Ex 2 code

by Naomi Getzler

Submission date: 12-Mar-2019 04:43PM (UTC+0000)
Submission ID: 102185119
File name: Ex_2_code.txt (14.78K)
Word count: 1385
Character count: 9107

```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 10 15:11:19 2019

@author: naomi
"""


#----------------------------------------
#import the libraries required in the code
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
import time
from matplotlib import cm
from scipy.sparse import diags
import scipy.sparse.linalg
from mpl_toolkits.mplot3d import axes3d
import copy
import random
import math
#----------------------------------------
#Grid size
Nx = 30
```

```
Ny = 30

#grid spacing

delta = 1

#inital guess of the matrix

Vguess = random.randint(0,10)


#general Gauss-seidel method

def Gauss_seidel_gen(Nx,Ny,delta,x):

    nx = Nx-1

    ny = Ny-1


    X, Y = np.meshgrid(np.arange(0, Nx), np.arange(0, Ny))


    V = np.empty((Nx, Ny))

    V.fill(Vguess)


    Vtop = 0

    Vbottom = 0

    Vleft = 0

    Vright = 0


    V[(ny):, :] = Vtop

    V[:1, :] = Vbottom
```

```
    V[:, (nx):] = Vright
    V[:, :1] = Vleft
    it_count=0
    #convergence condtion
    while True:
        V_old = copy.deepcopy(V)
        for i in range(1, nx, delta):
            for j in range(1, ny, delta):
                V[i, j] = 0.25 * (V[i+1][j] + V[i-1][j] + V[i][j+1] + V[i][j-1])
        it_count += 1

        if abs(np.sum(V_old.flatten()) - np.sum(V.flatten())) < x:
            #print(Iteration number:, it_count)
             break
    return V,it_count,Vtop
#general Jacobi method
def Jacobi_gen(Nx,Ny,delta,x):
    nx = Nx-1
    ny = Ny-1

    X, Y = np.meshgrid(np.arange(0, Nx), np.arange(0, Ny))

    V = np.empty((Nx, Ny))
```

```
V.fill(Vguess)


Vtop = 0

Vbottom = 0

Vleft = 0

Vright = 0


V[(ny):, :] = Vtop

V[:1, :] = Vbottom

V[:, (nx):] = Vright

V[:, :1] = Vleft


it_count = 0

#convergence conditon

while True:

    V_new = V.copy()

    V_old = copy.deepcopy(V)

    for i in range(1, nx, delta):

        for j in range(1, ny, delta):

            V[i, j] = 0.25 * (V_new[i+1][j] + V_new[i-1][j] + V_new[i][j+1] + V_new[i][j-1])

    it_count += 1


    if abs(np.sum(V_old.flatten()) - np.sum(V.flatten())) < x:
```

```
            #print(Iteration number:, it_count)
             break
    return V,it_count,Vtop
#standard deviation error for Gauss
def Gauss_gen_error(x):
    V, it_count, Vtop = Gauss_seidel_gen(Nx,Ny,delta,x)
    V_av = np.mean(V)
    std = math.sqrt(np.mean(abs(V - V_av))**2)
    return std
#test to see if the grid points converge to the boundaries
def Gauss_gen_test(x):
    V, it_count, Vtop = Gauss_seidel_gen(Nx,Ny,delta,x)
    V_av = np.mean(V)
    abs_err = (abs(V_av-Vtop))
    return abs_err
#standard deviation error for Jacobi
def Jacobi_gen_error(x):
    V, it_count, Vtop = Jacobi_gen(Nx,Ny,delta,x)
    V_av = np.mean(V)
    std = math.sqrt(np.mean(abs(V - V_av))**2)
    return std
#test to see if the grid points converge to the boundaries
def Jacobi_gen_test(x):
```

```
    V, it_count, Vtop = Jacobi_gen(Nx,Ny,delta,x)
    V_av = np.mean(V)
