UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO
INE5202 - CÁLCULO NÚMERICO EM COMPUTADORES

LEONARDO DE SOUSA MARQUES

RELATÓRIO II: FENÔMENO DE RUNGE

Docente:

Drª. Priscila Cardoso Calegari

Florianópolis
2024

Sumário

Lista de tabelas

1 INTRODUÇÃO

O presente relatório tem como objetivo apresentar soluções aproximadas para um problema que ocorre quando tentamos fazer a interpolação polinomial de uma função utilizando pontos igualmente espaçados, conhecido como **Fenômeno de Runge**. Esse fenômeno se caracteriza pela crescente oscilação do polinômio de interpolação, especialmente nas extremidades do intervalo, à medida que o número de pontos de interpolação aumenta. Esse comportamento ocorre mesmo que a função original seja suave e sem descontinuidades, como é o caso da função de Runge $f(x) = \frac{1}{1+25x^2}$.

Sendo assim, o objetivo deste trabalho é realizar a interpolação polinomial da função f no intervalo $[a, b] = [-5, 5]$, utilizando $N + 1$ pontos igualmente espaçados, incluindo as extremidades do intervalo. Para os testes, consideraremos os valores $N \in \{5, 10, 20\}$, com o intuito de demonstrar que, à medida que aumentamos o número de pontos — e, consequentemente, o grau do polinômio interpolador — o erro na interpolação tende a crescer. A fim de contornar esse erro, vamos, então, implementar uma solução que faz uso de splines lineares e cúbicas.

2 SPLINES: DEFINIÇÃO

Splines são funções matemáticas utilizadas para aproximar dados ou construir curvas suaves que passam por um conjunto de pontos dados. Ao contrário da interpolação polinomial, que usa um único polinômio de grau elevado, as splines utilizam um conjunto de polinômios de baixo grau, conectados de forma contínua, diminuindo as oscilações.

2.1 Spline Linear

Uma spline linear é uma função composta por segmentos de reta, que interpola os pontos dados de maneira que a função seja contínua. Ou seja, entre dois pontos consecutivos, a spline será representada por uma reta.

2.2 Spline Cúbica

A spline cúbica, por sua vez, utiliza polinômios de grau três entre os pontos dados. Diferentemente da spline linear, ela proporciona uma suavidade maior, pois além de ser contínua, também possui derivadas contínuas até a segunda ordem.

3 IMPLEMENTAÇÃO DO MODELO

Para implementar as soluções, vamos utilizar três bibliotecas fundamentais: **numpy**, para a resolução de sistemas algébricos lineares; **scipy**, para realizar a interpolação utilizando splines lineares e cúbicas; e **matplotlib**, para a geração e exibição dos gráficos. Portanto, fazemos as seguintes importações:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.interpolate import interp1d
```

Com isso, podemos implementar as funções básicas que retornam o valor de $f(x)$, os valores de x com o intervalo dividido em N partes e o erro máximo cometido, quando comparamos a função com os valores obtidos através da interpolação.

