Assignment_2_notebook_Noah_Steinberg

September 21, 2017

```
In [1]: import numpy as np
        import pylab as py
        import matplotlib.pyplot as plt
        import math
        import time
```

NUMBERS 1 & 3

We begin by solving Poisson's equation $\Delta\Phi(x,y) = 4\pi\rho(x,y)$

with the charge density given by $\rho(x,y) = e^{-16((x-.5)^2+(y-.5)^2)}$

We will solve this on a square lattice in the region $0 \leq x \leq 1$, $0 \leq y \leq 1$ with three different methods. We will begin with basic relaxation, then Gauss Seidel over relaxation, and finally with the multi grid method.

Our boundary conditions will be

$\Phi(0,y) = 1$

$\Phi(1,y) = 0$

$\Phi(x,1) = 1 - x$

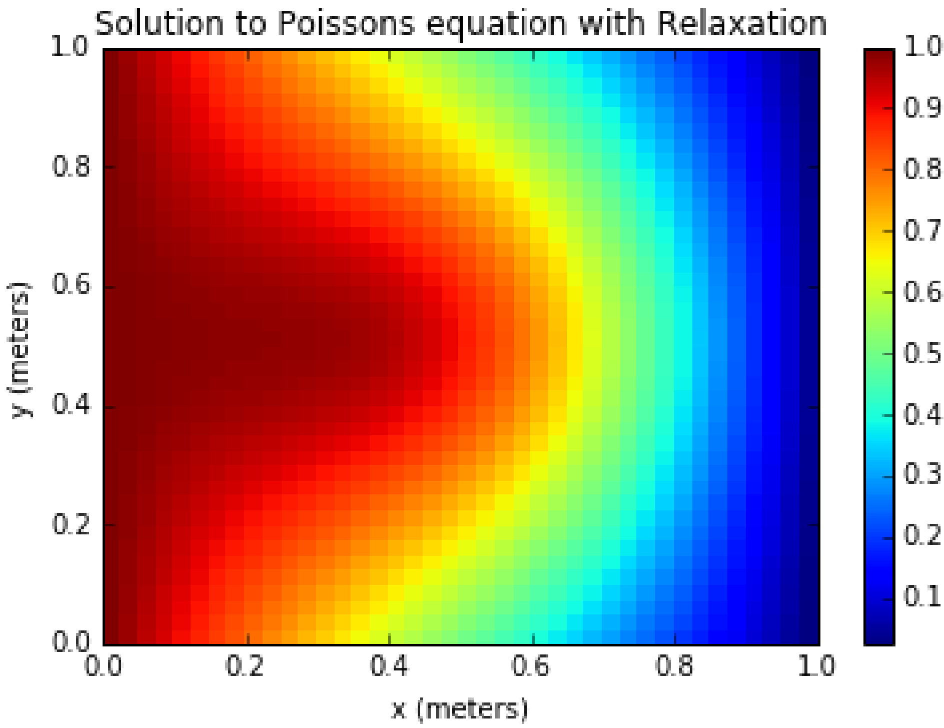
$\Phi(x,0) = 1 - x$

```
In [27]: #RELAXATION

         start = 0
         Boxsize = 1 #0 <= x <= 1, 0 <= y <= 1

         #Define the charge density function
         def rho(x,y):
             return np.exp(-16*(((x - .5)**2) + ((y - .5)**2)))

         #Our grid will be M+1 by M+1, and target will be the precision we aim for
         def relaxation(M, target):
             delta = 1

             #Create arrays for our x and y values
             xs = np.linspace(start, start+Boxsize, M + 1)
             ys = np.linspace(start, start+Boxsize, M + 1)
             del_x = xs[1] - xs[0]

             X,Y = np.meshgrid(xs,ys)
```

```
    #Create an array of our charge density on the grid
    charge = rho(X,Y)

    #Put in our boundary conditions
    phi = np.zeros((M+1,M+1), float)
    phi[:,0] = 1 - xs[:]
    phi[:,M] = 1 - xs[:]
    phi[0,:] = 1
    phi[M,:] = 0

    new_phi = np.zeros((M+1,M+1), float)
    new_phi[:,0] = 1 - xs[:]
    new_phi[:,M] = 1 - xs[:]
    new_phi[0,:] = 1
    new_phi[M,:] = 0

    #Start a timer
    start_time = time.time()

    #Relaxation
    while delta > target:
        for i in range(1,M):
            for j in range(1,M):

                new_phi[i,j] = (1/4)*(phi[i + 1,j]
                + phi[i - 1,j] + phi[i, j + 1]
                + phi[i, j - 1] + 4*(math.pi)*charge[i,j]*(del_x**2))

        #We define our delta as the maximum difference
        #between elements in our previous and current matrix
        delta = np.max(abs(new_phi - phi))
        phi, new_phi = new_phi, phi

    #End the timer
    end_time = time.time()

    print('Time was: ', end_time - start_time, ' seconds')

    return new_phi, X, Y

#We'll start with a 40x40 grid and aim for a precision of 1e-4
phi, X, Y = relaxation(40, 1e-4)
plt.pcolor(X,Y, phi.T)
plt.colorbar()
plt.xlabel('x (meters)')
plt.ylabel('y (meters)')
plt.title('Solution to Poissons equation with Relaxation')
```

```
plt.show()
```

