Trabalho Prático 2 Cálculo Numérico

Alunos:

Tarik Salles Paiva - 5059

Manuel Ferreira Ribeiro di Simões - 5091

Ítallo Winícios Ferreira Cardoso - 5101

Renan Grassi de Freitas Procópio - 3987

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sym
import math
import seaborn as sns
import sys
from copy import copy
```

0.0.1 Implementação da função 1, Interpolação por Lagrange:

[ ]:
```
def interpolacao_lagrange(X, Y, x):

    valor_interpolado = 0.0

    for i in range(len(X)):
        L_i = 1.0
        for j in range(len(X)):
            if i != j:  #Não é possível i = j, pois resulta em divisão zero em
 ↪(X[i] - X[j])

              # Calcula a parte do polinômio de Lagrange para o ponto i
                L_i *= (x - X[j]) / (X[i] - X[j])

        valor_interpolado += L_i * Y[i]  #+= L(i) * f(i)

    return valor_interpolado

X = np.array([0, 1, 2,4], dtype=float)
```

```python
Y = np.array([1, 3,5,7], dtype=float)
x = 2.3

valor_interpolado = interpolacao_lagrange(X, Y, x)
print(f"O valor interpolado em x = {x} é: {valor_interpolado}")
```

```
O valor interpolado em x = 2.3 é: 5.525249999999999
```

0.0.2 Implementação da função 2, Interpolação por Diferenças Finitas:

[ ]:
```python
def interpolacao_diferencas_finitas(X, Y, x):
    tabela_diferencas_finitas = np.zeros((len(X), len(X)))
    tabela_diferencas_finitas[:, 0] = Y


    for j in range(1, len(X)):
        for i in range(len(X) - j):
            tabela_diferencas_finitas[i, j] = tabela_diferencas_finitas[i + 1,␣
↪j - 1] - tabela_diferencas_finitas[i, j - 1] #diferença finita de ordem j no␣
↪índice i = (diferença finita de ordem j-1 no índice i+ 1) + (diferença␣
↪finita de ordem j-1 no índice i)

        print(f"Tabela de Diferenças Finitas após preencher a coluna {j}:")

        # Impressão da tabela de diferenças finitas com alguns espaços para␣
↪melhorar a visualização, não envolvendo nenhum cálculo
      ␣
↪#-----------------------------------------------------------------------------------------------------------

        nome_colunas = ['x', 'y'] + [f'Δ^{i}y' for i in range(1, j + 1)]
        comp_coluna = max(len(name) for name in nome_colunas) + 5

        header = ''.join(f"{name:>{comp_coluna}}" for name in nome_colunas)
        print(header)
        print('-' * len(header))

        for i in range(len(X)):
            linha = f"{X[i]:>{comp_coluna}.2f} {tabela_diferencas_finitas[i, 0]:
↪>{comp_coluna}.4f}"
            for k in range(1, j + 1):
                if i + k < len(X):
                    linha += f"{tabela_diferencas_finitas[i, k]:>{comp_coluna}.
↪4f}"
                else:
                    linha += ' ' * comp_coluna
            print(linha)
        print()
```

```
␣
↪#----------------------------------------------------------------------------------------------




    # Nesse caso o valor interpolado é calculado usando o polinômio de Newton
    valor_interpolado = tabela_diferencas_finitas[0, 0]
    polinomio = 1.0
    delta_x = X[1] - X[0]  # Diferença inicial (x1 - x0)
    x_diferenca_x0 = (x - X[0])  # Diferença (x - x0)
    diferenca_normalizada = x_diferenca_x0 / delta_x  # (x - x0) / (x1 - x0)

    for k in range(1, len(X)):
        polinomio *= (diferenca_normalizada - (k - 1)) / k  # Atualiza o termo␣
↪do polinômio
        valor_interpolado += polinomio * tabela_diferencas_finitas[0, k]  #␣
↪Adiciona o termo ao valor interpolado

    return valor_interpolado

X = np.array([0, 1, 2,4], dtype=float)
Y = np.array([1, 3,5,7], dtype=float)
x = 2.3

valor_interpolado = interpolacao_diferencas_finitas(X, Y, x)
print(f"O valor interpolado em x = {x} é: {valor_interpolado}")
```

```
Tabela de Diferenças Finitas após preencher a coluna 1:
        x        y     Δ^1y
---------------------------
     0.00    1.0000   2.0000
     1.00    3.0000   2.0000
     2.00    5.0000   2.0000
     4.00    7.0000

