UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Projeto Prático II
Estimando a ordem via quadrados mínimos

SME0602 - Cálculo Numérico
Professor: Elias Salomão Helou Neto

Enrico Vicenzo Salvador Robazza
nº USP: 9806738

29 de Julho de 2020

Questão 1

Interpole a função

$$f(x) = \frac{1}{1 + 25x^2}$$

nos pontos $x_i = -1 + 2i/k$ com $i \in \{0, ..., k\}$, por um polinômio. Denomine este interpolador p_k.
Calcule o valor de $e_k := max_{x \in [-1,1]} |f(x) - p_k(x)|$ e trace um gráfico de e_k por k. Descreva o que você percebe. Este resultado contradiz o Teorema da Aproximação de Weierstrass? Explique.

R: Para a interpolação dos pontos, foram utilizados valores de $k \in \{2, ..., 100\}$. Primeiramente, é necessário encontrar os valores de x_i com $i \in \{0, ..., k\}$ para cada um dos k, então é interpolado o polinômio p_k através do método de Lagrange, e calculado o erro máximo daquele polinômio no intervalo $[-1, 1]$ com 200 pontos. Esses resultados são adicionados em uma lista de e_k's. Por fim, a função retorna todos os k's e e_k's.

```
def get_eks_lagrange(fx):
  eks = []
  ks = []
  for k in range(2, 100):
    x_nodes = get_x_nodes_for_k(k)
    y_nodes = fx(x_nodes)
    pk = lagrange(x_nodes, y_nodes)
    x_plot = np.linspace(-1,1,200)
    ek = max(np.absolute(fx(x_plot)-pk(x_plot)))
    ks.append(k)
    eks.append(ek)
  return (ks, eks)
```

Com isso, foi traçado o seguinte gráfico de $log(e_k)$ por k referente aos erros máximos das interpolações:

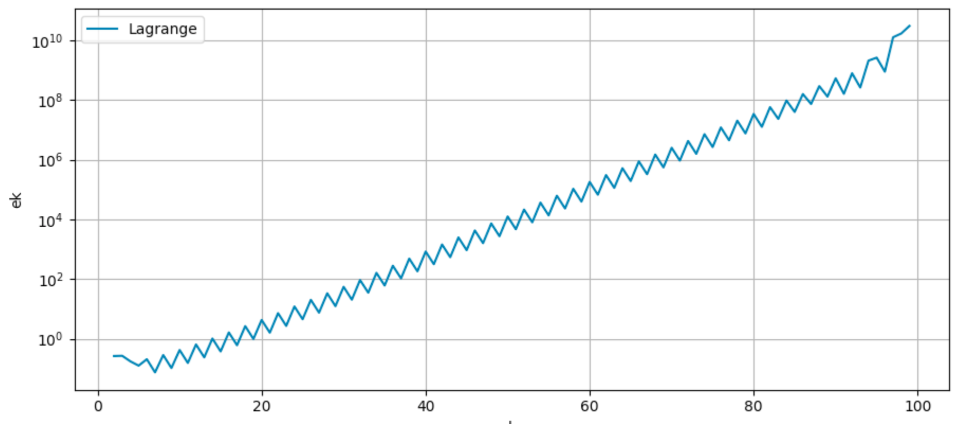

Figure 1: Gráfico de log(ek) por k para Lagrange.

Sobre estes resultados, podemos notar que $e_k \to \infty$ à medida que $k \to \infty$. Isso poderia indicar que o resultado contradiz o Teorema da Aproximação de Weirstrass, entretanto, o teorema não menciona nada a respeito dos pontos da interpolação, e se alterarmos os pontos utilizando os Nós de Chebyshev, de acordo com a equação:

$$x_i = \frac{a+b}{2} - \frac{b-a}{2} cos(\frac{k}{n}\pi), \qquad \forall k = 0, 1, ..., n$$

em que a e b é o intervalo da distribuição e n é a quantidade de pontos, obtemos o seguinte resultado:

