Relatório EP1 - MAP3121 (Métodos Numéricos e Aplicações)

Gabriel Souza Lima - 11820106
Lucas Pereira da Fonseca - 11808126

1 de maio de 2022

1 Introdução

Neste Exercício-Programa, foi implementado o um algortimo para fazer a decomposição LU de uma matriz tridiagonal A n x n. Além disso, também foi criado um algoritmo para a resolução de um sistema tridiagonal usando a decomposição LU da matriz.

Os algoritmos foram testados na resolução do sistema leinar tridiagonal cíclico $Ax = d$, cujos coefiecientes foram descritos no enunciado do Exercício-Progama.

2 Descrição

Abaixo, iremos explicar o funcionamento dos algoritmos utilizados na resolução das tarefas e do teste final.

2.1 Decomposição LU

Realizar a Decomposição LU de uma matriz A é uma forma simples de resolver sistemas linear. Seja A uma matriz triangularizável pelo Método de Eliminação de Gauss, queremos encontrar as matrizes L e U tais que $A = LU$. Assim, podemos calcular os coeficientes de A nxn como:

$$A_{ij} = \sum_{k=1}^{n} L_{ik} U_{kj}$$

No entanto, sabemos que L será uma matriz triangular inferior e U uma matriz triangular superior. Logo, podemos evitar algumas contas que já sabemos que serão nulas. Ao invés de iterar até n, podemos parar no mínimo entre i e j:

$$A_{ij} = \sum_{k=1}^{min(i,j)} L_{ik} Ukj$$

Demonstração: Suponha que $j > i$. Neste caso, sabemos que $L_{i,i+1} = L_{i,i+1} = ... = L_{i,n} = 0$. De forma similar, se $i > j$, sabemos que $U_{0,j} = U_{1,j} = ...U_{j-1,j} = 0$. Percebe-se que a última iteração que irá gerar um coeficiente não nulo é a iteração *min(i,j)*.
Podemos, também, nos aproveitar que $L_{ii} = 1$ e calcular os coeficientes de L e U usando o *Método da Eliminação de Gauss* em uma ordem invertida:

$$U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik}U_{kj}$$
$$L_{ji} = \frac{(A_{ji} - \sum_{k=1}^{i-1} L_{jk}U_{ki})}{U_{ii}}$$

2.2 Matrizes Tridiagonais

Para o caso em que A é tridiagonal, podemos representá-la por suas diagonais não nulas:

- a = $(0, A_{2,1}, A_{3,2}, ..., A_{n,n-1})$
- b = $(A_{1,1}, A_{2,2}, A_{3,2}, ..., A_{n,n})$
- c = $(A_{1,2}, A_{2,3}, A_{3,4}, ..., A_{n-1,n}, 0)$

Dessa forma, podemos ignorar os coeficientes que são nulos. Além disso, iremos representar as matrizes U, L por u, l tal que $u_i = U_{i,i}$ e $l_{i+1} = L_{i+1,i}$. Dessa forma, calculamos os vetores l e u por:

$$l_i = \frac{a_i}{u_{i-1}}$$
$$u_i = b_i - \frac{l_i}{c_{i-1}}$$

Demonstração: Seja A_i a matriz A após i iterações do *Método de Eliminação de Gauss*. Tal matriz A_i continuará sendo tridiagonal, uma vez que os coeficientes nulos das linhas irão zerar as divisões que envolvam os coeficientes $A_i[i, i+k+1]$ ou os coeficientes $A_i[i+k+1, i]$. Sabendo que a matriz continua a ser tridiagonal, é trivial perceber que $l_i = a_i/u_{i-1}$, uma vez que u_{i-1} é pivô da linha $A_i[i-1, 1:n]$ e a_i é o primeiro elemento não-nulo da linha $A_i[i, 1:n]$.
Para encontrar u_i, basta lembrar do *Método de Eliminação de Gauss*. Uma vez que o coeficiente da linha i e coluna $i+k, k \geq 0$, da matriz tridiagonal A_i será definida por

