Trabalho final

SME0892 - Cálculo Numérico para Estatística

Universidade de São Paulo (USP)

Gabriel Bueno Barbosa NUSP: 11371263
Tiago Chaves Bezerra Rocha NUSP: 14609637
Yago Augusto Bardelotte NUSP: 11320724

1 - Implementação

Para resolver o sistema resultante $Ax = b$, procedemos da seguinte maneira: decompomos a matriz em duas matrizes triangulares L e U ,de forma que $A = LU$. L é uma matriz triangular inferior e U é uma matriz triangular superior. Com a fatoração pronta,basta resolver os dois sistemas triangulares resultantes: $Ly = b$ e $Ux = y$.

Para fazer a fatoração LU e resolver os sistemas triangulares resultantes,utilizamos duas rotinas diferentes: uma delas se beneficia da estrutura de bandas da matriz e a outra não se beneficia.A implementação por bandas prevê as posições da matriz que contém valores nulos e evita cálculos desnecessários.

Os códigos comentados das implementações estão listados abaixo:

```python
import numpy as np
import time
import matplotlib.pyplot as plt
import seaborn as sns

def modelo ( n, w, T, L = 1.2e2 , E = 2.9e7 , I = 1.21e2 ):
    """
    Constroi a matriz e o vetor resultado de um sistema linear.

    Parametros:
        - n: fator de discretizacao.
        - w: carga aplicada na viga.
        - T: tracao.

    Retorna:
        - A: Matriz do sistema linear.
        - f: Vetor com o resultado das equacoes do sistema linear.
    """
    a = T / ( E * I )
    b = w / ( 2 * E * I )
    h = L / ( n - 1 )

    A = np.zeros( ( n - 2, n - 2 ) )
    v = -( h**2 * a + 2 )

    A[ range( 0, n - 3 ), range( 1, n - 2 ) ] = 1.0
    A[ range( 0, n - 2 ), range( 0, n - 2 ) ] = v
    A[ range( 1, n - 2 ), range( 0, n - 3 ) ] = 1.0

    X = np.linspace( h, ( n - 2 ) * h, n - 2 )

    bh_3 = b * ( h ** 3 )
    f = np.linspace( bh_3 , ( n - 2 ) * bh_3 , n - 2 ) * ( L - X )
    return ( A, f )
```

Listing 1: Função para gerar as matrizes

```python
def LU(A):
    """
    Realiza a fatoracao LU de uma matriz A.

    Parametros:
```

```
        - A: Matriz de entrada (numpy.array).

    Retorna:
        - L: Matriz triangular inferior resultante da fatoracao LU.
        - U: Matriz triangular superior resultante da fatoracao LU.
    """
    n = len(A)
    L = np.eye(n)  # Matriz identidade inicial para L
    U = np.copy(A)  # Copia de A para U

    for k in range(n-1):
        for i in range(k+1, n):
            # Colculo do multiplicador
            L[i, k] = U[i, k] / U[k, k]

            # Atualizacao da matriz U
            U[i, k:] -= L[i, k] * U[k, k:]

    return ( L, U )

def solve_lower_triangular(L, b):
    """
    Resolve um sistema linear com uma matriz triangular inferior.

    Parametros:
        - L: Matriz triangular inferior (numpy.array).
        - b: Vetor de solucoes (numpy.array).

    Retorna:
        - x: Vetor solucao do sistema linear.
    """
    n = len(b)
    x = np.zeros(n)

    for i in range(n):
        x[i] = (b[i] - np.dot(L[i, :i], x[:i])) / L[i, i]

    return ( x )


def solve_upper_triangular(U, b):
    """
    Resolve um sistema linear com uma matriz triangular superior.

    Parametros:
        - U: Matriz triangular superior (numpy.array).
        - b: Vetor de solucoes (numpy.array).

    Retorna:
        - x: Vetor solucao do sistema linear.
    """
    n = len(b)
    x = np.zeros(n)

    for i in range(n-1, -1, -1):
        x[i] = (b[i] - np.dot(U[i, i+1:], x[i+1:])) / U[i, i]

