    z = N*(.1309 - 0.1194*theta**2 + .2500j*theta)
    w = N*(- 2*0.1194*theta + .2500j)
    c = 1.0j/N*np.exp(z)*w
    plt.plot(np.real(z),np.imag(z),'o-')
    u = 0
    for k in range(int(N/2)):
        u -=  c[k]/((z[k] + 1))
    errors[pos] = np.abs(2*np.real(u)-0.36787944117144232159552338)
    pos += 1
plt.xlabel("Re(z)")
plt.ylabel("Im(z)")
plt.axis([-40,10,0,30])
print("Errors =",errors)
```

```
Errors = [   1.99561386e-02   1.20370371e-02   1.20968913e-03   2.35466
489e-05
   1.67282948e-05   1.77472479e-06   1.04223948e-08   2.68294421e-08
   5.22062821e-09   6.15275608e-10   4.65067984e-11   5.75650638e-13
   5.37236922e-13   1.12965193e-13   1.43218770e-14   4.99600361e-16]
```

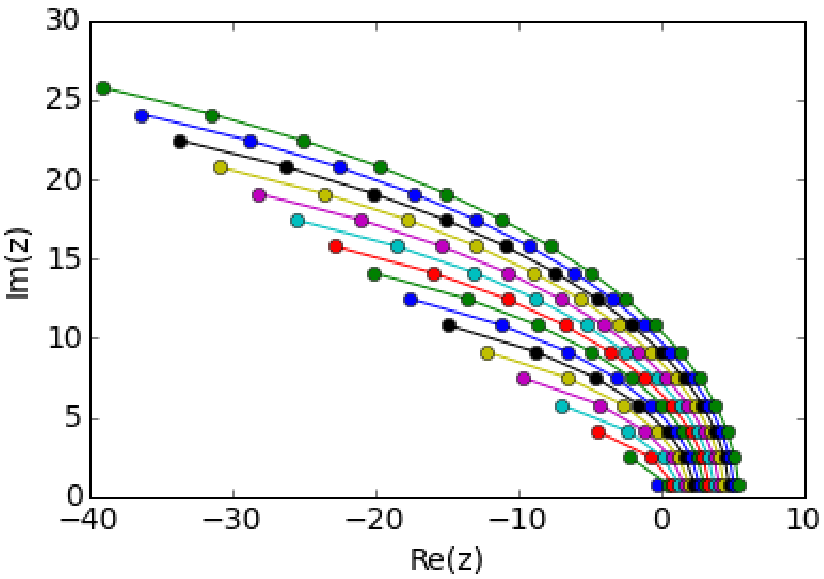


Approximation Using Cotangent Contour

In [94]:

```
#compute exp
errors_cot = np.zeros(len(Npoints))
pos = 0
for N in Npoints:
    theta = np.pi*np.arange(1,N,2)/N
    z = N*(-.6122 + 0.5017*theta/np.tan(.6407*theta)+ .2645j*theta)
    w = N*((0. + 0.2645j) + 0.5017/np.tan(0.6407*theta) - 0.3214391900
0000004*theta/(np.sin(0.6407*theta)**2))
    c = 1.0j/N*np.exp(z)*w
    plt.plot(np.real(z),np.imag(z),'o-')
    u = 0
    for k in range(int(N/2)):
        u -=  c[k]/((z[k] + 1))
    errors_cot[pos] = np.abs(2*np.real(u)-0.36787944117144232159552 38)
    pos += 1
plt.xlabel("Re(z)")
plt.ylabel("Im(z)")
plt.axis([-40,10,0,30])
print("Errors =",errors_cot)
```