$$A_i[i, i+k] = A_i[i, i+k] - A_i[i, i-1]/A_i[i-1, i-1] \times A_i[i-1, i+k]$$

e sabendo que: $A_i[i,i] = U[i,i] = u_i$, $A_i[i, i-1] = L[i, i-1] = l_i$, $A_i[i-1, i] = c_{i-1}$, substituímos $k = 0$ na expressão do *Método de Eliminação de Gauss* e chegamos que:

$$u_i = b_i - \frac{l_i}{c_{i-1}}$$

3 Funções principais

Agora, vamos ver quais funções foram usadas para resolver as tarefas.

3.1 decompoeTridiagonalLU()

O código abaixo faz a decomposição LU de uma matriz tridiagonal.
Para usar a função, basta passar como argumento os vetores *a, b, c* definidos na função *criaCoeficientes()*

```
def decompoeTridiagonalLU(a,b,c):
    """Faz a decomposicao em LU de uma matriz A nxn tridiagonal, com
    coeficientes a, b, c.
```

```
    Args:
        a, b, c: diagonais nao nulas da matriz tridiagonal A
    Return:
        l: multiplicadores da matriz L
        u: diagonal principal de U
    """
    n = len(a)
    u = np.zeros(n)
    l = np.zeros(n)
    u[0] = b[0]
    for i in range(1,n,1):
        l[i] = a[i]/u[i-1]
        u[i] = b[i]-l[i]*c[i-1]
    return l, u
```

3.2 resolveSlTridiagonal()

O código abaixo faz a decomposição LU de uma matriz tridiagonal. Para isso, foi utilizado o *Algoritmo de Thomas*, o qual é derivado do Método de *Eliminação de Gauss* aplicado para matrizes tridiagonais. A principal vantagem de tal algoritmo é evitar os coeficientes nulos, diminuindo o tempo de execução. Para usar a função, basta passar como parâmetros as diagonais não nulas a,b,c e o vetor d tal que Ax = d.

```
def resolveSlTridiagonal(a,b,c,d):
    """Resolve um sistema linear tridiagonal
    Args:
        a, b, c: diagonais nao nulas da matriz tridiagonal A
        d: vetor tal que Ax = d
    Return:
        x: solucao do sistema linear
    """
    #gerando os vetores que representam L e U
    l, u = decompoeTridiagonalLU(a,b,c)

    n = len(a)
    y = np.zeros(n)
    x = np.zeros(n)

    #Ly = d
    y[0] = d[0]
    for i in range(1,n,1):
        y[i] = d[i] - l[i]*y[i-1]

    #Ux = y
    x[n-1] = y[n-1]/u[n-1]
    for i in range(n-2,-1,-1):
        x[i] = (y[i] - c[i]*x[i+1])/u[i]

    return x
```

3.3 resolveSistemaTridiagonalCiclico()

Essa função encontra a solução de um sistema tridiagonal cíclica. Para isso, serão usadas as duas funções anterioes.
Para usá-la, basta passar como parâmetro os vetores a, b, c, d criados na função *criaCoeficientes()*.

```
def resolveSistemaTridiagonalCiclico(a,b,c,d):
```

```
    """Resolve um sistema tridiagonal ciclico
    Args:
        a, b, c: diagonais nao nulas da matriz tridiagonal A
        d: vetor tal que Ax = d
    Return:
        x: solucao do sistema linear
    """
    n = len(a)

    #v = (a1,0,...,0,cn-1)T
    v = np.zeros(n-1)
    v[0], v[n-2] = a[0], c[n-2]
    v = v.T

    # w = (cn,0,...,0,an)T
    w = np.zeros(n-1)
    w[0], w[n-2] = c[n-1], a[n-1]
    w = w.T

    # d_ = (d1,...,dn-1)T
    d_ = d[:n-1].T

    # T    a submatriz principal de A.
    # Representamos T pelos vetores a_sub=(0,a2,...,an-1), b_sub=(b1,...,bn-1) e c_sub(c
    a_sub = np.concatenate((np.array([0]),a[1:n-1]))
    b_sub = b[:n-1]
    c_sub = np.concatenate((c[:n-2],np.array([0])))

    # Ty_ = d_
    y_ = resolveSlTridiagonal(a_sub, b_sub, c_sub, d_)

    # Tz_ = v
    z_ = resolveSlTridiagonal(a_sub, b_sub, c_sub ,v)

    # Aqui iremos encontrar x, a solu   o de Ax = d
    x = np.zeros(n).T
    x[n-1] = (d[n-1]-c[n-1]*y_[0]-a[n-1]*y_[n-2])/(b[n-1]-c[n-1]*z_[0]-a[n-1]*z_[n-2])
    x[:n-1] = y_ - x[n-1]*z_

    return x
```

