**SME0602 - Cálculo Numérico – 1º semestre 2020**

**Prof. Elias Salomão Helou Neto**

**Projeto Prático 2**

**Estimando a ordem via quadrados mínimos**

**Alunos:**

**Paulo Katsuyuki Muraishi Kamimura 10277040**

**Guilherme Eiji Ichibara 10310700**

**Data: 26/07/2020**

[**1. Introdução 3**](#_Toc46695002)

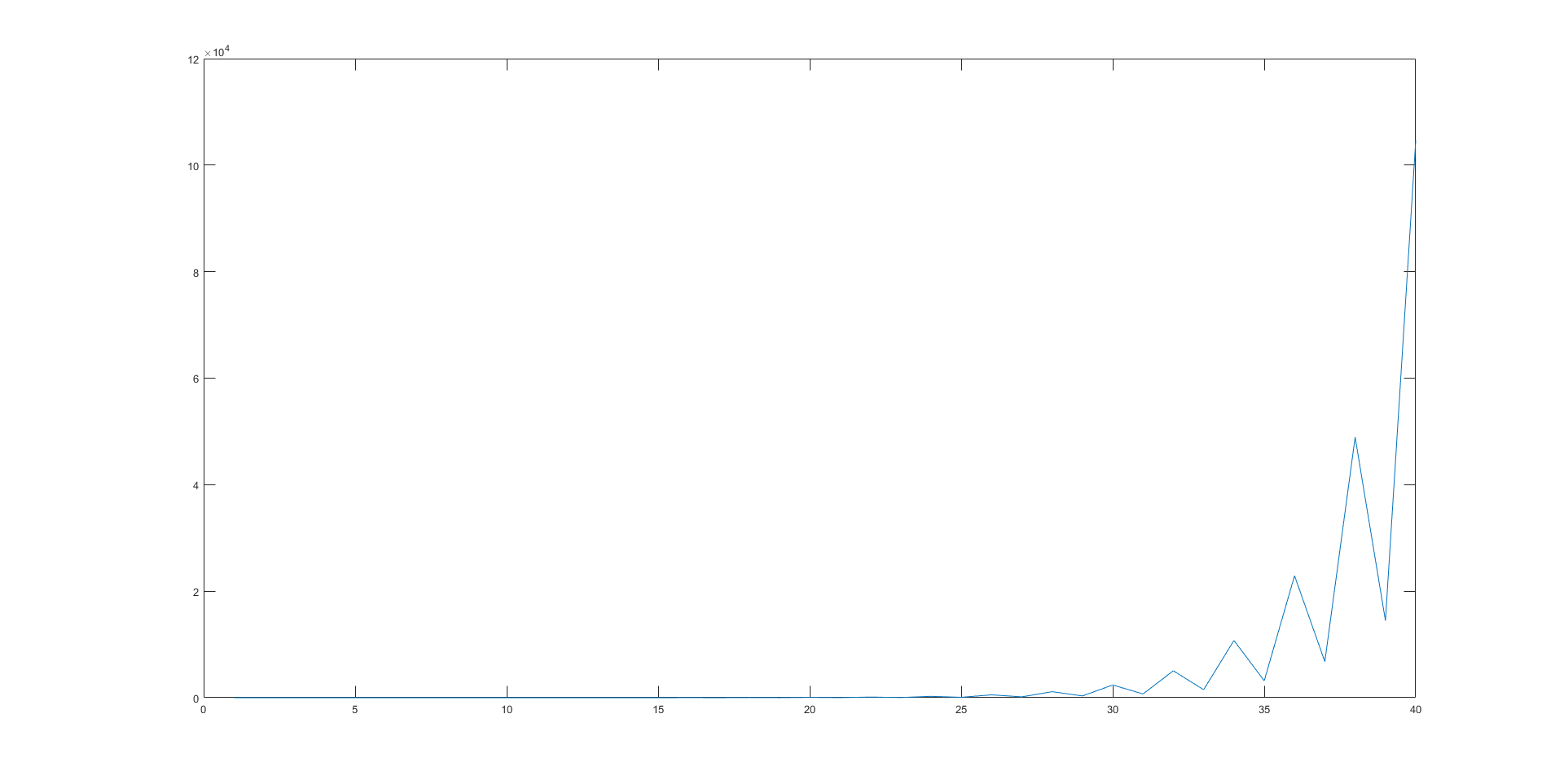
# Introdução

Neste trabalho será analisado métodos de interpolação, inicialmente realizando uma interpolação polinomial e em seguida interpolação utilizando-se de splines cúbicas polinomiais, essa análise é realizada verificando o máximo do módulo da diferença entre a função a ser interpolada e a função resultante da interpolação, em um dado intervalo. Por fim para o caso das splines naturais será estimado a ordem do valor do erro na interpolação por splines cúbicas utilizando quadrados mínimos.

Para este trabalho a implementação foi feita utilizando o GNU Octave utilizando rotinas de interpolação da própria linguagem. Vale ressaltar que para a implementação das splines cúbicas **foi necessário instalar um pacote gratuito do Octave** utilizando dois simples comandos, a execução correta destes passos será explicada posteriormente.