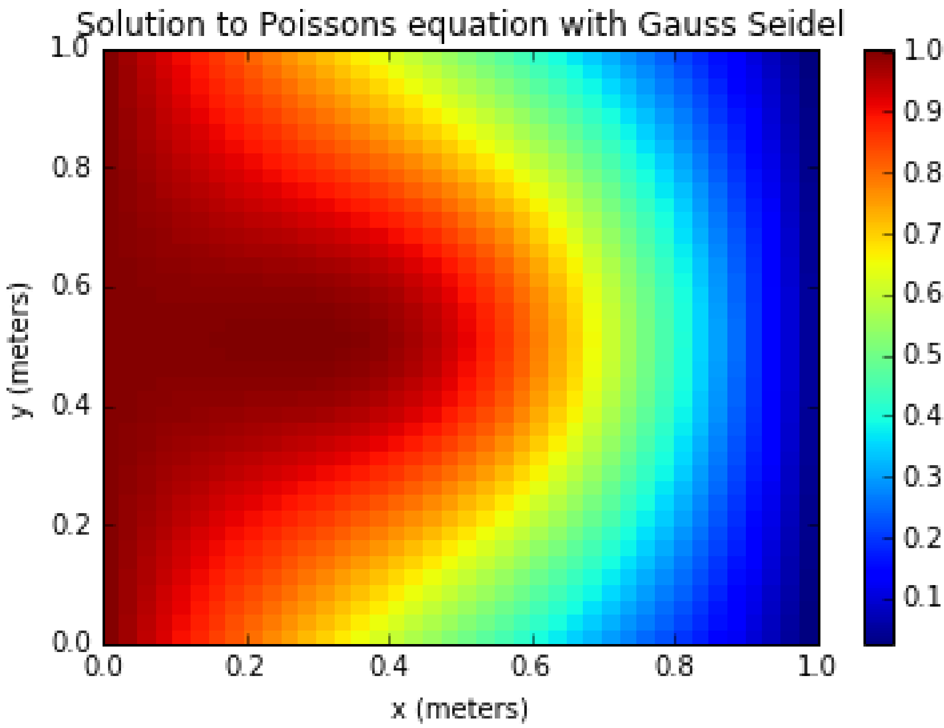
```
Time was:  3.2477588653564453  seconds
```




```
In [28]: #GAUSS SEIDEL

         def gauss_seidel(M, target, w):
             delta = 1

             xs = np.linspace(start, start+Boxsize, M + 1)
             ys = np.linspace(start, start+Boxsize, M + 1)
             del_x = xs[1] - xs[0]

             X,Y = np.meshgrid(xs,ys)
             charge = rho(X,Y)

             phi = np.zeros((M+1,M+1), float)
             phi[:,0] = 1 - xs[:]
             phi[:,M] = 1 - xs[:]
             phi[0,:] = 1
             phi[M,:] = 0

             phi_prime = np.zeros((M+1,M+1), float)
```

```
    phi_prime[:,0] = 1 - xs[:]
    phi_prime[:,M] = 1 - xs[:]
    phi_prime[0,:] = 1
    phi_prime[M,:] = 0

    start_time = time.time()

    #Gauss Seidel
    while delta > target:
        for i in range(1,M):
            for j in range(1,M):

                #We first do a relaxation step to define phi_prime
                phi_prime[i,j] = (1/4)*( phi[i + 1,j]
                + phi[i - 1,j] + phi[i, j + 1]
                + phi[i, j - 1]+ 4*(math.pi)*charge[i,j]*((del_x**2)))

                #Then we calculate the difference between phi and phi_prime
                v_delta = phi_prime[i,j] - phi[i,j]

                #We over correct by this difference multiplied by w
                phi[i,j] = phi[i,j] + w*v_delta

        delta = np.max(abs(phi_prime - phi))
        phi_prime, phi = phi, phi_prime

    end_time = time.time()

    print('Time was: ', end_time - start_time, ' seconds')

    return phi, X, Y

#We'll start with a 40x40 grid and aim for a precision of 1e-4
phi, X, Y = gauss_seidel(40, 1e-4, 1.8)
plt.pcolor(X,Y, phi.T)
plt.colorbar()
plt.xlabel('x (meters)')
plt.ylabel('y (meters)')
plt.title('Solution to Poissons equation with Gauss Seidel')

plt.show()
```

```
Time was:  0.6750340461730957  seconds
```