```
1  ...
2
3  # Funcao original
4  def f(x):
5      return 1 / (1 + 25 * x**2)
6
7  # Funcao de erro maximo (||f(z) - p(z)||)
8  def calcular_erro(f_values, p_values):
9      return np.max(np.abs(f_values - p_values))
10
11 # Divisao do intervalo
12 def divisao_do_intervalo(a, b, N):
13     total = b - a
14     distancia = total / N
15     x_vals = np.zeros(N + 1)
16     x_vals[0], x_vals[N] = a, b
17     for i in range(1, N):
18         x_vals[i] = x_vals[i - 1] + distancia
19     return x_vals
20
21 ...
```

3.1 INTERPOLAÇÃO POLINOMIAL

A interpolação polinomial tem como objetivo aproximar uma função $f(x)$ por meio de um polinômio de grau N que passa pelos pontos $(x_0, f(x_0)), (x_1, f(x_1)), \ldots, (x_N, f(x_N))$. O polinômio interpolador é dado por:

$$P_N(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_N x^N$$

onde $c_0, c_1, \ldots, c_N$ são os coeficientes que devemos determinar para que o polinômio passe pelos pontos dados.

A matriz A é construída a partir dos valores x_i, onde $i \in \{0, 1, 2, \ldots, N\}$, e é uma matriz $(N+1) \times (N+1)$ que contém as potências de x. Cada linha da matriz A corresponde a um valor de x_i e cada coluna corresponde ao respectivo poder de x.

A matriz A para N pontos igualmente espaçados $x_0, x_1, \ldots, x_N$ é dada por:

$$A = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^N \\ 1 & x_1 & x_1^2 & \cdots & x_1^N \\ 1 & x_2 & x_2^2 & \cdots & x_2^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^N \end{pmatrix}$$

Cada entrada $A[i][j]$ da matriz é calculada como x_i^j, onde i é o índice do ponto e j é o índice da coluna correspondente ao grau do polinômio. Após a construção dessa matriz, os coeficientes $c_0, c_1, \ldots, c_N$ são obtidos resolvendo o sistema linear $A \cdot c = y$, onde y é o vetor contendo os valores $y_i = f(x_i)$ para os pontos dados.

3.1.1 Exemplo de Matriz A

Tomando como exemplo $N = 5$ e os valores $\vec{x} = [-5, -3, -1, 1, 3, 5]$, a matriz A terá o seguinte formato:

$$A = \begin{pmatrix} 1 & -5 & 25 & -125 & 625 & -3125 \\ 1 & -3 & 9 & -27 & 81 & -243 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 3 & 9 & 27 & 81 & 243 \\ 1 & 5 & 25 & 125 & 625 & 3125 \end{pmatrix}$$

3.1.2 Solução do Sistema Linear

A solução para os coeficientes $c_0, c_1, \ldots, c_5$ é obtida resolvendo o sistema linear $A \cdot c = y$, onde y é o vetor de valores da função $f(x)$ nos pontos x_i:

$$y = \begin{pmatrix} f(-5) \\ f(-3) \\ f(-1) \\ f(1) \\ f(3) \\ f(5) \end{pmatrix}$$

Após encontrar os coeficientes c, o polinômio interpolador pode ser avaliado em novos pontos z.

No código Python, a matriz A é construída no seguinte trecho:

```
A = np.zeros((N + 1, N + 1))  # Matriz A
for j in range(N + 1):
    for i in range(N + 1):
        A[i][j] = x_vals[i]**j
```

Aqui, a matriz A é preenchida com os valores de x_i elevados aos respectivos poderes. Após isso, o sistema linear é resolvido para encontrar os coeficientes c, utilizando o módulo **linalg** do numpy:

```
c = np.linalg.solve(A, y_vals)
```

Uma vez encontrados os coeficientes, o polinômio interpolador pode ser avaliado para novos pontos z através da função:

```
def polin(x, c):
    y = c[0]
    n = len(c)
    for i in range(1, n):
        y += c[i] * x**i
    return y
```

Esse processo foi repetido para diferentes valores de N, permitindo comparar os erros de interpolação para diferentes números de pontos.

3.1.3 Gráficos e cálculo do erro

A fim de observarmos o comportamento do polinômio gerado, é necessário plotar os gráficos. Outra forma de mensurar a precisão do polinômio é realizando o cálculo do erro, definido no código pela função **calcula_erro(f_values, p_values)**, vista anteriormente, visto que temos a função f. Para calcular o erro máximo, utilizaremos a fórmula abaixo, que define o erro como sendo a diferença entre o valor da função $f(z_i)$ e o valor interpolado $p_N(z_i)$ nos pontos z_i:

$$\text{erro} = \max_i |f(z_i) - p_N(z_i)|$$

Os pontos z são dados pelo conjunto de pontos formados por $z_i = a + h \cdot i$, com $h = \frac{b-a}{50}$, onde i varia entre 0 e 50. Esses pontos são utilizados para avaliar a precisão da interpolação em uma sequência de amostras uniformemente distribuídas no intervalo $[a, b]$.

Dessa forma, o erro máximo nos dá uma medida quantitativa da precisão do polinômio de interpolação para os pontos de amostragem z_i. Esse valor é fundamental para entendermos como o polinômio se aproxima ou se afasta da função real f, especialmente à medida que o número de pontos de interpolação aumenta.

Com isso, em sequência, apresentamos os gráficos e erros gerados para $N = 5, 10, 20$.

3.1.4 Gráfico de polinômio para $N = 5$

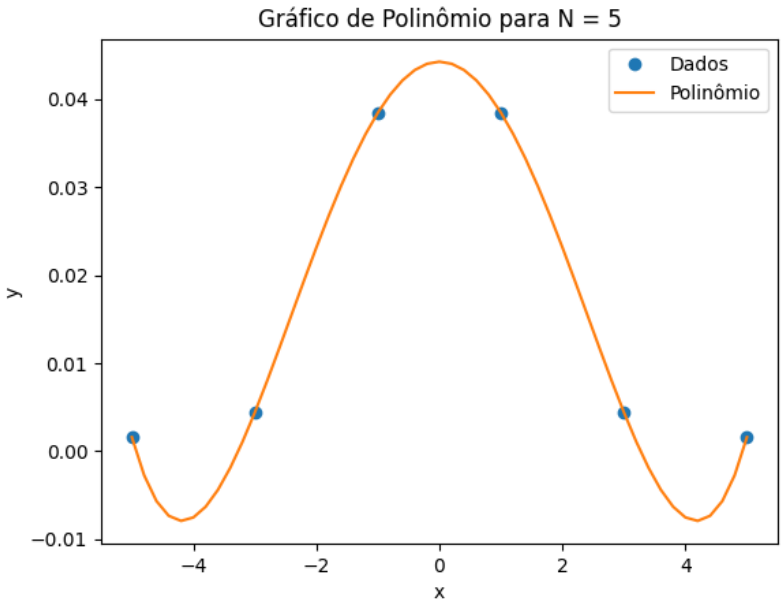

Figura 1