4 Funções secundárias

Agora, iremos falar sobre as funções secundárias, que ajudaram a avaliar os resultados das funções principais.

4.1 erroSlTridiagonal()

Usaremos essa função para verificar o erro na solução gerada pela função *resolveSlTridiagonal()*. Para usá-la, basta passar os vetores a,b,c,d e o vetor solução x.

```
def erroSlTridiagonal(a,b,c,d,x):
    """Encontra o maior erro na solucao de um sistema tridiagonal
    Args:
        a, b, c: diagonais nao nulas da matriz tridiagonal A
        x: solucao do sistema linear
```

```
        d: vetor tal que Ax = d
    Return:
        maior_erro: maior erro da solucao
    """
    n = len(a)
    erro = np.zeros(n)
    erro[0] = d[0] - (b[0]*x[0]+c[0]*x[1])
    for i in range(1,n-1):
        erro[i] = d[i] - (a[i]*x[i-1]+b[i]*x[i]+c[i]*x[i+1])
    erro[n-1] = d[n-1] - (a[n-1]*x[n-2]+b[n-1]*x[n-1])
    maior_erro = np.max(np.abs(erro))
    return maior_erro
```

4.2 erroSlTridiagonalCiclico()

Usaremos essa função para verificar o erro na solução gerada pela função *resolveSlTridiagonal()*.
Para usá-la, basta passar como argumento os vetores a,b,c,d gerados pela função *criaCoeficientes()* e o vetor solução x gerado pela função *resolveSistemaTridiagonalCiclico()*.

```
def erroSlTridiagonalCiclico(a,b,c,d,x):
    """Encontra o maior erro na solucao de um sistema tridiagonal ciclico
    Args:
        a, b, c: diagonais nao nulas da matriz tridiagonal A
        x: solucao do sistema linear
        d: vetor tal que Ax = d
    Return:
        maior_erro: maior erro da solucao
    """
    n = len(a)
    erro = np.zeros(n)
    erro[0] = d[0] - (b[0]*x[0]+c[0]*x[1]+a[0]*x[n-1])
    for i in range(1,n-1):
        erro[i] = d[i] - (a[i]*x[i-1]+b[i]*x[i]+c[i]*x[i+1])
    erro[n-1] = d[n-1] - (c[n-1]*x[0]+a[n-1]*x[n-2]+b[n-1]*x[n-1])
    maior_erro = np.max(np.abs(erro))
    return maior_erro
```

4.3 calculaResiduoParaGrafico()

Essa função será usada somente para plotar o gráfico. Ela será vetorizada posteriormente, o que diminuirá o tempo de execução.
Para usá-la, basta passar como argumento o tamanho n do vetor.

```
def calculaResiduoGrafico(n):
    """Calcula o residuo maximo de uma solucao. Essa funcao sera usada para
    plotar o grafico
    Args:
        n: tamanho do vetor
    Return:
        erro_max: residuo maximo
    """
    a,b,c,d = criaCoeficientes(n)
    x = resolveSistemaTridiagonalCiclico(a,b,c,d)
    erro = np.zeros(n)
    erro[0] = d[0] - (b[0]*x[0]+c[0]*x[1]+a[0]*x[n-1])
    for i in range(1,n-1):
        erro[i] = d[i] - (a[i]*x[i-1]+b[i]*x[i]+c[i]*x[i+1])
```

```
    erro[n-1] = d[n-1] - (c[n-1]*x[0]+a[n-1]*x[n-2]+b[n-1]*x[n-1])
    erro_max = np.max(np.abs(erro))
    return erro_max
```