We have used a gauss seidel over relaxation factor of w = 1.8. I found that past this point (between 1.8 and 2) the algorithm diverged. We see that Gauss Seidel is almost 5 times as fast!

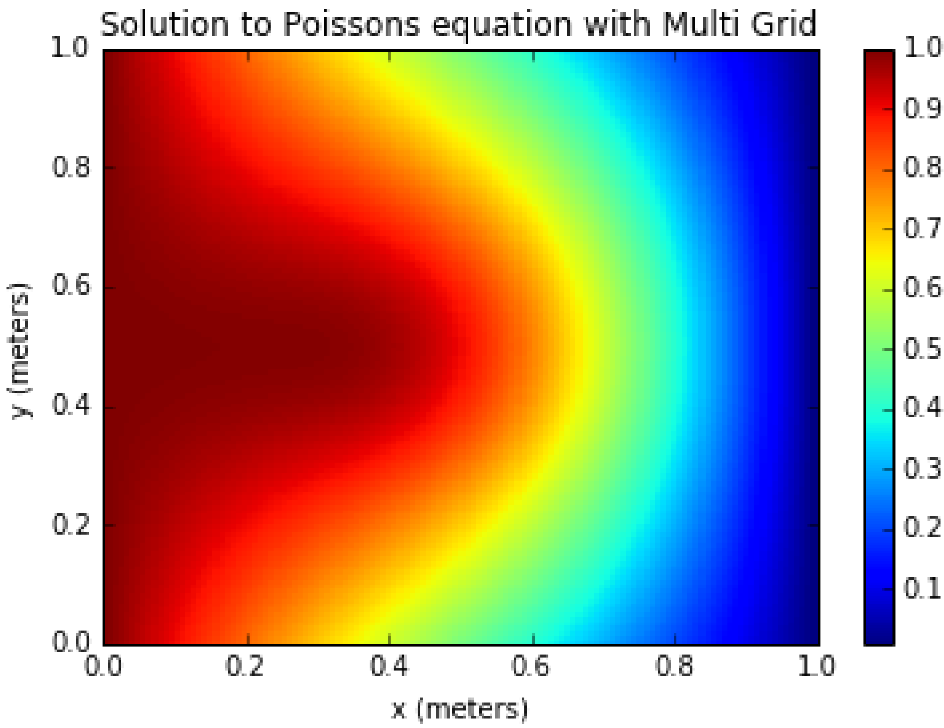
Next we will use the multigrid method. In this method, we start with a small grid size, 5x5 and we perform relaxation. Then we increase the grid size to 9x9, then 17x17 . . . 2N-1x2N-1. We use average interpolation to fill in new grid points. After reaching our finest grid size, we use Gauss Seidel, w = 1.8, to do a final smoothing of our solution.

```
In [26]: #MULTI_GRID

         def multi_grid(N, target, iterations, w):

             delta = 1

             xs = np.linspace(start, start+Boxsize, N + 1)
             ys = np.linspace(start, start+Boxsize, N + 1)
             del_x = xs[1] - xs[0]

             X,Y = np.meshgrid(xs,ys)
             charge = rho(X,Y)

             phi = np.zeros((N+1,N+1), float)
             phi[:,0] = 1 - xs[:]
             phi[:,N] = 1 - xs[:]
             phi[0,:] = 1
```

```python
phi[N,:] = 0

new_phi = np.zeros((N+1,N+1), float)
new_phi[:,0] = 1 - xs[:]
new_phi[:,N] = 1 - xs[:]
new_phi[0,:] = 1
new_phi[N,:] = 0

start_time = time.time()
while delta > target:
    for i in range(1,N):
        for j in range(1,N):

            new_phi[i,j] = (1/4)*(
            phi[i + 1,j] + phi[i - 1,j]
            + phi[i, j + 1] + phi[i, j - 1]
            + 4*(math.pi)*charge[i,j]*(del_x**2))

    delta = np.max(abs(new_phi - phi))
    phi, new_phi = new_phi, phi

for q in range(1, iterations + 1):
    N = 2*N
    re_phi = np.zeros((N + 1,N + 1), float)

    #copy over the points from last array
    re_phi[::2,::2] = phi

    #Average every other column in a given row
    #(Skiping rows with all zero entries)
    for i in range(0,N+1,2):
        for j in range(1,N,2):
