Análise Numérica

Alexandre Silva
João Temudo
Pedro Leite
Pedro Carvalho

May 2021

1 Introdução

Neste trabalho, temos de aplicar os métodos dados no capítulo 3 para analisar as diferenças entre o polinómio interpolador e o spline cúbico natural, em dois exercícios diferentes.

2 Primeiro Exercício

2.1 Contexto

No primeiro exercício, teremos como fonte de pontos a seguinte tabela:

mês	1	2	3	4	5	6	7	8	9	10	11	12
evaporação	8.6	7.0	6.4	4.0	2.8	1.8	1.8	2.3	3.2	4.7	6.2	7.9

2.2 Spline Cúbico Natural

Na criação de um spline cúbico para um dado conjunto de n+1 pontos, é necessário criar n funções de terceiro grau, cada uma viajando de um dado ponto para o seguinte. Como os nossos pontos têm todos a mesma distância entre si, temos um fator mitigante na quantidade de trabalho a realizar, nomeadamente:

Todas as ocurrências de $\mathrm{h_i}$ podem ser substituídas por 1.

Agora, para cada uma das nossas n funções, denominadas $\mathrm{S_i(x)}$, temos a seguinte fórmula:

$$S_i(x) = M_{i-1}\frac{(x_i - x)^3}{6h_i} + M_i\frac{(x - x_{i-1})^3}{6h_i} + \left(f_{i-1} - M_{i-1}\frac{h_i^2}{6}\right)\frac{x_i - x}{h_i} + \left(f_i - M_i\frac{h_i^2}{6}\right)\frac{x - x_{i-1}}{h_i} \tag{1}$$

Como estamos a tentar cálcular um spline cúbico natural, podemos, então, calcular $\mathrm{M_i}$ através da matriz gerada pelo seguinte conjunto de condições:

$$\frac{h_i}{6}M_{i-1}+\frac{h_i+h_{i+1}}{3}M_i+\frac{h_{i+1}}{6}M_{i+1}=\frac{f_{i+1}-f_i}{h_{i+1}}-\frac{f_i-f_{i-1}}{h_i}, i=1,\ldots,n{-}1 \quad (2)$$

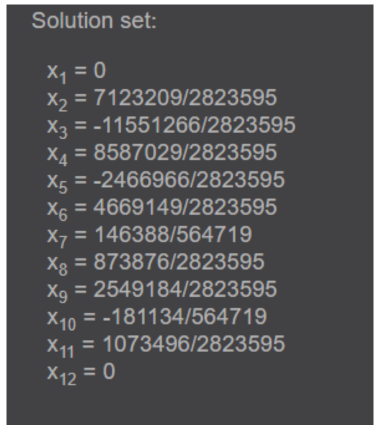
$$\mathrm{M_0}=0$$
$$\mathrm{M_n}=0$$

Assim, temos a matriz seguinte, onde $\mathrm{f_i}$ é a evaporação do mês i+1, o que resulta no sistema seguinte (onde $\mathrm{M_i}$ é igual a $\mathrm{x_{i+1}}$):

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	b
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1/6	2/3	1/6	0	0	0	0	0	0	0	0	0	1
3	0	1/6	2/3	1/6	0	0	0	0	0	0	0	0	-9/5
4	0	0	1/6	2/3	1/6	0	0	0	0	0	0	0	6/5
5	0	0	0	1/6	2/3	1/6	0	0	0	0	0	0	1/5
6	0	0	0	0	1/6	2/3	1/6	0	0	0	0	0	1
7	0	0	0	0	0	1/6	2/3	1/6	0	0	0	0	1/2
8	0	0	0	0	0	0	1/6	2/3	1/6	0	0	0	2/5
9	0	0	0	0	0	0	0	1/6	2/3	1/6	0	0	3/5
10	0	0	0	0	0	0	0	0	1/6	2/3	1/6	0	0
11	0	0	0	0	0	0	0	0	0	1/6	2/3	1/6	1/5
12	0	0	0	0	0	0	0	0	0	0	0	1	0

Figure 1: Matriz por resolver para cálcular os valores de $\mathrm{M_i}$, i = 1,...,n-1. Resolvido utilizando a aplicação matrix.rehish, link nas referências.

Esta matriz, quando resolvida através de eliminação gaussiana, resulta nos seguintes valores:



Solution set:

x_1 = 0
x_2 = 7123209/2823595
x_3 = -11551266/2823595
x_4 = 8587029/2823595
x_5 = -2466966/2823595
x_6 = 4669149/2823595
x_7 = 146388/564719
x_8 = 873876/2823595
x_9 = 2549184/2823595
x_{10} = -181134/564719
x_{11} = 1073496/2823595
x_{12} = 0

Figure 2: Resolução da matriz para cálcular os valores de M_i, i = 1,...,n-1.

Assim, já podemos calcular os valores de cada $S_i(x)$, que irá constituir a porção da reta de {i-1 $\leq$ x $\leq$ i}. Depois de calculados e anexados, podemos verificar que os splines individuais ($S_i(x)$) alinham e são contínuos nos seus intervalos e no intervalo da função. Para além disso, podemos, também, verificar que $S_i(x_i)$ = $S_{i-1}(x_i)$. Sabemos, então, que o spline cúbico natural está bem construído.

Figure 3: Gráfico do spline, criado utilizando a aplicação web "Desmos", link colocado após a conclusão.

2.3 Polinómio Interpolador

Para criar um polinómio interpolador pelo método de Lagrange, é necessário realizar a seguinte fórmula:

$$p_n(x) = I_0(x)f_0 + I_1(x)f_1 + \ldots + I_n(x)f_n \tag{3}$$

Onde cada $\mathrm{I_k(x)}$ é dado pela fórmula abaixo:

$$I_k(x) = \frac{(x - x_0) \ldots (x - x_{k-1})\,(x - x_{k+1}) \ldots (x - x_n)}{(x_k - x_0) \ldots (x_k - x_{k-1})\,(x_k - x_{k+1}) \ldots (x_k - x_n)} \tag{4}$$

Como temos 12 pontos, calcular cada $\mathrm{I_k(x)}$, mesmo que só parcialmente à mão, demoraria demasiado tempo, portanto decidimos programar a criação do polinómio interpolador completamente por código na linguagem de programação "Python".

Para o polinómio interpolador, começamos por criar um vetor com 12 posições, preenchidos com os valores de f(x) dos pontos, em ordem. Depois, entramos num ciclo que irá percorrer todos os pontos, ao qual chamaremos o ciclo "k". Dentro desse ciclo, começamos outro, ao qual chamamos ciclo "i", que irá percorrer todos os pontos exceto o atual em "k", e, a cada iteração, multiplica a variável "l" (que volta ao valor 1 com cada iteração de "k") pela fórmula seguinte:

$$l = l\frac{(x - i)}{(k - i)} \tag{5}$$

Quando o ciclo "i" é completado, passamos esta fração (que contém o nosso $\mathrm{I_k(x)}$), multiplicada por $\mathrm{f_k}$ para a variável "sum", que irá conter a soma de $\mathrm{I_k(x)f_k}$ gerados pelo ciclo "k" de 1 a 12. Ficamos, então, com o seguinte gráfico, mapeado pelo polinómio guardado em "sum":

Observando este gráfico e comparando-o com o do spline, podemos observar

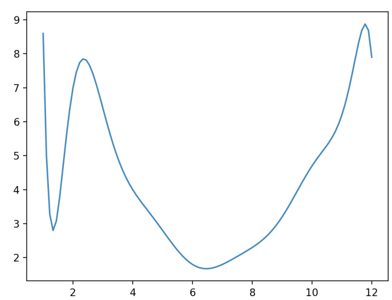


Figure 4: Gráfico do polinómio interpolador criado através de código Python.

que, enquanto ambos passam pelos pontos (k, $\mathrm{f_k}$), o spline apresenta uma amplitude muito menor nos valores entre cada ponto quando comparado com o polinómio interpolador. Isto é particularmente observável perto das pontas dos gráficos.

3 Exercício 2

3.1 Contexto

No exercício 2, é-nos fornecida a expressão seguinte, com domínio [-3, 3]:

$$f(x) = x - \cos(x)^3 \tag{6}$$

3.2 Alínea A

Na primeira alínea do exercício, devemos criar um conjunto de n+1 = 7 pontos. Esta geração é feita da seguinte forma:

- Observar a dimensão do domínio, neste caso, 3-(-3) = 6.
- Dividir a dimensão do domínio pelo número de pontos, através da qual obtemos a distância entre cada ponto. Neste caso, a distância será 6 / 6 = 1.
- Gerar n+1 pontos, ou seja, fazer -3 + 1 x i, i(0)n

Assim, ficamos com o seguinte conjunto de pontos xi e valores f(xi):

i	0	1	2	3	4	5	6
xi	-3	-2	-1	0	1	2	3
fi	-2.02972	-1.92793	-1.15772	-1.0	0.84227	2.07207	3.97028

Como podemos ver, os pontos têm todos um h_i = 1, logo, podemos substituir todas as ocurrências de h_i por 1.

3.3 Alínea B

3.3.1 Spline

Na criação de um spline, como mencionado acima, é necessário criar uma matriz para cálcular os vários M_i para o spline. Aplicando a fórmula (2), obtemos a seguinte matriz:

$$
\begin{cases}
1\,x_1 + 0\,x_2 + 0\,x_3 + 0\,x_4 + 0\,x_5 + 0\,x_6 + 0\,x_7 = 0 \\
1/6\,x_1 + 2/3\,x_2 + 1/6\,x_3 + 0\,x_4 + 0\,x_5 + 0\,x_6 + 0\,x_7 = 0.66842 \\
0\,x_1 + 1/6\,x_2 + 2/3\,x_3 + 1/6\,x_4 + 0\,x_5 + 0\,x_6 + 0\,x_7 = -0.61249 \\
0\,x_1 + 0\,x_2 + 1/6\,x_3 + 2/3\,x_4 + 1/6\,x_5 + 0\,x_6 + 0\,x_7 = 1.68455 \\
0\,x_1 + 0\,x_2 + 0\,x_3 + 1/6\,x_4 + 2/3\,x_5 + 1/6\,x_6 + 0\,x_7 = -0.61247 \\
0\,x_1 + 0\,x_2 + 0\,x_3 + 0\,x_4 + 1/6\,x_5 + 2/3\,x_6 + 1/6\,x_7 = 0.66841 \\
0\,x_1 + 0\,x_2 + 0\,x_3 + 0\,x_4 + 0\,x_5 + 0\,x_6 + 1\,x_7 = 0
\end{cases}
$$

Figure 5: Matriz por resolver para cálcular os valores de M_i, i = 1,...,n-1. Aplicação utilizada foi matrixcalc, link nas referências.

Esta matriz, quando resolvida, resulta nos valores abaixo:

Solution set:

$x_1 = 0$
$x_2 = 7123209/2823595$
$x_3 = -11551266/2823595$
$x_4 = 8587029/2823595$
$x_5 = -2466966/2823595$
$x_6 = 4669149/2823595$
$x_7 = 146388/564719$
$x_8 = 873876/2823595$
$x_9 = 2549184/2823595$
$x_{10} = -181134/564719$
$x_{11} = 1073496/2823595$
$x_{12} = 0$

Figure 6: Resolução da matriz para cálcular os valores de M_i, i = 1,...,n-1.

Tendo obtido estes valores, podemos aplicar a fórmula (1) para obter o seguinte gráfico, que contem o nosso spline cúbico natural para estimar f(x):

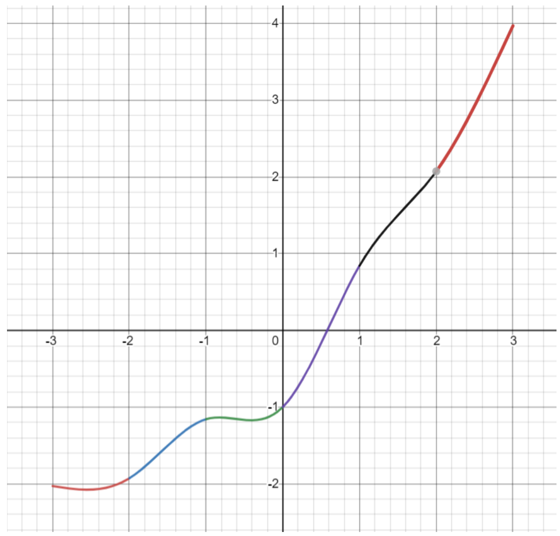


Figure 7: Gráfico do spline, criado utilizando a aplicação web "Desmos", link colocado após a conclusão.

Podemos, então, concluir que o spline está bem construído, dado que podemos observar que é contínuo no intervalo e $S_i(x_i) = S_{i-1}(x_i)$.

3.3.2 Polinómio Interpolador

Para criar o polinómio interpolador, repetimos o método que utilizamos para o primeiro exercício, mas com um ciclo que vai de -3 a 3, invés de 1 a 12, e com os f_k novos. Isto resultou no polinómio interpolador seguinte: Note-se que, apesar

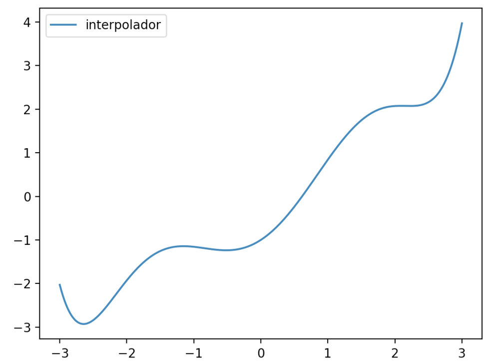


Figure 8: Gráfico do polinómio interpolador, criado utilizando código python

de os valores serem parecidos no centro do gráfico, quanto mais próximo fica das pontas, mais se diferencia o polinómio interpolador do spline.

3.4 Alínea C

3.4.1 Comparação

Na imagem abaixo, podemos comparar os três gráficos: o do polinómio interpolador, do spline e da função f(x). Como podemos observar, o spline, em média, é muito mais próximo dos valores de f(x) que o polinómio interpolador, ou seja, é mais viável neste contexto.

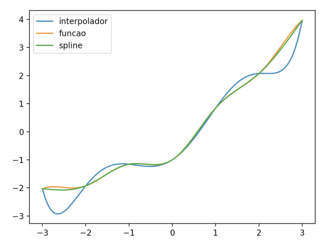


Figure 9: Gráfico do spline (verde), polinómio interpolador (azul) e função f(x) (laranja).

3.4.2 Erro Absoluto

Na imagem abaixo, podemos observar o erro absoluto das duas funções de aproximação, ou seja, sendo "p" o polinómio interpolador e "s" o spline, é demonstrado abaixo $\mid f - p \mid$ e $\mid f - s \mid$. Como observámos também na comparação das funções, o spline apresenta um erro muito menor que o polinómio interpolador, sendo a diferença maior junto aos extremos e menor junto ao centro.

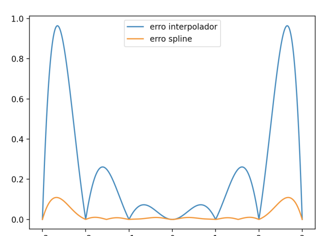


Figure 10: Gráfico com o erro do spline (laranja) e do polinómio interpolador (azul).

3.5 Alínea D

Na alínea D, é-nos pedido para cálcular o majorante do erro do spline e do polinómio interpolador.

Para este efeito, teremos de usar duas fórmulas. A primeira a demonstrar será a do spline:

$$|f(x) - S(x)| \leq \frac{5}{384} M h^4 \tag{7}$$

Nesta fórmula, podemos tratar "M" como o valor absoluto máximo da quarta derivada de f(x), ou seja, $|f^4(x)|$, e "h" como o valor máximo de h_i. No entanto, como foi dito acima, sabemos que h_i será sempre 1, ou seja, "h" = 1. Calculando "M" na aplicação wolframalpha (link nas referências), sabemos que o seu valor será igual a **19.8762**. Temos, então, que o majorante do erro do spline será igual a **0.2588046875** para os dois pontos.

Para o polinómio interpolador, a fórmula utilizada será a seguinte:

$$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(c_x)\, \pi_{n+1}(x) \tag{8}$$

Nesta fórmula, sabemos que $\pi_{n+1}(x) = (x - x_0)(x - x_1) \ldots (x - x_n)$, ou seja, $\pi_{n+1}(x) = (x - (-3))(x - (-2)) \ldots (x - 3)$ e c_x é o valor de x tal que a sétima derivada de f(x) assume o seu valor máximo no intervalo. Sabemos que o seu valor é, então, **547.125**. Calculando o majorante do erro para 0.1, obtemos o valor **0.38549948765625**, que é superior ao valor obtido no cálculo do majorante do erro no spline.
Computando a mesma conta para 2.6, obtemos o valor **10.05099264**, que é de longe superior ao majorante do erro a calcular 0.1 com o polinómio interpolador e, consequentemente, ainda maior que o majorante do erro a calcular qualquér ponto com o spline.
Pode-se, então, concluir que o spline terá uma margem de erro não só mais consistente ao longo de todos os pontos, mas muito menor quando o erro se aproxima das pontas.

4 Conclusão

Após a resolução deste projeto, achamos seguro concluir que, apesar de termos feito o spline parcialmente à mão e, portanto, tendo demorado mais tempo a cálculá-lo, é de maior confiança quando todos os pontos têm um $h_i = 1$ que o polinómio interpolador, desde que a função seja de grau menor ao número de pontos.

No entanto, dado que cálculamos o spline parcialmente à mão, também podemos observar que, numa escala de pontos maior, teriamos de automatizar melhor a criação dos $S_i(x)$ ou utilizar um polinómio interpolador, que é mais fácil de transferir para código.

5 Refências e ferramentas usadas

https://www.desmos.com/calculator?lang=pt-BR
https://www.symbolab.com/solver/simplify-calculator/
https://www.wolframalpha.com/
https://matrixcalc.org/pt/
https://matrix.reshish.com/gauss-jordanElimination.php
Linguagem de programação *Python*
Capítulo 3 de Análise Numérica 2020/21