Assignment 1

by Naomi Getzler

Submission date: 19-Feb-2019 02:09PM (UTC+0000)
Submission ID: 100892757
File name: Assingment_1.txt (13.19K)
Word count: 1256
Character count: 8654

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 18 13:08:50 2019

@author: naomi
"""

from mpl_toolkits.mplot3d import axes3d
import time
import numpy as np
from scipy import linalg
import matplotlib.pyplot as plt
from math import sqrt
import pylab

#generates a random matrix
def randomMatrix(N):
    return np.random.randint(101,size=(N,N),dtype='int64')
#generates a minor of the matrix
def minorsofMatrix(M,i,j):
    return np.delete(np.delete(M, j, 1), i, 0)
#generates the determinant of an Nxn matrix
def detofMatrix(M,N):
```

```
    if N == 2:
        return M[0,0]*M[1,1]-M[0,1]*M[1,0]
    else:
        det = 0
        for j in range(N):
            det += ((-1)**(j))*(M[0,j])*detofMatrix(minorsofMatrix(M,0,j),N-1)
        return det
#generates the cofactors of the matrix
def cofactorsofMatrix(M,N):
    if N == 2:
        return np.array([[M[1,1],-1*M[1,0]],[-1*M[0,1],M[0,0]]])
    else:
        C = np.zeros((N,N))
        for row in range(N):
            for col in range(N):
                C[row,col] = ((-1)**(row+col))*detofMatrix(minorsofMatrix(M,row,col),N-1)
        return C
#generates the inverse
def inverseofMatrix(M,N):
        return (1/detofMatrix(M,N))*(cofactorsofMatrix(M,N).T)


def builtin(N):
    builtinInverse = linalg.inv(M)
```

```
    return builtinInverse
#tests that the inverse dotted with the orginal matrix produces the identity matrix
def iTest0(M,N):
    I = inverseofMatrix(M,N)
    return np.dot(M,I)


def MatrixB(N):
    return np.random.randint(11,size=(N,1))
#solves simultanoues equations using the analytical method
def solveAnalytic(A,B,N):
     return np.dot(inverseofMatrix(A,N),B)
#solves simultanoues equations using the lu decompositon method
def solveLUD(A,B):
    LU, P = linalg.lu_factor(A)
    return  linalg.lu_solve((LU, P), B)
#solves simultanoues equations using the singular value decompositon method
def solveSVD(A,B):
   U,S,V = np.linalg.svd(A)
   C = np.dot(U.T,B)
   Q = np.linalg.solve(np.diag(S),C)
   return  np.dot(V.T,Q)


def tensor2D(x,z):
```

```
    x1= x
    x2 = x-15
    z1 = 8-z
    z2 = 8-z
    p = np.array([[z1/sqrt(x1**2+z1**2),z2/sqrt(x2**2+z2**2)],
                  [x1/sqrt(x1**2+z1**2),x2/sqrt(x2**2+z1**2)]])
    return p

def tensor3d(x,y,z):
    x1= x
    x2 = x-15
    x3 = x-7.5

    z1 = z
    z2 = z
    z3 = z

    y1 = y
    y2 = y
    y3 = y-8

    norm1 = sqrt((x1**2+y2**2+z1**2))
    norm2 = sqrt(x2**2+y2**2+z2**2)
```

```
    norm3 = sqrt(x3**2+y3**2+z3**2)
    matrixC = np.array([[x1/norm1,x2/norm2,x3/norm3],
            [y1/norm1,y2/norm2,y3/norm3],
            [z1/norm1,z2/norm2,z3/norm3]])

    return matrixC

g= 9.81
Mass = 70
w = Mass*g

def weight2d(w):
    return np.array([w,0])

def weight3D(w):
    return np.array([0,0,w])

def tension2D(x,z):
    inverse = np.linalg.inv(tensor2D(x,z))
    return np.dot(inverse,weight2d(w))

def tension3D(x,y,z,w):
   U,S,V = np.linalg.svd(tensor3d(x,y,z))
```

```
    C = np.dot(U.T,weight3D(w))
    Q = np.linalg.solve(np.diag(1/S),C)
    return  np.dot(V.T,Q)



def check(N):    #this function is defined as there are many user inputs later on
    while True:
        try:
            value = int(input(N))
        except ValueError:
            print('Please enter an integer value') #prevents code from crashing for errors
            continue
        else:
            break
    return value


MyInput = '0'
while MyInput != 'q':
    MyInput = input('Enter a choice, \n "a)" to analytically find the inverse of a matrix,\n
"b)" to compare the methods of solving a martix, \n "c)" to compare the LUD verse the
SVD method,\n "d)" to explore the singularity behaviour, \n "e)" 2D tension plot, \n "f)"
3D tension plot,\n "q)" to quit: ')
    print('You entered the choice: ',MyInput)
```