```

```
    return ( x )

def resol_base_tri (A, b):
    """
    Resolve um sistema de equacoes lineares triangular superior com
     matriz banda usando fatoracao PA=LU.

    Parametros:
        A: Matriz de coeficientes do sistema (matriz banda superior
    )
        b: Vetor de termos independentes

    Retorna:
        x: Solucao do sistema
    """
    n = len(b)

    L, U = LU(A)
    y = solve_lower_triangular(L, b)
    x = solve_upper_triangular(U, y)

    return ( x )
```

Listing 2: Funções para resolver as matrizes sem utilizar estrutura por bandas

```
def LU_banda ( A, p, n):
    """
    Realiza a fatoracao LU por bandas de uma matriz A.

    Parametros:
        - A: Matriz de entrada (numpy.array).

    Retorna:
        - L: Matriz triangular inferior resultante da fatoracao LU.
        - U: Matriz triangular superior resultante da fatoracao LU.
    """
    U = np.array(A)
    L = np.eye( n )
    for j in range( n - 1 ):
        v = min( n, j + p + 1 )
        for i in range( j + 1, v ):
            L[ i, j ] = U[ i, j ] / U[ j, j ]
            U[ i, j : v ] = U[ i, j : v ] - L[ i, j ] * U[ j, j : v
     ]
    return ( L, U )

def solve_lower_band(L, b, n):
    """
    Resolve um sistema de equacoes lineares triangular inferior com
     matriz banda.

    Parametros:
        L: Matriz de coeficientes do sistema (matriz banda inferior
    )
        b: Vetor de termos independentes
        n: Discretizacao

    Retorna:
```

```
        x: Solucao do sistema
    """
    y = [0] * (n - 2)

    for i in range (n - 2):
        if i != 0:
            y[i] = (b[i] - L[i, i - 1] * y[i - 1]) / L[i, i]
        else:
            y[i] = b[i] / L[i, i]

    return ( y )

def solve_upper_band(U, y, n):
    """
    Resolve um sistema de equacoes lineares triangular superior com
     matriz banda.

    Parametros:
        U: Matriz de coeficientes do sistema (matriz banda superior
    )
        y: Vetor de termos independentes
        n: Discretizacao

    Retorna:
        x: Solucao do sistema
    """
    x = [0] * (n - 2)

    for i in range (n - 3, -1, -1):
        if i != n - 3:
            x[i] = (y[i] - U[i, i + 1] * x[i + 1]) / U[i, i]
        else:
            x[i] = (y[i]) / U[i, i]

    return ( x )

def resol_base_band (A, b, n, p):
    """
    Resolve um sistema de equacoes lineares triangular superior com
     matriz banda usando fatoracao PA=LU.

    Parametros:
        A: Matriz de coeficientes do sistema (matriz banda superior
    )
        b: Vetor de termos independentes
        n: Discretizacao
        p: largura de banda

    Retorna:
        x: Solucao do sistema
    """
    L, U = LU_banda(A, p, n-2)
    y = solve_lower_band(L, b, n)
    x = solve_upper_band(U, y, n)

```

```
    return ( x )
```

Listing 3: Funções para resolver as matrizes utilizando estrutura por bandas

2 - Resultados

A partir dos nossos resultados e através da análise da complexidade computacional dos algoritmos esperamos que o método de matriz com bandas seja mais rápido, já que a eliminação Gaussiana tradicional atrelado à fatoração $A = LU$ produz um algoritmo com complexidade $O(n^3)$, enquanto o método de matriz com bandas produz um algoritmo com $O(4n)$ para obtenção das Matrizes L e U, e $O(5n)$ para realizar operações, resultando em uma complexidade menor em relação a eliminação Gaussiana tradicional, pois possui mais zeros do que uma matriz triangular superior ou inferior e, sinalizando esses zeros no algoritmo, realizamos menos operações para obter a solução do sistema linear.

2.1 - Tempo de Execução

Usando W = 500 e T = 150 fixos e variando n é evidente que o tempo de execução com bandas é mais rápido que o tempo de execução sem bandas já que o algoritmo com bandas possui complexidade computacional menor comparado ao algoritmo sem bandas.