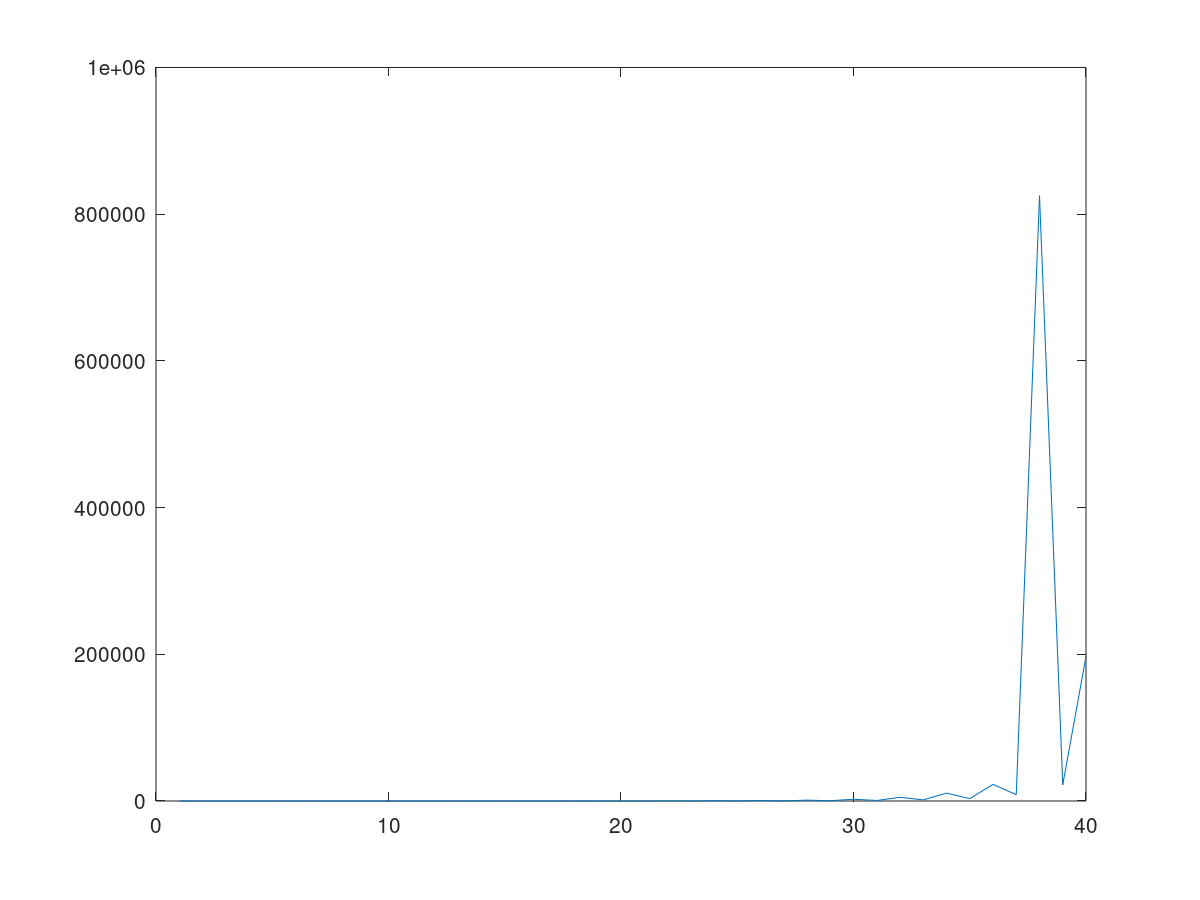
# Questão 1

Para a primeira questão deve-se realizar a interpolação da função de Runge, utilizando apenas um polinômio, e como visto nas notas de aula, o polinômio interpolador para n+1 pontos distintos, têm um grau menor ou igual a n. No caso da questão devemos avaliar a função nos pontos que vão de 0 até k, desse modo temos k+1 pontos e consequentemente o polinômio terá no máximo grau igual a k.  
 Para a implementação decidiu-se que o polinômio teria sempre grau igual a k, a fim de tentar aproximar o polinômio interpolador, cada vez mais da função. Após realizar a interpolação variando o k de 1 até 40 analisamos o valor de para cada k, e traçando um gráfico obtivemos o seguinte:



k

Figura 1 Gráfico de por k gerado no **Matlab**

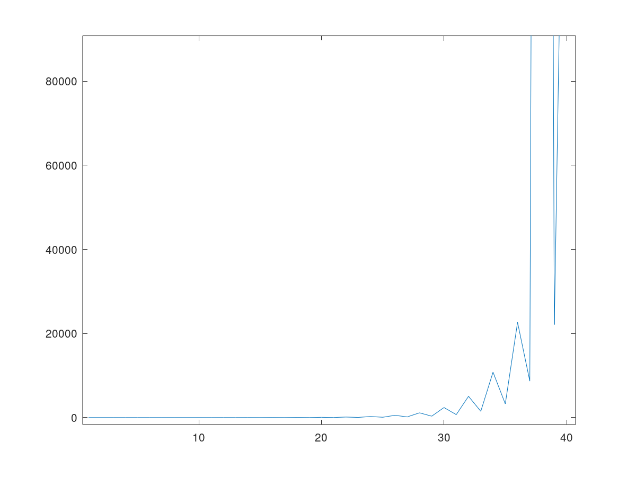


k

Figura 2 Gráfico de por k gerado no **Octave**