Para esse caso, aplicando na função **calcula_erro**, teremos o valor $erro = 0.9557546591213079$. Percebemos que é um valor relativamente alto para o erro, pois queremos que ele seja o mais próximo de zero possível.

3.1.5 Gráfico de polinômio para $N = 10$

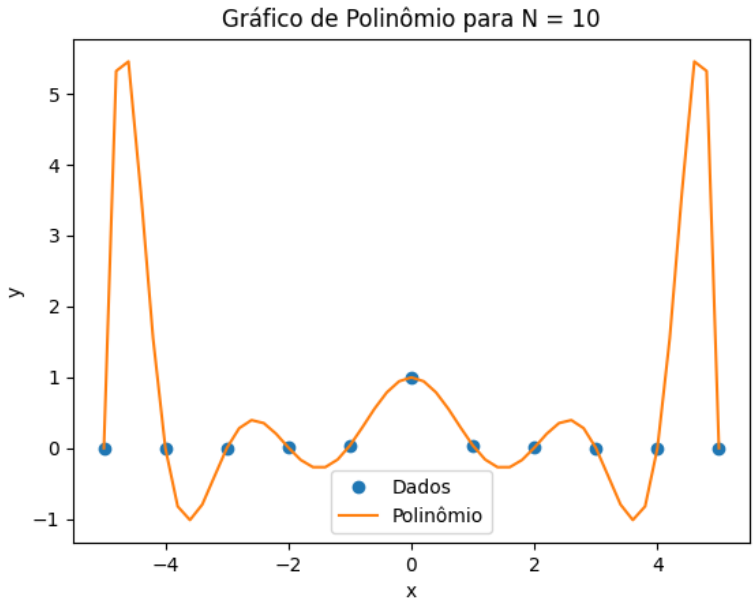

Figura 2

Aplicando na função **calcula_erro**, teremos o valor $erro = 5.453562457215162$. Percebemos um aumento do erro conforme o grau do polinômio aumenta, conforme o Fenômeno de Runge indica.

3.1.6 Gráfico de polinômio para $N = 20$

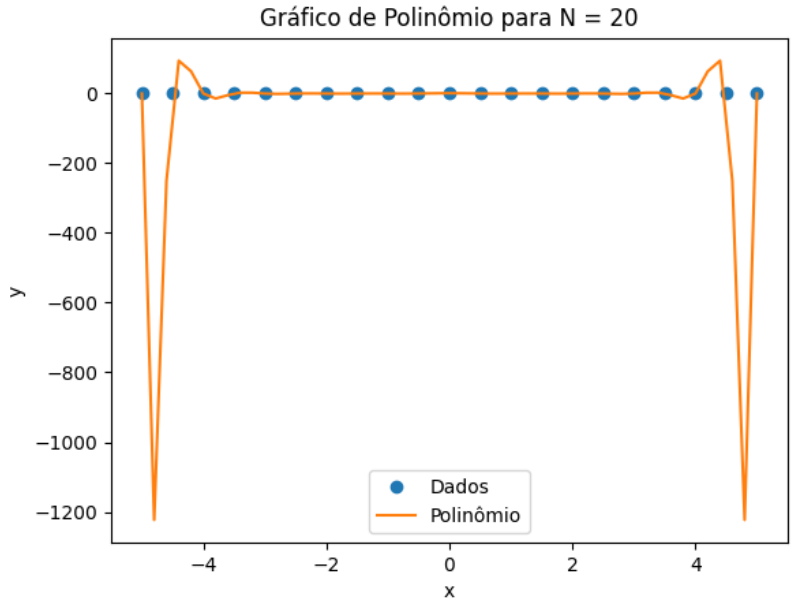

Figura 3

Aplicando na função **calcula_erro**, teremos o valor $erro = 1223.0841051674208$. Nesse caso, já se torna inviável utilizar a interpolação polinomial, visto que o erro é exorbitante.

3.1.7 Conclusões sobre a interpolação polinomial

Notamos que conforme o grau de p_N aumenta, o erro se torna muito grande. Portanto, a fim de contornar esse problema, vamos buscar interpolar a função f utilizando splines lineares e cúbicas.

3.2 INTERPOLAÇÃO COM SPLINE LINEAR

Para realizar a implementação, vamos utilizar o módulo **interpolate** da biblioteca **scipy**. Mais especificamente, vamos utilizar o método `interp1d`, que permite realizar interpolação de dados em uma dimensão. O método `interp1d` é versátil e oferece a opção de usar diferentes tipos de interpolação.

No caso da interpolação *linear*, o método conecta os pontos de dados adjacentes com segmentos de reta. Isso significa que, para cada intervalo entre dois pontos consecutivos, a interpolação retorna uma linha reta conectando esses pontos. No código, definimos a função **interpolacao_spline_linear(x_vals, y_vals, z_vals)**:

```
# Interpolacao linear usando interp1d
def interpolacao_spline_linear(x_vals, y_vals, z_vals):
    spline = interp1d(x_vals, y_vals, kind='linear')
    p_vals = spline(z_vals)
    return p_vals
```

Onde:

- `x_vals` são os pontos de amostragem no eixo x,
- `y_vals` são os valores correspondentes da função nos pontos de amostragem $y = f(x)$,
- O parâmetro `kind='linear'` especifica que a interpolação será feita por meio de segmentos lineares.
- `z_vals` são os valores de z para realização dos testes após obtenção da spline.

Para ilustrar o comportamento dessa interpolação, vamos criar o gráfico das retas e realizar o cálculo do erro para cada valor de N.

3.2.1 Gráficos de spline linear para $N = 5$

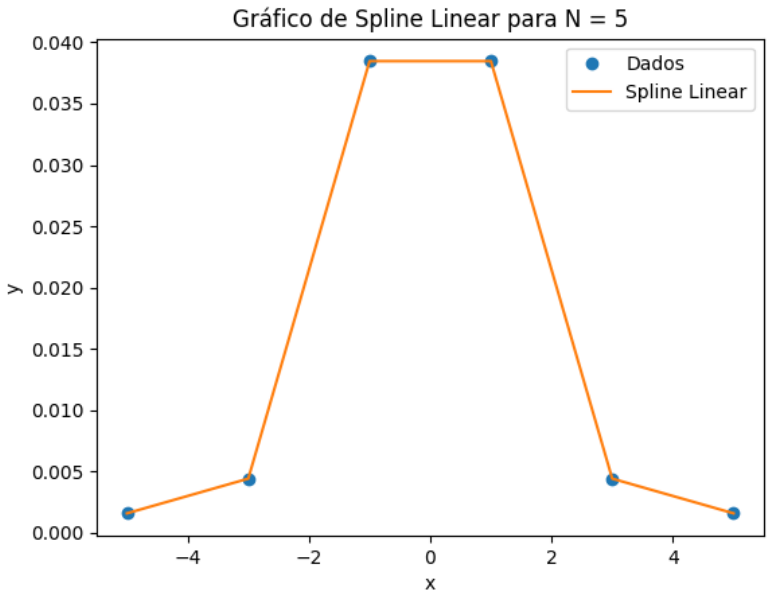

Figura 4

Realizando o calculo do erro, obtemos $erro = 0.9615384615384616$. Percebe-se que em relação a interpolação por polinômios, temos um aumento percentual de 0.605% no erro. Isso se dá pois o número de pontos ainda é pequeno, então não esperamos encontrar mudanças bruscas nos erros.

3.2.2 Gráficos de spline linear para $N = 10$

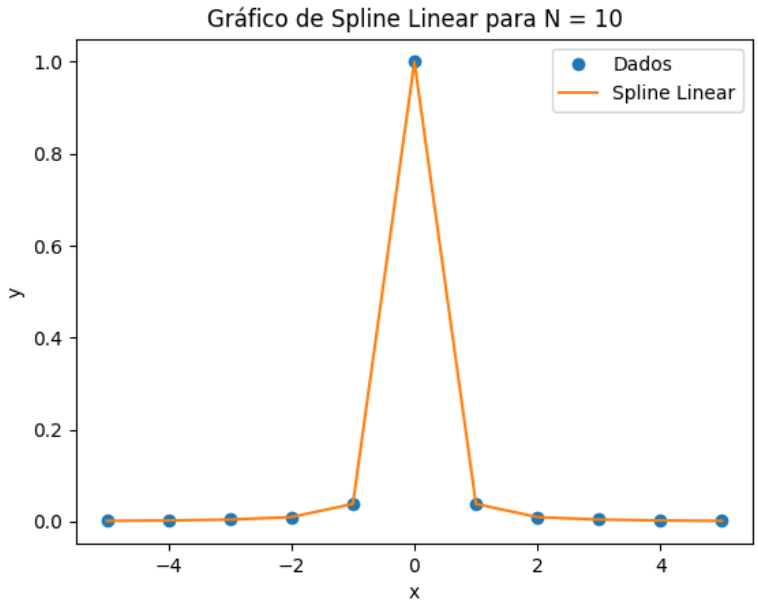

Figura 5

Para $N = 10$, ao calcular o erro obtemos um valor de 0.41538461538461546. Quando comparado ao erro ocasionado pelo polinômio interpolador, obtemos uma decrescimento de aproximadamente 92.38%. Observa-se que a spline linear fornece, então, uma aproximação muito melhor da função, visto que o número de pontos aumentou.

3.2.3 Gráficos de spline linear para $N = 20$

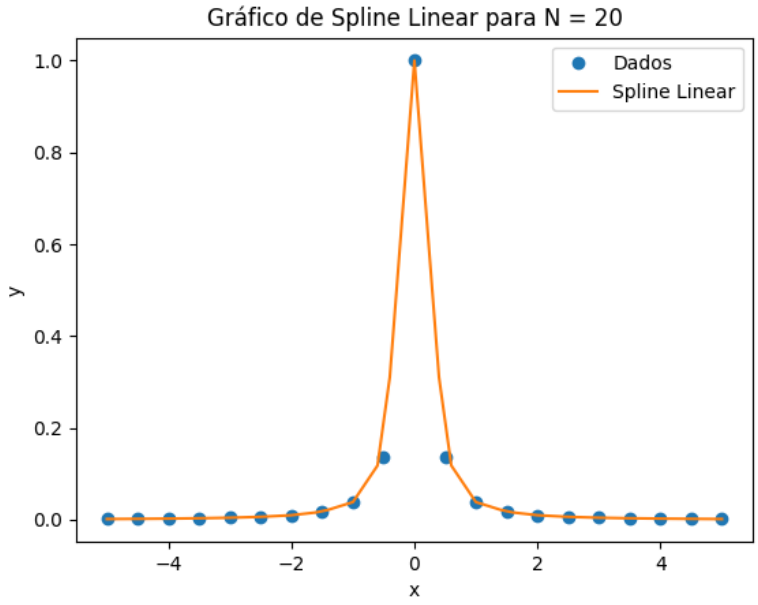

Figura 6

Por fim, ao calcular o erro para $N = 20$ da spline linear, obtivemos $erro = 0.15517241379310365$, que representa um decrescimento de 99.98% em relação à interpolação polinomial. Esse é o melhor caso para observamos os impactos do Fenômeno de Runge.

3.2.4 Conclusões sobre a interpolação com splines lineares

Observamos que conforme o número de pontos aumenta, o efeito provocado pela interpolação por splines é contrário ao da interpolação por polinômios. Apesar de que os valores de erro ainda não estão completamente próximos de zero, temos uma grande taxa de descrescimento na maioria dos testes.

3.3 INTERPOLAÇÃO COM SPLINE CÚBICA

A implementação é semelhante à interpolação com spline linear, mas precisamos alterar o parâmetro da função **interp1d** para **kind = 'cubic'**.