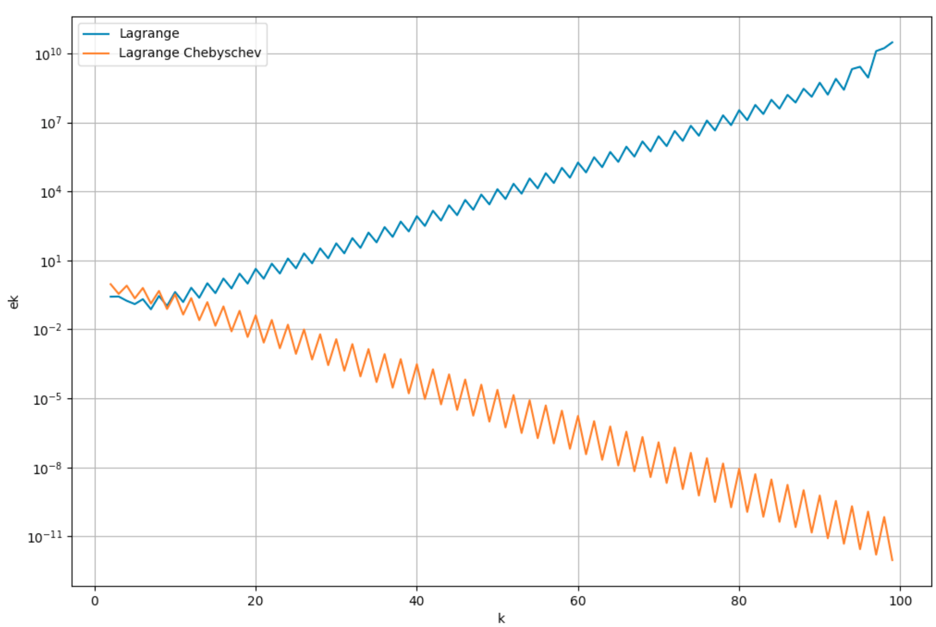

Figure 2: Gráfico de log(ek) por k para Lagrange com nós de Chebyshev.

Este fenómeno é conhecido como Fenómeno de Runge e ocorre quando os pontos de amostra são igualmente espaçados, o que é o caso da primeira interpolação.

Questão 2

Interpole a mesma função da questão 1, nos mesmos pontos, agora utilizando *splines* cúbicas naturais.
Avalie os resultados estudando a quantidade $max_{x \in [-1,1]} |f(x) - s(x)|$ para cada um dos casos. Faça gráficos desses erros conforme aumenta o número de pontos de interpolação e interprete os resultados. Compare com os obtidos na questão 1.

Supondo que o erro incorrido seja da forma Ch^q, onde h é a distância entre dois sucessivos pontos de interpolação, estime q utilizando quadrados mínimos para o caso das splines naturais e para o caso da derivada conhecida nos extremos.

R: Para interpolar com splines cúbicas, foi utilizada uma função bem similiar a do método de Lagrange, em que a quantidade de pontos de interpolação varia em $k \in \{2, \dots, 100\}$:

```
def get_eks_spline(fx):
  eks = []
  ks = []
  for k in range(2, 100):
    x_nodes = get_x_nodes_for_k(k)
    y_nodes = fx(x_nodes)
    sk = CubicSpline(x_nodes, y_nodes, bc_type='natural')
    x_plot = np.linspace(-1,1,200)
    ek = max(np.absolute(fx(x_plot)-sk(x_plot)))
    ks.append(k)
    eks.append(ek)
  return (ks, eks)
```

Foi utilizada a implementação das Splines Cúbicas do pacote scipy, com o $bc\_type = "natural"$, o que significa que as derivadas de segunda ordem nas extremidades são iguais a 0.
Com a interpolação, foi traçado o seguinte gráfico de $log(e_k)$ por k:

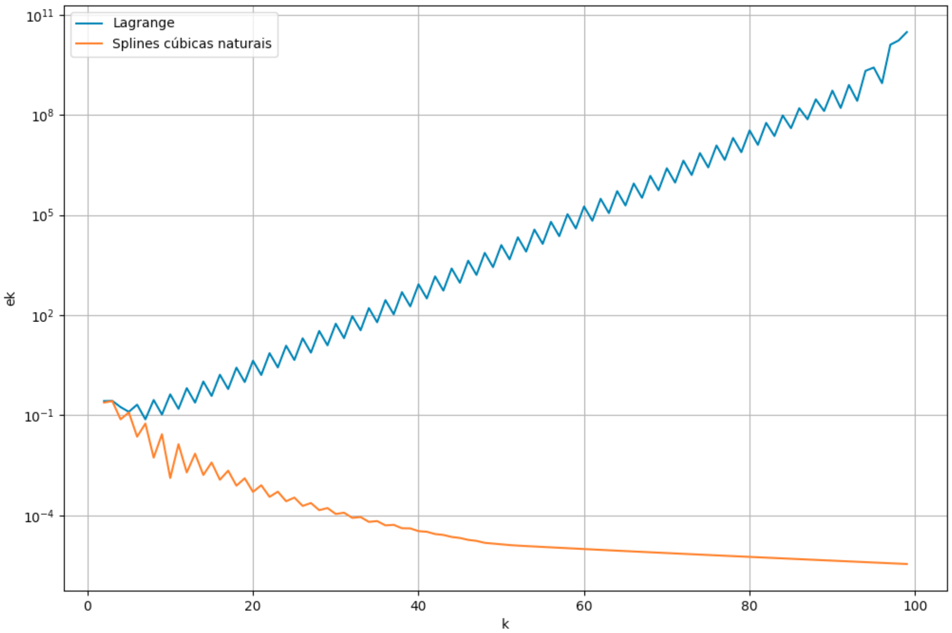

Figure 3: Gráfico de log(ek) por k para Splines e Lagrange.

De acordo com o gráfico, podemos notar que mesmo com os pontos igualmente espaçados, a medida que o número de pontos k aumenta, e_k diminui.
Podemos utilizar também a implementação das Splines Cúbicas com o caso das derivadas conhecidas nos extremos, uma vez que é possível calcular as derivadas da função. Para isso, foi utilizada a seguinte função:

```
def get_eks_spline_known_derivative(fx, fd2x):
  eks = []
  ks = []
  for k in range(2, 100):
    x_nodes = get_x_nodes_for_k(k)
    y_nodes = fx(x_nodes)
    dx1 = fd2x(x_nodes[0])
    dxn = fd2x(x_nodes[-1])
    sk = CubicSpline(x_nodes, y_nodes, bc_type=((2, dx1), (2, dxn)))
    x_plot = np.linspace(-1,1,200)
    ek = max(np.absolute(fx(x_plot)-sk(x_plot)))
    ks.append(k)
    eks.append(ek)
  return (ks, eks)
```

que recebe como parâmetros a função a ser interpolada e sua segunda derivada. Então, para cada k, são calculadas as derivadas nas extremidades, que por sua vez são utilizadas na interpolação por Splines Cúbicas. Com essa implementação, obtemos o seguinte gráfico de log(ek) por k:

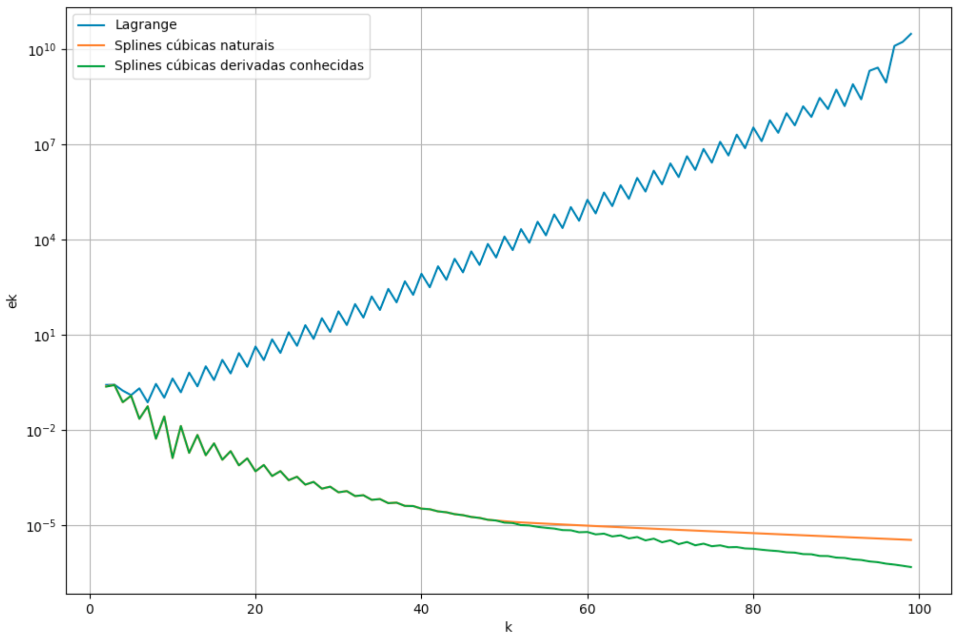

Figure 4: Gráfico de log(ek) por k para Splines Naturais, Derivadas Conhecidas e Lagrange.