Como podemos ver no gráfico,  passa a crescer indefinidamente conforme k aumenta, outra observação é que o valor de oscila, isso se deve ao fato que a magnitude da derivada de k-ésima ordem dessa função em particular cresce rapidamente conforme k aumenta e a equidistância dos pontos faz com que a constante de Lebesgue também cresça rapidamente conforme k aumenta.

Além disso realizamos o teste do código tanto no Octave quanto no Matlab, e por algum motivo o crescimento do valor de é mais comportado no Matlab, já que podemos verificar um pico de para o k = 38 no gráfico gerado no Octave, a forma como o método foi implementado no Octave pode ser a justificativa desse resultado pontual inesperado.



|  |  |  |  |
| --- | --- | --- | --- |
| **k** |  | **k** |  |
| ... | ... | ... | ... |
| 35 | 3177.34 | 35 | 3259.67 |
| 36 | 22901.23 | 36 | 22649.46 |
| 37 | 6771.90 | 37 | 8673.72 |
| 38 | 48907.21 | 38 | 825315.57 |
| 39 | 14467.18 | 39 | 22102.53 |
| 40 | 104667.69 | 40 | 197146.10 |
| **Matlab** | | **Octave** | |

Tabela 1 Valores de calculados formatado condicionalmente por cor.

