Moran_HW2

September 20, 2017

Shannon Moran Phys 514 HW 2 Due: Sept 21

0.1 Import required packages

In [1]:
```
%matplotlib inline
import numpy as np
from math import *
import matplotlib.pyplot as plt
import time
```

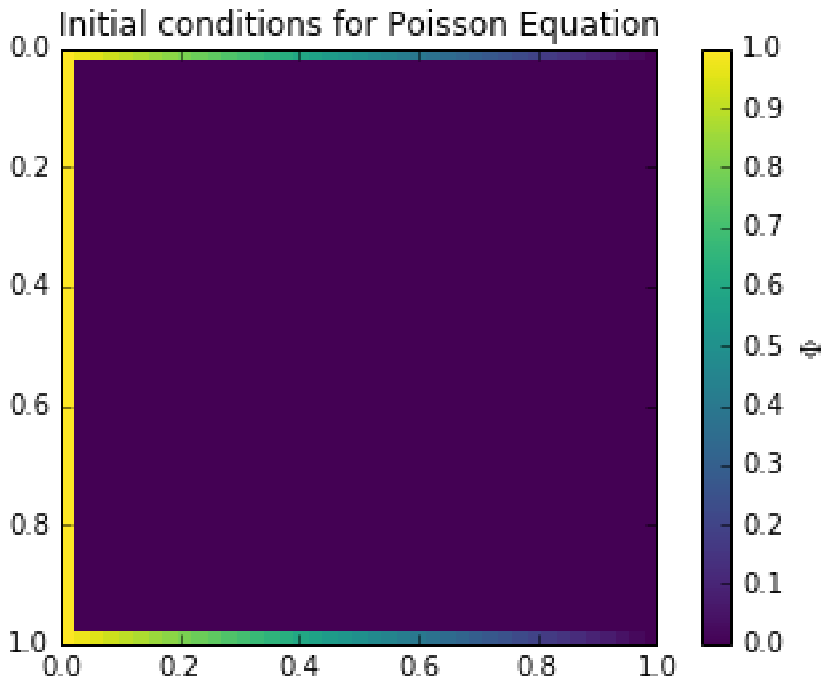
0.2 1, 3: Solving Poisson's Equation

In [2]:
```
# 0) Initialize phi given boundary conditions at t=0
def initialize(N):
    h = 1/N
    phi = np.zeros((N+1,N+1))
    for i in range(N+1):
        phi[i][0] = 1
        phi[i][-1] = 0
        x_values = np.arange(0,1+h,h)
        for j in range(N+1):
            phi[0][j] = 1-x_values[j]
            phi[-1][j] = 1-x_values[j]
    return phi

# 0) Define rho(x,y) for RHS of Poisson's Equation
def f(x,y):
    rho = -4*pi*exp(-16*((x-0.5)**2+(y-0.5)**2))
    return rho

# 1) Relaxation Method
def relaxation(N,tol):
    t0 = time.time()
    h = 1/N
    e,t = 1,1
    phi = initialize(N)
    phi_0 = initialize(N)
    while e>tol:
        phi_new = np.copy(phi_0)
        for i in np.arange(1,N):
            for j in np.arange(1,N):
                phi_new[i][j] = 1/4*(phi[i-1][j]+phi[i+1][j]+phi[i][j-1]
                                     +phi[i][j+1]-h**2*f(j/N,i/N))
```