```
Errors = [   1.68880829e-03   5.53557671e-03   2.71697153e-04   2.80811
239e-05
   3.45788438e-08   1.16177229e-07   8.09777750e-09   1.41766099e-10
   2.53143617e-11   2.94064773e-12   1.66200387e-13   3.88578059e-15
   7.54951657e-15   2.10942375e-15   4.69069228e-14   4.53526106e-14]
```

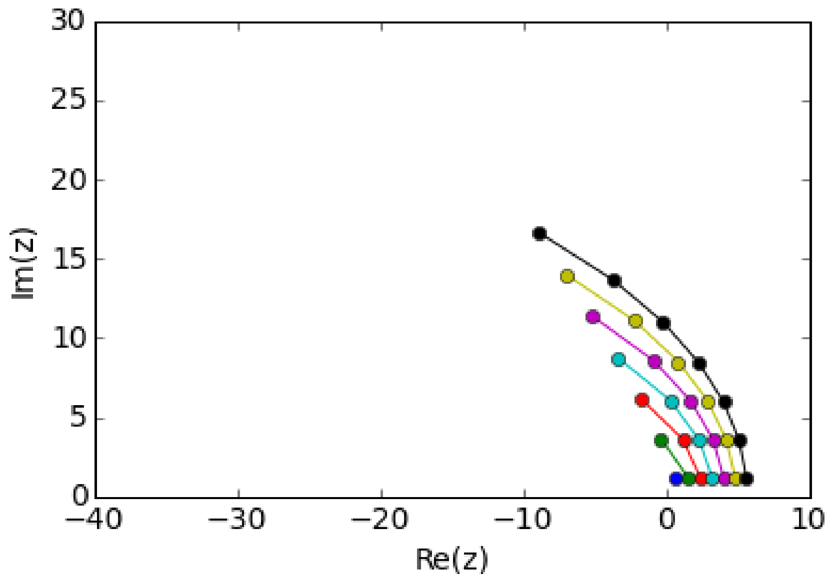


Approximation Using Best Rational Approximation

In [95]:
```python
Npoint_rat = (2,4,6,8,10,12,14)
errors_rat = np.zeros(len(Npoint_rat))
pos = 0
#######
N = 2
u = 0
z = [ 0.585051560655138 + 1.185847251723686j,]
c = [ 0.169152633611546 - 0.809801111544530j,]
for k in range(int(N/2)):
    u -=  c[k]/((z[k] + 1))
errors_rat[pos] = np.abs(2*np.real(u)-0.36787944117144232159552 38)
pos += 1
plt.plot(np.real(z),np.imag(z),'o-')
#######
N = 4
u = 0
z = [ 1.548400570539483 + 1.191825853927648j, -0.367838314399776 + 3.6
58133272063430j]
c = [  -0.061683522554915 - 1.905059455980079j,  0.073392419234124 +
0.450004915853843j]
for k in range(int(N/2)):
    u -=  c[k]/((z[k] + 1))
errors_rat[pos] = np.abs(2*np.real(u)-0.36787944117144232159552 38)
pos += 1
plt.plot(np.real(z),np.imag(z),'o-')
#######
N = 6
u = 0
z = [  2.400602938933259 + 1.193129308402001j,  1.158552571719549 +
3.614772600819200j, -1.781988275920806 + 6.196512467345827j]
c = [   -0.579013004031147 - 4.286888564581573j,  0.663006870528764 +
1.451412919902496j,   -0.083581617156413 - 0.106429260747820j]
for k in range(int(N/2)):
    u -=  c[k]/((z[k] + 1))
errors_rat[pos] = np.abs(2*np.real(u)-0.36787944117144232159552 38)
pos += 1
plt.plot(np.real(z),np.imag(z),'o-')
#######
N = 8
u = 0
z = [ 3.220945245027020 + 1.193619605401522j,
      2.292249147807734 + 3.600771496022284j,
      0.269490987390844 + 6.082032592598592j,
     -3.408539501369338 + 8.773034564213896j]
c = [ -1.831771710723487 - 9.525608129508791j,
      2.436240733056933 + 3.716755640617416j,
     -0.632588053778158 - 0.443923102637992j,
      0.028129757165822 + 0.011577384564594j]
for k in range(int(N/2)):
    u -=  c[k]/((z[k] + 1))
errors_rat[pos] = np.abs(2*np.real(u)-0.36787944117144232159552 38)
pos += 1
plt.plot(np.real(z),np.imag(z),'o-')
#######
N = 10
u = 0
z = [  4.027732482090957 + 1.193856067342547j,