4.4 criaVetoresTridiagonais

Essa função gera vetores a,b,c,d tais que:

- a = (0,-1,-1,...,-1)
- b = (2,2,2,...,2)
- c = (-1,-1,...,-1,0)
- d = (1,1,1,...,1)

Ela será usada para avaliar os erros da função *resolveSlTridiagonal()*.
Para usá-la, basta passar como argumento o tamanho n dos vetores.

```
def criaVetoresTridiagonais(n):
    """Cria os coeficientes da matriz tridiagonal e o vetor d, para testar a
    funcao resolveSlTridiagonal()
    Args:
        n: tamanho dos vetores
    Return:
        a, b, c e d
    """
    a = np.repeat(-1,n)
    b = np.repeat(2,n)
    c = np.copy(a)
    a[0] = 0
    c[n-1] = 0
    d = np.repeat(1,n)
    return a,b,c,d
```

4.5 criaCoeficientes()

Essa função é responsável por criar os coeficientes dos vetores a, b e c da matriz A e vetor d, do lado direito da equação. Os vetores criados são os mesmos descritos no enunciado do EP.

- $a_i = \frac{2i-1}{4i}, 1 \leq i \leq n-1, a_n = \frac{2n-1}{2n}$
- $c_i = 1 - a_i, 1 \leq i \leq n$
- $b_i = 2, 1 \leq i \leq n,$
- $d_i = cos(\frac{2\pi i^2}{n^2})$

Para usá-la, basta passar como parâmetro o tamanho n dos vetores.

```
def criaCoeficientes(n):
    """Cria os coeficientes da matriz tridiagonal ciclica e o vetor d
    Args:
        n: tamanho dos vetores
    Return:
        a, b, c e d"""
    a = (2*np.arange(1,n+1)-1)/(4*np.arange(1,n+1))
    a[n-1] = (2*n-1)/(2*n)
    c = 1 - a
    b = np.repeat(2,n)
    d = np.array([math.cos(i) for i in 2*math.pi*(np.arange(1,n+1)**2)/(n**2)])
    return a,b,c,d
```

5 Tarefas

Vamos ver agora quais foram os resultados obtidos ao rodarmos os algoritmos.

5.1 Primeira Tarefa

Essa parte se tratava de fazer a decomposição LU de uma matriz tridiagonal A nxn.
Para n = 20, obtivemos os seguintes vetores l e u:

- Vetor l: [0. 0.1875 0.22408964 0.23522214 0.2415735 0.24587182 0.2489774 0.25132199 0.25315296 0.25462172 0.25582577 0.2568306 0.25768181 0.25841208 0.25904544 0.25959998 0.26008953 0.26052488 0.26091455 0.52253075]
- Vetor u: [2. 1.859375 1.85994398 1.86278709 1.86411491 1.8647705 1.86513724 1.86536322 1.86551249 1.86561631 1.86569147 1.86574764 1.86579072 1.8658245 1.86585147 1.86587334 1.86589134 1.86590631 1.86591891 1.73185922]

5.2 Segunda Tarefa

Na segunda tarefa, implementamos um algoritmo que resolve um sistema linear tridiagonal usando a decomposição LU.
Para testar esse algoritmo, criamos os vetores a,b,c,d tais que:

- a = (0,-1,-1,...,-1)
- b = (2,2,2,...,2)
- c = (-1,-1,...,-1,0)
- d = (1,1,1,...,1)

Usando as funções *resolveSlTridiagonal()* e *erroSlTridiagonal()* para n=20, encontramos que:

- Solução = [10. 19. 27. 34. 40. 45. 49. 52. 54. 55. 55. 54. 52. 49. 45. 40. 34. 27. 19. 10.]
- Resíduo máximo: 1.4210854715202004e-14