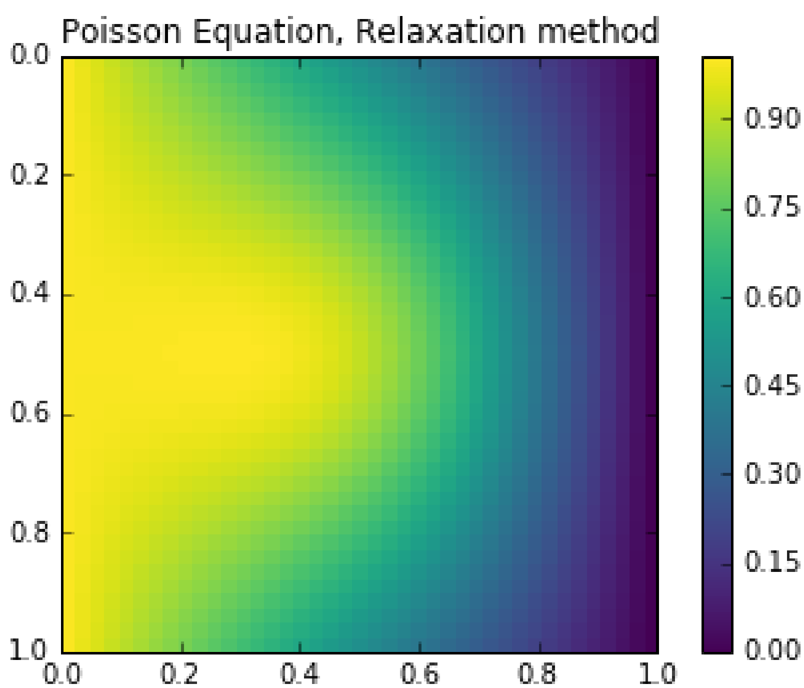
```
        e = np.amax(np.absolute(np.subtract(phi,phi_new)))
        phi = np.copy(phi_new)
        t+=1
    print('Relaxation, compute time: %0.2f s, %d iterations' % (time.time()-t0,t-1))
    plt.imshow(phi,interpolation="nearest",cmap="viridis",extent=[0,1,1,0])
    plt.title("Poisson Equation, Relaxation method")
    plt.colorbar()
    plt.show()
    return

# 2) Gauss-Seidel method
def gauss_seidel(N,tol):
    t0 = time.time()
    h = 1/N
    w = 0.75
    e,t = 1,1
    phi = initialize(N)
    while e>tol:
        phi_old = np.copy(phi)
        delta_phi = np.zeros((N+1,N+1))
        for i in np.arange(1,N):
            for j in np.arange(1,N):
                delta_phi[i][j] = (1/4)*(phi[i-1][j]+phi[i+1][j]+phi[i][j-1]
                                        +phi[i][j+1]-h**2*f(j/N,i/N))-phi[i,j]
        phi = np.copy(np.add(phi,w*(delta_phi)))
        e = np.amax(np.absolute(np.subtract(phi,phi_old)))
        t += 1
    print('Gauss-Seidel, compute time: %0.2f s, %d iterations' % (time.time()-t0,t-1))
    plt.imshow(phi,interpolation="nearest",cmap="viridis",extent=[0,1,1,0])
    plt.title("Poisson Equation, Gauss-Seidel method")
    plt.colorbar()
    plt.show()
    return

# 3) Multigrid method
# Helper function that interpolates grid for N = N_refined > N_coarse
def mg_interpolate(N,phi):
    phi_interpolate = initialize(N)
    for x in np.arange(1,N):
        for y in np.arange(1,N):
            i,j = y/2,x/2
            if (x%2==0 and y%2==0): # ie, if phi exists at this point
                phi_interpolate[y][x] = phi[i][j]
            elif (y%2==0): # horizontal sweep, assumes x%2!=0
                phi_interpolate[y][x] = (1/2)*(phi[i][j-1/2]+phi[i][j+1/2])
            elif (x%2==0): # vertical sweep, assumes y%2!=0
                phi_interpolate[y][x] = (1/2)*(phi[i-1/2][j]+phi[i+1/2][j])
            elif (j%2!=0 and i%2!=0):
                phi_interpolate[y][x] = (1/4)*(phi[i-1/2][j-1/2]+phi[i+1/2][j+1/2]
                                              +phi[i+1/2][j-1/2]+phi[i-1/2][j+1/2])
            else: pass
    return phi_interpolate

# Starts from coarse grid and refines to a grid equal to other methods
```

```python
def multi_grid(N,tol):
    t0 = time.time()
    N_grid = [int(((N+1)/2+1)/2), int((N+1)/2), int(N)]
    phi = initialize(int(N_grid[0]))
    for i in range(len(N_grid)):
        N = N_grid[i]
        if N==N_grid[-1]: pass
        else: Nnext=N_grid[i+1]
        h = 1/N
        phi_new = np.copy(initialize(N))
        e,t = 1,1
        while e>tol:
            phi_new = np.copy(initialize(N))
            for i in np.arange(1,N):
                for j in np.arange(1,N):
                    phi_new[i][j] = 1/4*(phi[i-1][j]+phi[i+1][j]+phi[i][j-1]
                                          +phi[i][j+1]-h**2*f(j/N,i/N))
            e = np.amax(np.absolute(np.subtract(phi,phi_new)))
            phi = np.copy(phi_new)
            t += 1
        if N==N_grid[-1]: pass
        else: phi = np.copy(mg_interpolate(Nnext,phi_new))
    print('Multi-grid, compute time: %0.2f s, %d iterations' % (time.time()-t0,t-1))
    plt.imshow(phi,interpolation="nearest",cmap="viridis",extent=[0,1,1,0])
    plt.title("Poisson Equation, Multi-grid method")
    plt.colorbar()
    plt.show()
    return
```