Tabela de Diferenças Finitas após preencher a coluna 2:
        x        y     Δ^1y     Δ^2y
------------------------------------
     0.00    1.0000   2.0000   0.0000
     1.00    3.0000   2.0000   0.0000
     2.00    5.0000   2.0000
     4.00    7.0000

Tabela de Diferenças Finitas após preencher a coluna 3:
        x        y     Δ^1y     Δ^2y     Δ^3y
```

```
---------------------------------------------
     0.00     1.0000    2.0000    0.0000    0.0000
     1.00     3.0000    2.0000    0.0000
     2.00     5.0000    2.0000
     4.00     7.0000

O valor interpolado em x = 2.3 é: 5.6
```

0.0.3 Implementação da função 3, Interpolação por Diferenças Divididas:

[ ]:
```python
def proterm(i, value, x):
    pro = 1
    for j in range(i):
        pro = pro * (value - x[j])
    return pro

def dividedDiffTable(x, y, n):
    for i in range(1, n):
        for j in range(n - i):
            y[j][i] = (y[j][i - 1] - y[j + 1][i - 1]) / (x[j] - x[i + j])
    return y

def applyFormula(value, x, y, n):
    sum = y[0][0]
    for i in range(1, n):
        sum = sum + (proterm(i, value, x) * y[0][i])
    return sum

def printDiffTable(y, n):
    for i in range(n):
        for j in range(n - i):
            print(round(y[i][j], 4), "\t", end=" ")
        print("")

def divided_difference(n, x_list, y_list, value):
    y = [[0 for i in range(n)] for j in range(n)]
    for i in range(n):
        y[i][0] = y_list[i]

    y = dividedDiffTable(x_list, y, n)
    printDiffTable(y, n)

    result = round(applyFormula(value, x_list, y, n), 2)
    return result

# Example usage for Newton Divided Differences
```

```python
n = 3
x_list = [11, 12, 13]
y_list = [1400, 2500, 3250]
value = 10

resultado_dividido = divided_difference(n, x_list, y_list, value)
print(f"O valor interpolado em {value} é: {resultado_dividido}")
```

```
1400     1100.0          -175.0
2500     750.0
3250
O valor interpolado em 10 é: -50.0
```

0.0.4 Implementação da função 4, Sistemas Lineares: Gauss

[ ]:
```python
def metodo_gauss(A, B):
    # Combina a matriz A e o vetor B em uma única matriz aumentada [A|B]
    AB = np.hstack([A, B.reshape(-1, 1)])
    n = len(B)

    # Aplicando eliminação gaussiana
    for i in range(n):
        # Pivotamento parcial
        max_row = np.argmax(np.abs(AB[i:, i])) + i
        AB[[i, max_row]] = AB[[max_row, i]]

        # Eliminação
        for j in range(i + 1, n):
            factor = AB[j, i] / AB[i, i]
            AB[j, i:] -= factor * AB[i, i:]

    # Substituição para encontrar as soluções
    x = np.zeros(n)
    for i in range(n - 1, -1, -1):
        x[i] = (AB[i, -1] - np.dot(AB[i, i + 1:n], x[i + 1:])) / AB[i, i]

    return AB, x

# Exemplo de uso:
# A = np.array([[1, -2, 1], [2, -3, 1], [1, 4, 2]], dtype=float)
# B = np.array([-1, -3, 7], dtype=float)

# A = np.array([[4, 1, 2], [3, 5, 1], [1, 1, 3]], dtype=float)
# B = np.array([4, 7, 3], dtype=float)

A = np.array([[0, 1, -2], [2, -2, 3], [1, 3, 1]], dtype=float)
B = np.array([7, -10, 8], dtype=float)
```

```
AB, x = metodo_gauss(A, B)
print("Matriz A:\n", A)
print("Matriz B:\n", B)
print("\nMatriz aumentada [A|B] após eliminação gaussiana:")
print(AB)
print("\nSolução do sistema, vetor x:")
print(x)
```

```
Matriz A:
 [[ 0.  1. -2.]
 [ 2. -2.  3.]
 [ 1.  3.  1.]]
Matriz B:
 [  7. -10.   8.]

