UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ: CAMPUS DE FOZ DO IGUAÇU

CENTRO DE ENGENHARIAS E CIÊNCIAS EXATAS

**Cálculo Numérico**

***Atividade #3***

**Instruções**:

* Entrega individual, via “Tarefas” do Teams e arquivo único em .pdf;
* Use este arquivo .docx para fazer sua atividade, e ao finalizar, gere o .pdf.
* Além de incluir os algoritmos no .pdf, eles devem ser upados em anexo, cada um individualmente e um arquivo txt;
* **Discente**: Daniel Marques da Silva

1. Desenvolva uma rotina que crie, automaticamente, uma matriz *n* x *m* que adiciona números negativos a cada linha, pulando uma coluna, mas nunca atribui valores à primeira coluna.

|  |
| --- |
| **Resposta**: Conforme definido no enunciado, foi elaborado uma rotina em Python para elaboração de uma matriz *n* x *m* que adiciona a cada nova linha um número negativo no próximo item de cada linha. Ex. Linha 3, Coluna 4, número -3. As demais colunas possuem valor 0. Como forma de ser versátil, foi inserida a necessidade do usuário inserir o valor de *n* e *m* que se deseja, conforme figura 1 a seguir:    Figura 1 - Linhas de *Input* para Python  A seguir temos na figura 2, a estrutura para geração da matriz n x m    Figura 2 – Função *Generatmatrix*  Aqui, a lógica implementada é bastante simples e intuitiva, sendo exemplos facilmente encontrados na internet. As condições para adicionar ou não o valor na linha são definidas por *elif, if* e *else.*  Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento. |

1. Desenvolva uma rotina que crie, automaticamente, uma matriz *n* x *n* que atribui na diagonal principal o quadrado da posição da linha, e na diagonal secundária a soma dos elementos da linha e coluna a qual se encontra (exceto sua própria posição).

|  |
| --- |
| **Resposta**: Seguindo a mesma linha de raciocínio do exercício anterior, foi elaborado um projeto-código que pedia ao usuário as dimensões da matriz, contudo o diferencial aqui é a necessidade de atribuir valores para as diagonais principal e secundaria da matriz em questão. A forma de elaboração da função é bastante semelhante a anterior, conforme figura 3. Aqui as condições para atribuição de valor na linha e coluna seguem um principio de verificação se a linha e a coluna forem iguais e/ou as as colunas são equivalente a condição de coluna menos a linha, além de a diagonal secundaria ser a soma dos valores contidos na linha.    Figura 3 – Função *Genematrix*  Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento. |

1. Desenvolva uma rotina que crie uma matriz *Y* a partir das entradas *P* e *V*. A matriz *P* apresenta as posições, e *V* o valor relacionado com a posição de *P*. O valor de *V,* mas com sinal contrário, deve ser armazenado nas coordenadas apresentadas em *P*, e de forma simétrica. Isto é, se a posição de *P* for (1; 2) e de *V* for 10; é preciso armazenar o valor de -10 nas posições (1; 2) e (2; 1) da matriz *Y*. A matriz *P* nunca terá valores iguais na mesma linha, como por exemplo, (2; 2)...

|  |
| --- |
| **Resposta**: A necessidade aqui era entrar com duas matrizes, uma denominada ‘P’, que continha as posições que devem ser colocadas os valores que contidas em ‘V’.    Figura 4 – Entradas das matrizes P e V    Figura 5 – Função de Geração  A função apresentada acima (figura 5), apresenta a função que cria a matriz conforme as definições dadas nas matrizes P e V. Inicialmente é gerada uma matriz de zeros com as dimensões da matriz V, que apresenta os valores que devem ser apresentados em cada linha. Também, é realizada as operações matemáticas necessárias e descritas no enunciado, segundo o ‘*for x...”.*  Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento. |