    abs_err = (abs(V_av-Vtop))
    return abs_err
#Capacitor model using the Gauss Seidel method
def Gauss_seidel_cap(Nx,Ny,delta):
    V = np.empty((Nx+1,Ny+1))
    V.fill(Vguess)
    Vtop = 0
    Vbottom = 0
    Vleft = 0
    Vright = 0

    nx = Nx-1
    ny = Ny-1

    V[-1:, :] = Vtop
    V[:1, :] = Vbottom
    V[:, -1:] = Vright
    V[:, :1] = Vleft

    sep = 4
    len_plate=10
```

```
# Boundary condition
halflen_plate=int((len_plate/2))
mid_px=int((Nx/2))
mid_py=int((Ny/2))

plate1=int(mid_px-sep)
plate2=int(mid_px+sep)
min_py = mid_py-halflen_plate
max_py = mid_py+halflen_plate
V[plate1,min_py:max_py] = -100
V[plate2,min_py:max_py] = 100
it_count = 0
while True:
   V_old = copy.deepcopy(V)
   for i in range(1,nx,delta):
       for j in range(1,ny,delta):
           if i == plate1 or i == plate2:
               continue
           if j < min_py and j> max_py:
              continue
           else:
               V[i, j] = 0.25*(V[i+1,j] + V[i-1,j] + V[i,j+1] + V[i,j-1])
           it_count += 1
```

```
        if abs(np.sum(V_old.flatten()) - np.sum(V.flatten())) < 1e-10:
            #print(Iteration number:, it_count)
            break

    return  V
#conditions for the furnance and no heat loss
def PhiT_vect_furnace():
    n = 11
    T_H = 1000 + 273
    T_C = 20 + 273

    phiT=np.empty((n,1))
    phiT[0]= T_H
    phiT[1:]= T_C

    return phiT
#conditions for the furnance and the ice bath
def PhiT_vect_ice():
    n = 11
    T_H = 1000 + 273
    T_C = 20 + 273
    T_ice = 273
```

```
    phiT=np.empty((n,1))


    phiT[0]= T_H

    phiT[1:]= T_C

    phiT[n-1,0:]= T_ice


    return phiT

#Diffusion coefficents with dirichlet BC's

def Tridia_mat_dir():

    diff = 59

    spec_heat = 450

    density = 7900

    alpha = diff/(spec_heat*density)

    rodlen = 0.50

    nx = 10

    dx = rodlen/(nx-1)

    dt  = 1


    a = alpha*dt/(dx**2)


    diagonal = np.zeros(nx+1)

    lower    = np.zeros(nx)

    upper    = np.zeros(nx)
```

```
    # Precompute sparse matrix
    diagonal[:] = 1 + 2*a
    lower[:] = -a  #1
    upper[:] = -a  #1
    # Insert boundary conditions

    diagonal[0] = 1
    upper[0] = 0
    lower[-1] = -a
    diagonal[nx] = 1 + a

    A = scipy.sparse.diags(
        diagonals=[diagonal, lower, upper],
        offsets=[0, -1, 1], shape=(nx+1, nx+1),
        format='csr')
    return A.todense()

#Diffusion coefficents with dirichlet BC's

def Tridia_mat_heatloss():
    diff = 59
    spec_heat = 450
```

```
density = 7900

alpha = diff/(spec_heat*density)

rodlen = 0.50

nx = 10

dx = rodlen/(nx-1)

dt  = 0.1


a = alpha*dt/(dx**2)


diagonal = np.zeros(nx+1)

lower    = np.zeros(nx)

upper    = np.zeros(nx)


# Precompute sparse matrix

diagonal[:] = 1 + 2*a

lower[:] = -a  #1

upper[:] = -a  #1

# Insert boundary conditions

diagonal[0] = 1

upper[0] = 0

diagonal[nx] = 1

lower[-1] = 0
```

```
    A = scipy.sparse.diags(
        diagonals=[diagonal, lower, upper],
        offsets=[0, -1, 1], shape=(nx+1, nx+1),
        format='csr')
    return A.todense()



MyInput = '0'
while MyInput != 'q':