Podemos analisar que os resultados obtidos são muito parecido às Splines Naturais, entretanto para um $k \geq 50$, e_k passa a ser menor.

Supondo que o erro incorrido seja da forma $e(k) = Ck^q$, podemos linearizar essa função de forma que:

$$\begin{aligned} log(e(k)) &= log(Ck^q) \\ &= log(C) + q \ . \ log(k) \end{aligned} \tag{1}$$

Em que q seria o coeficiente angular da reta, e $log(C)$ seria o coeficiente linear.
Se traçarmos um gráfico $log(e_k)$ por $log(k)$ tanto para as Splines Cúbicas Naturais quanto para a de derivadas conhecidas, obtemos:

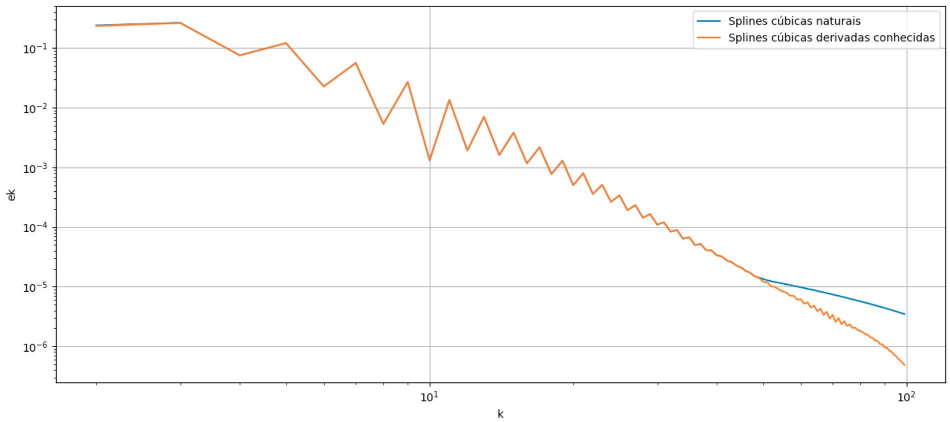

Figure 5: Gráfico de log(ek) por log(k) para Splines Naturais e Derivadas Conhecidas

Com isso, podemos estimar o valor de q, através da interpolação dessa reta pelo Método dos Quadrados Mínimos, e então encontrar os coeficientes linear e angular:

```
k=2
func = method.least_squares(np.log(ks2), np.log(eks2), k)
x_plot = np.array(np.log(ks2))
b = func(0)
a = (func(x_plot[1])-b)/x_plot[1]
```

Em que ks é o vetor com todos os $k \in \{2, \ ... \ , 100\}$, eks é o vetor contendo os erros, b é o coeficiente linear e a é o coeficiente angular.

Para a interpolação por Quadrados Mínimos, foi utilizada as seguintes funções:

```
def ft(t, x):
    soma = 0
    for i in range(len(x)):
        soma += x[i] * t ** i
    return soma

def least_squares(x_nodes, y_nodes, k):
    A = np.empty((len(y_nodes), k))
    for i in range(len(y_nodes)):
        A[i] = [x_nodes[i] ** j for j in range(k)]
    AA = np.dot(A.T, A)
    Ab = np.dot(A.T, y_nodes)
    x = np.linalg.solve(AA, Ab)
    return lambda t: ft(t, x)
```

A função *least_squares* recebe como parâmetro os pontos de interpolação (x, y), e um valor k que é referente ao grau do polinômio a ser interpolado.