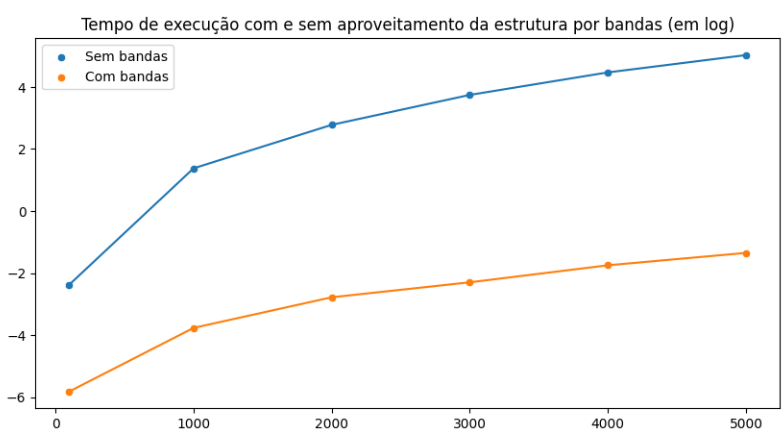

Figura 1: Variando n e mantendo T e w fixos

Com n = 3000 e T = 150 e variando a carga w obtemos o mesmo padrão.Aqui aproveitamos o fato de a tração ser fixa e a fatoração LU não mudar de um caso pra outro. Na primeira iteração temos a realização da fatoração $A = LU$ e percebe-se que, o não aproveitamento das banda aumenta muito o tempo de execução do programa. No entanto, ao se aproveitar da fatoração realizada, o tempo de execução torna-se muito menor em ambos os casos. Nota-se que a fatoração $A = LU$ por bandas tem um custo de tempo menor do que as iterações que não se aproveitam dessa estrutura, nota-se também que o algoritmo para a resolução dos sistemas triangulares resultantes com matrizes banda é mais eficiente do que o algoritmo que não prevê a estrutura de bandas.

As figuras 1 e 2 deixam claro o quanto é eficiente trabalhar um sistema de forma apropriada.

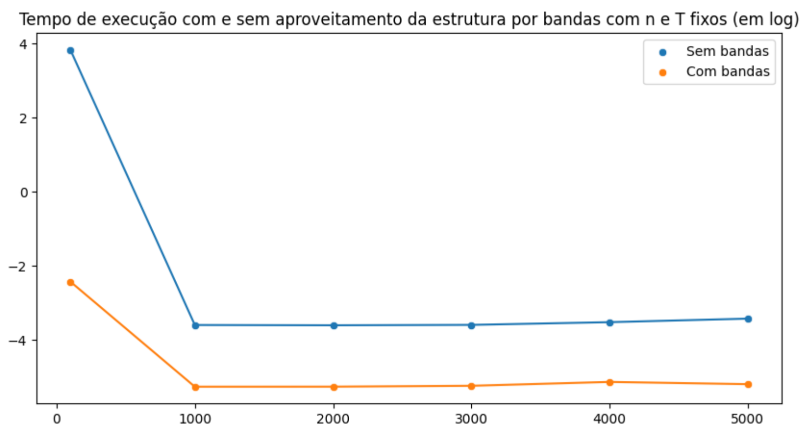

Figura 2: Variando w e mantendo T e n fixos

2.2 - Deformação da Viga

Observa-se na figura 3 que a carga tem impacto diretamente proporcional na deformação da viga, independente da tração.

O que é perceptível de diferença é o ponto de início da deformação, como pode ser visto nas tabelas 1 e 2 e também em ambas as figuras. Para $n = 500$, as pontas das vigas estão muito mais próximas de 0 em todas as cargas. No entanto, o ponto de maior deformação são os mesmos para ambas as discretizações e cargas.

Ao avaliar o impacto da tração, é possível observar que as diferenças estão a partir da terceira casa decimal. Neste caso não há um grande impacto na deformação ao alterar o valor da tração.