```
    if MyInput == 'a':
        print('You have chosen to analytically inverse the matrix')
        N=check('What would you like the dimensions of the matrix to be?')
        M = randomMatrix(N)
        print('This is the random matrix you have genterated: \n', M)
        #To check the determinant or the cofactors of the matrix unhastag below!
        #print("The cofactors of this Matrix are", cofactorsofMatrix(M,N))
        #print('The determinant of the matrix is',detofMatrix(M,N))
        print('The inverse of the random matrix generated by the analytical method is',
inverseofMatrix(M,N))
        print('Pythons builtin inverse matrix function is \n', builtin(N))
        print('Test to see if the identity matrix is generated to 15 demical places \n',
np.around(iTest0(randomMatrix(N),N),decimals = 15))



    elif (MyInput =='b'):
      print('You have chose to compare the time for analytic, LUD and SVD methods')
      t_analytic = []
      t_SVD = []
      t_LUD = []
```

```
NxN = []



#Dear marker, it is recommended that you stay in the range of 2 to 9 when testing this code
#please refer to the report to find out why!


for n in range(2,8):
    N = n
    A =  randomMatrix(N)
    B = MatrixB(N)


    start = time.time()
    X_analytic = solveAnalytic(A,B,N)
    end = time.time()
    t_analytic.append(end-start)


    start = time.time()
    x_LUD = solveLUD(A,B)
    end = time.time()
    t_LUD.append(end-start)


    start = time.time()
```

```
        x_SVD = solveSVD(A,B)
        end = time.time()
        t_SVD.append(end-start)

        NxN.append(n)

    plt.plot(NxN,t_analytic, color = "g",marker ="p",label = "Analytic")
    plt.plot(NxN,t_LUD, color = "r",marker ="^", label = "LUD")
    plt.plot(NxN,t_SVD, color = "b",marker ="x", label = "SVD")
    plt.legend(title = "Limit",fontsize = 15)
    plt.xlabel('$N$',fontsize = 15)
    plt.ylabel('Time (s)',fontsize = 15)
    plt.grid(True)
    plt.show()

elif (MyInput =='c'):
    print('You have chosen to compare the time for LUD and SVD methods')
    t_analytic = []
    t_SVD = []
    t_LUD = []
    NxN = []

    #it is recommended that you stay in the range of 2 to 9 when testing this code
```

```
#please refer to the report to find out why!


for n in range(2,450):
    N = n
    A =  randomMatrix(N)
    B = MatrixB(N)


    start = time.time()
    x_LUD = solveLUD(A,B)
    end = time.time()
    t_LUD.append(end-start)


    start = time.time()
    x_SVD = solveSVD(A,B)
    end = time.time()
    t_SVD.append(end-start)


    NxN.append(n)


plt.scatter(NxN,t_LUD, color = "r",marker ="^", label = "LUD")
plt.scatter(NxN,t_SVD, color = "b",marker ="x", label = "SVD")
plt.legend(title = "Limit",fontsize = 15)
plt.xlabel('$N$',fontsize = 15)
```

```
  plt.ylabel('Time (s)',fontsize = 15)
  plt.grid(True)
  plt.show()
elif(MyInput == 'd'):
  Xan = []
  Yan = []
  Zan = []


  XLUD = []
  YLUD = []
  ZLUD = []

  XSVD = []
  YSVD = []
  ZSVD = []

  Kval = []

  for k in np.linspace(0,50,51)/1e15:
     A = np.array([[1,1,1],[1,2,-1],[2,3,k]])
     r = np.array([5,10,15])
     N = len(A)
```

```
            solAn = solveAnalytic(A,r,N)
            Xan.append(solAn[0])
            Yan.append(solAn[1])
            Zan.append(solAn[2])
            solLUD = solveLUD(A,r)
            XLUD.append(solLUD[0])
            YLUD.append(solLUD[1])
            ZLUD.append(solLUD[2])
            solSVD = solveSVD(A,r)
            XSVD.append(solSVD[0])
            YSVD.append(solSVD[1])
            ZSVD.append(solSVD[2])

            Kval.append(k)
        print('Solutions generated using the analytic solving method for a small value of
k:')
        plt.plot(Kval,Xan, label='X')
        plt.plot(Kval,Yan,label='Y')
        plt.plot(Kval,Zan,label='Z')
        pylab.legend(loc='upper right')
        plt.xlabel('$K$',fontsize = 15)
        plt.ylabel('Solution',fontsize = 15)
        plt.show()
```