     3.283752898176212 + 3.594386774986297j,
     1.715406031287229 + 6.038934929977215j,
```

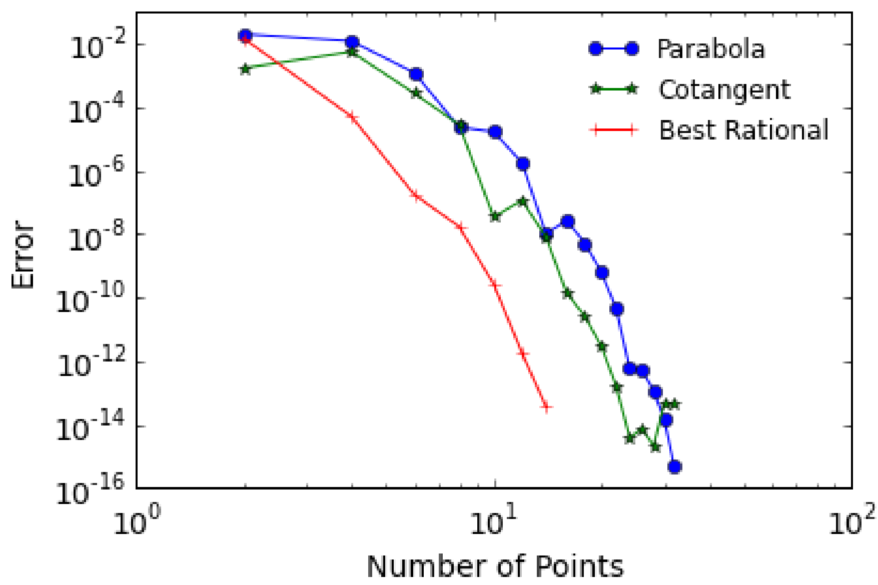
```
     -0.894404685089565 + 8.582756905805123j,
     -5.161191252117303 +11.375156263053537j,]
c = [-4.818382049903466 -21.054597563476996j,
     7.117165194831597 + 8.819533309710385j,
     -2.565584990864071 - 1.216385735509945j,
     0.272586984764519 + 0.014211728988400j,
     -0.005784903977276 + 0.000685850663472j]
for k in range(int(N/2)):
    u -=  c[k]/((z[k] + 1))
errors_rat[pos] = np.abs(2*np.real(u)-0.36787944117144232159552338)
pos += 1
plt.plot(np.real(z),np.imag(z),'o-')
#######
N = 12
u = 0
z = [ 4.827493721857821 + 1.193987934748583j,
      4.206124482242000 + 3.590920581795537j,
      2.917868831300586 + 6.017345604862655j,
      0.851707374831106 + 8.503832331032017j,
     -2.235968024314319 +11.109295524177000j,
     -6.998687843706477 +13.995915681397976j]
c = [-11.799385013095378 -46.411645036888196j,
     18.785985030295741 +20.237286725704049j,
     -8.238258735936171 - 2.796189543435955j,
      1.319411773710640 - 0.183524222891953j,
     -0.068571483418057 + 0.038419135920867j,
      0.000818433042302 - 0.000581354363764j,]
for k in range(int(N/2)):
    u -=  c[k]/((z[k] + 1))
errors_rat[pos] = np.abs(2*np.real(u)-0.36787944117144232159552338)
pos += 1
plt.plot(np.real(z),np.imag(z),'o-')
#######
N = 14
u = 0
z = [5.623151880088747 + 1.194068309420004j,
      5.089353593691644 + 3.588821962583661j,
      3.993376923428209 + 6.004828584136945j,
      2.269789514323265 + 8.461734043748510j,
     -0.208754946413353 +10.991254996068200j,
     -3.703276086329081 +13.656363257468552j,
     -8.897786521056833 +16.630973240336562j]
c = [ -0.278754565727894 - 1.021482174078080j,
      0.469337296545605 + 0.456439548888464j,
     -0.234984158551045 - 0.058083433458861j,
      0.048071353323537 - 0.013210030313639j,
     -0.003763599179114 + 0.003351864962866j,
      0.000094388997996 - 0.000171848578807j,
     -0.000000715408253 + 0.000001436094999j,]
for k in range(int(N/2)):
    u -=  100*c[k]/((z[k] + 1))
errors_rat[pos] = np.abs(2*np.real(u)-0.36787944117144232159552338)
pos += 1
plt.plot(np.real(z),np.imag(z),'o-')