Figura 3 Gráfico com aplicação de zoom percebe-se o pico em k = 38. Gráfico de por k gerado pelo **Octave**

Implementação:

1. function [max\_ek]=max\_ek(k)
2. x=-1:0.0001:1;          %Cria-se o domínio do tempo "contínuo"
3. f=1./(1+25\*x.\*x);       %Define o polinômio original
5. %Cria-se o vetor com k+1 pontos igualmente espaçados no intervalo entre
6. 1 e 1 (xi = -1 + 2i/k, com i pertencente a {0,...,k})
7. xi=linspace(-1,1,k+1);
9. %Define os valores de y a partir dos pontos de xi definidos acima
10. yi=1./(1+25\*xi.\*xi);

13. %A partir de interpolação polinomial, cria-se um polinômio de ordem k a
14. partir dos pontos xi e yi definidos
15. pk = polyfit(xi,yi,k);

18. %Cria-se um gráfico que consiste no erro do polinômio pk em relação ao
19. polinômio original
20. ek = abs(f-polyval(pk,x));

23. %Retorna o valor máximo da função ek definida acima
24. max\_ek = max(abs(f-polyval(pk,x)));
26. %{
27. figure(1);
28. subplot(4,1,1);
29. plot(x,f);
30. xlabel('x');
31. ylabel('y');
32. title('Equação original');
33. grid on;
35. subplot(4,1,2);
36. plot(xi,yi);
37. xlabel('x');
38. ylabel('y');
39. title('Pontos selecionados');
40. grid on;
42. subplot(4,1,3);
43. plot(x, polyval(pk,x));
44. xlabel('x');
45. ylabel('y');
46. title('Polinômio pk encontrado');
47. grid on;
49. subplot(4,1,4);
50. plot(x, ek);
51. xlabel('x');
52. ylabel('y');
53. title('Erros do polinômio');
54. grid on;
55. %}
56. end

Apesar dos resultados mostrarem que o polinômio interpolador não aproxima bem a função, para todos os pontos diferentes daqueles a serem usados para interpolação, esse resultado não contradiz o Teorema da Aproximação de Weierstrass, já que o teorema diz toda função real contínua cujo domínio é um intervalo compacto, pode ser aproximado uniformemente por polinômios.

Desse modo só conseguimos provar que a interpolação polinomial da forma que foi implementada não aproxima bem a função para todos os pontos do intervalo, para que o resultado pudesse contradizer o teorema ele teria que mostrar que qualquer sequência de polinômios onde o grau é menor ou igual a k, o limite não tenderia a 0.

O script utilizado está presente no arquivo ***max\_ek.m***, sendo implementada a função max\_ek(k). Para tanto, calcula-se o valor da função de Runge no intervalo de [-1,1] com pontos espaçados de 0.0001 a fim de criar um domínio x com uma grande quantidade de amostra para simular um domínio “contínuo”. Posteriormente foi gerado um array *xi* com os pontos seguindo as regras do enunciado, espaçados de uma distância de 1/k.

A partir dos valores *xi* é calculado o *yi* correspondente para a equação dada no enunciado, assim a partir de ambos esses vetores é feito a interpolação utilizando a função polyfit do próprio Octave, gerando um novo polinômio .

Do polinômio avaliamos os mesmos pontos que avaliamos a função de Runge para tirarmos o módulo da diferença de tais pontos, calculando assim o erro para cada ponto da equação, por fim pegamos o maior valor do vetor e em seguida é retornado pela função max\_ek(k).

A área comentada do código plota os gráficos que estão anexados com o relatório, o primeiro gráfico apresenta a função original, o segundo os pontos que serão usados para interpolação, o terceiro o polinômio resultante da interpolação e o quarto o erro entre a função original e o polinômio.

O Octave não é muito eficiente na alocação de memória então ao invés de usarmos uma função que vai acrescentando um novo valor ao vetor, decidiu-se realizar as iterações da interpolação da seguinte maneira:

1. ek\_results = [max\_ek(1) max\_ek(2) max\_ek(3) max\_ek(4) max\_ek(5) max\_ek(6) max\_ek(7) max\_ek(8) max\_ek(9) max\_ek(10) max\_ek(11) max\_ek(12) max\_ek(13) max\_ek(14) max\_ek(15) max\_ek(16) max\_ek(17) max\_ek(18) max\_ek(19) max\_ek(20) max\_ek(21) max\_ek(22) max\_ek(23) max\_ek(24) max\_ek(25) max\_ek(26) max\_ek(27) max\_ek(28) max\_ek(29) max\_ek(30) max\_ek(31) max\_ek(32) max\_ek(33) max\_ek(34) max\_ek(35) max\_ek(36) max\_ek(37) max\_ek(38) max\_ek(39) max\_ek(40)];
2. k\_domain\_1 = 1:1:40;
3. plot(k\_domain\_1,ek\_results\_1);

O código acima está no arquivo main.m, após obter os valores de podemos plotar o gráfico de por k. Todos os resultados da implementação também se encontram anexado em uma planilha Open Document.

