UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS JARDINS DE ANITA

CIÊNCIA DE DADOS

ERICK COUTINHO

536591

2° AVALIAÇÃO: MÉTODOS NUMÉRICOS

ITAPAJÉ, CE

2023

1.

Após diversas tentativas, nenhum dos métodos convergiu. Após uma análise buscando o motivo da não convergência:

Os autovalores da matriz A que são [−2.11750391, 3.20697238, 2.16092448, 0.74960704].

Observando esses autovalores, pode-se notar que a matriz não é diagonalizável, pois possui um autovalor negativo. A convergência dos métodos iterativos como Jacobi e Gauss-Seidel está relacionada aos autovalores da matriz. Para esses métodos convergirem, é necessário que a matriz seja diagonalizável ou que, pelo menos, seus autovalores estejam dentro do círculo de convergência.

No entanto, se a matriz não é diagonalizável ou tem autovalores fora do círculo de convergência, esses métodos podem não convergir. No caso, como há um autovalor negativo, isso pode ser uma razão para a não convergência dos métodos.

Mesmo com a não convergência dos métodos, trouxe o resultado do sistema:

Código:

```
print('A solução exata é:\n',np.linalg.inv(A)@b)
```

a expressão completa "np.linalg.inv(A)@b" está resolvendo o sistema linear Ax=b para x encontrando a multiplicação da matriz inversa de A pelo vetor b. Essa operação resulta no vetor x que representa a solução exata do sistema linear.

Saída:

```
Autovalores da matriz A: [-2.11750391  3.20697238  2.16092448  0.74960704]
A solução exata é:
 [[0.90909091]
 [0.81818182]
 [1.54545455]
 [1.27272727]]
```

2. Interpole um polinômio que passa pelos pontos : {(1, 2),(2, 0.4),(3, 3),(4, 3.5)}.

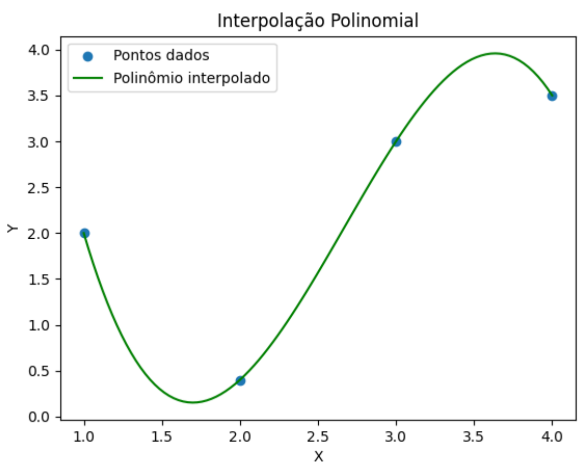
Código:

```python
import numpy as np
import matplotlib.pyplot as plt

#pontos
x1 = np.array([1,2])
x2 = np.array([2,0.4])
x3 = np.array([3,3])
x4 = np.array([4,3.5])
#arrays
x = np.array([x1[0], x2[0], x3[0], x4[0]])
y = np.array([x1[1], x2[1], x3[1], x4[1]])

grau_polinomio = len(x) - 1
# Ajuste do polinômio aos pontos
coeficientes = np.polyfit(x, y, grau_polinomio)
# Criar o polinômio a partir dos coeficientes
polinomio = np.poly1d(coeficientes)
print(f'O polinomia que interpola os pontos é:{polinomio}')
print('Os números "3" e "2" no print representam os graus dos termos do polinômio.')
print('Polinômio: \n-1.05x^3 + 8.4x^2 - 19.45x + 14.1 ')

# Avaliar o polinômio em pontos específicos para plotagem
x_valores = np.linspace(min(x), max(x), num: 100)
y_valores = polinomio(x_valores)

# Plotar os pontos e o polinômio interpolado
plt.scatter(x, y, label='Pontos dados')
plt.plot( *args: x_valores, y_valores, label='Polinômio interpolado', color='green')
plt.title('Interpolação Polinomial')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Saída:

```
O polinomia que interpola os pontos é:       3       2
-1.05 x + 8.4 x - 19.45 x + 14.1
Os números "3" e "2" no print representam os graus dos termos do polinômio.
Polinômio:
-1.05x^3 + 8.4x^2 - 19.45x + 14.1
```

Saida (Gráfico):



3. Obtenha a reta de regressão que melhor ajusta o conjunto de pontos abaixo plotando seu gráfico: {(1, 3.25), (1.5, 3.5), (2, 3.75), (2.5, 4), (3, 4.25), (3.5, 4.5)}.

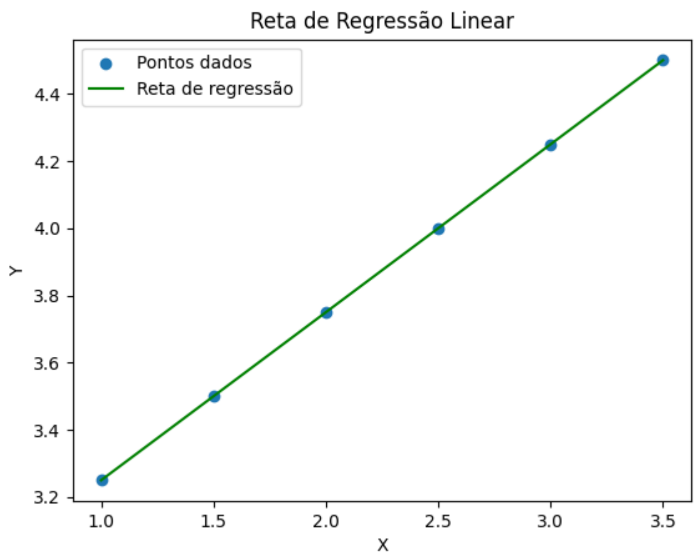
Código:

```python
import numpy as np
import matplotlib.pyplot as plt

# Pontos
pontos = np.array([(1, 3.25), (1.5, 3.5), (2, 3.75), (2.5, 4), (3, 4.25), (3.5, 4.5)])

# Separar os pontos em coordenadas x e y
x = pontos[:, 0]
y = pontos[:, 1]

# Realizar a regressão linear
coeficientes = np.polyfit(x, y, deg: 1)  # O argumento "1" indica uma regressão linear (reta)
reta_regressao = np.poly1d(coeficientes)
print('Equação da reta:', reta_regressao)
x_valores_reta = np.linspace(min(x), max(x), num: 100)
y_valores_reta = reta_regressao(x_valores_reta)
# Plotar os pontos e a reta de regressão
plt.scatter(x, y, label='Pontos dados')
plt.plot( *args: x_valores_reta, y_valores_reta, label='Reta de regressão', color='green')
plt.title('Reta de Regressão Linear')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.show()
```

Saída:

```
Equação da reta:
0.5 x + 2.75
```

Saida (Gráfico):



4. Plote a derivada exata e a derivada numérica da função com h = 0.1 e h = 0.001

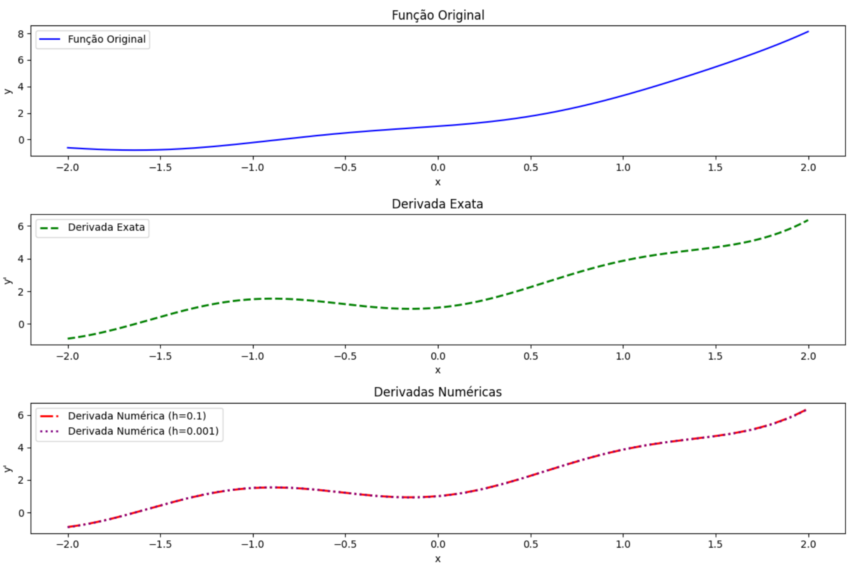
y = e^x + sen(x)^3

Código:

```python
import numpy as np
import matplotlib.pyplot as plt
def funcao(x):
    return np.exp(x) + np.sin(x)**3
def derivada_exata(x):
    return np.exp(x) + 3 * np.sin(x)**2 * np.cos(x)
def derivada_numerica(x, h):
    return (funcao(x + h) - funcao(x - h)) / (2 * h)
#Valores de x
x_valores = np.linspace(-2, 2, 500)
y_valores = funcao(x_valores)
derivada_exata_valores = derivada_exata(x_valores)
derivada_numerica_h_01 = derivada_numerica(x_valores, 0.1)
derivada_numerica_h_001 = derivada_numerica(x_valores, 0.001)
# Plote
plt.figure(figsize=(12, 8))
# Plotar a função original
plt.subplot(3, 1, 1)
plt.plot(x_valores, y_valores, label='Função Original', color='blue')
plt.title('Função Original')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
# Plotar a derivada exata
plt.subplot(3, 1, 2)
plt.plot(x_valores, derivada_exata_valores, label='Derivada Exata', linestyle='--', linewidth=2, color='green')
plt.title('Derivada Exata')
plt.xlabel('x')
plt.ylabel('y\'')
plt.legend()
# Plotar as derivadas numéricas
plt.subplot(3, 1, 3)
plt.plot(x_valores, derivada_numerica_h_01, label='Derivada Numérica (h=0.1)', linestyle='-.', linewidth=2, color='red')
plt.plot(x_valores, derivada_numerica_h_001, label='Derivada Numérica (h=0.001)', linestyle=':', linewidth=2, color='purple')
plt.title('Derivadas Numéricas')
plt.xlabel('x')
plt.ylabel('y\'')
```

```python
plt.xlabel('x')
plt.ylabel('y\'')
plt.legend()

plt.tight_layout()
plt.show()
```

Saida (Gráfico):



5. Sejam as integrais definidas:

(a) $\int_2^5 cos\,(2x)\,dx.$

(b) $\int_1^4 xln(x)dx.$

Construa duas tabelas com n = {4, 10, 25, 100}. A primeira deverá comparar a solução analítica com os métodos de Riemann usando o ponto médio, trapézios e Simpson enquanto que a segunda tabela deverá comparar os erros absolutos da solução analítica em relação a cada um desses três métodos citados.

Código (A):

```python
import numpy as np

# Função a ser integrada
6 usages
def f(x):
    return np.cos(2 * x)
# Solução analítica
2 usages
def solucao_analitica(a, b):
    return (1 / 2) * np.sin(4) - (1 / 4) * np.sin(2)
# Métodos de Riemann
2 usages
def ponto_medio(f, a, b, n):
    h = (b - a) / n
    xi = np.linspace(a + h / 2, b - h / 2, n)
    return h * np.sum(f(xi))
2 usages
def trapezios(f, a, b, n):
    h = (b - a) / n
    xi = np.linspace(a, b, n + 1)
    return h / 2 * (f(a) + 2 * np.sum(f(xi[1:-1])) + f(b))
2 usages
def simpson(f, a, b, n):
    if n % 2 != 0:
        n += 1  # Se n é ímpar, torná-lo par
    h = (b - a) / n
    xi = np.linspace(a, b, n + 1)
    resultado_simpson = f(a) + f(b)
    for i in range(1, n, 2):
        resultado_simpson += 4 * f(xi[i])
    for i in range(2, n - 1, 2):
        resultado_simpson += 2 * f(xi[i])
    return h / 3 * resultado_simpson
# Definindo intervalo
a, b = 2, 5
```

```python
# Definindo intervalo
a, b = 2, 5
# Valores de n
n_valores = [4, 10, 25, 100]
# Tabela 1: Comparando com a solução analítica
print("Tabela 1: Comparando com a solução analítica")
print("n | Solução Analítica | Ponto Médio | Trapézios | Simpson")
print("--|-------------------|-------------|-----------|--------")
for n in n_valores:
    ponto_medio_resultado = ponto_medio(f, a, b, n)
    trapezios_resultado = trapezios(f, a, b, n)
    simpson_resultado = simpson(f, a, b, n)
    analitico = solucao_analitica(a, b)
    print(f"{n} | {analitico:.6f} | {ponto_medio_resultado:.6f} | {trapezios_resultado:.6f} | {simpson_resultado:.6f}")
# Tabela 2: Comparando os erros absolutos
print("\nTabela 2: Comparando os erros absolutos")
print("n | Erro Ponto Médio | Erro Trapézios | Erro Simpson")
print("--|------------------|----------------|-------------")
for n in n_valores:
    ponto_medio_resultado = ponto_medio(f, a, b, n)
    trapezios_resultado = trapezios(f, a, b, n)
    simpson_resultado = simpson(f, a, b, n)
    analitico = solucao_analitica(a, b)
    erro_pm = np.abs(analitico - ponto_medio_resultado)
    erro_trap = np.abs(analitico - trapezios_resultado)
    erro_simpson = np.abs(analitico - simpson_resultado)
    print(f"{n} | {erro_pm:.6f} | {erro_trap:.6f} | {erro_simpson:.6f}")
```

Saída (A):

Tabela 1: Comparando com a solução analítica

n	Solução Analítica	Ponto Médio	Trapézios	Simpson
4	-0.605726	0.117061	0.085652	0.110430
10	-0.605726	0.108003	0.103180	0.106471
25	-0.605726	0.106646	0.105880	0.106392
100	-0.605726	0.106407	0.106359	0.106391

Tabela 2: Comparando os erros absolutos

n	Erro Ponto Médio	Erro Trapézios	Erro Simpson
4	0.722786	0.691378	0.716156
10	0.713729	0.708905	0.712196
25	0.712372	0.711605	0.712118
100	0.712132	0.712084	0.712116

CONTEXTO:

Solução Analítica:

A solução analítica de uma integral definida é a expressão matemática exata que representa a área sob a curva da função no intervalo especificado. Em outras palavras, é o resultado teórico exato da integral.

Métodos de Riemann:

Os métodos de Riemann são técnicas de aproximação numérica para calcular integrais definidas. Eles dividem o intervalo de integração em subintervalos e aproximam a área sob a curva usando somas de áreas de retângulos (no caso do ponto médio), trapézios (no caso dos trapézios) ou segmentos de parábolas (no caso de Simpson).

- Ponto Médio:

Este método utiliza o valor da função no ponto médio de cada subintervalo para calcular a altura do retângulo.

- Trapézios:

Este método usa segmentos de reta conectando os pontos da curva nos extremos de cada subintervalo, formando trapézios. A área de cada trapézio é calculada e somada para obter a aproximação da integral.

- Simpson:

Este método usa segmentos de parábolas para conectar os pontos da curva nos extremos e no ponto médio de cada subintervalo.

Código (B):

```python
print('\nItem B:\n')
# Função a ser integrada
def g(x):
    return x * np.log(x)
# Solução analítica
def solucao_analitica_b(a, b):
    return b * (np.log(b) - 1) - a * (np.log(a) - 1)
# Métodos de Riemann
def ponto_medio_b(f, a, b, n):
    h = (b - a) / n
    xi = np.linspace(a + h / 2, b - h / 2, n)
    return h * np.sum(f(xi))
def trapezios_b(f, a, b, n):
    h = (b - a) / n
    xi = np.linspace(a, b, n + 1)
    return h / 2 * (f(a) + 2 * np.sum(f(xi[1:-1])) + f(b))
def simpson_b(f, a, b, n):
    if n % 2 != 0:
        n += 1  # Se n é ímpar, torná-lo par
    h = (b - a) / n
    xi = np.linspace(a, b, n + 1)
    resultado_simpson = f(a) + f(b)
    for i in range(1, n, 2):
        resultado_simpson += 4 * f(xi[i])
    for i in range(2, n - 1, 2):
        resultado_simpson += 2 * f(xi[i])
    return h / 3 * resultado_simpson
```

```python
# Definindo intervalo
a, b = 1, 4
# Valores de n
n_valores_b = [4, 10, 25, 100]
# Tabela 1: Comparando com a solução analítica
print("Tabela 1: Comparando com a solução analítica")
print("n | Solução Analítica | Ponto Médio | Trapézios | Simpson")
print("--|-------------------|-------------|-----------|--------")
for n in n_valores_b:
    ponto_medio_resultado = ponto_medio_b(g, a, b, n)
    trapezios_resultado = trapezios_b(g, a, b, n)
    simpson_resultado = simpson_b(g, a, b, n)
    analitico_b = solucao_analitica_b(a, b)
    print(
        f"{n} | {analitico_b:.6f} | {ponto_medio_resultado:.6f} | {trapezios_resultado:.6f} | {simpson_resultado:.6f}")
# Tabela 2: Comparando os erros absolutos
print("\nTabela 2: Comparando os erros absolutos")
print("n | Erro Ponto Médio | Erro Trapézios | Erro Simpson")
print("--|------------------|-----------------|--------------")
for n in n_valores_b:
    ponto_medio_resultado = ponto_medio_b(g, a, b, n)
    trapezios_resultado = trapezios_b(g, a, b, n)
    simpson_resultado = simpson_b(g, a, b, n)
    analitico_b = solucao_analitica_b(a, b)
    erro_pm = np.abs(analitico_b - ponto_medio_resultado)
    erro_trap = np.abs(analitico_b - trapezios_resultado)
    erro_simpson = np.abs(analitico_b - simpson_resultado)
    print(f"{n} | {erro_pm:.6f} | {erro_trap:.6f} | {erro_simpson:.6f}")
```

Saída (B):

```
Item B:

Tabela 1: Comparando com a solução analítica
n | Solução Analítica | Ponto Médio | Trapézios | Simpson
--|-------------------|-------------|-----------|--------
4 | 2.545177 | 7.308197 | 7.404954 | 7.341614
10 | 2.545177 | 7.335165 | 7.350742 | 7.340395
25 | 2.545177 | 7.339523 | 7.342018 | 7.340356
100 | 2.545177 | 7.340303 | 7.340459 | 7.340355

Tabela 2: Comparando os erros absolutos
n | Erro Ponto Médio | Erro Trapézios | Erro Simpson
--|------------------|----------------|-------------
4 | 4.763019 | 4.859776 | 4.796437
10 | 4.789988 | 4.805564 | 4.795217
25 | 4.794346 | 4.796841 | 4.795178
100 | 4.795125 | 4.795281 | 4.795177
```

Link para acessar repositório: https://github.com/ErickCoutinho/AV_2_M-todos_Numericos