UNIVERSIDADE ESTADUAL DE SANTA CRUZ
MESTRADO EM MODELAGEM COMPUTACIONAL

APROXIMAÇÃO DE FUNÇÕES COM SPLINES CÚBICOS

Mateus Chaves Barbosa (202210182)

Ilhéus-Bahia
2022

Mateus Chaves Barbosa (202210182)

Aproximação de funções com splines cúbicos

Relatório apresentado como parte dos critérios de avaliação da disciplina: Metodos numéricos I.

Professor(a): Dany Sanchez Dominguez.

Ilhéus-Bahia

2022

Sumário

1 Introdução **2**

2 Interpolação por spline **3**

2.1 Spline natural 5

2.2 Spline natural: Implementação do método 6

2.3 Spline natural: Integrais e derivadas 10

2.4 Spline Fixo: Implementação do método 18

2.5 Spline Fixo: Integrais e derivadas 22

3 Conclusões **25**

1 Introdução

Interpolação é o método que nos permite construir um conjunto de dados através de dados discretos que já são conhecidos. Quando se trata de experimentos ou amostragens nas mais diversas áreas das ciências e engenharia, é comum estarmos munidos de dados pontuais, também conhecido como conjunto degenerado, que não possuem continuidade e isso na maioria das vezes não condiz com o que é realmente observado na natureza.

Com isso, através da interpolação é possível construir uma função que se ajusta nesses dados pontuais, o que nos dá uma continuidade que é conveniente. Uma outra aplicação da interpolação é fazer a aproximação de funções complicadas para funções mais simples facilitando assim a análise e a partir dessa função mais simples é possível obter dados aproximados a da função mais complexa e a depender dessa aproximação, os erros associados podem sem compensados.

Quando se trata de interpolação, existem alguns tipos que podem ser utilizados, são eles:

- Interpolação linear: É o tipo mais simples de interpolação, ela conecta os pontos a partir de uma função do tipo linear.
- Interpolação polinomial: É o tipo de interpolação no qual a função interpoladora é um polinômio.
- Interpolação trigonométrica: É o tipo de interpolação para o caso de um polinômio trigonométrico. Essa interpolação é mais utilizada para os casos onde temos funções periódicas
- Interpolação spline: Interpolação através de uma curva que é definida por n pontos onde $n > 2$
- Interpolação bilinear : É uma generalização da interpolação linear.

Neste relatório, o trabalho foi feito sobre interpolação do tipo spline considerando duas condições de contorno: Natural e fixada. E os resultados, incluindo as integrais e derivadas dos polinômios obtidos, serão discutidas no corpo desse relatório.

2 Interpolação por spline

Existem diversas formas de se fazer interpolações, a essa altura do curso já vimos as interpolações através dos polinômios de Taylor e também a interpolação de Lagrange. Porém, todas elas apresentam determinados problemas em relação a sua eficácia e confiabilidade como foi discutido no corpo desse relatório, no capítulo de introdução. Para conseguir evitar esses problemas, é necessário uma forma mais eficiênte de se realizar a interpolação e essa técnica é conhecida como spline.

A interpolação segmentada, ou splines, é uma técnica de interpolação que dado um conjunto $(x_i, yi)_{i=1}^{n}$ de n pontos onde $x_{i+1} > x_i$ ou seja as abcissas são distintas e estão em ordem crescente a função linear que interpola os pontos x_{i+1} e x_i é dada por:

$$P_i(x) = y_i \frac{x_{i+1} - x}{x_{i+1} - x_i} + y_{i+1} \frac{x - x_i}{x_{i+1} - x_i} \tag{1}$$

E o resultado dessa interpolação linear segmentada é uma função contínua que está definida em partes do intervalo $[x_1, x_n]$

$$f(x) = P_i(x), \quad x \in [x_i, x_{i+1}] \tag{2}$$

E como o nome já sugere, essa interpolação vai ser feita através de funções lineares o que nos gerará retas que farão a interpolação dos pontos. A desvantagem desse método é que por mais que eu tenha funções contínuas interpolando os pontos, essas funções não são deriváveis em todos os pontos, por não serem suaves (apresentam bicos) e quando temos esse tipo de comportamento em uma função, neste ponto de "bico"teríamos infinitas retas tangentes a este ponto, e por consequencia, a derivada neste ponto não existe.

No entanto, esse conceito de spline linear pode ser ampliado para polinômios de mais alto grau, o que é o caso das splines cúbicas. A escolha de um polinômio de mais alto grau implica em uma maior liberdade na construção da interpolação, pois há um número maior de coeficiêntes e parte dessa maior liberdade pode ser empregada na condição de suavidade da função de interpolação.

Dado um conjunto n de pontos, $p = (x_j, yj)_{j=1}^{n}$ tais que $x_{j+1} > x_j$, novamente temos abcissas distintas e em ordem crescente, um spline de ordem m que interpola esses pontos é uma função s que possui as seguintes propriedades:

- Em cada intervalo $[x_j, x_{j+1}), j = 1, 2, ...n-2$ e no segmento $[x_{n-1}, x_n]$ s é um po-

linômio de grau menor ou igual a m.

- Em algum dos intervalos, s é um polinômio de grau m
- Em cada $x_j \in p, s(x_j) = y_j$, ou seja, o spline interpola os pontos dados.
- s é uma função de classe C^{m-1}, ou seja, ela é uma função $m-1$ vezes continuamente diferenciável.

São $n-1$ intervalos e em cada um deles existe $m+1$ coeficientes. As condições iii e iv impostas pela definição correspondem respectivamente a n e $m(n-2)$ equações. Estas últimas, se devem à exigência de continuidade nos pontos internos. Portanto, há $m-1$ coeficientes a mais do que o número de equações e, à exceção do caso $m = 1$ que é a interpolação linear segmentada, o problema é subdeterminado. Ou seja, uma vez fixada a ordem $m > 1$, existem infinitos splines de ordem que interpolam os pontos do conjunto p.

Para um spline cúbico, ou seja $m = 3$ temos que, seja s um spline cúbico que interpola o conjunto de pontos $p = (x_j, yj)_{j=1}^{n}$ tais que $x_{j+1} > x_j$. Se $y'^{n}_{j\,j=1}$ é o conjunto dos valores da derivada de s em x_j, então em cada intervalo $[x_j, x_{j+1})$ o spline s_j será igual a:

$$s_j = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \tag{3}$$

onde

$$a_j = y_j, \quad c_j = 3\frac{y_{j+1} - y_j}{h_j^2} - \frac{y'_{j+1} + 2y'_j}{h_j} \tag{4}$$

$$b_j = y'_j, \quad d_j = -2\frac{y_{j+1} - y_j}{h_j^3} + \frac{y'_{j+1} + 2y'_j}{h_j^2} \tag{5}$$

Considerando também que $h_j = x_{j+1} - x_j$, $j = 1, 2, ..., n-1$.

Dado um conjunto de pontos $p = (x_j, y_j)_{j=1}^n \subset \mathbb{R}^2$ tais que $x_{j+1} > x_j$, as derivadas de um spline cúbico que interpola os pontos p, y'_j, $j = 1, 2, ...n$ satisfazem o seguinte sistema de equações algébricas lineares:

$$h_j y'_{j-1} + 2(h_{j-1} + h_j)y'_j + h_{j-1}y'_{j+1} = 3\left(h_j \frac{y_j - y_{j-1}}{h_{j-1}} + h_{j-1}\frac{y_{j+1} - y_j}{h_j}\right) \tag{6}$$

onde $j = 2, 3, ..., n-1$ e $h_j = x_{j+1} - xj$.

O sistema de equações (6) é subdeterminado. Ou seja, temos n variáveis e $n-2$ equações. A inclusão de duas equações adicionais linearmente independentes das $n-2$ equações (6) possibilita a existência de uma única solução. Normalmente essas equações adicionais envolvem o comportamento do spline na fronteira ou em sua vizinhança. Para este trabalho em questão foram consideradas duas delas, o spline natural e o spline fixado.

2.1 Spline natural

Uma das formas de se adicionar as duas equações afim de se completar o sistema (6) é impor condições de fronteira de forma natural (ou livres), portanto:

$$s''(x_1) = s''(x_n) = 0 \tag{7}$$

E de acordo com a equação (3), teremos respectivamente:

$$c_1 = 0, \quad e \quad 2c_{n-1} + 6d_{n-1}h_{n-1} = 0 \tag{8}$$

ou seja:

$$\begin{cases} 2y'_1 + y'_2 = 3\dfrac{y_2 - y_1}{h_1} \\ \\ y'_{n-1} + 2y'_n = 3\dfrac{y_n - y_{n-1}}{h_{n-1}} \end{cases} \tag{9}$$

Essas duas equações em conjunto com o sistema (6) formam um sistema de n equações algébricas lineares do tipo $Ay' = z$, onde:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0+h_1) & h_1 & & \\ 0 & h_1 & 2(h_1+h_2) & h_2 & \\ \vdots & & \ddots & \ddots & 0 \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

Note que, é diagonal dominante estrita e com isso, o sistema $Ay' = z$ possui solução única. Calculando y', os valores de a_j, b_j, c_j, d_j são obtidos diretamente através das expressões (4) e (5).

2.2 Spline natural: Implementação do método

No problema em questão, devemos escontrar os splines para dois conjuntos de pontos. Para o primeiro conjunto nós temos que $x = 0; 0,25; 0,5; 0,75; 1$, e o segundo conjunto é dado por: $x = 0; 0,125; 0,250; 0,375, 0,5; 0,625; 0,75; 0,875; 1.0$. Os pontos no eixo y são dados pela equação $f(x) = cos(\pi x)$, com isso, para cada valor de x teremos um valor de y associado.

O algoritmo usando para fazer os splines é o mesmo para os dois conjuntos de pontos, o que muda obviamente são os pontos que serão dados como entrada e o número de polinômios que serão gerados na saída. Segue o algoritmo abaixo escrito na linguagem python:

```
1
2 import numpy as np
3 import math
4
5 def spline(x, y): #funcao para calcular os splines
6
7
8   n = len(x)
9   a = {k: v for k, v in enumerate(y)} #obtendo os valores de a em uma
     estrutura de dicionario
10  h = {k: x[k+1] - x[k] for k in range(n - 1)} #obtendo os valores de h
     como um dicionario
11
12  A = [[1] + [0] * (n-1)] # adicionando a primeira linha da matriz A
13  for i in range(1, n - 1):# adicionando as demais linhas da matriz A
14    linha = [0] * n
15    linha[i-1] = h[i-1]
16    linha[i] = 2*(h[i-1] + h[i])
17    linha[i + 1] = h[i]
18    A.append(linha)
19
20  A.append([0] * (n - 1) + [1]) # adicionando a ultima linha da matriz A
21
22
23  B = [0] # primeiro elemento do vetor B
24
25  for k in range(1, n-1):
26    valor = 3*(a[k+1] -a[k]) / h[k] - 3*(a[k] - a[k-1]) / h[k-1] #
     adicionando os demais elementos do vetor B
27    B.append(valor)
28  B.append(0) #ultimo elemento do vetor B
29
30  c = dict(zip(range(n), np.linalg.solve(A, B))) # resolve o sistema Ax
     = B
31
32  b = {}
33  d = {}
34  for k in range(n - 1): # laco que calcula os coeficientes b e d
35    b[k] = (1/h[k]) * (a[k + 1] - a[k]) - (h[k]/3) * (2*c[k]+c[k+1])
```

```
36      d[k] = (c[k+1] - c[k])/ (3 * h[k])
37
38
39    s = {}
40
41    for k in range(n - 1):
42      eq = f'{a[k]}{b[k]:+} * (x{-x[k]:+}){c[k]:+}*(x {- x[k]:+})**2{d[k
       ]:+}*(x{-x[k]:+})**3'
43      s[k] = {'eqs': eq, 'dominio': [x[k], x[k+1]]} #retorna os polinomios
44
45    for k, v in s.items():
46      print(f'S{k}', v)
47
48
49    return s
50
51 x_1 = [0., 0.25, 0.5, 0.75, 1.]
52 x_2 = [0., 0.125, 0.250, 0.375, 0.5, 0.625, 0.75, 0.875, 1.]
53 y_1 = [math.cos(math.pi*i) for i in x_1]
54 y_2 = [math.cos(math.pi*i) for i in x_2]
55
56
57 eqs = spline(, ) # aqui vao os pontos x e y desejados
58 print("\n")
```

O código acima além de calcular os coeficientes, já gera o polinômio referente ao spline com os devidos coeficientes calculados, vale salientar que os polinômios seguem a forma da equação (3). A tela de saída do método segue nos terminais abaixo:

E como foi dito anteriormente, se temos um conjunto com 5 pontos, iremos gerar 4 polinômios. Segue abaixo a tabela com os valores dos coeficientes separadamente:

a_j	b_j	c_j	d_j
1.0	0.7573	0.0	6.627
0.7071	1.999	4.970	6.627
6.123×10^{-17}	3.2426	1.894×10^{-15}	6.627
0.7071	2.0	4.9705	6.627

O mesmo algoritmo acima foi usado para o segundo conjunto de pontos, o que nos

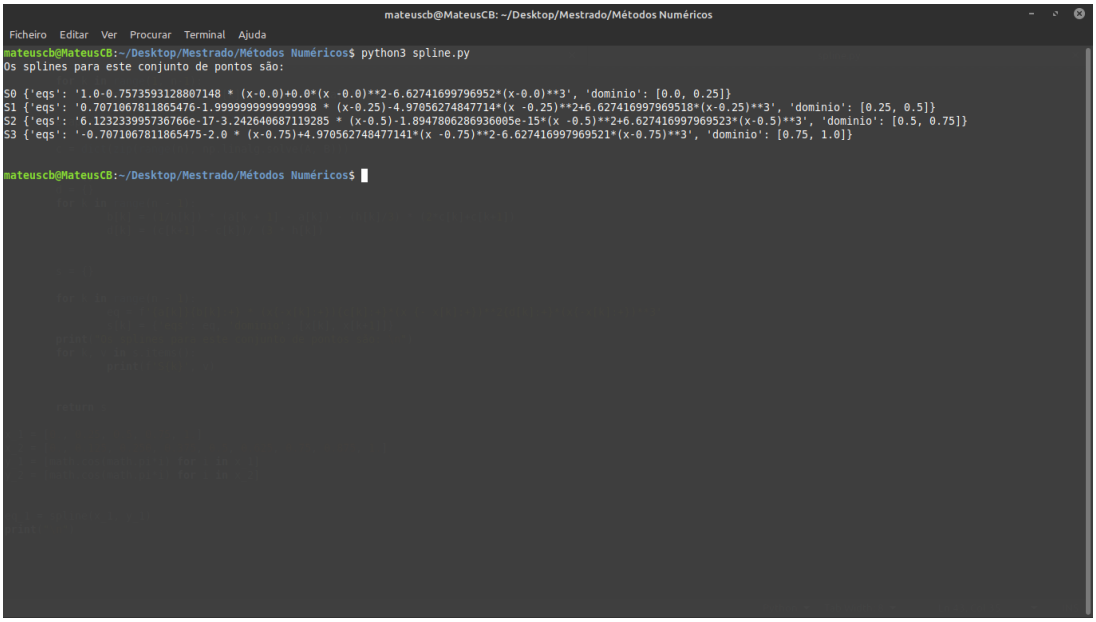

Figura 1: Terminal de execução dos splines para o primeiro conjunto de pontos

gerou 8 polinômios. A saída do método segue no terminal abaixo:

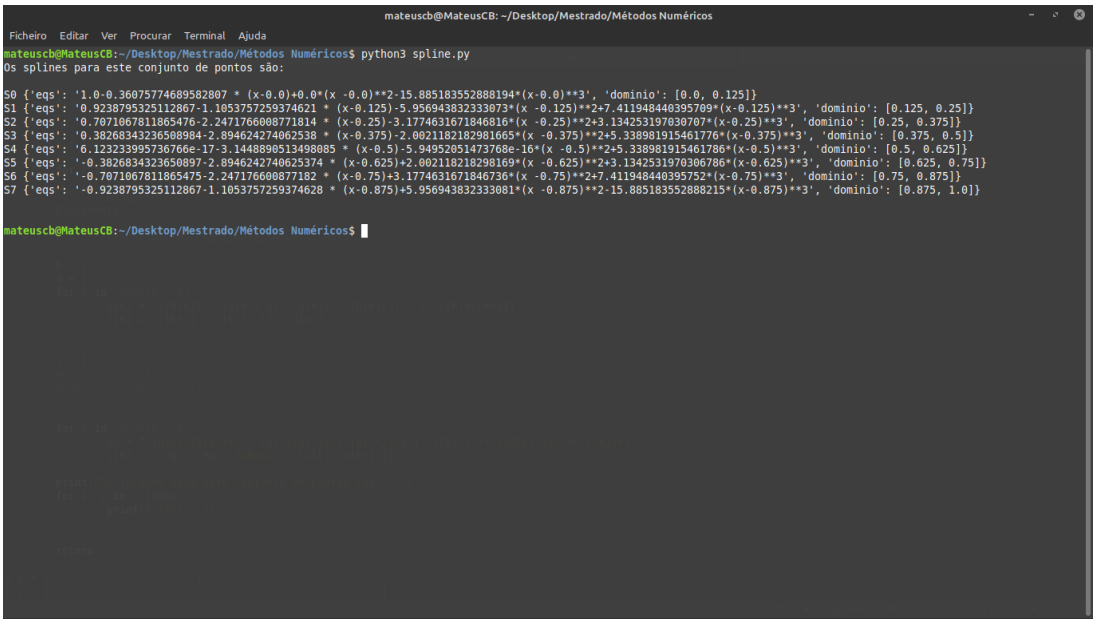

Figura 2: Terminal de execução dos splines para o segundo conjunto de pontos

Segue abaixo os valores dos coeficientes encontrados para o segundo conjunto de pontos:

a_j	b_j	c_j	d_j
1.0	0.3607	0.0	15.8851
0.923	1.1053	5.956	7.4119
0.7071	2.247	3.1774	3.1342
0.382	2.894	2.00211	5.3389
6.123×10^{-17}	3.144	5.9495	5.3389
-0.382	2.8946	2.0021	3.13425
-0.707	2.2471	3.1774	+7.4119
-0.923	1.1053	5.9569	15.8851

Para ambos conjutos de pontos, os valores dos coeficientes estão sendo considerados em módulo. Os sinais dos coeficientes foram utilizados, obviamente, para compor as equações dos splines.

2.3 Spline natural: Integrais e derivadas

Para fazer as integrais, foi utilizada a biblioteca simpy da linguagem python. Por meio dessa biblioteca, é possível fazer calculos simbolicos(e também numéricos) na linguagem python o que facilita o processo de obtenção dos resultados.

Primeiramente foi calculada a integral da função original, dada por: $f(x) = cos(\pi x)$. Segue abaixo os comando utilizados para o cálculo das integrais, tanto da função original como também para as dos splines para o primeiro conjunto de pontos:

```
1
2 from sympy import *
3 import math
4
5 init_printing(pretty_print=True)
6
7 x = Symbol('x')
8
9 Integral(cos(pi*x), (x, 0, 1)).doit().evalf(2) # calcula a integral da
    funcao original com duas casas decimais.
10
11 #Segue agora o calculo das integrais para cada um dos splines
12
```

```
S_0 = Integral(1.0 - (0.7573593128807148 * (x - 0.0)) + (0.0 * (x - 0.0)
    **2) - (6.627416997969 52*(x-0.0)**3), (x, 0, 0.25)).doit().evalf(2)
print(S_0)

S_1 = Integral(0.7071067811865476 - (1.9999999999999998 * (x-0.25)) -
    (4.97056274847714*(x -0.25)**2)+ (6.627416997969518*(x-0.25)**3), (x,
     0.25, 0.5)).doit().evalf(2)
print(S_1)

S_2 = Integral(6.123233995736766e-17 - (3.242640687119285 * (x-0.5)) -
    (1.8947806286936005e-15 * (x -0.5)**2)+ (6.627416997969523*(x-0.5)
    **3), (x, 0.5, 0.75)).doit().evalf(2)
print(S_2)

S_3 = Integral(-0.7071067811865475 - (2.0 * (x-0.75)) +
    (4.970562748477141*(x -0.75)**2) - (6.627416997969521*(x-0.75)**3), (
    x, 0.75, 1)).doit().evalf(2)
print(S_3)

total = S_1 + S_2 + S_3 + S_0
print(total)
```

Segue a tabela com os resultados das integrais em cada um dos intervalos

Spline	Valor da integral	Intervalos
S_0	0.22	[0, 0.25]
S_1	0.095	[0.25, 0.5]
S_2	-0.095	[0.5, 0.75]
S_3	-0.022	[0.75, 1]

Podemos ver facilmente que, ao somar cada um desses resultados teremos o valor 0 que é o valor real da integral, o que também é obtido integrando a função original $f(x) = cos(\pi x)$ no intervalo [0, 1].

O procedimento para se obter as integrais para o segundo conjunto de ponto foi identico ao anterior, a diferença é temos mais splines (8 para sermos exatos) e portanto temos 8 integrais para implementar. Segue abaixo a integração de cada um dos splines para o segundo conjunto de pontos:

```
1
2 from sympy import *
3 import math
4
5 init_printing(pretty_print=True)
6
7 x = Symbol('x')
8
9 Integral(cos(pi*x), (x, 0, 1)).doit().evalf(2) # calcula a integral da
      funcao original com duas casas decimais.
10
11 #Segue agora o calculo das integrais para cada um dos splines
12
13 S_0 = Integral(1.0 - (0.36075774689582807 * (x-0.0)) + (0.0*(x -0.0)**2)
      - (15.885183552888194*(x-0.0)**3), (x, 0, 0.125)).doit().evalf(2)
14 print(S_0)
15
16 S_1 = Integral(0.9238795325112867-(1.1053757259374621 * (x-0.125)) -
      (5.956943832333073*(x -0.125)**2) + (7.411948440395709*(x-0.125)**3),
       (x, 0.125, 0.25)).doit().evalf(2)
17 print(S_1)
18
19 S_2 = Integral(0.7071067811865476- (2.2471766008771814 * (x-0.25)) -
      (3.1774631671846816*(x -0.25)**2) + (3.134253197030707*(x-0.25)**3),
      (x, 0.25, 0.375)).doit().evalf(2)
20 print(S_2)
21
22 S_3 = Integral(0.38268343236508984-(2.894624274062538 * (x-0.375))-
      (2.0021182182981665*(x -0.375)**2) + (5.338981915461776*(x-0.375)**3)
      , (x, 0.375, 0.5)).doit().evalf(2)
23 print(S_3)
24
25 S_4 = Integral(6.123233995736766e-17 - (3.1448890513498085 * (x-0.5)) -
      (5.94952051473768e-16*(x -0.5)**2) + (5.338981915461786*(x-0.5)**3),
      (x, 0.5, 0.625)).doit().evalf(2)
26 print(S_4)
27
28 S_5 = Integral(-0.3826834323650897- (2.8946242740625374 * (x-0.625)) +
      (2.002118218298169*(x -0.625)**2) + (3.1342531970306786*(x-0.625)**3)
```

```
    , (x, 0.625, 0.75)).doit().evalf(2)
print(S_5)

S_6 = Integral(-0.7071067811865475-(2.247176600877182 * (x-0.75)) +
    (3.1774631671846736*(x -0.75)**2) + (7.411948440395752*(x-0.75)**3),
    (x, 0.75, 0.875)).doit().evalf(2)
print(S_6)

S_7 = Integral(-0.9238795325112867- (1.1053757259374628 * (x-0.875)) +
    (5.956943832333081*(x -0.875)**2) - (15.885183552888215*(x-0.875)**3)
    , (x, 0.875, 1)).doit().evalf(2)
print(S_7)

total = S_1 + S_2 + S_3 + S_0 + S_4 + S_5 + S_6 + S_7
print(total)
```

Segue agora a tabela com os resultados das integrais para o segundo conjunto de pontos em seus respectivos intervalos.

Spline	Valor da integral	Intervalo
S_0	0.12	[0, 0.125]
S_1	0.10	[0.125, 0.25]
S_2	0.069	[0.25, 0.375]
S_3	0.024	[0.375, 0.5]
S_4	-0.024	[0.5, 0.625]
S_5	-0.069	[0.625, 0.75]
S_6	-0.10	[0.75, 0.875]
S_7	-012	[0.875, 1]

E ao fazer o somatório dos valores das integrais em cada um dos intervalos vamos ter que o resultado será 0, tal qual o cálculo da integral da função original. Ou seja, temos um resultado muito satisfatório em relação a essa interpolação, devemos salientar também que no cálculo das integrais foram consideradas apenas 2 casas decimais, caso fossem consideradas mais casas a soma das integrais daria uma valor muito próximo de zero, porém não exatamente zero, como foi o caso aqui calculado.

Para o cálculo das derivadas também foi utilizada a biblioteca simpy do python. E o processo que foi empregado foi muito semelhante ao das integrais. Segue abaixo os códigos utilizados para o cálculo das derivadas para o primeiro conjunto de pontos.

```
1
2 from sympy import *
3 import math
4
5 init_printing(pretty_print=True)
6
7 x = Symbol('x')
8
9 cos(pi * x).diff(x, 1).subs({x: 0.5}) #calcula a primeira derivada da
      funcao original no ponto 0.5.
10 cos(pi * x).diff(x, 2).subs({x: 0.5}) #calcula a segunda derivada da
      funcao original no ponto 0.5.
11
12 #Segue agora o calculo das derivadas para cada um dos splines
13
14 D_0 = (1.0 - (0.7573593128807148 * (x - 0.0)) + (0.0 * (x - 0.0)**2) -
      (6.62741699796952*(x-0.0)**3)).diff(x,1).subs({x:0.5})
15 D_0_2 = (1.0 - (0.7573593128807148 * (x - 0.0)) + (0.0 * (x - 0.0)**2) -
       (6.62741699796952*(x-0.0)**3)).diff(x,2).subs({x:0.5})
16 print("Resultado da derivada primeira: ", D_0)
17 print("Resultado da derivada segunda: ", D_0_2)
18
19 D_1 = (0.7071067811865476 - (1.9999999999999998 * (x-0.25)) -
      (4.97056274847714*(x -0.25)**2)+ (6.627416997969518*(x-0.25)**3)).
      diff(x,1).subs({x:0.5})
20 D_1_2 = (0.7071067811865476 - (1.9999999999999998 * (x-0.25)) -
      (4.97056274847714*(x -0.25)**2)+ (6.627416997969518*(x-0.25)**3)).
      diff(x,2).subs({x:0.5})
21 print("Resultado da derivada primeira: ", D_1)
22 print("Resultado da derivada segunda: ", D_1_2)
23
24 D_2 = (6.123233995736766e-17 - (3.242640687119285 * (x-0.5)) -
      (1.8947806286936005e-15 * (x -0.5)**2)+ (6.627416997969523*(x-0.5)
      **3)).diff(x,1).subs({x:0.5})
25 D_2_2 = (6.123233995736766e-17 - (3.242640687119285 * (x-0.5)) -
```

```
    (1.8947806286936005e-15 * (x -0.5)**2)+ (6.627416997969523*(x-0.5)
    **3)).diff(x,2).subs({x:0.5})
print("Resultado da derivada primeira: ", D_2)
print("Resultado da derivada segunda: ", D_2_2)

D_3 = (-0.7071067811865475 - (2.0 * (x-0.75)) + (4.970562748477141*(x
    -0.75)**2) - (6.627416997969521*(x-0.75)**3)).diff(x,1).subs({x:0.5})
D_3_2 = (-0.7071067811865475 - (2.0 * (x-0.75)) + (4.970562748477141*(x
    -0.75)**2) - (6.627416997969521*(x-0.75)**3)).diff(x,2).subs({x:0.5})
print("Resultado da derivada primeira: ", D_3)
print("Resultado da derivada segunda: ", D_3_2)
```

Segue na tabela abaixo com os resultados das derivadas em cada intervalo:

Spline	Valores de f'(x) e f"(x)	Intervalos
S_0	-5.72, -19.88	[0, 0.25]
S_1	-3.24, -3.55	[0.25, 0.5]
S_2	-3.24, -3.55	[0.5, 0.75]
S_3	-5.72, 19.88	[0.75, 1]

Agora seguem os códigos utilizados para calcular as derivadas primeira e segunda para o segundo conjunto de pontos:

```

from sympy import *
import math

init_printing(pretty_print=True)

x = Symbol('x')

cos(pi * x).diff(x, 1).subs({x: 0.5}) #calcula a primeira derivada da
    funcao original no ponto 0.5.
cos(pi * x).diff(x, 2).subs({x: 0.5}) #calcula a segunda derivada da
    funcao original no ponto 0.5.

#Segue agora o calculo das derivadas para cada um dos splines

D_0 = (1.0 - (0.36075774689582807 * (x-0.0)) + (0.0*(x -0.0)**2)-
    (15.885183552888194*(x-0.0)**3)).diff(x, 1).subs({x: 0.5})
```

```
15 D_0_2 = (1.0 - (0.36075774689582807 * (x-0.0)) + (0.0*(x -0.0)**2)-
       (15.885183552888194*(x-0.0)**3)).diff(x, 2).subs({x: 0.5})
16 print("Resultado da derivada primeira: ", D_0)
17 print("Resultado da derivada segunda: ", D_0_2)
18
19 D_1 = (0.9238795325112867-(1.1053757259374621 * (x-0.125)) -
       (5.956943832333073*(x -0.125)**2) + (7.411948440395709*(x-0.125)**3))
       .diff(x, 1).subs({x: 0.5})
20 D_1_2 = (0.9238795325112867-(1.1053757259374621 * (x-0.125)) -
       (5.956943832333073*(x -0.125)**2) + (7.411948440395709*(x-0.125)**3))
       .diff(x, 2).subs({x: 0.5})
21 print("Resultado da derivada primeira: ", D_1)
22 print("Resultado da derivada segunda: ", D_1_2)
23
24
25 D_2 = (0.7071067811865476- (2.2471766008771814 * (x-0.25)) -
       (3.1774631671846816*(x -0.25)**2) + (3.134253197030707*(x-0.25)**3)).
       diff(x, 1).subs({x: 0.5})
26 D_2_2 = (0.7071067811865476- (2.2471766008771814 * (x-0.25)) -
       (3.1774631671846816*(x -0.25)**2) + (3.134253197030707*(x-0.25)**3)).
       diff(x, 2).subs({x: 0.5})
27 print("Resultado da derivada primeira: ", D_2)
28 print("Resultado da derivada segunda: ", D_2_2)
29
30 D_3 = (0.38268343236508984-(2.894624274062538 * (x-0.375))-
       (2.0021182182981665*(x -0.375)**2) + (5.338981915461776*(x-0.375)**3)
       ).diff(x, 1).subs({x: 0.5})
31 D_3_2 = (0.38268343236508984-(2.894624274062538 * (x-0.375))-
       (2.0021182182981665*(x -0.375)**2) + (5.338981915461776*(x-0.375)**3)
       ).diff(x, 2).subs({x: 0.5})
32 print("Resultado da derivada primeira: ", D_3)
33 print("Resultado da derivada segunda: ", D_3_2)
34
35 D_4 = (6.123233995736766e-17 - (3.1448890513498085 * (x-0.5)) -
       (5.94952051473768e-16*(x -0.5)**2) + (5.338981915461786*(x-0.5)**3)).
       diff(x, 1).subs({x: 0.5})
36 D_4_2 = (6.123233995736766e-17 - (3.1448890513498085 * (x-0.5)) -
       (5.94952051473768e-16*(x -0.5)**2) + (5.338981915461786*(x-0.5)**3)).
       diff(x, 2).subs({x: 0.5})
```

```
print("Resultado da derivada primeira: ", D_4)
print("Resultado da derivada segunda: ", D_4_2)


D_5 = (-0.3826834323650897- (2.8946242740625374 * (x-0.625)) +
    (2.002118218298169*(x -0.625)**2) + (3.1342531970306786*(x-0.625)**3)
    ).diff(x, 1).subs({x: 0.5})
D_5_2 = (-0.3826834323650897- (2.8946242740625374 * (x-0.625)) +
    (2.002118218298169*(x -0.625)**2) + (3.1342531970306786*(x-0.625)**3)
    ).diff(x, 2).subs({x: 0.5})
print("Resultado da derivada primeira: ", D_5)
print("Resultado da derivada segunda: ", D_5_2)

D_6 = (-0.7071067811865475-(2.247176600877182 * (x-0.75)) +
    (3.1774631671846736*(x -0.75)**2) + (7.411948440395752*(x-0.75)**3)).
    diff(x, 1).subs({x: 0.5})
D_6_2 = (-0.7071067811865475-(2.247176600877182 * (x-0.75)) +
    (3.1774631671846736*(x -0.75)**2) + (7.411948440395752*(x-0.75)**3)).
    diff(x, 2).subs({x: 0.5})
print("Resultado da derivada primeira: ", D_6)
print("Resultado da derivada segunda: ", D_6_2)

D_7 = (-0.9238795325112867- (1.1053757259374628 * (x-0.875)) +
    (5.956943832333081*(x -0.875)**2) - (15.885183552888215*(x-0.875)**3)
    ).diff(x, 1).subs({x: 0.5})
D_7_2 = (-0.9238795325112867- (1.1053757259374628 * (x-0.875)) +
    (5.956943832333081*(x -0.875)**2) - (15.885183552888215*(x-0.875)**3)
    ).diff(x, 2).subs({x: 0.5})
print("Resultado da derivada primeira: ", D_7)
print("Resultado da derivada segunda: ", D_7_2)
```

E como de constume, segue abaixo a tabela com os resultados das derivadas primeira e segunda para o segundo conjunto de pontos:

Spline	Valores de f'(x) e f"(x)	Intervalo
S_0	-12.27, -47.65	[0, 0.125]
S_1	-2.44, 4.76	[0.125, 0.25]
S_2	-3.24, -1.65	[0.25, 0.375]
S_3	-3.14, 0	[0.375, 0.5]
S_4	-3.14, 0	[0.5, 0.625]
S_5	-3.24, 1.65	[0.625, 0.75]
S_6	-2.44, -4.76	[0.75, 0.875]
S_7	-12.27, 47.65	[0.875, 1]

Para as derivadas temos um resultado não muito satisfatório. Apenas alguns dos splines conseguem ter derivadas primeira e segunda próximas ao do valor da função original. Os splines S_3 e S_4 são os que apresentam esses valores satisfatórios, todos os demais apresentam derivadas primeira e segunda com um grande erro associado.

Os erros variam entre 3.18% a 82.1% para o primeiro conjunto de pontos e 0% a 290% para o segundo conjunto de pontos, quando comparadas com o valor real da primeira derivada. Para a segunda derivada temos um erro indo de 355% a 1988% para o primeiro conjunto de pontos e 0% a 4765% para o segundo conjunto de pontos.

2.4 Spline Fixo: Implementação do método

O método de interpolação por Spline fixo é muito semelhante ao natural, a diferença é que com o spline fixo as condições de contorno são definidas em termos das derivadas da função $f(x)$, com isso, o vetor B e a matriz A do sistemas $Ax = B$ vão sofrer algumas alterações. As alterações ocorrem na primeira e última linha da matriz A e também no primeiro e último elemento do vetor B e essas mudanças seguem da seguinte forma:

O primeiro e os demais elementos da primeira linha da matriz do spline fixo é da dado por $A[0][0] = 1, \quad A[0][n] = 0$. Agora para o spline fixo vamos ter que: $A[0][0] = 2h_0, \quad A[0][1] = h_0, \quad A[0][n-2] = 0$. Essas são as mudanças para a matriz A, para o vetor B, o primeiro elemento vai ser agora dado por:

$$B[0] = \frac{3}{h_0}(a_1 - a_0) - 3f'(a) \tag{10}$$

$$B[n-1] = 3f'(b) - \frac{3}{h_{n-1}}(a_n - a_{n-1}) \tag{11}$$

Com isso, os códigos apresentados acima, para a implementação do spline natural, terão pequenas alterações. Essas alterações estão indicadas abaixo:

```
1
2 import numpy as np
3 import math
4
5 def splinefixo(x, y): Funcao para obter os splines
6
7
8   n = len(x)
9   a = {k: v for k, v in enumerate(y)}
10  h = {k: x[k+1] - x[k] for k in range(n - 1)}
11
12  A = [[2*h[0], h[0]] + [0] * (n - 2)] # Adiciona a primeira linha da
     matriz A
13  for i in range(1, n - 1): #adiciona as outras linhas da matriz A
14    linha = [0] * n
15    linha[i-1] = h[i-1]
16    linha[i] = 2*(h[i-1] + h[i])
17    linha[i + 1] = h[i]
18    A.append(linha)
19
20  A.append([0]*(n-2) + [h[n-2], 2*h[n-2]]) # Adiciona a ultima linha da
     matriz A
21
22
23  B.append((3 * (a[1]-a[0])) / h[0] - 3 * math.pi*math.sin(math.pi*x[0])
     ) # Adiciona o primeiro elemento do vetor B
24
25  for k in range(1, n-1):
26    valor = 3*(a[k+1] -a[k]) / h[k] - 3*(a[k] - a[k-1]) / h[k-1] #
     adiciona os outros elementos do vetor B
27    B.append(valor)
28  B.append(-3*(math.pi)*(math.sin(math.pi*x[n-1])) - (3/h[n-2])*(a[n-1]-
     a[n-2])) # Adiciona o ultimo elemnto do vetor B
```

```
29
30    c = dict(zip(range(n), np.linalg.solve(A, B))) # Resolve o sistema Ax=
       B
31
32    b = {}
33    d = {}
34    for k in range(n - 1): # Calcula os coeficientes b e d
35      b[k] = (1/h[k]) * (a[k + 1] - a[k]) - (h[k]/3) * (2*c[k]+c[k+1])
36      d[k] = (c[k+1] - c[k])/ (3 * h[k])
37
38
39    s = {}
40
41    for k in range(n - 1):
42      eq = f'{a[k]}{b[k]:+} * (x{-x[k]:+}){c[k]:+}*(x {- x[k]:+})**2{d[k
       ]:+}*(x{-x[k]:+})**3'
43      s[k] = {'eqs': eq, 'dominio': [x[k], x[k+1]]}
44
45    for k, v in s.items():
46      print(f'S{k}', v)
47
48
49    return s
50
51 x_1 = [0., 0.25, 0.5, 0.75, 1.]
52 x_2 = [0., 0.125, 0.250, 0.375, 0.5, 0.625, 0.75, 0.875, 1.]
53 y_1 = [math.cos(math.pi*i) for i in x_1]
54 y_2 = [math.cos(math.pi*i) for i in x_2]
55
56
57 eq_1 = splinefixo() # aqui vao os valores dos pontos
58 print("\n")
```

E da mesma forma que o spline natural, esse código gerou os coeficientes e com eles é possível também montar os polinômios. Segue a saída desse código, com os polinômios montados usando o spline fixo para o primeiro conjunto de pontos:

E abaixo a tabela com os coeficientes dos polinômios do spline:

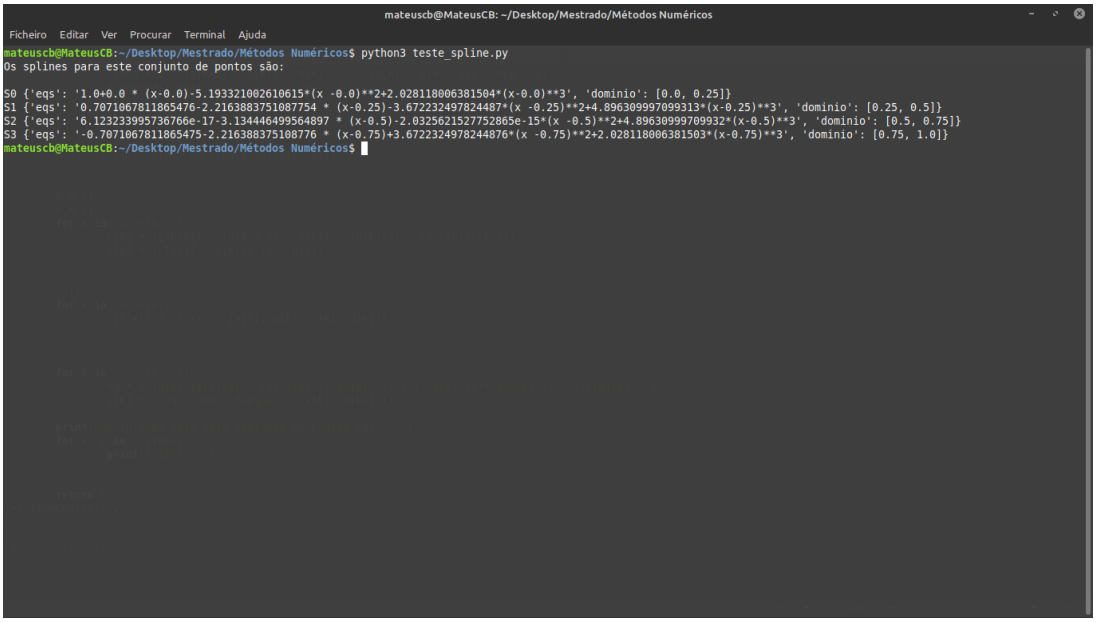

Figura 3: Terminal de execução dos splines fixo para o primeiro conjunto de pontos

a_j	b_j	c_j	d_j
1.0	0.0	5.193	2.028
0.7071	2.216	3.672	4.896
6.12×10^{-17}	3.134	2.032×10^{-15}	4.896
-0.7071	2.2163	3.672	2.028

E agora segue o terminal e a tabela com os valores dos coeficientes para o segundo conjunto de pontos:

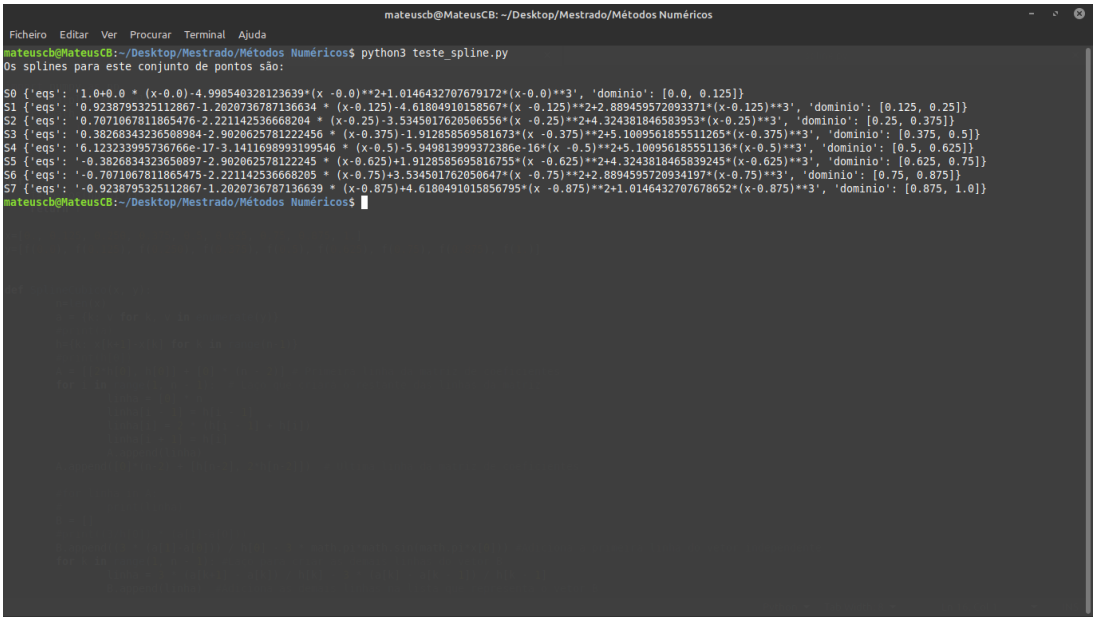

Figura 4: Terminal de execução dos splines fixo para o segundo conjunto de pontos

a_j	b_j	c_j	d_j
1.0	0.0	4.998	1.0146
0.9238	1.202	4.618	2.889
0.7071	2.221	3.534	4.324
0.3826	2.9020	1.912	5.1009
6.123×10^{-17}	3.141	5.949	5.1009
-0.382	2.8946	1.912	4.324
-0.707	2.2471	3.534	2.889
-0.923	1.1053	4.618	1.0146

2.5 Spline Fixo: Integrais e derivadas

O processo de obtenção das integrais e derivadas no caso do spline fixo foi exatamente igual ao do spline natural, a diferença obviamente é que foi integrado e derivado polinômios diferentes (Já que os coeficientes dos polinômios gerados no spline natural são diferentes dos gerados no splie fixo). Por conta disso, os códigos para fazer a integração e derivação não precisam ser expostos aqui novamente, com isso, segue as tabelas para o resultados das integrais e derivadas para o primeiro e segundo conjunto de pontos, respectivamente:

Spline	Valor da integral	Intervalos
S_0	0.22	[0, 0.25]
S_1	0.093	[0.25, 0.5]
S_2	-0.093	[0.5, 0.75]
S_3	-0.022	[0.75, 1]

Spline	Valor da integral	Intervalo
S_0	0.12	[0, 0.125]
S_1	0.10	[0.125, 0.25]
S_2	0.069	[0.25, 0.375]
S_3	0.024	[0.375, 0.5]
S_4	-0.024	[0.5, 0.625]
S_5	-0.069	[0.625, 0.75]
S_6	-0.10	[0.75, 0.875]
S_7	-012	[0.875, 1]

Os resultados das integrais para esse spline também são muito satisfatórios e identicos aos resultados obtidos através do spline natural. Aqui também temos a soma de todas as integrais convergindo para zero, o que é um bom sinal já que é o resultado da integral da função $f(x) = cos(\pi x)$ no intervalo [0, 1], e isso ocorre para os dois conjuntos de pontos.

Spline	Valores de f'(x) e f"(x)	Intervalos
S_0	-3.67, -4.30	[0, 0.25]
S_1	-3.13, -3.55$\times 10^{-15}$	[0.25, 0.5]
S_2	-3.13, -3.55$\times 10^{-15}$	[0.5, 0.75]
S_3	-3.67, 4.30	[0.75, 1]

Spline	Valores de f'(x) e f"(x)	Intervalo
S_0	-4.23, -6.95	[0, 0.125]
S_1	-3.44, -2.73	[0.125, 0.25]
S_2	-3.17, -0.58	[0.25, 0.375]
S_3	-3.14, -1.77$\times 10^{-15}$	[0.375, 0.5]
S_4	-3.17, 0.58	[0.5, 0.625]
S_5	-3.17, 0.58	[0.625, 0.75]
S_6	-3.44, 2.73	[0.75, 0.875]
S_7	-4.23, 6.95	[0.875, 1]

Já para as derivadas, o mau comportamento ainda permanece. Temos uma boa aproximação do valor real da derivada apenas nos splines S_3 para o segundo conjunto de pontos, porém o erro associado as derivadas obtidas dos splines fixos são menores do que os erros das derivadas obtidas através do spline natural.

Para o primeiro conjunto de pontos o erro vai de 0.03% a 16.8% para a primeira derivada e 0% a 430% para a segunda derivada. No segundo conjunto de pontos nós temos o erro indo de 0% até 34.7% para a primeira derivada e 0.3% a 695% para a segunda.

De qualquer forma, os resultados das derivadas para os dois conjuntos de pontos não são satisfatórios.

3 Conclusões

Dos resultados obtidos, podemos ver que a interpolação por splines cubicos é uma boa opção para este tipo de procedimento, os gráficos a seguir mostram as interpolações naturais e fixas para os dois conjunto de pontos:

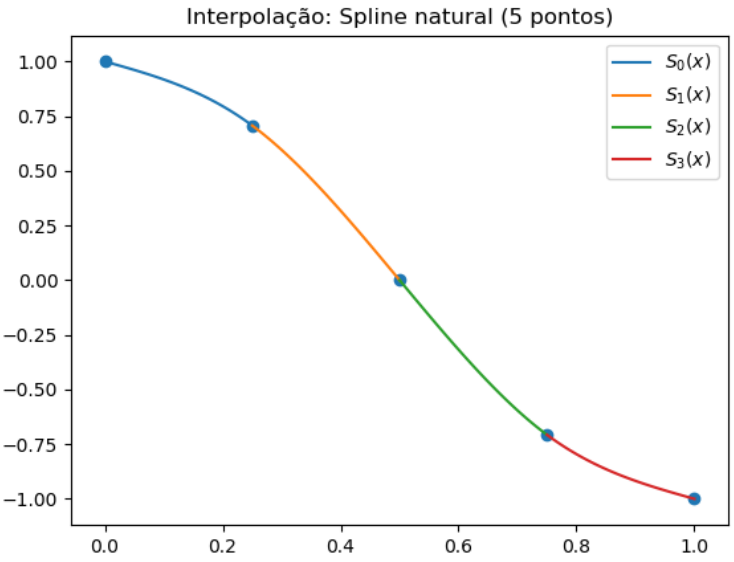

Figura 5: Interpolação por spline natural para o primeiro conjunto de pontos

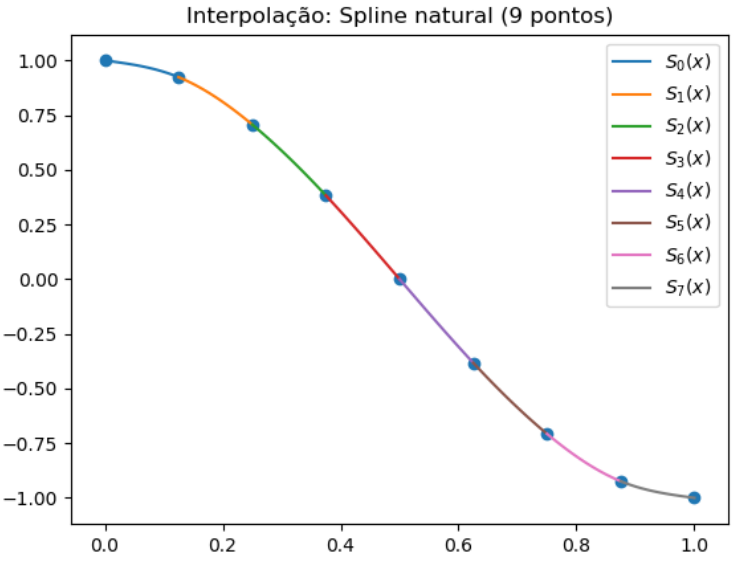

Figura 6: Interpolação por spline natural para o segundo conjunto de pontos

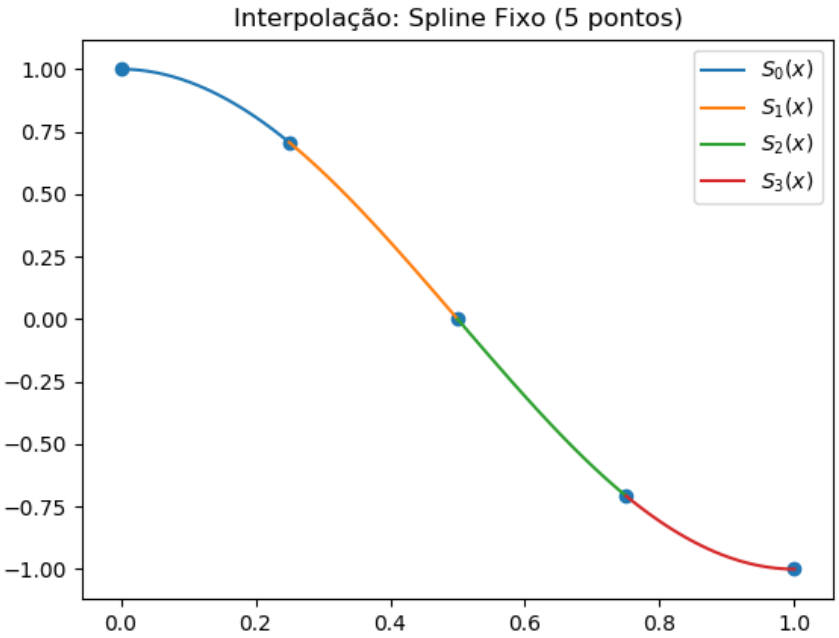

Figura 7: Interpolação por spline fixo para o primeiro conjunto de pontos

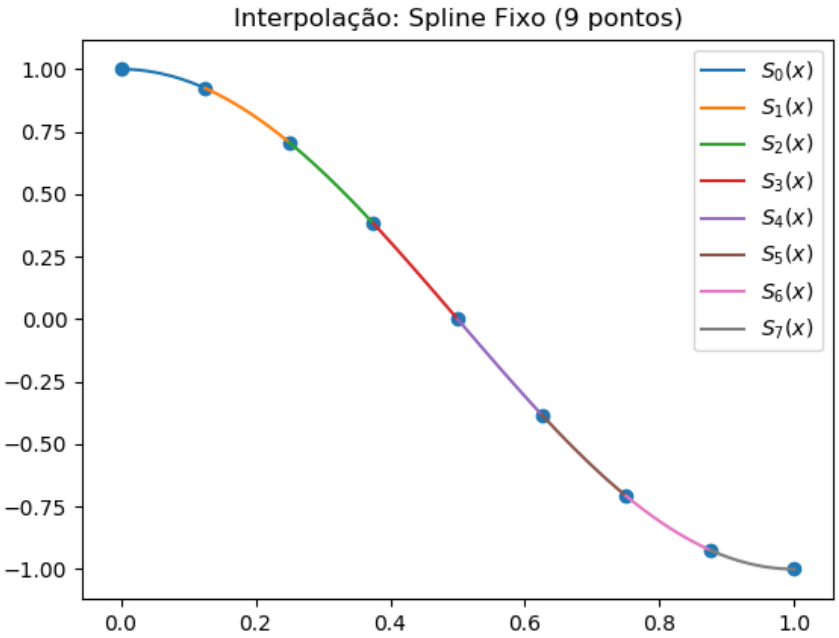

Figura 8: Interpolação por spline fixo para o segundo conjunto de pontos

Em relação aos valores das integrais e derivadas, os splines cúbicos fixados apresentam uma melhor aproximação quando comparados com os splines naturais. Vemos que temos uma aproximação muito melhor das derivadas nesse spline do que no spline cúbico natural, mas em relação as integrais os dois são satisfatórios, já que ambos conseguem convergir para o valor real da integral da função $f(x) = cos(\pi x)$.