```
  print('Solutions generated using the LUD solving method for a small value of k:')
  plt.plot(Kval,XLUD, label='X')
  plt.plot(Kval,YLUD,label='Y')
  plt.plot(Kval,ZLUD,label='Z')
  pylab.legend(loc='upper right')
  plt.xlabel('$K$',fontsize = 15)
  plt.ylabel('Solution',fontsize = 15)
  plt.show()
  print('Solutions generated using the SVD solving method for a small value of k:')
  plt.plot(Kval,XSVD, label='X')
  plt.plot(Kval,YSVD,label='Y')
  plt.plot(Kval,ZSVD,label='Z')
  pylab.legend(loc='upper right')
  plt.xlabel('$K$',fontsize = 15)
  plt.ylabel('Solution',fontsize = 15)
  plt.show()


elif (MyInput =='e'):
  print('You have chosen to explore the 2D trapezie artist problem!')
  T1=[]
  T2=[]
```

```
X = []
Z = []

for x in range(1,16,1):
  for z in range(1,8,1):
     X.append(x)
     Z.append(z)
     t= tension2D(x,z)
     T1.append(t[0])
     T2.append(t[1])



max_T1 = max(T1)
max_x1 = X[np.argmax(T1)]
max_z1= Z[np.argmax(T1)]
print('Maximum tension in wire 1 is:',max_T1,'N')
print('This occurs at the x and y points:(', max_x1,',',max_z1,')')

max_T2 = max(T2)
max_x2 = X[np.argmax(T2)]
max_z2 = Z[np.argmax(T2)]
print('Maximum tension in wire 2 is:',max_T2,'N')
print('This occurs at the x and y points:(', max_x2,',',max_z2,')')
```

```
  print('Plot of the tension versus the position for wire 1')
  fig = plt.figure()
  ax = fig.gca(projection='3d')
  new_t_vals = np.array(T1)
  ax.plot_trisurf(X, Z, T1)
  plt.xlabel("x (m)")
  plt.ylabel("z(m)")
  ax.set_zlabel("T1(N)")
  plt.show()

  print('Plot of the tension versus the position for wire 2')
  fig = plt.figure()
  ax = fig.gca(projection='3d')
  new_t_vals = np.array(T2)
  ax.plot_trisurf(X, Z, T2)
  plt.xlabel("x (m)")
  plt.ylabel("z(m)")
  ax.set_zlabel("T2(N)")
  plt.show()

elif (MyInput =='f'):
  print('You have chosen to explore the 3D method')
```

```
T1=[]
T2=[]
T3=[]
X = []
Y = []
Z = []


for x in np.arange(1,16,0.2):
    for y in np.arange(1,8,0.2):
        for z in np.arange(1,7,0.2):
            if y > (7.5/8)*x:
                continue
            if y > -((7.5/8)*x)+16:
                continue
            t= tension3D(x,y,z,w)
            X.append(x)
            Y.append(y)
            Z.append(z)
            T1.append(t[0])
            T2.append(t[1])
            T3.append(t[2])
maxT1 = max(T1)
ind1=T1.index(max(T1))
```

```
maxT2 = max(T2)
ind2=T2.index(max(T2))

maxT3 = max(T3)
ind3=T3.index(max(T3))



print('Maximum tension is in wire 1 is:',maxT1,'N')
print('which occurs at the x,y,z points','(', X[ind1],',',Y[ind1],',',Z[ind1],')')
print('Maximum tension is in wire 2 is:',maxT2,'N')
print('which occurs at the x,y,z point','(', X[ind2],',',Y[ind2],',',Z[ind2],')')
print('Maximum tension is in wire 3 is:',maxT3,'N')
print('which occurs at the x,y,z point','(', X[ind3],',',Y[ind3],',',Z[ind3],')')



print('Plot of the positon with tension for wire 1')
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
p = ax.scatter(X, Y, Z, c=T1, cmap=plt.hot())
plt.xlabel("x (m)")
plt.ylabel("y(m)")
```

```
ax.set_zlabel("z(m)")
fig.colorbar(p)
plt.show()

print('Plot of the positon with tension for wire 2')
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
p = ax.scatter(X, Y, Z, c=T2, cmap=plt.hot())
plt.xlabel("x (m)")
plt.ylabel("y(m)")
ax.set_zlabel("z(m)")
fig.colorbar(p)
plt.show()

print('Plot of the positon with tension for wire 3')
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
p = ax.scatter(X, Y, Z, c=T3, cmap=plt.hot())
plt.xlabel("x (m)")
plt.ylabel("y(m)")
ax.set_zlabel("z(m)")
fig.colorbar(p)
plt.show()
```

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

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