1. Elabore um algoritmo genérico do método da eliminação de Gauss **ingênuo** e com **pivotamento** **parcial**. Compare a resposta do programa desenvolvido para os seguintes sistemas lineares  usando o método de Gauss ingênuo e com pivoteamento parcial. Compare com a solução direta do software.

|  |
| --- |
| **Resposta**: Nesse exercício eram pedidos dois códigos para a solução do método de eliminação por Gauss. Um sem o a necessidade de troca de linhas(pivotamento) e outro com a troca. Além de verificada o tempo e comparação das soluções com função já integrada ao Python. A figura a seguir apresenta a janela de saída do processo executado para o Gauss Ingenuo.    Figura 6 – Janela Gauss Ingenuo  A próxima figura demonstra a janela de saída para o método de Gauss com o Pivotamento.    Figura 7 – Gauss com Pivotamento  Como é possível observar na ultima linha antes do comando para fechar a janela, os tempos de execução para o método com pivotamento são mais rápidos, o que proporciona menor tempo computacional para os cálculos, uma diferença de 1 ms.  A diferença entre as funções de Gauss ingênuo e com pivotamento são resumidas ao incremento de uma verificação e uso de uma função que executa a troca das linhas antes das operações dentro da matriz.  Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento. |

1. Elabore um algoritmo genérico para cálculo do determinante, considerando um sistema resolvido por pivotamento parcial. A entrada deve ser apenas a matriz [A]. A matriz [B] não precisa ser inserida. Faça o exercício para o mesmo sistema do exercício anterior, e compare sua resposta com uma função pronta do Python, que já resulte no determinante.

|  |
| --- |
| **Resposta**: Conforme descrito, era necessário executar uma solução de determinante para o sistema anterior, considerando ainda o pivotamento. A única adição feita em comparação ao anterior foi uma função para execução do determinante. A figura 8 apresenta a janela de saída do projeto.    Figura 8 – Janela Saída  A penúltima linha apresenta o resultado calculado do determinante da matriz gerada anteriormente. A primeira linha apresenta o calculo do determinante executado pelo processo de funções elaboradas, por ultimo temos o tempo necessário para esses processos.  Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento. |

1. Elabore um algoritmo genérico no Python do método de decomposição LU **com e sem pivotamento parcial**. Compare o resultado obtido diretamente pelo Python. Aplique o método para a matriz de exercícios anteriores e verifique a resposta. (Os Exercicios 6 e 7 podem ser resumidos segundo o código apresentado a seguir)

|  |
| --- |
| **Resposta**: Para esse foi elaborado algoritmo que executasse a decomposição LU. O código a seguir pode ser facilmente atualizado como forma de se adaptar as definições requeridas nos Ex. 6 e 7. Consiste na execução de uma geração, a partir da matriz A em outras duas outras, L e U, que contem a triangulação regular de A e a triangular superior com os coeficientes que zeram as linhas da triangular inferior respectivamente. Além foi necessário utilizar uma matriz P que controla o pivotamento na matriz L. A janela a seguir apresenta a matriz resposta para o sistema apresentado em 7.    Figura 9 – Matriz resposta para LU  Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento. |

1. Elabore uma rotina de cálculo de uma matriz inversa de ordem 50x50 de números randômicos entre -50 e 200, utilizando a decomposição LU **com** e **sem** pivotamento parcial, eliminação de Gauss **com** e **sem** pivotamento parcial, e compare os resultados com a solução direta pelo Python.

|  |
| --- |
| **Resposta**: A elaboração do método LU para determinar a inversa junto de um calculo do seu tempo foi bastante trabalhoso. Foram necessárias considerações de pivotamento, trabalho para execução da inversa além de uma definição mais abrangente para a matriz, onde foi implementada a dimensão requerida pelo usuário, não se limitando apenas a 50x50. A figura a seguir apresenta a janela saída para execução da inversa por LU com pivotamento.    Figura 10 – Inversa por LU  É possível notar o tempo necessário para executar e percorrer as funções para execução da inversa. A figura a cima ainda é resultado de uma inversa com pivotamento, e foram necessários 8 segundos para sua execução.  Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento. |