Podemos notar que em n = 50 as extremidades das curvas da tração são diferentes, e quando n = 500 as extremidades das curvas da tração são iguais, isso acontece porque quanto maior a variável de discretização n maior a precisão da aproximação por diferenças finitas, e neste caso essa diferença sútil não afeta nossas conclusões e interpretação sobre o problema, mas em um cenário real onde não sabemos tudo sobre o problema em que estamos estudando uma aproximação mais grosseira poderia gerar uma interpretação diferente e afetar nossas conclusões e com isso vemos mais uma aplicação de métodos que são computacionalmente mais eficientes, uma precisão maior no momento de interpretar o problema que estamos estudando, além da maior eficiência computacional.

Tabela 1: Pontos de maior e menor deformação de uma viga sob diferentes cargas e graus de discretização para T = 100

Discretização	Carga	Ponto Deformação	
		Mínimo	Máximo
50	100	-0.005	-0.0769
	1000	-0.0502	-0.7693
	2000	-0.1005	-1.5386
	3000	-0.1507	-2.3079
	4000	-0.2009	-3.0772
	5000	-0.2511	-3.8464
500	100	-0.0005	-0.0769
	1000	-0.0049	-0.7694
	2000	-0.0099	-1.5388
	3000	-0.0148	-2.3082
	4000	-0.0197	-3.0777
	5000	-0.0247	-3.8471

Tabela 2: Pontos de maior e menor deformação de uma viga sob diferentes cargas e graus de discretização para T = 1000

Discretização	Carga	Ponto Deformação	
		Mínimo	Máximo
50	100	-0.005	-0.0769
	1000	-0.0502	-0.769
	2000	-0.1004	-1.538
	3000	-0.1506	-2.307
	4000	-0.2008	-3.076
	5000	-0.251	-3.845
500	100	-0.0005	-0.0769
	1000	-0.0049	-0.7691
	2000	-0.0099	-1.5383
	3000	-0.0148	-2.3074
	4000	-0.0197	-3.0765
	5000	-0.0247	-3.8456

