EP1- Decomposição LU para matrizes tridiagonais

MAP3121- Métodos Numéricos e Aplicações

Arthur Pedroso Porto Belli - 11804608
Letícia Harumi Furusawa - 11965585

1 de maio de 2022

1 Introdução

Este relatório descreve os resultados obtidos no exercício programa, no qual estudamos decomposição LU de matrizes e resolução de sistemas lineares para matrizes tridiagonais e cíclicas, um caso particular da solução de sistemas lineares de equações que é recorrente na resolução de problemas de engenharia.

O grupo optou pela implementação em Python 3.9.7 e utilizou as bibliotecas Numpy 1.21.1 e math (nativa).

Como escolha de projeto, o grupo decidiu que o *input* de dados será feito exclusivamente pela leitura de arquivos externos que seguem a seguinte formatação:

- Linha 1: Valores da diagonal a: $[a_1, a_n]$;
- Linha 2: Valores da diagonal b: $[b_1, b_n]$;
- Linha 3: Valores da diagonal c: $[c_1, c_n]$;
- Linha 4: Valores do vetor d: $[d_1, d_n]$.

2 Funções utilizadas

2.1 LU_to_A (Exercício 1)

Realiza a operação de multiplicação das matrizes L e U Recebe como parâmetros somente as matrizes L e U e retorna a matriz A, resultante da multiplicação das matrizes de entrada.

```
def LU_to_A(L, U):
    dim = L.shape[0]
    A = np.zeros((dim, dim), dtype=np.double) #matriz A tem mesma dimens o que as de
                                                entrada, mas inicializada com zeros
    for i in range(dim): #iterando as linhas
        for j in range(dim): #iterando as colunas
            for k in range(min(i, j)+1): #iterando na regra de forma   o
                A[i][j] += L[i][k]*U[k][j]
    return A
```

2.2 A_to_LU (Exercício 2)

Realiza a operação inversa da função (2.1).

```
def A_to_LU(A):
  dim = A.shape[0]
    L = np.eye(dim, dtype=np.double) #matriz identidade
    U = np.zeros((dim, dim), dtype=np.double)

    for i in range(dim):
        U[i, i:] = A[i, i:] - np.dot(L[i, :i], U[:i, i:])
        L[(i+1):, i] = (A[(i+1):, i] - np.dot(L[(i+1):, :], U[:, i]))/U[i,i]
    return L, U
```

2.3 A_to_LU_tridig

Versão otimizada da função (2.3) para matrizes tridiagonais.
Recebe como parâmetros os vetores a, b, c que armazenam as diagonais da matriz tridiagonal e retorna os vetores U$_{ii}$, $U_{i+1}e$ L_{i+1}.

```
def A_to_LU_tridig(a, b, c): #recebe as diagonais
    dim = b.shape[0] #n mero de elementos na diagonal principal    a dimmens o da
                                                            matriz
    l_ip1 = np.zeros(dim, dtype=np.double) #l_i+1
    u_ii = np.zeros(dim, dtype=np.double) #u_ii
    u_ip1 = c.copy() #u_i+1

    u_ii[0] = b[0]
    for i in range(1, dim):
        l_ip1[i] = a[i]/u_ii[i-1] #multiplicadores
        u_ii[i] = b[i]-l_ip1[i]*c[i-1]

    return u_ii, u_ip1, l_ip1
```

2.4 solve_lin_sys_tridiag

Resolve sistemas de equações lineares ridiagonais usando a decomposição LU. Recebe como parâmetros os vetores U_{ii}, $U_{i+1}e$ L_{i+1} (valores em forma vetorial armazenamento otimizado) e retorna os valores que resolvem o sistema no vetor x.

```
def solve_lin_sys_tridiag(u, u_ip1, l, d):
    #Ly = d
    dim = u.shape[0]
    y = np.zeros((dim, 1), dtype=np.double) #vetor coluna
    y[0][0] = d[0]

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

    #Ux = y
    x = np.zeros((dim, 1), dtype=np.double) #vetor coluna
    x[dim-1] = y[dim-1][0]/u[dim-1]

    for i in range(dim-1, -1, -1):
        x[i-1] = (y[i-1][0]-u_ip1[i-1]*x[i])/u[i-1]

    return x
```

2.5 A_to_abc

Retorna as diagonais $a,\ b,\ c$ da matriz A.

```
#retorna as diagonais em forma de vetor de forma mais generalizada, funciona tanto
                                            para tridiagonal, como tridiagonal ciclica
def A_to_abc(A):
    dim = A.shape[0]
    a = np.zeros(dim, dtype=np.double)
    b = np.zeros(dim, dtype=np.double)
    c = np.zeros(dim, dtype=np.double)

    for i in range(dim):
        for j in range(dim):
            if i == j:
                b[i] = A[i][j]
            elif i - j == 1 or (i == 0 and j == dim-1):
                a[i] = A[i][j]
            elif i - j == -1 or (i == dim-1 and j == 0):
                c[i] = A[i][j]
    return a, b, c
```