            re_phi[i,j] = (re_phi[i,j-1] + re_phi[i,j+1])/2

    #Average every other row in a given column
    for j in range(0,N+1):
        for i in range(1,N,2):
            re_phi[i,j] = (re_phi[i-1,j] + re_phi[i+1,j])/2

    if (q != iterations):
        phi, re_phi = re_phi, phi

#Now do another relaxation step on the final matrix size
xs = np.linspace(start, start+Boxsize, N + 1)
ys = np.linspace(start, start+Boxsize, N + 1)
del_x = xs[1] - xs[0]

X,Y = np.meshgrid(xs,ys)
```

```
        charge = rho(X,Y)

        phi_prime = np.zeros((N+1,N+1), float)
        phi_prime[:,0] = 1 - xs[:]
        phi_prime[:,N] = 1 - xs[:]
        phi_prime[0,:] = 1
        phi_prime[N,:] = 0

        delta_2 = 1

        while delta_2 > target:
            for i in range(1,N):
                for j in range(1,N):

                    phi_prime[i,j] = (1/4)*( re_phi[i + 1,j]
                    + re_phi[i - 1,j] + re_phi[i, j + 1]
                    + re_phi[i, j - 1]+ 4*(math.pi)*charge[i,j]*((del_x**2)))

                    v_delta = phi_prime[i,j] - re_phi[i,j]

                    re_phi[i,j] = re_phi[i,j] + w*v_delta

            delta_2 = np.max(abs(phi_prime - re_phi))
            phi_prime, re_phi = re_phi, phi_prime

        end_time = time.time()

        print('Time was: ', end_time - start_time, ' seconds')
        print('Final solution is ', N, 'x',N)

        return re_phi, X, Y

    phi, X, Y = multi_grid(4, 1e-4, 5, 1.8)

    plt.pcolor(X,Y, phi.T)
    plt.colorbar()
    plt.xlabel('x (meters)')
    plt.ylabel('y (meters)')
    plt.title('Solution to Poissons equation with Multi Grid')

    plt.show()
```

```
Time was:  2.655183792114258  seconds
Final solution is  128 x 128
```



We see the multi grid method is tremendously quick at producing large grids! We have produced a 128x128 grid, notice its smoothness!