Matriz aumentada [A|B] após eliminação gaussiana:
[[  2.     -2.      3.    -10.   ]
 [  0.      4.     -0.5    13.   ]
 [  0.      0.     -1.875   3.75 ]]

Solução do sistema, vetor x:
[ 1.  3. -2.]
```

0.0.5 Implementação da função 5, Sistemas Lineares: Decomposição LU

[ ]:
```
import numpy as np

def decomposicao_LU(A):
    n = len(A)
    L = np.zeros((n, n))
    U = np.zeros((n, n))

    for i in range(n):
        L[i][i] = 1
        for j in range(i, n):
            U[i][j] = A[i][j] - sum(L[i][k] * U[k][j] for k in range(i))
        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]

    return L, U

def forward_substitution(L, b): # resolvendo o sistema triangular inferior Ly =
 ↪b
    y = np.zeros_like(b)
    for i in range(len(b)):
```

```python
        y[i] = b[i] - sum(L[i][j] * y[j] for j in range(i))
    return y

def backward_substitution(U, y): # resolvendo o sistema triangular superior Ux = y
    x = np.zeros_like(y)
    for i in range(len(y)-1, -1, -1):
        x[i] = (y[i] - sum(U[i][j] * x[j] for j in range(i+1, len(y)))) / U[i][i]
    return x

def solve_lu(A, B):
    L, U = decomposicao_LU(A)
    y = forward_substitution(L, B)
    x =     (U, y)
    return L, U, x

# Exemplo de uso
A = np.array([[1, -2, 1], [2, -3, 1], [1, 4, 2]], dtype=float)
B = np.array([-1, -3, 7], dtype=float)

L, U, x = solve_lu(A, B)

print("Matriz A:")
print(A)
print("\nMatriz B:")
print(B)
print("\nMatriz L:")
print(L)
print("\nMatriz U:")
print(U)
print("\nSolução x:")
print(x)
```

```
Matriz A:
[[ 1. -2.  1.]
 [ 2. -3.  1.]
 [ 1.  4.  2.]]

Matriz B:
[-1. -3.  7.]

Matriz L:
[[1. 0. 0.]
 [2. 1. 0.]
 [1. 6. 1.]]
```

```
Matriz U:
[[ 1. -2.  1.]
 [ 0.  1. -1.]
 [ 0.  0.  7.]]

Solução x:
(array([[ 1., -2.,  1.],
       [ 0.,  1., -1.],
       [ 0.,  0.,  7.]]), array([-1., -1., 14.]))
```

0.0.6 Implementação da função 6, Sistemas Lineares: Gauss-Jacobi

[ ]:
```python
def gauss_jacobi(A, b, x0, tol=1e-6, max_iter=1000):
    n = len(b)
    x = np.copy(x0)

    for iter_count in range(max_iter):
        x_new = np.copy(x)

        for i in range(n):
            sum1 = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x_new[i] = (b[i] - sum1) / A[i, i]

        error = np.linalg.norm(x_new - x, ord=np.inf)

        if error < tol:
            return x_new

        x = x_new

    raise ValueError("A solução não converge dentro do número máximo de␣
↪iterações.")

A = np.array([[4, 1, 2], [3, 5, 1], [1, 1, 3]], dtype=float)
b = np.array([4, 7, 3], dtype=float)
x0 = np.zeros_like(b)

solucao = gauss_jacobi(A, b, x0)
print(f"A solução aproximada é: x = {solucao}")
```

```
A solução aproximada é: x = [0.50000036 1.00000038 0.50000034]
```

0.0.7 Implementação da função 7, Sistemas Lineares: Gauss-Siedel

[ ]:
```python
def gauss_seidel(A, b, x0, tol=1e-6, max_iter=1000):
    n = len(b)
    x = np.copy(x0)
```

```
    for _ in range(max_iter):
        x_old = np.copy(x)
        for i in range(n):
            sum1 = sum(A[i, j] * x[j] for j in range(n) if j != i)
            x[i] = (b[i] - sum1) / A[i, i]

        error = np.linalg.norm(x - x_old, ord=np.inf)

        if error < tol:
            return x

    raise ValueError("A solução não converge dentro do número máximo de␣
↪iterações.")

A = np.array([[4, 1, 2], [3, 5, 1], [1, 1, 3]], dtype=float)
b = np.array([4, 7, 3], dtype=float)
x0 = np.zeros_like(b)

solucao = gauss_seidel(A, b, x0)
print(f"A solução aproximada é: x = {solucao}")
```

```
A solução aproximada é: x = [0.50000005 0.99999998 0.49999999]
```