2.6 abc_to_A

Realiza a operação inversa da função (2.5)

```
def abc_to_A(a, b, c):
    dim = b.shape[0]
    A = np.zeros((dim, dim), dtype=np.double)
    for i in range(dim):
        for j in range(dim):
            if i == j:
                A[i][j] = b[i]
            elif i - j == 1 or (i == 0 and j == dim-1):
                A[i][j] = a[i]
            elif i - j == -1 or (i == dim-1 and j == 0):
                A[i][j] = c[i]
    return A
```

2.7 get_T

Recebe como parâmetro a matriz tridiagonal A e retorna T, submatriz diagonal principal de A.

```
#retorna a submatriz diagonal principal de A
def get_T(A):
    dim = A.shape[0]-1 #submatriz diagonal principal
    T = np.zeros((dim, dim), dtype=np.double)
    for i in range(dim):
        for j in range(dim):
            T[i][j] = A[i][j]
    return T
```

2.8 get_w_v

Recebe como parâmetros a matriz tridiagonal A e a string ”option”que determina qual vetor a função retornará. No final, retorna o vetor escolhido (v ou w) na forma transposta.

```
#a depender do valor de option, retorna o vetor v, ou o vetor w da matrix A
def get_w_v(A, option):
    dim = A.shape[0]-1
    answer = np.zeros(dim, dtype=np.double)
    if option == "v":
        for i in range(dim):
            answer[i] = A[i][dim]
    else:
        for i in range(dim):
            answer[i] = A[dim][i]
    return answer.T #retornamos transposto pois desejamos que seja um vetor coluna
```

2.9 solve_lin_sys_tridig_cyclic

Resolve sistema linear descrito por matriz tridiagonal cíclica usando o algoritmo da decomposição LU, como descrito no enunciado.

```
#recebe as diagonais da matriz A e o vetor d, retorna os valores que resolvem Ax = d
def solve_lin_sys_tridig_cyclic(d, a, b, c):
    A = abc_to_A(a, b, c) #iremos construir a matriz A com suas diagonais c clicas
    dim = A.shape[0]

    x = np.zeros((dim, 1), dtype=np.double)
    x_barr = np.zeros(dim, dtype=np.double) #~x

    T = get_T(A)
    v = get_w_v(A, "v")
    w = get_w_v(A, "w")

    #T*~y = ~d = d[0:-1]
    y_barr = solve_lin_sys_tridiag(*A_to_LU_tridig(*A_to_abc(T)), d[0:-1]) #pegamos os
                                                      valores de d at   o pen ltimo
    #T*~z = v
    z_barr = solve_lin_sys_tridiag(*A_to_LU_tridig(*A_to_abc(T)), v)


    x_n = (d[-1] - c[-1]*y_barr[0]-a[-1]*y_barr[-1])/(b[-1]-c[-1]*z_barr[0]-a[-1]*
                                                     z_barr[-1])
    x_barr = y_barr - x_n*z_barr

    x = np.append(x_barr, np.array(x_n))
    return x
```

3 Resultados

O enunciado pede a resolução do sistema $Ax = d$, onde os coeficientes da matriz A são:

$$a_i = \frac{2i-1}{4i},\ 1 \leq i \leq n-1,\ a_n = \frac{2n-1}{2n},$$

$$c_i = 1 - a_i,\ 1 \leq i \leq n,$$

$$b_i = 2,\ 1 \leq i \leq n;$$

e o lado direito do sistema linear é dado por

$$d_i = \cos\left(\frac{2\pi i^2}{n^2}\right),\ 1 \leq i \leq n.$$

Usando n=20 e montando esse sistema $20x20$ temos o seguinte vetor resposta com 4 casas decimais:[1]

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{20} \end{pmatrix} = \begin{pmatrix} 0.3644 \\ 0.3175 \\ 0.2424 \\ 0.0234 \\ -0.2865 \\ -0.3978 \\ 0.0190 \\ 0.5041 \\ -0.0332 \\ -0.5901 \\ 0.5589 \\ -0.0472 \\ -0.4770 \\ 0.7951 \\ -0.9274 \\ 0.9554 \\ -0.9124 \\ 0.7615 \\ -0.4537 \\ 0.1131 \end{pmatrix}$$

Os valores acima foram conferidos com calculadoras online e outros testes foram realizados com matrizes menores, que confirmaram a veracidade dos resultados obtidos.

4 Conclusões

O presente exercício-programa implementado pelo grupo atingiu os objetivos propostos, e foi uma ótima oportunidade de aprendizado e de colocar em prática os conhecimentos adquiridos em sala de aula.

[1] Valores mais precisos podem ser encontrados no apêndice.

5 Apêndice

Os valores exatos do sistema proposto pelo enunciado são:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{20} \end{pmatrix} = \begin{pmatrix} 0.36437899819211517 \\ 0.3174607509401019 \\ 0.24242554798306953 \\ 0.02337147051527826 \\ -0.2864789314799596 \\ -0.3977658016044437 \\ 0.019012694661708674 \\ 0.5040996917791497 \\ -0.03325746395681847 \\ -0.5901523387499714 \\ 0.5588887827549721 \\ -0.04716297755636626 \\ -0.4770582629284569 \\ 0.7951538372368097 \\ -0.9274109191353269 \\ 0.9553928058487235 \\ -0.9124105014581908 \\ 0.7615446461408618 \\ -0.45369315861629994 \\ 0.11311975252039647 \end{pmatrix}$$