NUMBER 2

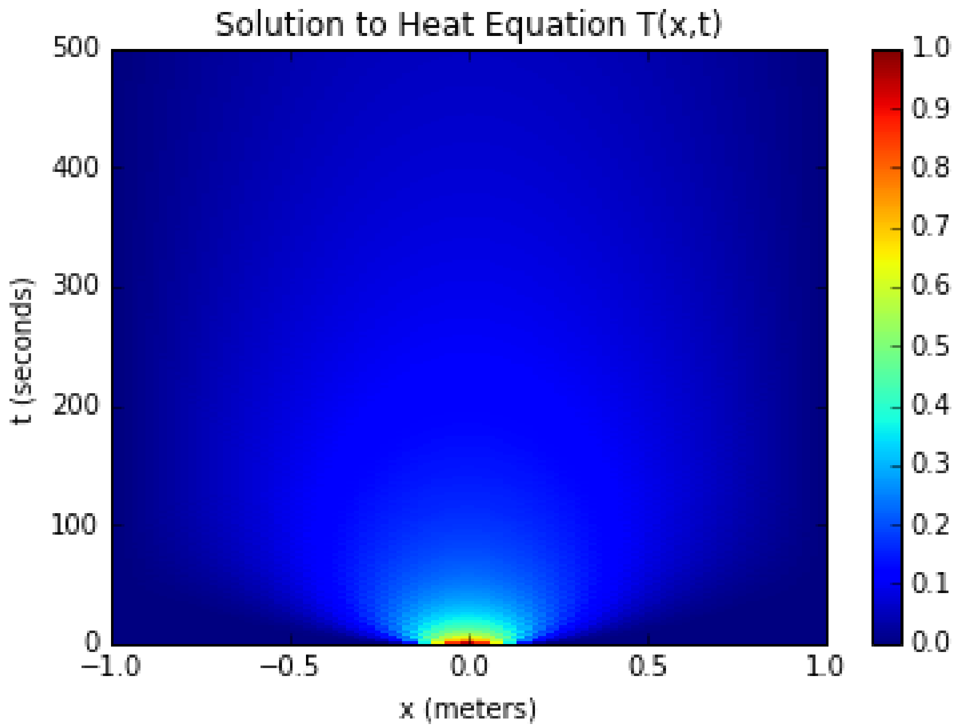
Now we will solve the one dimensional heat equation using the method of lines. We will solve this in the region $-1 \leq x \leq 1$ with boundary conditions

$T(-.1 \leq x \leq .1, t = 0) = 1$ else $T(x, t = 0) = 0$

$T(-1, t) = T(1, t) = 0$

In [29]:
```
K0=1
C0=20
rho0=50
xsteps0=100
tsteps0=2500
t_end0=500

def getxt(i,j):
    x = -1 + (2/xsteps0)*i
    t = 0 + (t_end0/tsteps0)*j
    return x,t

def heat_equation(K,C,rho, xsteps, tsteps, t_end):
    x_a = -1
    x_b = 1
```

```
    xstep = (x_b - x_a)/xsteps
    tstep = (t_end)/tsteps

    T = np.zeros((xsteps + 1,tsteps + 1), float)

    #initial conditions T(|x| < 0.1, t = 0) = 1
    for i in range(0,xsteps + 1):
        x,t = getxt(i,0)
        #print(x)
        if abs(x) < 0.1:
            T[i,0] = 1
        else:
            T[i,0] = 0

    for j in range(0,tsteps):
        for i in range(0,xsteps):
            x,t = getxt(i,j)
            if x == -1 or x == 1:
                T[i,j+1] = 0
            else:
                T[i, j+1] = T[i,j] + (K*tstep/(C*rho*(xstep**2)))*(T[i + 1,j]
                + T[i - 1, j] - 2*T[i,j])


    return T

T = heat_equation(K0,C0,rho0,xsteps0,tsteps0,t_end0)


x = np.linspace(-1, 1, xsteps0 + 1)
t = np.linspace(0, t_end0, tsteps0 + 1)

X,Time = np.meshgrid(x,t)
plt.pcolor(X,Time, T.T)
plt.colorbar()
plt.xlabel('x (meters)')
plt.ylabel('t (seconds)')
plt.title('Solution to Heat Equation T(x,t)')
plt.show()
```


Solution to Heat Equation T(x,t)
500
400
300
200
100
0
t (seconds)
−1.0
−0.5
0.0
0.5
1.0
x (meters)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0

```
In [ ]:
```