Podemos ver os resíduos máximos para diferentes valores de n na tabela abaixo:

n	Resíduo Máximo	Duração (s)
10	3.552713678800501e-15	0.0
100	2.2737367544323206e-13	0.009158611297607422
1000	2.9103830456733704e-11	0.04319596290588379
10000	3.725290298461914e-09	0.4311087131500244
100000	4.76837158203125e-07	4.702253103256226

5.3 Resolução Sistema Tridiagonal Cíclico

Finalmente, testamos os algortimos anteriores na resolução de um sistema tridiagonal cíclico $Ax = d$, onde os coeficientes são:

$$a_i = \frac{2i-1}{4i}, 1 \leq i \leq n-1, a_n = \frac{2n-1}{2n},$$
$$c_i = 1 - a_i, 1 \leq i \leq n,$$
$$b_i = 2, 1 \leq i \leq n,$$
$$d_i = cos\left(\frac{2\pi i^2}{n^2}\right)$$

Usando as funções *resolveSistemaTridiagonalCiclico()* e *erroSlTridiagonalCiclico* com n=20:

- Solução: [0.33031512 0.33369784 0.33082061 0.32458573 0.3105381 0.28498139 0.24375728 0.18349137 0.10274415 0.00360629 -0.10669724 -0.2147279 -0.30113746 -0.34330813 -0.32097501 -0.22451082 -0.0638644 0.12580676 0.28713644 0.35589205]
- Resíduo máximo: 1.1102230246251565e-16
- Duração: 0.0009970664978027344 s

Vamos observar qual foi o desempenho do algoritmo para alguns valores de n na tabela abaixo:

n	Resíduo Máximo	Duração (s)
10	1.1102230246251565e-16	0.0
100	2.2.220446049250313e-16	0.002995014190673828
1000	2.220446049250313e-16	0.034737348556518555
10000	2.220446049250313e-16	0.3099980354309082
100000	2.220446049250313e-16	3.409693717956543

Podemos, também, visualizar como o algoritmo performa para diferentes valores de n no gráfico abaixo:

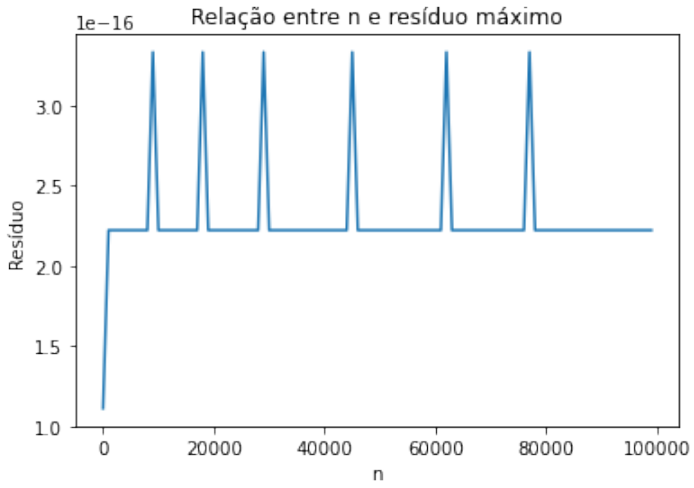

Figura 1: Resíduos máximos

Podemos ver que os resíduos máximos ficaram na ordem de 10^{-16}, indicando grande precisão.

6 Conclusões

Pudemos aplicar neste Exercício-Programa técnicas para decompor matrizes tridiagonais, as quais foram usadas para resolver tanto sistemas tridiagonais quanto sistemas tridiagonais cíclicos.
Foi observada grande eficiência na resolução, devido ao pequeno tempo de duração do código mesmo para vetores com mais de 100000 coeficientes, além de obter precisões boas para a resolução dos sistemas.
Os algoritmos criados aqui serão úteis em trabalhos futuros, nos quais poderemos aplicar novas técnicas de Cálculo Numérico para resolver problemas ainda mais complexos.