Universidade Federal do Espírito Santo

Prova 2

Códigos

Algoritmos numéricos - Explicação dos códigos

Italo Jezo de Oliveira Silva e Agenor Andre Almeida Dias

Vitória

2022

Todos os códigos estão implementados de forma computacional no github (compartilhado com saulobortolon). A seguir, daremos uma explicação sobre o funcionamento do código desenvolvido para cada questão.

Problema 1) Derivação Numérica (Q1.ipynb)

O código tem como objetivo calcular a primeira e segunda derivadas da função tangente (f(x) = tan(x)) e da função (f(x) = exp(x/3) + x^2) usando diferenças finitas. As aproximações são realizadas usando métodos de diferenças finitas de quarta ordem, o que significa que o erro em cada aproximação é de O(h^4), onde h é o tamanho do passo.

O código calcula a primeira e segunda derivadas para 6 valores diferentes de x, armazenados no vetor x. O resultado é então comparado com as derivadas exatas, armazenadas nas variáveis dydx e dy2dx2, respectivamente. Finalmente, o erro em cada aproximação é calculado e impresso na tela.

- **Resultados do código:**

  - **Para f(x) = tan(x):**

x	y	f'(x)	f''(x)	Erro f'(x)	Erro f''(x)
2.1	-1.7098465429045075	3.899344249999988	-12.315779166666713	0.6175732347569759 %	8.210546978357927 %
2.2	-1.3738230567687948	2.8924854166666654	-8.034799166666655	-0.17647860958872685 %	-1.2765299660312297 %
2.3	-1.1192136417341325	2.249704083333334	-5.034229166666671	0.130295723430561 %	0.16142857387096224 %
2.4	-0.9160142896734107	1.8429592499999978	-3.3140691666666697	-0.2108155447661104 %	1.637810391501147 %
2.5	-0.7470222972386602	1.544209916666667	-2.874319166666694	0.8878061778839896 %	-23.478738079200497 %
2.6	-0.6015966130897586	1.3554963333333316	-1.8938416666666844	0.47155183067346185 %	-15.573209680303263 %

  - **Para f(x) = exp(x/3) + x^2:**

x	y	f'(x)	f''(x)	Erro f'(x)	Erro f''(x)
-3.0	9.367879441171443	-5.877357500000001	2.0406916666667234	0.00027256408776091707 %	0.009007251905309314 %
-2.8	8.233240720868597	-5.468923333333338	2.043666666666697	-6.534421839074953e-05 %	0.0013087463594900265 %
-2.6	7.1803503845086825	-5.059884166666663	2.04674166666669	-1.9002472009040488e-05 %	-0.0017622655636565108 %
-2.4	6.209328964117222	-4.6502199999999965	2.0499166666666437	7.910646563566749e-05 %	0.00042800536313479916 %
-2.2	5.3203053010898005	-4.23991083333334	2.0531916666666796	-0.00029718551201989257 %	0.0085512715506402 %
-2.0	4.513417119032592	-3.8288533333333414	2.0571499999999667	0.0001991973384997416 %	-0.005038945328725581 %

Problema 2) Integração Numérica (Q2.ipynb)

Este código implementa o método 1/3 de Simpson para calcular a integral tridimensional de uma função num intervalo [a, b]. O método é baseado na regra do ponto médio composto e usa a fórmula de Simpson para aproximar o valor da integral.

A função integral_3d recebe como argumentos o número de pontos n, o intervalo inferior a e o intervalo superior b. A função gera três vetores equidistantes, um para cada dimensão da integral, que são x, y e z. Em seguida, a função calcula o comprimento de cada subintervalo para cada dimensão.

Em seguida, a função usa três loops aninhados para calcular a integral. O primeiro loop itera sobre os valores de x, o segundo loop itera sobre os valores de y e o terceiro loop itera sobre os valores de z. Em cada iteração, a função verifica se o comprimento do subintervalo é igual a zero. Se for, a iteração é ignorada. Caso contrário, o valor da integral é calculado usando a fórmula de Simpson. O valor é acumulado em variáveis temporárias (sz, sy, sx) que representam a integral parcial em relação a z, y e x, respectivamente.

Por fim, a função retorna o valor da integral total (sx). O código também imprime o valor da integral aproximada e o erro relativo comparado à solução analítica, que é conhecida como 1/12.

- **Resultados do código:**

  - **x = [0; 0.25; 0.5; 0.75; 1]; y e z divididos em 4 intervalos**

Este código implementa uma aproximação numérica para a integral tridimensional de uma função, utilizando o método de quadratura de Simpson. O intervalo de integração é [0, 1] e n subintervalos são usados. O resultado da integral é exibido juntamente com o erro relativo em relação à solução analítica.

```
Integral aproximada: 0.082031250000000
Erro relativo comparado à solução analítica: 1.5624999999999994%
```

- **x = [0; 0.125; 0.25; 0.375; 0.5; 0.625; 0.75; 0.875; 1]; y e z divididos em 8 intervalos**

Este código calcula uma aproximação numérica da integral de uma função de três variáveis usando o método dos trapézios. O vetor x é definido como equidistante entre "a" e "b", com "n" pontos, e os vetores y e z são definidos como equidistantes entre outras expressões. A integral é então calculada usando o método dos trapézios em cada dimensão e o resultado é retornado. No final, o resultado é impresso junto com o erro relativo comparado à solução analítica. O código usa a regra 1/3 de Simpson para calcular a integral numérica. Isso é evidente no cálculo da integral em cada dimensão, onde os valores são acumulados e, ao final de cada dimensão, são multiplicados pelo comprimento de seu respectivo subintervalo e divididos por 3.

```
Integral aproximada: 0.083251953125000
Erro relativo comparado à solução analítica: 0.097656249999994%
```

- **x = [0; 0.1; 0.2;…; 0.9;1.0]; y e z divididos em 10 intervalos**

Este código implementa uma aproximação numérica para a integral tripla de uma função em 3 dimensões, usando o método de 1/3 de Simpson. A função é integrada em relação aos eixos x, y e z. A região de integração é dividida em 10 intervalos para cada eixo (x, y e z).

A integração é realizada com 3 loops aninhados, um para cada eixo. O primeiro loop calcula a integral no eixo y, o segundo loop calcula a integral no eixo z e, finalmente, o terceiro loop calcula a integral no eixo x. O tamanho de cada intervalo (dx, dy e dz) é calculado a partir dos pontos (x, y e z) e o número de intervalos (n).

A aproximação da integral é feita usando o método de 1/3 de Simpson, onde cada termo da soma é ponderado por 1, 4 ou 2, dependendo da sua posição no intervalo. A soma dos termos é então multiplicada pelo tamanho do intervalo dividido por 3. O resultado final da integral é armazenado na variável "sx".

O código também calcula e imprime o erro relativo da aproximação, comparando-a com a solução analítica conhecida, que é 1/12. O erro é calculado como a diferença absoluta entre a aproximação e a solução analítica, dividida pela solução analítica e multiplicada por 100 para obter a porcentagem.

```
Integral aproximada: 0.083300000000000
Erro relativo comparado à solução analítica: 0.039999999999996%
```

Problema 3) Interpolação por SPLines (Q3.ipynb)

O código importa as bibliotecas numpy, scipy e matplotlib para plotar e calcular uma spline cúbica que interpola dados.

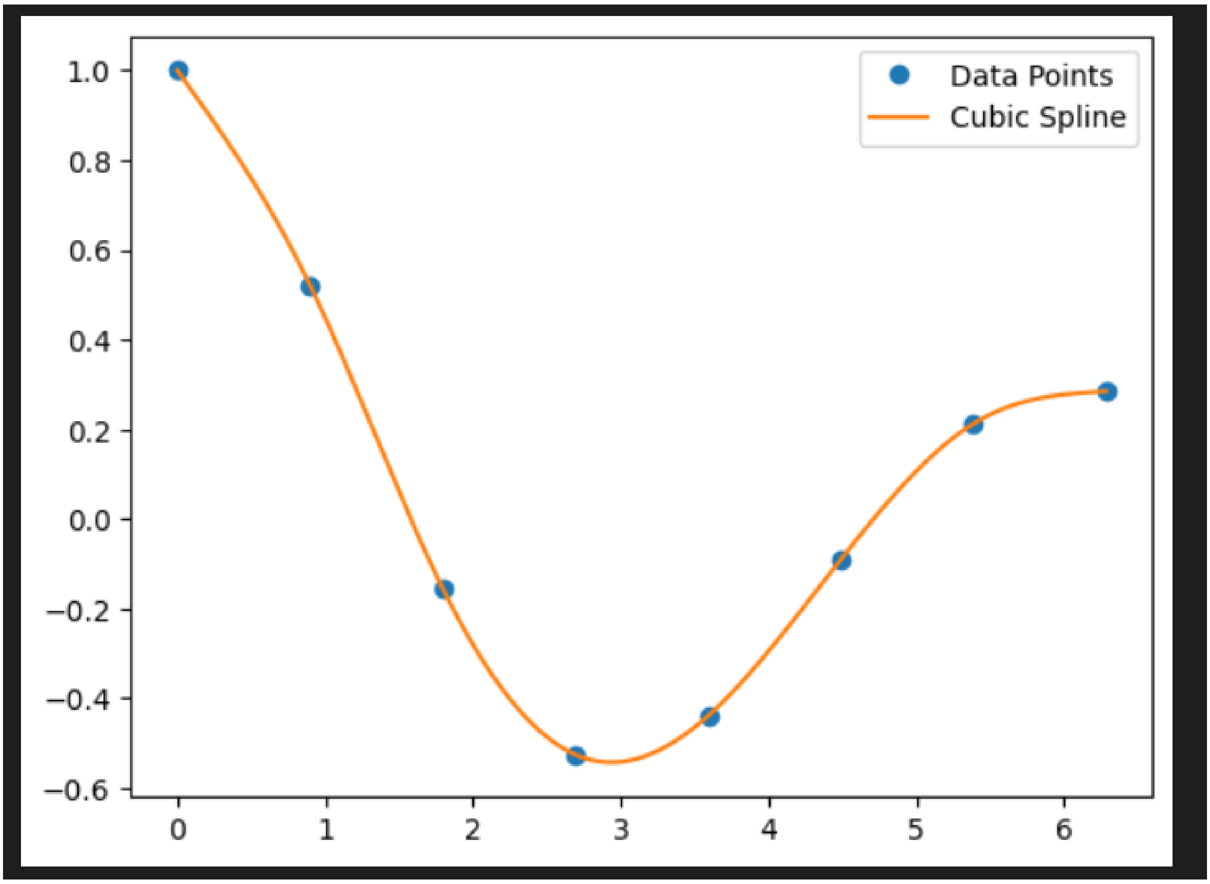
A função CubicSpline da biblioteca scipy.interpolate é usada para aproximar a curva spline para os dados fornecidos, com bc_type = 'natural' definido como tipo de condições de contorno para a spline cúbica.

O conjunto de dados é então interpolado para 1000 pontos, utilizando np.linspace para gerar 1000 valores de x no intervalo de x mínimo a x máximo. O método cs (x_interp) é usado para avaliar a spline cúbica nos pontos interpolados.

O gráfico dos pontos de dados e da spline cúbica é plotado usando a biblioteca matplotlib.

A seguir, a função spline é avaliada para dois valores de x, x = 3 e x = 3.5903916, e o erro relativo em relação ao f(x) original f(x) = x^(-0.2x)cos(x) é calculado. Além disso, as primeiras e segundas derivadas da spline são avaliadas em x = 3.5903916 e o erro relativo em relação às derivadas do f(x) original é calculado. Todos os resultados são impressos na tela.

- **Resultados do código:**




```
Valor da função spline aproximada em x = 3:  -0.5418737990383534
Erro relativo em relação ao f(x) original em x = 3:  -0.002660686765483778
Valor da função spline aproximada em x = 3.5903916:  0.29457476170969565
Valor da primeira derivada da função spline aproximada em x = 3.5903916:  0.29457476170969565
Valor da segunda derivada da função spline aproximada em x = 3.5903916:  0.3483569297671761
Erro relativo da função spline aproximada em relação ao f(x) original em x = 3.5903916:  -1.6704141954175014
Erro relativo da primeira derivada da função spline aproximada em relação ao f(x) original em x = 3.5903916:  0.016374463060602588
Erro relativo da segunda derivada da função spline aproximada em relação ao f(x) original em x = 3.5903916:  1.6106501054291817
```

Problema 4) Mínimos Quadrados (Q4.ipynb)

Este código importa as bibliotecas NumPy, SciPy e Matplotlib para trabalhar com cálculo numérico, otimização e plotagem de gráficos, respectivamente. Em seguida, são definidos três conjuntos de dados x, y1, y2 e y3. Em seguida, três funções polinomial, reta e exponencial são definidas como modelos para ajustar os dados.

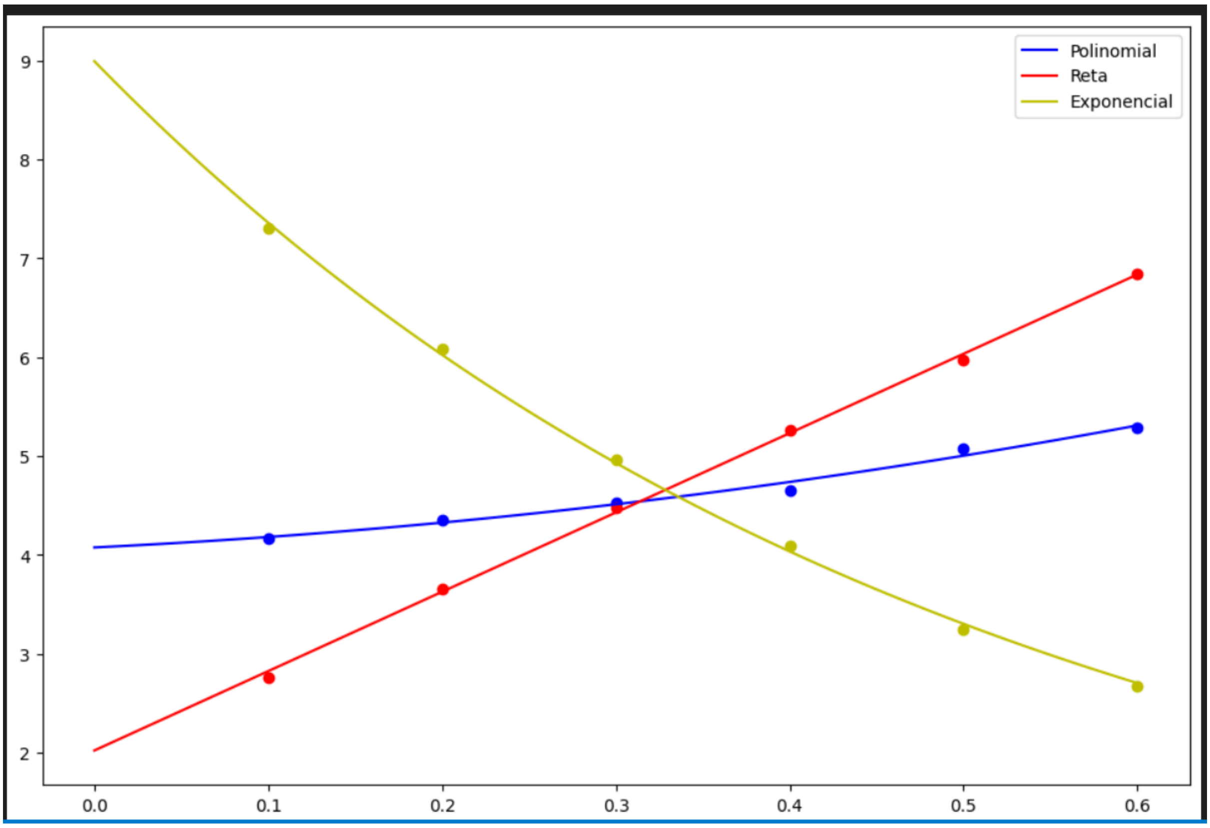
O método de ajuste de curva "curve_fit" da biblioteca SciPy é usado para encontrar os parâmetros a, b, c, d, e, m e n que melhor ajustam os dados y1, y2 e y3 às funções polinomial, reta e exponencial, respectivamente. Os resultados são impressos na tela.

O código então plota os gráficos dos modelos ajustados e dos dados originais usando o Matplotlib. Em seguida, as funções f1, f2 e f3 são definidas como as funções polinomial, reta e exponencial com os parâmetros ajustados. Note que, para fins de didática e comparações, colocamos a cor das funções na mesma cor que do gráfico do enunciado.

Por fim, o código usa o loop "for" para calcular o ponto de interseção entre as três funções: f1, f2 e f3. O ponto de interseção é encontrado como o ponto no intervalo [0,1] que tem a soma mínima das diferenças entre os valores das três funções nesse ponto. O resultado é impresso na tela.

- **Resultados do código:**

```
Polinomial: a = 2.003571, b = 0.855214, c = 4.074800
Reta: d = 8.028286, e = 2.019933
Exponencial: m = 8.995238, n = 2.004564
```




```
x =  0.3287287287287287
f1(x) =  4.572444597480804
f2(x) =  4.659061490052105
f3(x) =  4.654049382684386
```