plt.xlabel("Re(z)")
plt.ylabel("Im(z)")
plt.axis([-40,10,0,30])
print("Errors =",errors_rat)
```

```
Errors = [   1.45997682e-02   5.09689032e-05   1.74062861e-07   1.74426
180e-08
   2.67894040e-10   1.80194748e-12   3.69149156e-14]
```



In [96]:
```
plt.loglog(Npoints,errors,'o-',label="Parabola")
plt.loglog(Npoints,errors_cot,'*-',label="Cotangent")
plt.loglog(Npoint_rat,errors_rat,'+-',label="Best Rational")
plt.legend()
plt.xlabel("Number of Points")
plt.ylabel("Error")
```

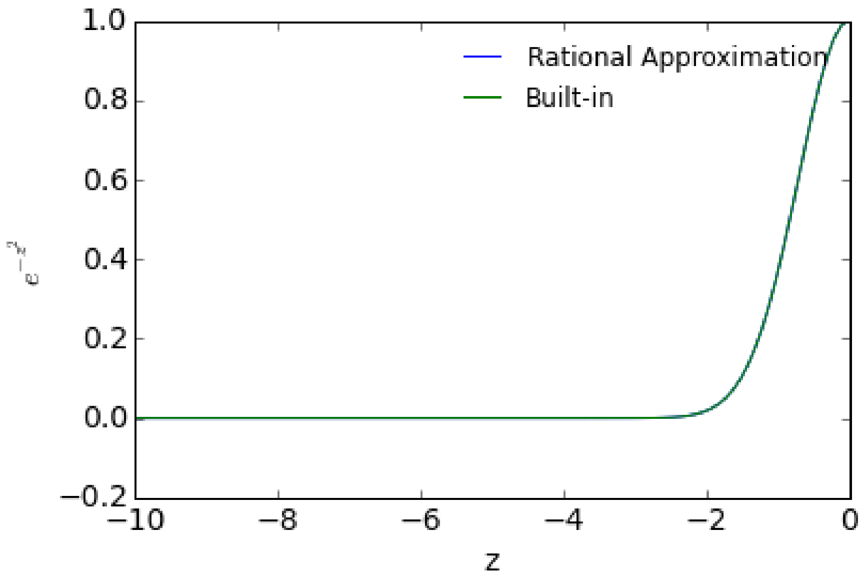
Out[96]: <matplotlib.text.Text at 0x1156c8e10>

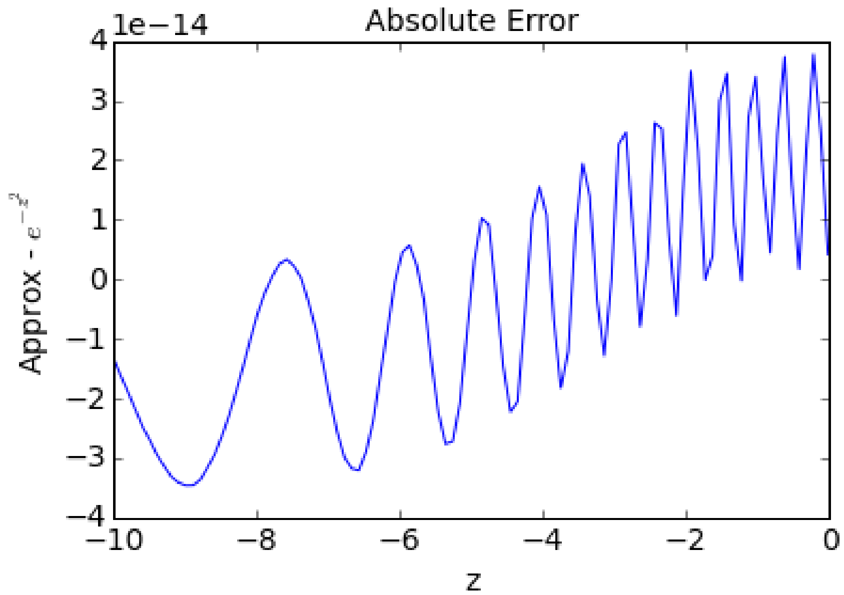


For Fun: Exp[-z^2]

In [107]:

```
N = 14
z = [5.623151880088747 + 1.194068309420004j,
      5.089353593691644 + 3.588821962583661j,
      3.993376923428209 + 6.004828584136945j,
      2.269789514323265 + 8.461734043748510j,
     -0.208754946413353 +10.991254996068200j,
     -3.703276086329081 +13.656363257468552j,
     -8.897786521056833 +16.630973240336562j]
c = [ -0.278754565727894 - 1.021482174078080j,
      0.469337296545605 + 0.456439548888464j,
     -0.234984158551045 - 0.058083433458861j,
      0.048071353323537 - 0.013210030313639j,
     -0.003763599179114 + 0.003351864962866j,
      0.000094388997996 - 0.000171848578807j,
     -0.000000715408253 + 0.000001436094999j,]
argument = np.linspace(-10,-0.01,100)
y = -argument**2
u = np.zeros(100)
for k in range(int(N/2)):
    u -=  100*c[k]/((z[k] - y))
plt.plot(argument,2*np.real(u),label="Rational Approximation")
plt.plot(argument,np.exp(-argument**2),label="Built-in")
plt.legend()
plt.xlabel('z')
plt.ylabel('$e^{-z^2}$')
plt.show()
plt.plot(argument, 2*np.real(u)-np.exp(-argument**2))
plt.title("Absolute Error")
plt.xlabel('z')
plt.ylabel('Approx - $e^{-z^2}$')
plt.show()
```

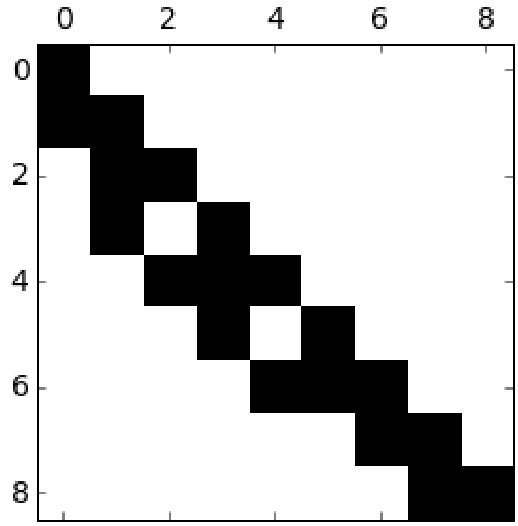




Depletion Example

In [84]:

```python
#cross-sections in barns
position = {'28':0,'29':1,'20':2,'39':3,'30':4,'49':5,'40':6,'41':7,'5
1':8}
previous_cap = {'28':-1,'29':0,'20':1,'39':-1,'30':3,'49':-1,'40':5,'4
1':6,'51':-1}
previous_beta = {'28':-1,'29':-1,'20':-1,'39':1,'30':2,'49':3,'4
0':4,'41':-1,'51':7}
sig_gamma = 1.0E-24*np.array([2.7,22.0,0,60,274,290,326,532])
sig_a = 1.0E-24*np.array([12.0,22,0,60,0,274+698,290+53,326+938,535])
lam = np.array([0,42.4737,1.17982,0.294956,138.629,0,0,0.000131877,0])

A = np.zeros((9,9))

phi = 1.0e14 * 60 * 60 * 24 #10^14 1/cm^2/s in 1/cm^2/day
for i in position:
    row = position[i]
    A[row,row] = -lam[row] - phi*sig_a[row]
    if previous_cap[i]>=0:
            A[row,previous_cap[i]] = phi*sig_gamma[previous_cap[i]]
    if previous_beta[i]>=0:
            A[row,previous_beta[i]] = lam[previous_beta[i]]
plt.spy(A)
b = np.zeros(9)
b[0] = 1.0
```



Example Matrix Exponential

In [89]:
```
Npoints = (12,) #(2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32)

for N in Npoints:
    pos = 0
    theta = np.pi*np.arange(1,N,2)/N
    z = N*(.1309 - 0.1194*theta**2 + .2500j*theta)
    w = N*(- 2*0.1194*theta + .2500j)
    c = 1.0j/N*np.exp(z)*w
    #plt.plot(np.real(z),np.imag(z),'o-')
    #plt.show()
    u = np.zeros(9)
    for k in range(int(N/2)):
        n,code = splinalg.gmres(z[k]*sparse.identity(9)  - A*365,b, to
l=1e-12, maxiter=2000)
        if (code):
            print(code)
        u = u- c[k]*n
    u = 2*np.real(u)
    print(u)
```