# Questão 2

De maneira similar a primeira questão foi feita a interpolação da função desta vez utilizando splines cúbicas natural, para isso apenas alteramos o método que interpola a função de Runge por polinômios e passamos a usar um que recebe como parâmetro os pontos a serem interpolados utilizando as splines cúbicas e um terceiro parâmetro condicional sobre os pontos das extremidades, como pode ser visto no código o parâmetro utilizado é o parâmetro “variational”, que funciona de maneira idêntica ao “natural” utilizado no exemplo dado em aula, que faz com que a derivada segunda do primeiro e último ponto seja igual a zero.

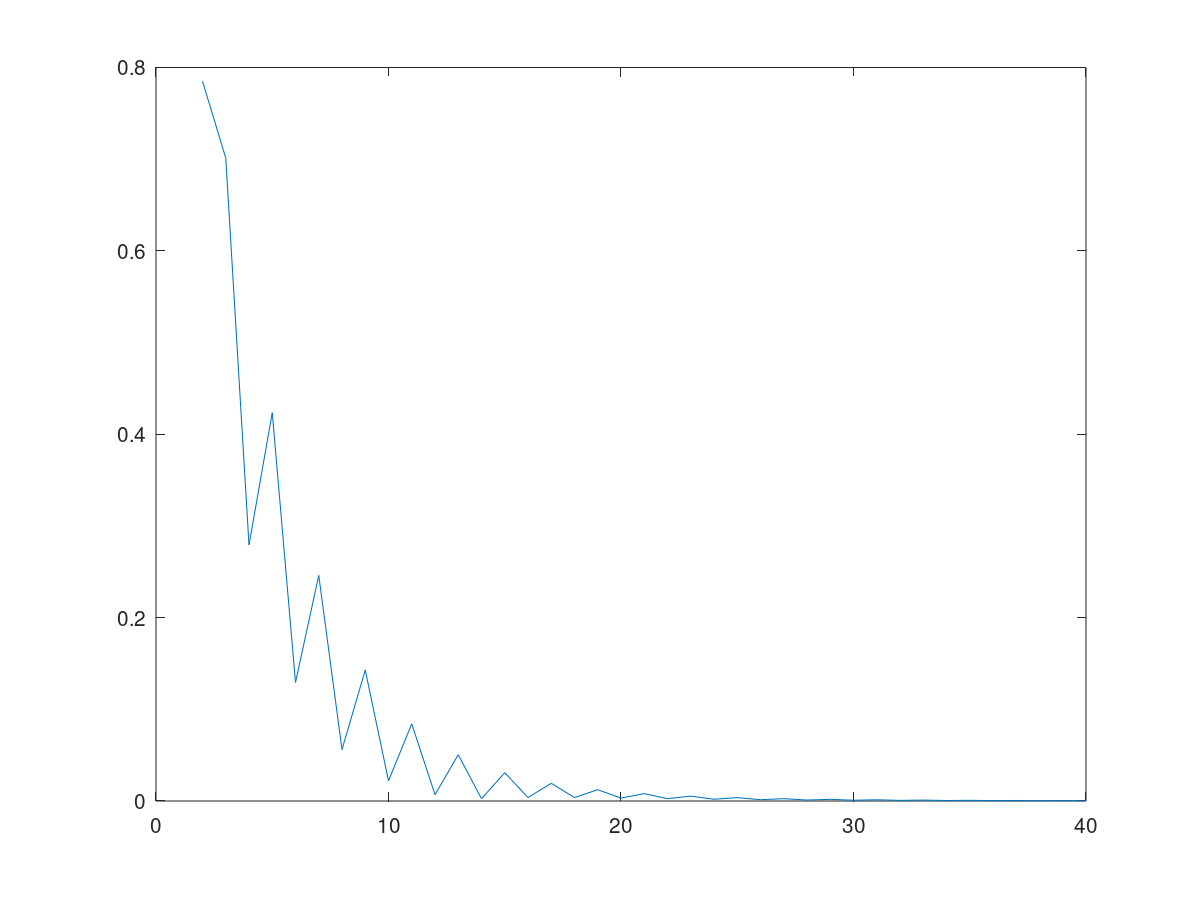
A função responsável pela realização do método de splines cúbicas natural é o *csape* que faz parte do pacote gratuito do Octave. Para a devida utilização na janela de comandas é necessário executar o seguinte comando.