```
1 # Interpolacao cubica usando interp1d
2 def interpolacao_spline_cubica(x_vals, y_vals, z_vals):
3     spline = interp1d(x_vals, y_vals, kind='cubic')
4     p_vals = spline(z_vals)
5     return p_vals
```

Tendo feito isso, podemos partir para a confecção dos gráficos e cálculos dos erros associados.

3.3.1 Gráficos de spline cúbica para $N = 5$

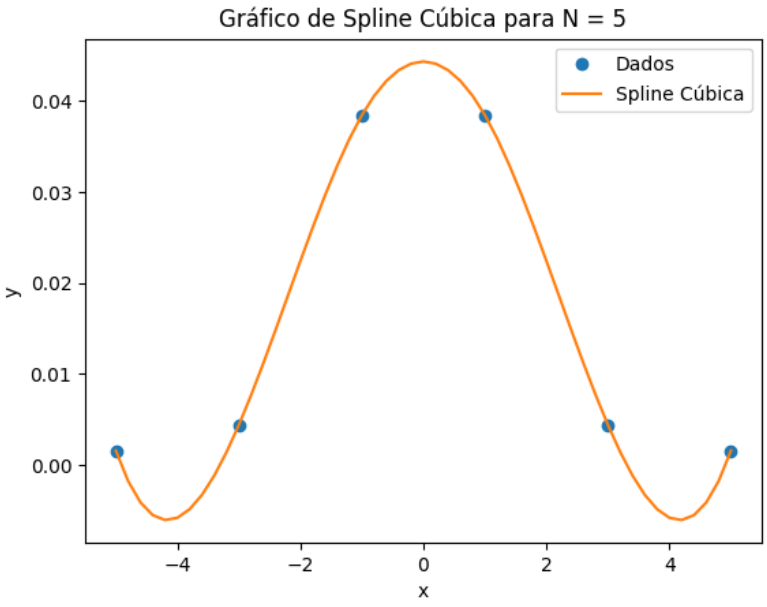

Figura 7

Observamos que a curvatura da função é muito similar a do polinômio interpolador, mas elas não são exatamente iguais, uma vez que o cálculo do erro para essa spline cúbica é igual a $erro = 0.9556527119576683$. Tal valor representa um decrescimento de 0.0107% em relação ao erro da interpolação polinomial. Novamente, pelo fato de serem poucos pontos, não espera-se encontrar diferenças significativas entre os métodos.

3.3.2 Gráficos de spline cúbica para $N = 10$

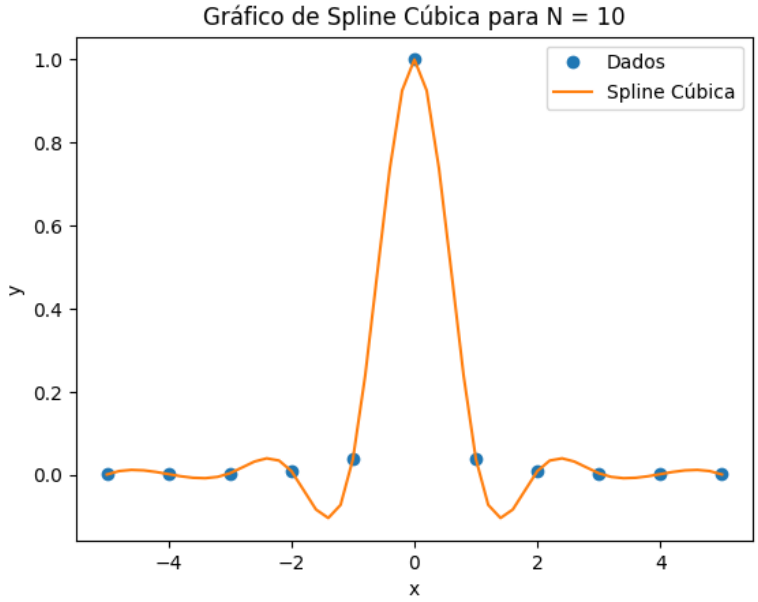

Figura 8

Já para $N = 10$, percebe-se uma grande diferença dos gráficos de cada interpolação (polinomial, linear e cúbica). Em relação ao erro, temos um valor de 0.5373013829127719, que representa um decrescimento de 90.15% em relação ao erro ocasionado pelo polinômio.

3.3.3 Gráficos de spline cúbica para $N = 10$

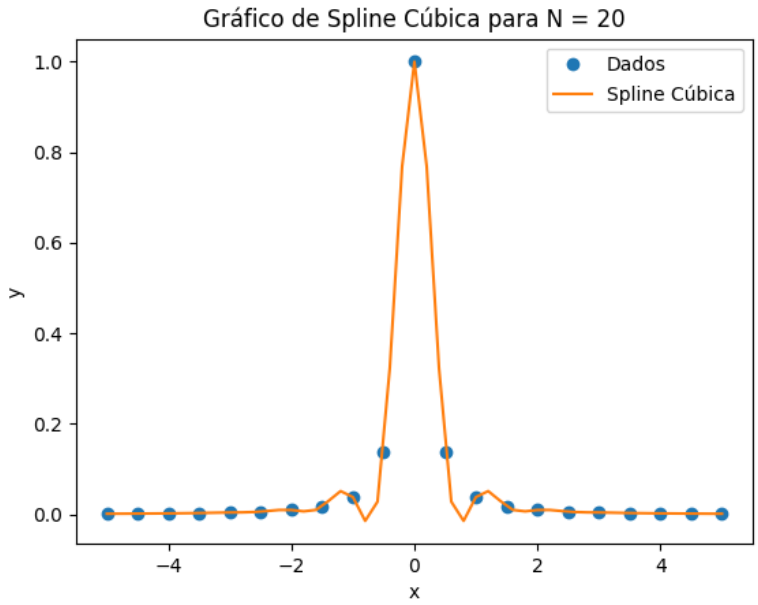

Figura 9

Por fim, para $N = 20$, temos um erro associado de 0.2684057965151454, que representa um descrescimento de 99.97% quando comparado ao erro da interpolação polinomial.

3.3.4 Conclusões sobre a interpolação com splines cúbicas

Percebemos, então, que os valores obtidos na interpolação são muito mais próximos dos valores de f quando comparados aos encontrados pela interpolação polinomial, como esperado. Ademais, não pode-se apontar que a spline cúbica possui um desempenho melhor que a spline linear, visto a proximidade no cálculos dos erros.

4 CONSIDERAÇÕES FINAIS

Conforme a tabela abaixo que sumariza os erros em cada método, podemos observar com mais clareza os efeitos do fenômeno sobre a interpolação polinomial, visto os erros exorbitantes. Além disso, concluímos que para o exemplo dado, o método da spline linear apresentou a média com menores erros.

N	Erro Polinômio	Erro Spline Linear	Erro Spline Cúbica
5	0.9557546591213079	0.9615384615384616	0.9556527119576683
10	5.453562457215162	0.41538461538461546	0.5373013829127719
20	1223.0841051674208	0.15517241379310365	0.2684057965151454

Tabela 1 – Erros máximos para diferentes valores de N e métodos de interpolação

5 REFERÊNCIAS

PETERS, S. & SZEREMETA, J.F. **Cálculo Numérico Computacional**. Editora da UFSC, 2018.