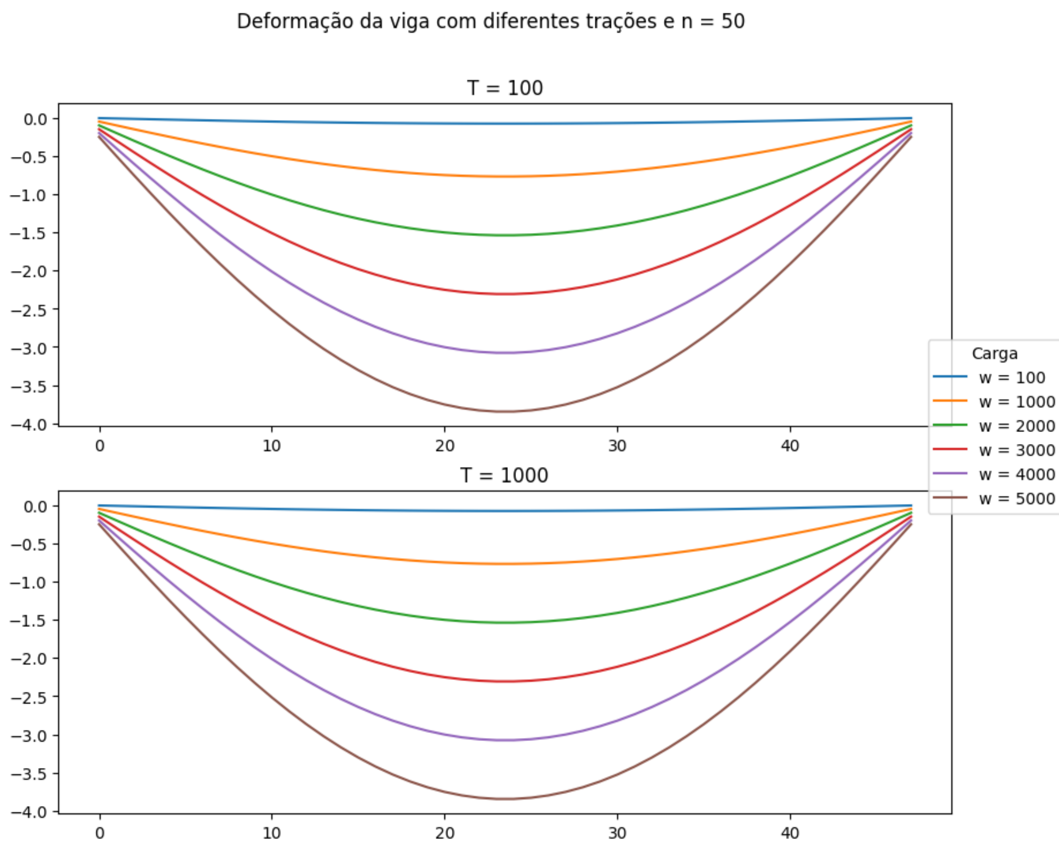

Figura 3: Deformação da viga para diferentes trações e cargas, fixado discretização n = 50

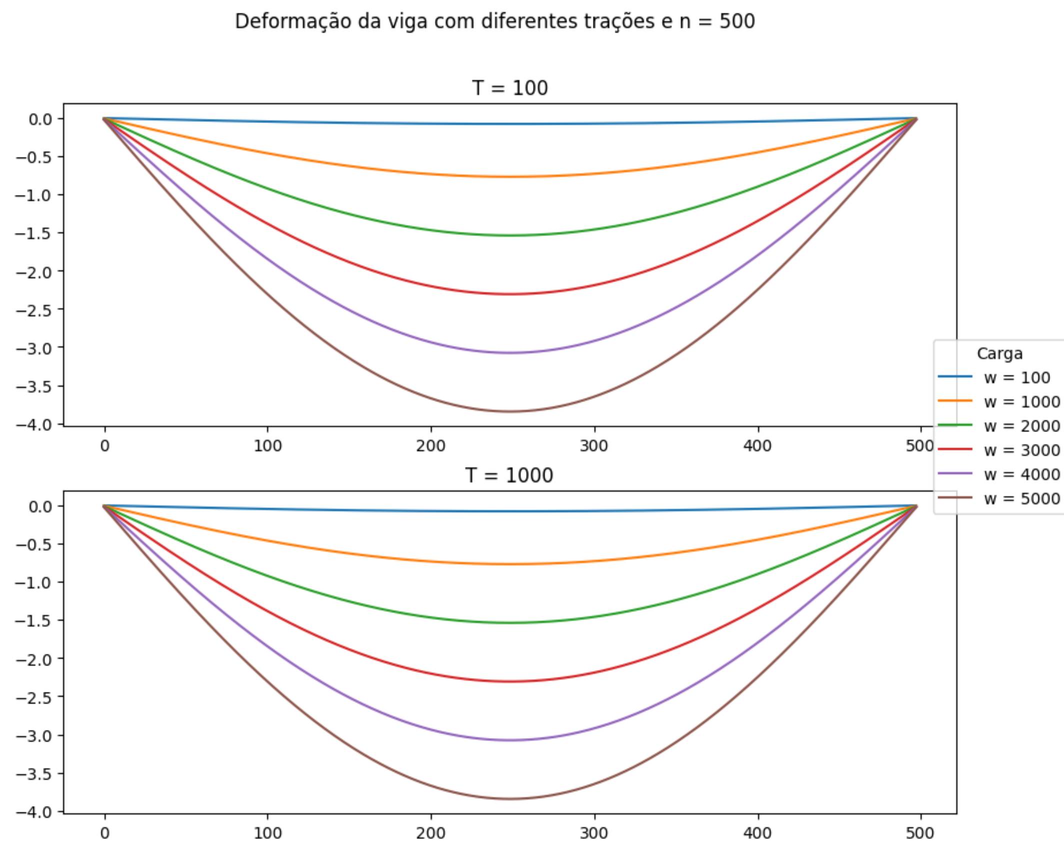

Figura 4: Deformação da viga para diferentes trações e cargas, fixado discretização n = 500