|  |
| --- |
| **pkg install -forge splines** |

O comando acima irá realizar o download da biblioteca. Em seguida é necessário instalar a biblioteca utilizando o seguinte comando.

|  |
| --- |
| **pkg load splines** |

 Com a interpolação realizada variando o k calculamos o novo onde dessa vez substituiremos (x) por s(x), o novo gráfico de por k fica da seguinte maneira:



k

Figura 4 Gráfico de por k gerado no **Octave**

Implementação:

1. function [max\_ek]=spline(k)
2. x=-1:0.0001:1;
3. f=1./(1+25\*x.\*x);
5. xi=linspace(-1,1,k+1);
6. yi=1./(1+25\*xi.\*xi);
8. s = csape(xi, yi,'variational')
9. s\_function = fnval(s,x);
10. ek = abs(f-s\_function);
11. max\_ek = max(ek);
13. %{
14. figure(1);
15. subplot(4,1,1);
16. plot(x,f);
17. xlabel('x');
18. ylabel('y');
19. title('Equação original');
20. grid on;
22. subplot(4,1,2);
23. plot(xi,yi);
24. xlabel('x');
25. ylabel('y');
26. title('Pontos selecionados');
27. grid on;
29. subplot(4,1,3);
30. plot(x,s\_function,'k-',xi,yi,'ro')
31. xlabel('x');
32. ylabel('y');
33. title('Polinômio s encontrado');
34. grid on;
36. subplot(4,1,4);
37. plot(x, ek);
38. xlabel('x');
39. ylabel('y');
40. title('Erros do polinômio');
41. grid on;
42. %}
43. end

Como podemos ver o valor de para splines cúbicas naturais também oscila, mas diferentemente da primeira questão ele vai diminuindo com as iterações satisfazendo assim o teorema de Weierstrass.

Por fim, a partir dos valores de obtidos e supondo que ele seja da forma como dito no enunciado, foi feito a estimativa da ordem de q através de quadrados mínimos, para isso é necessário inicialmente linearizar essa aproximação para que o método do mínimos quadrados possa ser utilizado, sabendo que h é a distância entre um ponto de interpolação ao outro, :

Dessa maneira armazena-se os valores de em um array y. Já em uma matriz X, preenche-se a primeira coluna com 1 e a segunda coluna com os valores de -. Com isso basta utilizar o operador “\” que realiza uma “left division” entre a segunda e a primeira estrutura de dados, assim obtendo como resultado um vetor *b* com dois valores onde o segundo valor é a ordem de grandeza estimada que queríamos, assim obtemos que o seguinte valor:

q = 3.4641

# Conclusão

A análise dos dois métodos demonstrou que a interpolação polinomial tem um bom desempenho somente nas primeiras iterações de k, já o método com splines cúbicas consegue se beneficiar com a presença de um grande número de amostras, isso já era esperado pela própria lógica da implementação em que a interpolação é realizada em várias partes, dividindo os pontos.

Já com o método dos mínimos quadrados é possível analisar o quão rápido o método de splines cúbicas consegue se aproximar de um resultado mais próximo do real, diminuindo o erro e, uma vez que o **,** tal velocidade é de fato exponencial e a precisão aumenta ao iterar por valores altos de k.