Com isso, é necessário encontrar

$$\underset{x \in \mathbb{R}^n}{min} ||Ax - b||^2$$

resolvendo o sistema $A^T A = A^T b$, sendo que A é uma matriz $m \times k$ em que m é a quantidade de pontos da amostra, como se segue:

$$
\begin{bmatrix}
x_1^0 & x_1^1 & x_1^2 & ... & x_1^k \\
x_2^0 & x_2^1 & x_2^2 & ... & x_2^k \\
 & & . & & \\
 & & . & & \\
 & & . & & \\
x_m^0 & x_m^1 & x_m^2 & ... & x_m^k
\end{bmatrix}
$$

Encontrados os valores do vetor x, é retornada uma função $ft(t,x)$ que é a soma polinomial de todos os fatores.

Com isso, os resultados obtidos para as Splines Naturais foram:

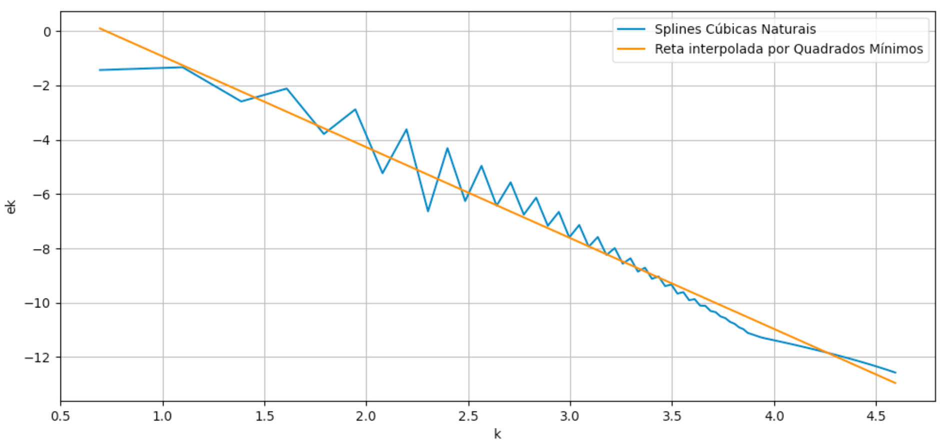

Figure 6: Interpolação por mínimos quadrados de ek para Splines Naturais

A partir dessa função, conseguimos encontrar os valores de $a = -3.3459959287040517$ e $b = 2.41919397032667$.
Os resultados obtidos para as Splines com Derivadas Conhecidas foram:

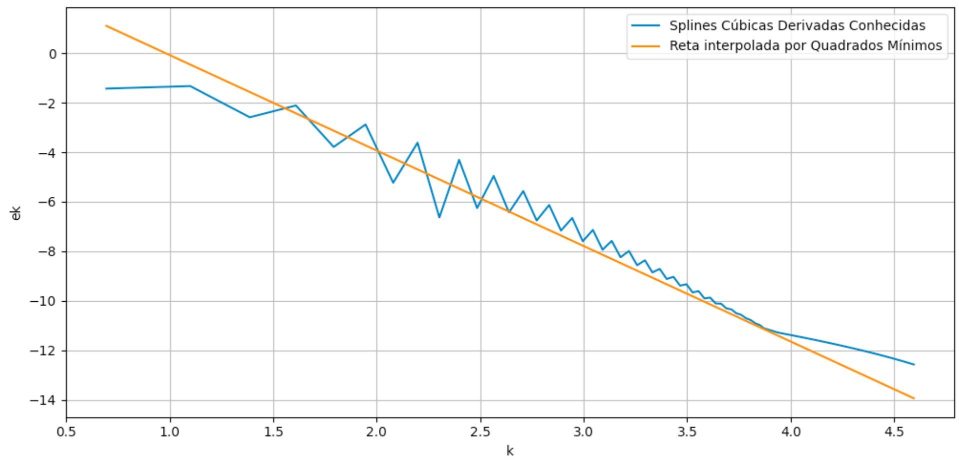

Figure 7: Interpolação por mínimos quadrados de ek para Splines Derivadas Conhecidas

A partir dessa função, conseguimos encontrar os valores de $a = -3.854345366863061$ e $b = 3.774497170458169$.
Com esses resultados, podemos chegar a conclusão de que o valor de q para as Splines Cúbicas Naturais e para as Splines Cúbicas com Derivadas Conhecidas são, respectivamente, -3.3459959287040517 e -3.854345366863061, e portanto, a segunda tem uma ordem de convergência maior do que a primeira.