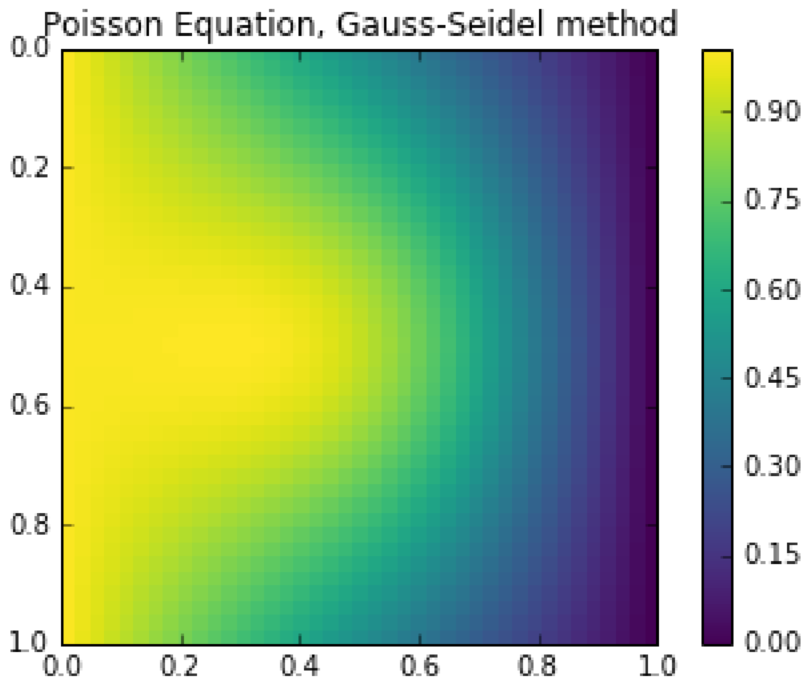
In [3]:
```python
# Number of grid points, translate to step sizes
N = 40

# Set tolerance, defined as max absolute difference between any grid point in steps t1 and t2
tol = 10**(-13)

# Display initial conditions
plt.imshow(initialize(N),interpolation="nearest",cmap="viridis",extent=[0,1,1,0])
plt.title("Initial conditions for Poisson Equation")
plt.colorbar().set_label(r'${\Phi}$')
plt.show()

# Call method to display results
relaxation(N,tol)
gauss_seidel(N,tol)
multi_grid(N,tol)
```



Relaxation, compute time: 65.56 s, 7878 iterations

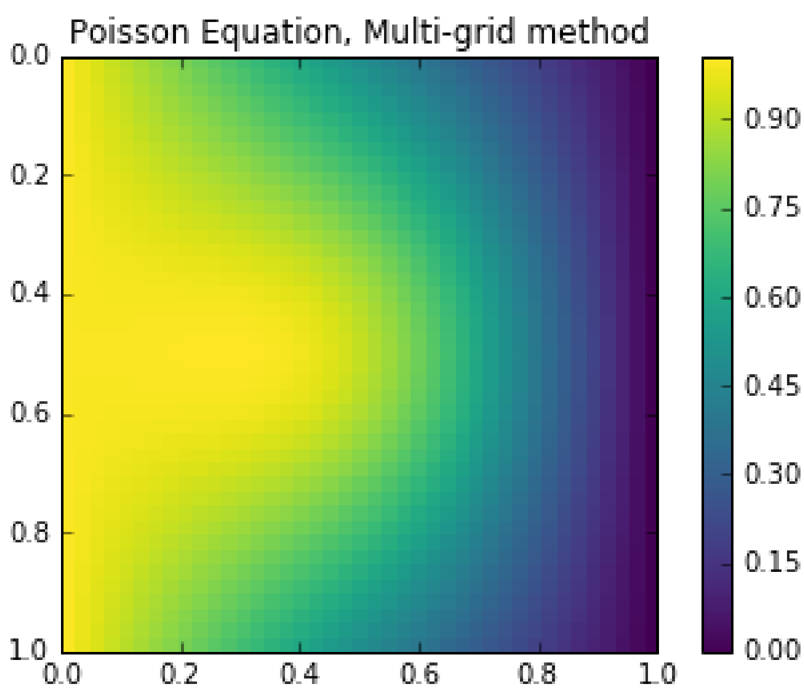


Gauss-Seidel, compute time: 76.77 s, 10382 iterations




```
Multi-grid, compute time: 54.06 s, 5637 iterations
```

```
/Users/shannonmoran/miniconda3/lib/python3.5/site-packages/ipykernel/__main__.py:78: DeprecationWarning:
/Users/shannonmoran/miniconda3/lib/python3.5/site-packages/ipykernel/__main__.py:74: DeprecationWarning:
/Users/shannonmoran/miniconda3/lib/python3.5/site-packages/ipykernel/__main__.py:76: DeprecationWarning:
/Users/shannonmoran/miniconda3/lib/python3.5/site-packages/ipykernel/__main__.py:72: DeprecationWarning:
```



0.3 2: Solving Heat Equation, 1D

Assume the equation is simply:

$$\frac{\partial T}{\partial t} = \nabla^2 T(x,t)$$

Which we can discretize as:

$$T(x_n, t_{n+1}) = T(x_n, t_n) + \frac{\Delta t}{\Delta x^2}[T(x_{n+1,t_n}) - 2T(x_n, t_n) + T(x_{n-1}, t_n]$$

In [8]:
```
def initialize_heat(dx):
    x = np.arange(-1,1+dx,dx)
    N = len(x)
    T = np.zeros(N)
    for i in range(N):
        if (abs(x[i])<0.1): T[i]=1
    return x, T

def forward_euler(trun,dt,dx):
    plt.figure(figsize=(10,5))
    plt.title('Temperature evolution over time')
    plt.ylabel('Temperature')
    plt.xlabel('x-coordinate')
    plt.xticks(np.arange(-1, 1.1, 0.1))
    x,T = initialize_heat(dx)
    F = dt/(dx**2)
    print('F needs to be less than 0.5 to successfully converge. F: %0.2f' %F)
    T_new = np.zeros(len(x))
    tplot = [0.01000, 0.05000, 0.10000, 0.50000, 1.00000]
    t = 0
    plt.plot(x,T,label="t=%0.2f s" %t)
    while t<trun:
        T_new = np.zeros(len(x))
        for i in range(len(x)-1):
            T_new[i] = T[i] + F*(T[i-1] - 2*T[i] + T[i+1])
        # Enforce boundary conditions
        T_new[0],T_new[-1] = 0,0
        T = np.copy(T_new)
        if (round(t,5) in tplot):
            plt.plot(x,T,label="t=%0.2f s" %t)
        t += dt
    plt.legend(bbox_to_anchor=(1.2, 1))
    plt.show()
    return

# Simulation settings
trun = 2
dt = 10**(-5)
dx = 10**(-2)

# Run Forward Euler routine to simulate heating of strip over time
forward_euler(trun,dt,dx)
```

```
F needs to be less than 0.5 to successfully converge. F: 0.10
```