Lista de Códigos

#################################################################

### EXERCÍCIO 1

#===================================================================================================

#------------------------------------ Automatic Generation of Matrix -------------------------------

# This generate matrix with negative numbers

# Autor : Daniel Marques

# Electrical Engeneering - 2021

#======================================================================

import numpy as np

#----------------------------------------------------------------------

def generatmatrix(m,n,matrix):

a = 2

x = -1

for i in range(1, m+1):

a+=1

if a>=n:

a = 2

line=[]

for j in range(1,n+1):

if j==(i+1):

line.append(x)

x=x-1

elif i>=n and a==j:

line.append(x)

x=x-1

else:

line.append(0)

matrix.append(line)

#----------------------------------------------------------------------

m=(int(input('Número de Linhas:')))

n=(int(input('Número de Colunas:')))

#----------------------------------------------------------------------

ft=[]

generatmatrix(m,n,ft)

print(ft)

#################################################################

### EXERCÍCIO 2

#===================================================================================================

#------------------------------------ Automatic Generation of Matrix -------------------------------

# This generate matrix with conditions for main diagonal and secundary

# Autor : Daniel Marques

# Electrical Engeneering - 2021

#======================================================================

import numpy as np

#----------------------------------------------------------------------

def genematrix(m,n,matrix):

a = n+1

for i in range(1, m+1):

line=[]

a-= 1

for j in range(1,n+1):

if j==i:

line.append(j\*\*2)

elif j==a:

line.append((j\*\*2)+(i\*\*2))

else:

line.append(0)

matrix.append(line)

#----------------------------------------------------------------------

m=(int(input('Número de Linhas:')))

n=(int(input('Número de Colunas:')))

#----------------------------------------------------------------------

ft=[]

genematrix(m,n,ft)

print(ft)

#################################################################

### EXERCÍCIO 3

#==============================================================================

# ============================ Generate Matrix ================================

#################################

# Gens matrix with matrix containing values and positions

# Autor : Daniel Marques

# Electrical Engeneering - 2021

#################################

#==============================================================================

import time as time

import numpy as np

# ============================== Space for Functions ==========================

def genepv(a,b):

matrix =[]

k=np.size(b)

for i in range(k+1):

for j in range(k+1):

matrix = np.zeros([i,j])

for x in range(k):

t=int(a[x][0])

u=int(a[x][1])

matrix[t-1][u-1]=-b[0][x]

matrix[u-1][t-1]=-b[0][x]

for r in range(k):

s=0

for l in range(k):

s=s+float(matrix[r][l])

if l==k-1:

matrix[r][r]=-s

return matrix

# ============================== Space for Input ==============================

p = np.array([[1, 3], [2, 4], [1, 2], [3, 4]],dtype=float)

v = np.array([[1,10,5,100]],dtype=float)

matrix = []

# ============================== Error Definition =============================

# ============================== Main Loop/Output =============================

start=time.time()

matrix = genepv(p,v)

end=time.time()

# ============================== Space for Plots ==============================

print(end-start)

print(matrix)

#################################################################

### EXERCÍCIO 4

#==============================================================================

# ============================ Gauss Ingenuo ===============================

#################################

# Python function for 'Gauss Ingenuo' without pivoting

# Autor : Daniel Marques

# Electrical Engeneering - 2021

#################################

#==============================================================================

import time

import numpy as np

import matplotlib.pyplot as mp

# ============================== Space for Functions ==========================

def zeraelem(mtx, lin, c, k):

dy = mtx[c,c]/mtx[lin,c]

for i in range(k+1):

mtx[lin,i] = mtx[lin,i]\*dy-mtx[c,i]

return mtx

def gaussingenuo(a,b):

matrix=[]

k=np.size(b)

for i in range(k+1):

for j in range(k+2):

matrix=np.zeros([i,j])