    MyInput = input('Enter a choice, \n "a)" to explore convergence conditions,\n "b)" to
plot the capaticator, \n "c)" to explore the diffusion rod, \n "q)" to quit: ')
    print('You entered the choice: ',MyInput)



    if MyInput == 'a':
        print('You have chosen to explore the convergence conditions of the Gauss and
Jacobi methods')
        print('The respective errors for Jacobi and Gauss Seidel are:')
        print(Jacobi_gen_test(x=1e-3))
        print(Gauss_gen_test(x=1e-3))

        X = []
        IT1 = []
```

```
IT2 = []

for x in np.logspace(-30,-50,10):
    Nx = 10
    Ny = 10
    V1, it_count1,Vtop = Gauss_seidel_gen(Nx,Ny,delta,x)
    V2, it_count2,Vtop = Jacobi_gen(Nx,Ny,delta,x)
    IT1.append(it_count1)
    IT2.append(it_count2)
    X.append(math.log(x))

plt.plot(X,IT1, color = "r",marker ="^", label = "Gauss")
plt.plot(X,IT2, color = "b",marker ="^", label = "Jacobi")
plt.legend(title = "Method",fontsize = 15)
plt.xlabel('Convergence tolerance',fontsize = 15)
plt.ylabel('Number of iterations',fontsize = 15)
plt.grid(True)


plt.show()


def plotofgrid_den(x):
```

```python
N = []

IT1 = []

IT2 = []


for Nx in range(0,30,3):

    for Ny in range(0,30,3):


        V, it_count1,Vtop = Gauss_seidel_gen(Nx,Ny,delta,x)

        V, it_count2,Vtop = Jacobi_gen(Nx,Ny,delta,x)


        IT1.append(it_count1)

        IT2.append(it_count2)

        N.append(Nx)


IT1=np.array(IT1)

IT1=np.reshape(IT1,(10,10))


IT2=np.array(IT2)

IT2=np.reshape(IT2,(10,10))


N=np.array(N)

N=np.reshape(N,(10,10))
```

```
    plt.plot(N[:,1],IT1[9], color = "r",marker ="^", label = "Gauss")
    plt.plot(N[:,1],IT2[9], color = "b",marker ="^", label = "Jacobi")
    plt.legend(title = "Limit",fontsize = 15)
    plt.xlabel('Sqrt of the no. grid points',fontsize = 15)
    plt.ylabel('Number of iterations',fontsize = 15)
    plt.grid(True)

plt.show(plotofgrid_den(x=1e-3))

print('The standard deviation on the mean of Jacboi method is',
Jacobi_gen_test(x=1e-3))
print('The standard deviation on the mean of Gauss-seidel method
is',Gauss_gen_test(x=1e-3))


def plotofgrid_sd(x):
    N = []
    Jac = []
    Gau = []

    for Nx in range(0,30,3):
        for Ny in range(0,30,3):
            j = Jacobi_gen_error(x=1e-3)
```

```
        g = Gauss_gen_error(x=1e-3)

        Jac.append(j)
        Gau.append(g)
        N.append(Nx)

    plt.plot(N,Jac, color = "r",marker ="^", label = "Gauss")
    plt.plot(N,Gau, color = "b",marker ="^", label = "Jacobi")
    plt.legend(title = "Limit",fontsize = 15)
    plt.xlabel('Sqrt of the no. grid points',fontsize = 15)
    plt.ylabel('Standard deviation',fontsize = 15)
    plt.grid(True)

plt.show(plotofgrid_sd(x=1e-3))

def plotofconvergence_sd():
    X = []
    Jac = []
    Gau = []

    for x in np.logspace(-30,-50,10):
        j = Jacobi_gen_error(x)
        g = Gauss_gen_error(x)
```

```
            Jac.append(j)

            Gau.append(g)

            X.append(math.log(x))


        plt.plot(X,Jac, color = "r",marker ="^", label = "Gauss")

        plt.plot(X,Gau, color = "b",marker ="^", label = "Jacobi")

        plt.legend(title = "Method",fontsize = 15)