```
[  9.62866113e-01   5.28837728e-07   8.52081775e-11   7.60457800e-05
   2.85096857e-10   2.56945310e-03   1.15458593e-03   2.16755300e-04
   2.57285015e-06]
```

Your Task

1. Compute the solution using numpy's expm method
2. Compute the solution using Backward Euler
3. Compute the solution using one of the nice contours
4. Compute the solution using the best rational approximation (CRAM)

Plot the solutions from 0 to 540 days

For Fun: Lattice Problem exact in time

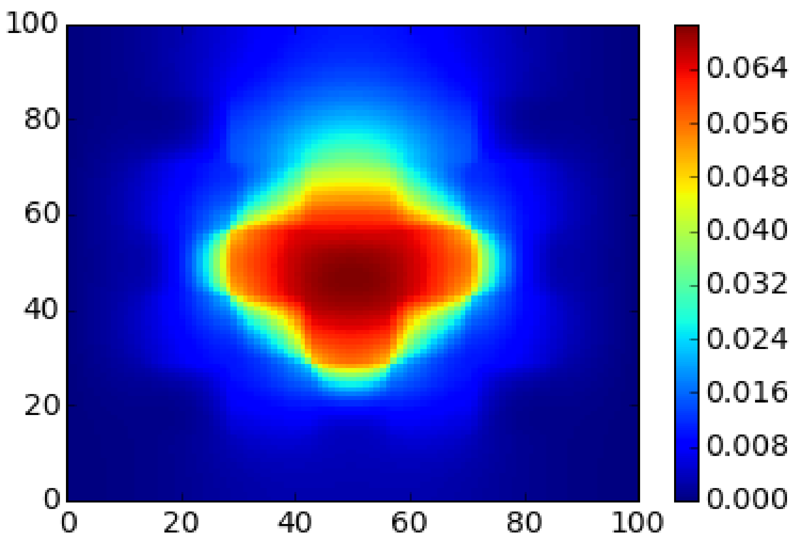
In [75]:
```
import numpy as np
import scipy as sp
import scipy.sparse as sparse
import scipy.sparse.linalg as splinalg
def coordLookup_l(i, j, k, I, J):
    """get the position in a 1-D vector
    for the (i,j,k) index
    """
    return i + j*I + k*J*I

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
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 fix
ed 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
these
        D,Sigma,Q:       Each is an array of size (I,J,K) containing t
he quantity
    Outputs:
        x,y,z:           Vectors containing the cell centers in each d
imension
        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
    return A,b

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 = 100
J = 100
K = 1
Nx = 7

BCs = np.ones((6,3))
BCs[:,0] = 0
BCs[:,2] = 0
BCs[0,:] = [1,0,0]
BCs[1,:] = [1,0,0]
D,Q,Sigma = lattice((Nx,Nx,1),(I,J,K))
A,b = diffusion_steady_fixed_source((I,J,K),(1.0,1.0,1.0),BCs,D,Sigm
a,Q)
```

In [80]:
```
Npoints = (12,) #(2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32)

for N in Npoints:
    pos = 0
    theta = np.pi*np.arange(1,N,2)/N
    z = N*(.1309 - 0.1194*theta**2 + .2500j*theta)
    w = N*(- 2*0.1194*theta + .2500j)
    c = 1.0j/N*np.exp(z)*w
    #plt.plot(np.real(z),np.imag(z),'o-')
    #plt.show()
    u = np.zeros(I*J*K)
    for k in range(int(N/2)):
        phi,code = splinalg.gmres(z[k]*sparse.identity(I*J*K)  +
A*.1,b, tol=1e-12, maxiter=2000)
        if (code):
            print(code)
        u = u- c[k]*phi
    u = 2*np.real(u)
    plt.pcolor(u.reshape((I,J)));
    plt.colorbar()
```



In [ ]: