24_1s_MTM1013T11_Q03_AndreGoncalvesdaSilva

July 22, 2024

MTM1013 - MÉT. NUM. ECOMPUTACIONAIS Prof. Charles R. P. Szinvelski

0.0.1 QUESTÃO (03): SISTEMAS LINEARES

24/1s - UFSM Data: 22/07/24 (Das 8h00min às 9h00min)

Acadêmico: André Gonçalves da Silva Matrícula: 202210071 Curso/Turma: 123/11

0.0.2 INSTRUÇÕES SOBRE A ELABORAÇÃO DOS ARQUIVOS

A tarefa QUESTÃO (02) tem por finalidade a aplicação direta dos aspectos teóricos e práticos dos métodos numéricos e computacionais abordados na disciplina (Sistemas Lineares e recursos de programação em linguagem PYTHON 3 empregados nas respectivas implementações). A produção do arquivo-resposta deverá seguir os moldes da produção dos arquivos respostas (formatos IPYNB e PDF e ambos deverão ser entregues via TAREFA aberta no MOODLE) para a QUESTÃO (03) de 22/07/2024.

OBSERVAÇÕES

- Salienta-se que observância da nomeação dos arquivos IPYNB e PDF (ver abaixo) e informações do cabeçalho (prencher devidamente) serão partes constituinte do escore (divisões sucessivas por 2):
  - aa_ss_MTMxxxxTtt_Q0x_NomeSobrenome.ipynb;
  - aa_ss_MTMxxxxTtt_Q0x_NomeSobrenome.pdf;
- Os algoritmos que devem ser empregados para as implementações em linguagem PYTHON 3 e *testes de mesas* são os algoritmos dados pelo material didático disponibilizado nos repositórios MEGA e DRIVE e AS RESOLUÇÕES NÃO PRODUZIDAS POR ESTES ALGORITMOS SERÃO DESCONSIDERADAS. Ressalta-se que os algoritmos do Material Didático são adequações dos algoritmos apresentados em Campos Filho (2018) (ou BURDEN E FAYRES(2016)[1]), e consequentemente estes algoritmos destas bibliografias poderão ser utilizados para a implementação;
- A elaboração das questões buscará caracterizações personalizadas para cada aluno, fato que resultará em um diversificado conjunto de situações-testes para a exploração dos aspectos teóricos e práticos da disciplina em detrimento, eventualmente, da aplicabilidade do tema abordado na questão.

[1] BURDEN, Richard L.; FAIRES, J D.; BURDEN, Annette M. Análise Numérica - Tradução da 10ª edição norte-americana. [Digite o Local da Editora]: Cengage Learning Brasil, 2016. E-book. ISBN 9788522123414. Disponível em: https://integrada.minhabiblioteca.com.br/#/books/9788522123414/. Acesso em: 18 abr. 2024.

OBTENÇÃO DE DADOS PARA ELABORAÇÃO DAS QUESTÕES Ao considerar o meu número de matrícula na UFSM, 3332882, monta-se a tabela

d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
0	0	0	3	3	3	2	8	8	2

e sobre essas informações, colocam-se as seguintes elaborações:

(I) Com as informações da matrícula:

- $A = |d9 - d10| = |8 - 2| = 6$, e se $A = 0$, considere $A = 2$;
- $B = \left\lceil \frac{d1+...+d4}{A} \right\rceil = \left\lceil \frac{0+0+0+3}{6} \right\rceil = \lceil 0,5 \rceil = 1$;
- $C = \left\lfloor \frac{d5+d6+d7+d8}{A} \right\rfloor = \left\lfloor \frac{3+3+2+8}{6} \right\rfloor = \lfloor 2,67 \rfloor = 2$. Se $C = 0$ considere $C = 1$;
- $Dados = [A, B, C] = [6, 1, 2]$;
- $d5 + d6 = 3 + 3 = 6$;
- $d6 + d7 = 3 + 2 = 5$;
- $d7 + d8 = 2 + 8 = 10$;

Considere as informações acima, matrícula e demais dados, e monte o seguinte sistema linear:

$$\begin{cases} (5A)x + (d_1 + d_2)y + (d_2 + d_3)z = (d5 + d6) \\ (d_1 + d_2)x + (5B)y + (d_4 + d_5)z = (d6 + d7) \\ (d_2 + d_3)x + (d_4 + d_5)y + (5C)z = (d8 + d9) \end{cases} \Leftrightarrow \begin{cases} 30x = 6 \\ 5y + 6z = 5 \\ 6y + 10z = 10 \end{cases}.$$

(Determinante via WolframAlpha.)

0.0.3 QUESTÕES.

1) (0,5 ponto) Faça o que se pede, a partir das intruções acima:

1.1) (0,25 ponto [Dados]) Informe $Matricula = [d1, d2, d3, d4, d5, d6, d7, d8, d9, d10]$ e $Dados = [A, B, C]$) como listas;

1.2) (0,25 ponto [Dados]) Informe o Sistema Linear, na forma acordada, associado ao tratamento dos dados;

2) (1,0 ponto) A partir de 1.2), faça o que se pede:

2.1) (0,3 ponto) Aplique os critérios de confirmação de convergência de matrizes;

2.2) (0,7 ponto) Aplique o método numérico adequado para a obtenção da solução e informe a solução;

Observação: Quantificação do escore da Questão 2;

- Método adequado 100% do escore;

- Método adequado possível 90% do escore;

- Método possível 50% do escore;

- Recurso computacional 20% do escore;

OBSERVAÇÃO: Colocar os documentos IPYNB e PDF da resolução no espaço TAREFA do MOODLE aberta, caso queira entregar o arquivo-resposta com esses recursos computacionais. Caso se resolva via Google COLAB, colocar link compartilhado sem restrições de acesso e também o arquivo PDF associado.

---

[1]:
```
import numpy as np
import math
```

Aqui está os códigos sobre sistemas lineares que eu escrevi:

[2]:
```
def elimination(A, b):
    # junta a matriz A e o vetor b como uma coluna
    M = np.concatenate((A, b), axis=1)

    # para cada coluna k de 0 a n-1 (que no caso n-2), pois vamos transformar␣
 ↪em tringular superior
    # trabalhando coluna por coluna, primeiro zerando os elementos abaixo da␣
 ↪diagonal principal
    # na primeira coluna, depois na segunda, mas a ultima coluna não precisa␣
 ↪ser zerada pois não
    # existem elementos abaixo da diagonal principal
    for k in range(0, M.shape[0]-1):
        # para cada linha i de k+1 a n (que no caso n-1), começa em k+1 pois␣
 ↪zeramos somente abaixo da diagonal principal
        for i in range(k+1, M.shape[0]):
            M[i] = M[i] - M[k] * M[i, k] / M[k, k]

    # retorna a matriz A e o vetor b separadamente
    return np.split(M, [M.shape[1]-1], axis=1)

def STS(A, b):
    n = len(b)
    x = np.zeros(n)

    # i vai variar de n-1 a 0 (por isso o n-1 no primeiro argumento, -1 porque␣
 ↪python vai ir até 0, e o último
```

```
    # -1 diz que deve decrementar -1 de i a cada iteração), o resto será o que␣
↪está a direita do x em questão,
    # por isso j vai de i+1 a n (que no caso sera n-1)
    for i in range(n-1, -1, -1):
        rest = sum([A[i, j] * x[j] for j in range(i+1, n)])
        # o .item() é para o numpy parar de reclamar
        x[i] = (b[i] - rest).item() / A[i,i]

    return x

def STI(A, b):
    n = len(b)
    x = np.zeros(n)

    # i vai variar de 0 a n-1, o resto será o que está a esquerda do x em␣
↪questão
    # logo j vai de 0 a i-1 (é python, então tudo começa em 0 e vai até um␣
↪valor antes do final)
    for i in range(n):
        rest = sum([A[i, j] * x[j] for j in range(i)])
        x[i] = (b[i] - rest).item() / A[i,i]

    return x

def LUdecomp(A):
    n = A.shape[0]
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    for i in range(n):
        for j in range(i, n):
            U[i, j] = A[i, j] - sum([L[i, k] * U[k, j] for k in range(i)])

        # queria poder usar i aqui, mas com j fica melhor no código, para U o i␣
↪significa a linha e j a coluna,
        # para L o i significa a coluna e o j linha
        for j in range(i+1, n):
            L[j, i] = (A[j, i] - sum([L[j, k] * U[k, i] for k in range(i)]))/
↪U[i,i]

    L = np.identity(n) + L

    return L, U

def solveLU(A, b):
    L, U = LUdecomp(A)
```

```
    y = STI(L, b)
    x = STS(U, y)

    return x

def partialpivot(A):
    P = np.identity(A.shape[0])
    for i in range(Anew.shape[0]-1):
        biggestindex = i
        for j in range(i+1, Anew.shape[0]):
            if abs(Anew[j,0]) > abs(Anew[biggestindex,0]):
                biggestindex = j

        temp = np.copy(P[i,:])
        P[i,:] = P[biggestindex,:]
        P[biggestindex,:] = temp

    return P

def positivedefinite(A):
    for i in range(A.shape[0]):
        if np.linalg.det(A[:i+1,:i+1]) <= 0:
            return False

    return True

def cholesky(A):
    if not positivedefinite(A):
        return None

    G = np.zeros(A.shape)
    G[0,0] = np.sqrt(A[0,0])

    for j in range(A.shape[0]):
        if j != 0:
            G[j,j] = np.sqrt(A[j,j] - sum([G[j,k]**2 for k in range(i)]))

        for i in range(j+1, A.shape[0]):
            if j == 0:
                G[i,j] = A[i,j] / G[0,0]

            else:
                G[i,j] = (A[i,j] - sum([G[i,k]*G[j,k] for k in range(j)]))/
 ↪G[j,j]

    return G
```

```python
def solvecholesky(A, b):
    G = cholesky(A)
    if G is None:
        return None

    y = STI(G, b)
    x = STS(G.T, y)

    return x

def nulldiagonalelem(m):
    for i in range(len(m[1,:])):
        if m[i,i] == 0:
            return True
    return False

def diagonaldominant(m):
    for i in range(len(m[1,:])):
        sum = 0.0
        for j in range(len(m[1,:])):
            if i != j:
                sum += abs(m[i,j])

        if abs(m[i,i]) <= sum:
            return False

    return True

def linecolcriterium(m):
    for i in range(len(m[1,:])):
        sum = 0.0
        for j in range(len(m[1,:])):
            if i != j:
                sum += abs(m[i,j])

        if sum >= 1:
            return False

    return True

def getLR(m):
    mnew = np.copy(m)
    for i in range(len(m[:,1])):
        mnew[i,:] = - mnew[i,:] / m[i,i]
        mnew[i,-1] = -mnew[i,-1]
        mnew[i,i] = 0
```

```python
    return mnew

def converges(m):
    #always necessary
    if np.linalg.det(m) == 0:
        return False

    #not always necessary
    if nulldiagonalelem(m):
        return False

    mnew = getLR(m)

    if diagonaldominant(m) or linecolcriterium(mnew) or linecolcriterium(mnew.
 ↪T):
        return True

    return False

def formatacaough(A):
    for i in range(A.shape[0]):
        if len(A.shape) == 2:
            for j in range(A.shape[1]):
                print("[{0:^9}]".format("%.3g" % A[i,j]), end='')
        else:
            print("[{0:^9}]".format("%.3g" % A[i]), end='')

        print()
```

[3]:
```python
d10 = 1.0
d9 = 7.0
d8 = 0.0
d7 = 0.0
d6 = 1.0
d5 = 2.0
d4 = 2.0
d3 = 0.0
d2 = 2.0
d1 = 0.0

A = abs(d9 - d10)
if A == 0:
    A = 2.0

B = math.ceil((d1 + d2 + d3 + d4)/A)
C = math.floor((d5 + d6 + d7 + d8)/A)
if C == 0:
```

```
    C = 1.0
```

[4]:
```
M = np.array([5.0*A, (d1 + d2), (d2 + d3), (d1 + d2), 5.0*B, (d4 + d5), (d2 +
 ↪d3), (d4 + d5), 5.0*C]).reshape(3,3)
print(M)
b = np.array([(d5 + d6), (d6 + d7), (d8 + d9)]).reshape(3,1)
print(b)
```

```
[[30.  2.  2.]
 [ 2.  5.  4.]
 [ 2.  4.  5.]]
[[3.]
 [1.]
 [7.]]
```

[5]:
```
print("Matrícula: %s"%([d1, d2, d3, d4, d5, d6, d7, d8, d9, d10]))
print("Dados: %s"%([A, B, C]))
```

```
Matrícula: [0.0, 2.0, 0.0, 2.0, 2.0, 1.0, 0.0, 0.0, 7.0, 1.0]
Dados: [6.0, 1, 1.0]
```

O sistema linear é dado pelas equações

$$\begin{aligned} 30x + 2y + 2z &= 3, \\ 2x + 5y + 4z &= 1, \\ 2x + 4y + 5z &= 7. \end{aligned} \tag{1}$$

[6]:
```
Mamp = np.concatenate((M, b), axis=1)
print("Sistema linear: ")
formatacaough(Mamp)
```

```
Sistema linear:
[   30    ][    2    ][    2    ][    3    ]
[    2    ][    5    ][    4    ][    1    ]
[    2    ][    4    ][    5    ][    7    ]
```

Os critérios de convergência podem ser checados com a função escrita em código, sendo eles: determinante $|M| \neq 0$, não ter elementos nulos na diagonal, ser estritamente diagonal dominante e, na decomposição da matriz em matrizes L e R, satisfazer os critérios de linhas e colunas:

[7]:
```
print("Determinante != 0: %s"%(np.linalg.det(M) != 0))
print("Elementos na diagonal não nulos: %s"%(not nulldiagonalelem(M)))
Mnew = getLR(M)
print("Critério de linhas: %s"%(linecolcriterium(Mnew)))
print("Critério de colunas: %s"%(linecolcriterium(Mnew.T)))
```

```
Determinante != 0: True
Elementos na diagonal não nulos: True
```

```
Critério de linhas: False
Critério de colunas: True
```

Vemos que não satisfaz o critério das linhas, mas não é uma condição necessária para o método que irei utilizar.

Temos que a matrix é positiva definida

[8]:
```python
print(np.linalg.det(M[:2,:2]))
print(np.linalg.det(M[:3,:3]))
print(np.linalg.det(M[:4,:4]))
```

```
146.0
261.99999999999994
261.99999999999994
```

E, como se trata de uma matrix simétrica e positiva definida, podemos usar o método de Cholesky

[9]:
```python
x = solvecholesky(M, b)
print(x)
```

```
[ 0.04198473 -2.5648855   3.4351145 ]
```

Obtemos a solução:

$$x = \begin{bmatrix} 0.04 \\ -2.56 \\ 3.44 \end{bmatrix} \tag{2}$$