        plt.xlabel('Convergence tolerance',fontsize = 15)

        plt.ylabel('Standard deviation',fontsize = 15)

        plt.grid(True)


    plt.show(plotofconvergence_sd())



elif (MyInput =='b'):


    #create 2-d mesh

    X, Y = np.meshgrid(np.linspace(0,1,Nx + 1), np.linspace(0,1,Ny + 1))


    start = time.time()

    V = Gauss_seidel_cap(Nx,Ny,delta)

    end = time.time()
```

```
print(end-start,'s')
# Set colour interpolation and colour map
colorinterpolation = 50
colourMap = plt.cm.hot #you can try: colourMap = plt.cm.coolwarm

# Configure the contour
plt.title("Contour of Voltage")
plt.contourf(X, Y, Gauss_seidel_cap(Nx,Ny,delta), colorinterpolation,
cmap=colourMap)
# Set Colorbar
plt.colorbar()

plt.show()

Grad = np.gradient(V)
u_val, v_val = np.gradient(Gauss_seidel_cap(Nx, Ny, delta))
u_val = np.zeros((Nx+1, Ny+1)) - u_val
v_val = np.zeros((Nx+1, Ny+1)) - v_val

E = np.sqrt(u_val*u_val + v_val*v_val) #magnitude of electric field

plt.title("Contour of Electric Field")
```

```
plt.contourf(X, Y, E, colorinterpolation, cmap=colourMap)
# Set Colorbar
plt.colorbar()
plt.show()



vector_u, vector_v = Grad
vector_u = np.zeros((Nx+1,Ny+1)) - vector_u
vector_v = np.zeros((Nx+1, Ny+1)) - vector_v
mag = np.sqrt(vector_u*vector_u + vector_v*vector_v)

width = 4*mag/mag.max()

plt.streamplot(X, Y, vector_v, vector_u, color= 'b' ,
                    linewidth=width)



fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, V, rstride=1, cstride=1, cmap=cm.coolwarm,
                linewidth=0, antialiased=False)
#ax.set_zlim(0, 110)
```

```
    ax.set_xlabel('Position x [in m]')
    ax.set_ylabel('Position y [in m]')
    ax.set_zlabel('Potential,V(x,y)')
    plt.title("Plot of Electric Potential V(x,y)")

    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.show()



elif (MyInput =='c'):
  M = Tridia_mat_dir()
  N = Tridia_mat_heatloss()

  b = PhiT_vect_furnace()
  c = PhiT_vect_ice()

  phi_prime1 = np.empty_like(b)
  phi_prime2 = np.empty_like(c)

  nx = 11
```

```
for t in range(1,8000,500):
    for i in range(round(t)):
     phi_prime1 = scipy.sparse.linalg.spsolve(M, b)
     phi_prime1, b = b, phi_prime1

    X = np.linspace(0,0.5,nx)

    plt.plot(X, phi_prime1, label= str(t) + 's')
    plt.xlabel("Postion along rod (m)")
    plt.ylabel("Temperature (K)")
    plt.legend(bbox_to_anchor=(1.04,1), borderaxespad=0)

plt.show()

for t in range(1,8000,500):
    for i in range(round(t)):
        phi_prime2 = scipy.sparse.linalg.spsolve(N,c)
        phi_prime2, c = c, phi_prime2

    X = np.linspace(0,0.5,nx)

    plt.xlabel('Postion on the rod\ m')
    plt.ylabel('Temperature\ K')
```

```
            plt.plot(X, phi_prime2, label= str(t) + 's')
            plt.xlabel("Postion along rod (m)")
            plt.ylabel("Temperature (K)")
            plt.legend(bbox_to_anchor=(1.04,1), borderaxespad=0)

          plt.show()

        elif MyInput != 'q':
            print('This is not a valid choice')

    print('You have chosen to finish - goodbye.')
```

Ex 2 code

GRADEMARK REPORT

FINAL GRADE

/20

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22