i=j=0

for i in range(k):

for j in range(k):

matrix[i,j]=float(a[i,j])

for i in range(k):

matrix[i,4]=float(b[0,i])

i=1

while(i<k):

for j in range(k+1):

if j < i:

matrix = zeraelem(matrix, i, j, k)

i+=1

i=1

m=np.zeros(k)

while i<k:

j=1

l=0

while j<i:

l+=(matrix[-i,-j-1]\*m[k-j])

j+=1

m[k-i]=(-l+matrix[-i,k-1])/matrix[-i,-j-1]

i+=1

print(matrix)

# ============================== Space for Input ==============================

a=np.array([[0.006, 2, -16, -7],[-7, 0.01, -16, 14],[-0.02, 8,-9,15],[2, -7, 9, 5]],dtype=float) #define the matrix for solutions

b=np.array([[14,-3, 17,-12]],dtype=float) #define solution matrix

# ============================== Error Definition =============================

# ============================== Main Loop/Output =============================

st=time.time\_ns()

gaussingenuo(a,b)

print(np.linalg.solve(a,(b.T)))

ed=time.time\_ns()

print((ed-st)\*10\*\*-6,'ms')

# ============================== Space for Plots ==============================

# -----------------------------------------------------------------------------

#################################################################

### EXERCÍCIO 5

#=========================================================================================

# ============================ Determinant with pivotament ===============================

#################################

# Considerations of project

# Autor : Daniel Marques

# Electrical Engeneering - 2021

#################################

#==============================================================================

import numpy as np

import time

# ============================== Space for Functions ==========================

def pivot (mtx,l,k):

i=0

while (i+l)<k:

if np.abs(mtx[l+i,l])>np.abs(mtx[l,l]):

a=mtx[l].copy()

mtx[l]=mtx[l+1]

mtx[l+1]=a

i+=1

return mtx

def zeraelem(mtx, lin, c, k):

dy = mtx[lin,c]/mtx[c,c]

for i in range(k):

mtx[lin,i] = mtx[lin,i]-dy\*mtx[c,i]

return mtx

def determinat (a):

k=np.size(a[0])

matrix=np.zeros([k,k])

x=1

for i in range(k):

for j in range(k):

matrix[i,j]=matrix[i,j]+[a[i,j]]

for i in range(k):

matrix=pivot(matrix,i,k)

i=1

while(i<k):

for j in range(k):

if j < i:

matrix =zeraelem(matrix, i, j, k)

i+=1

for i in range(k):

x\*=matrix[i,i]

print(x)

print (matrix)

# ============================== Space for Input ==============================

a=np.array([[0.006,2 , -16, -7],[-7, 0.01, -16, 14],[-0.02, 8,-9,15],[2, -7, 9, 5]],dtype='f8')

# ============================== Error Definition =============================

# ============================== Main Loop/Output =============================

st=time.time\_ns()

determinat(a)

dt= np.linalg.det(a)

ed=time.time\_ns()

print(dt)

print((ed-st)\*10\*\*-6,'ms')

# ============================== Space for Plots ==============================

# -----------------------------------------------------------------------------

#################################################################

### EXERCÍCIO 6

#==============================================================================

# ============================ LU Decomposion =================================

#################################

# This project make a LU Decomposion with partial pivoting

# Without pivoting comment the function on main def

# Autor : Daniel Marques

# Electrical Engeneering - 2021

#################################

#==============================================================================

import numpy as np

import time

# ============================== Space for Functions ==========================

def generated(mtxp,b,mtxl):

i=1

k=np.size(b)

m=mtxp@(b.T)

n=np.zeros(k,dtype=complex)

while i<k:

j=0

l=0

while j<i:

l+=(mtxl[i,j]\*m[j])

j+=1

n[i]=(-l+(m[i]))

i+=1

return n #Create matrix D

def generatmatrix(mtx,a,k): #Create matrix U

for i in range(k):

for j in range(k):

mtx[i,j]=(a[i,j])

return mtx

def matrixl (mtx,k):

mtx=np.zeros([k,k],dtype=complex)

for i in range(k):

mtx[i,i]=1

return mtx #Create matrix L

def zeraelem(mtx,mtz,lin, c, k):

dy = mtx[lin,c]/mtx[c,c]

mtz[lin,c]=dy

for i in range(k):

mtx[lin,i] = mtx[lin,i]-dy\*mtx[c,i]

return mtx,mtz #Made the up triangulantion

def pivot (mtxu,p,l,k):

i=0

while (i+l)<k:

if np.abs(np.real(mtxu[l+i,l]))>np.abs(np.real(mtxu[l,l])):

a=mtxu[l].copy()

x=p[l].copy()

mtxu[l]=mtxu[l+1]

mtxu[l+1]=a

p[l]=p[l+1]

p[l+1]=x

i+=1

return mtxu,p #Main Pivoting

def pivotl(mtxl,l,k):

i=0

while (i+l)<k:

if np.abs(mtxl[l+i,l])>np.abs(mtxl[l,l]):

a=mtxl[l].copy()

mtxl[l]=mtxl[l+1]

mtxl[l+1]=a

i+=1

return mtxl #Pivot the P matrix

# ============================== Main Function ================================

def gaussingenuo(a,b):

mtxu=[]

mtxl=[]

p=[]

k=np.size(b)

d=[]

#-------------- Generation of Matrix L,U and P ----------------------

mtxu=np.zeros([k,k],dtype=complex)

mtxl=matrixl(mtxl,k)

p=np.zeros([k,k])

mtxu=generatmatrix(mtxu,a,k)

for i in range(k):

p[i,i]=1

#--------------- Structure for L ------------------------------------

i=1

while(i<k):

mtxu,p=pivot(mtxu,p,i,k)

for j in range(k+1):

if j < i:

mtxu,mtxl = zeraelem(mtxu,mtxl ,i, j, k)

i+=1

d=generated(p,b,mtxl) #Function for D matrix

d=np.reshape(d,(1,4)) #Restruct the D matrix

x=np.zeros(k,dtype=complex) #Create the solution matrix

for i in range(k):

j=1

l=0

while j<(k):

l+=(mtxu[k-i-1,j]\*d[0,j])

j+=1

x[k-1-i]=(-l+d[0,k-i-1])/mtxu[k-i-1,j-i-1]

print(x)

# ============================== Space for Input ==============================

a=np.array([[3+2j, 2.3+2j, -16+2j, -17.7+7j],[-15-8j, 2.3+5j, -16+1j, 14-2j],[-5+13j, 8-9j, 60+3j, 7+9j],[-8+7j, -6+1j, 9+1j, 11+5j]],dtype= complex)

b=np.array([[6+5j, -7.2+5j, 3+9.7j, 2-12.9j]],dtype=complex)

# ============================== Error Definition =============================

# ============================== Main Loop/Output =============================

gaussingenuo(a,b)

#print(np.size(b))

#print(c@(b.T))

#print(np.linalg.solve(a,(b.T)))

# ============================== Space for Plots ==============================

# -----------------------------------------------------------------------------

#################################################################

### EXERCÍCIO 8

#==============================================================================

# ====================== LU Decomposion for Inverse ===========================

#################################

# This project make a LU Decomposion with partial pivoting

# Without pivoting comment the function on main def

# Autor : Daniel Marques

# Electrical Engeneering - 2021

#################################

#==============================================================================

import numpy as np

import time

import random as rd

# ============================== Space for Functions ==========================

def generated(mtxp,mtxl,b,l):

i=1

k=np.size(mtxp[0])

m=np.zeros(k)

n=np.zeros(k,dtype=float)

m[l]=1

n[l]=mtxl[0,0]

while i<k:

j=0

l=0

while j<i:

l+=(mtxl[i,j]\*m[j])

j+=1

n[i]=(-l+(m[i]))

i+=1

return n #Create matrix D

def generatmatrix(mtx,a,k):

for i in range(k):

for j in range(k):

mtx[i,j]=(a[i,j])

return mtx #Create matrix U

def matrixl (mtx,k):

mtx=np.zeros([k,k],dtype=float)

for i in range(k):

mtx[i,i]=1

return mtx #Create matrix L

def zeraelem(mtx,mtz,lin, c, k):

dy = mtx[lin,c]/mtx[c,c]

mtz[lin,c]=dy

for i in range(k):

mtx[lin,i] = mtx[lin,i]-dy\*mtx[c,i]

return mtx,mtz #Made the up triangulantion

def pivot (mtxu,p,l,k):

i=0

while (i+l)<k:

if np.abs(np.real(mtxu[l+i,l]))>np.abs(np.real(mtxu[l,l])):

a=mtxu[l].copy()

x=p[l].copy()

mtxu[l]=mtxu[l+1]

mtxu[l+1]=a

p[l]=p[l+1]

p[l+1]=x

i+=1

return mtxu,p #Main Pivoting

def pivotl(mtxl,l,k):

i=0

while (i+l)<k:

if np.abs(mtxl[l+i,l])>np.abs(mtxl[l,l]):

a=mtxl[l].copy()

mtxl[l]=mtxl[l+1]

mtxl[l+1]=a

i+=1

return mtxl #Pivot the P matrix

def solvinv(d,mtxu,x,k):

x[k-1]=d[0,k-1]/mtxu[k-1,k-1]

i = 1

while i < k:

j = 0

aux = 0

while j < i:

aux += (mtxu[k-i-1][k-j-1]\*x[k-j-1] )

j += 1

x[k -i-1] = (-aux+d[0,k-i-1] )/mtxu[i,i]

i += 1

x[0]=(x[0]/mtxu[0,0])\*10

return x

def generatea(m,n,matrix,k):

i=0

while i!=k:

j=0

while j<k:

matrix[i,j]=float(rd.uniform(-50,200))

j+=1

i+=1

return matrix

# ============================== Main Function ================================

def gaussingenuo(a,b):

mtxu=[]

mtxl=[]

p=[]

k=np.size(b)

d=[]

#-------------- Generation of Matrix L,U and P ----------------------

mtxu=np.zeros([k,k],dtype=float)

mtxl=matrixl(mtxl,k)

p=np.zeros([k,k])

mtxu=generatmatrix(mtxu,a,k)

for i in range(k):

p[i,i]=1

#--------------- Structure for L ------------------------------------

i=1

while(i<k):

mtxu,p=pivot(mtxu,p,i,k)

for j in range(k+1):

if j < i:

mtxu,mtxl = zeraelem(mtxu,mtxl ,i, j, k)

i+=1

x=np.zeros(k,dtype=float) #Create the solution matrix

out=np.zeros([k,k])

for i in range(k):

d=generated(p,mtxl,b,i) #Function for D matrix

d=np.reshape(d,(1,k)) #Restruct the D matrix

for j in range(k):

x = solvinv(d,mtxu,x,k)

out[j,i]=x[j]

print(out)

# ============================== Space for Input ==============================

m=input('Número de Linhas:')

n=input('Número de Colunas:')

aux1=int(m)

aux2=int(n)

b=np.zeros([aux1,1]) #base solution matrix

# ============================== Error Definition =============================

# ============================== Main Loop/Output =============================

st=time.time\_ns()

matrix=np.zeros([aux1,aux2],dtype=float) #generate a base matrix with zeros

k=np.size(matrix[0]) #number of therms

matrix=generatea(m,n,matrix,k) #generate a random matrix

gaussingenuo(matrix,b) #solve the matrix 'A' and 'B'

ed=time.time\_ns()

print((ed-st)\*10\*\*-6,'ms')

# ============================== Space for Plots ==============================