UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
Departamento de Ciências de Computação

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

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.

O script a ser rodado é o main.m ele faz a chamada das funções definidas nos arquivos max_ek.m e spline.m. No arquivo spline.m a função responsável pela realização do método de splines cúbicas natural é o *csape* que faz parte do pacote gratuito do Octave Forge. Para a devida utilização, é necessário executar o seguinte comando na janela de comandos.

```
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
```

2. 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 e_k para cada k, e traçando um gráfico obtivemos o seguinte:

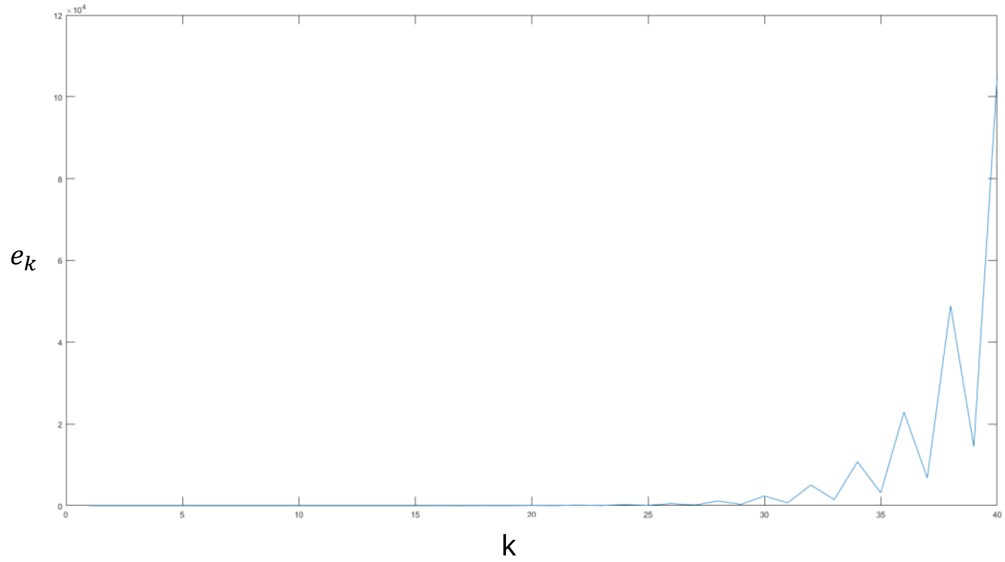

*Figura 1 Gráfico de e_k por k gerado no **Matlab***

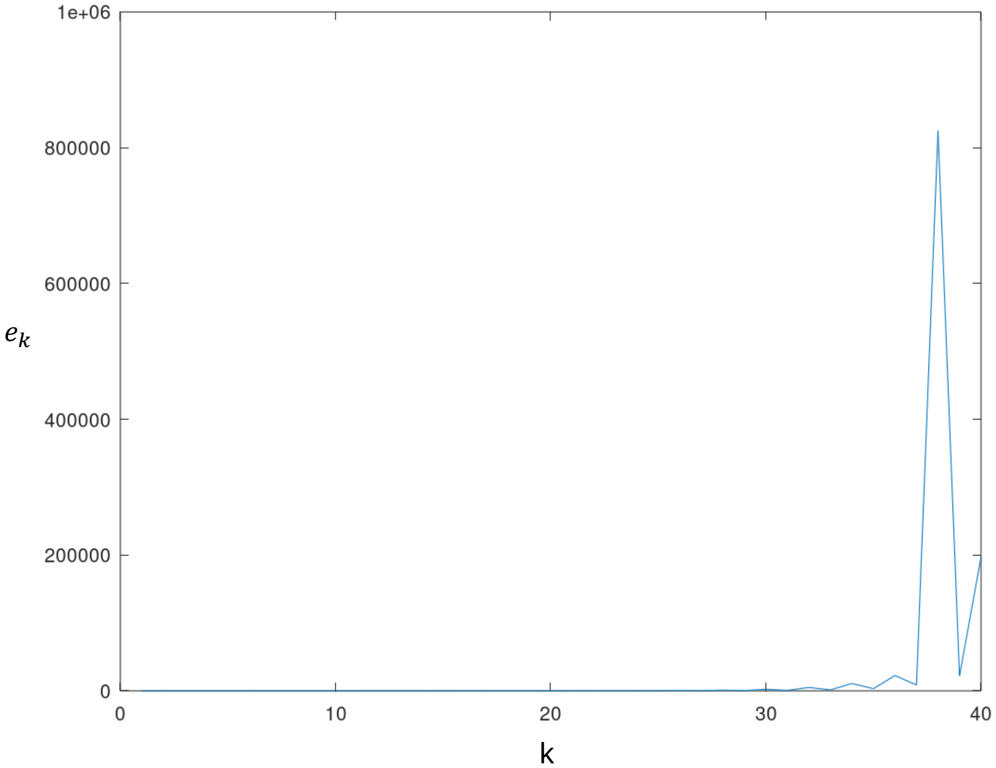

*Figura 2 Gráfico de e_k por k gerado no **Octave***

Como podemos ver no gráfico, e_k passa a crescer indefinidamente conforme k aumenta, outra observação é que o valor de e_k 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 e_k é mais comportado no Matlab, já que podemos verificar um pico de e_k 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.

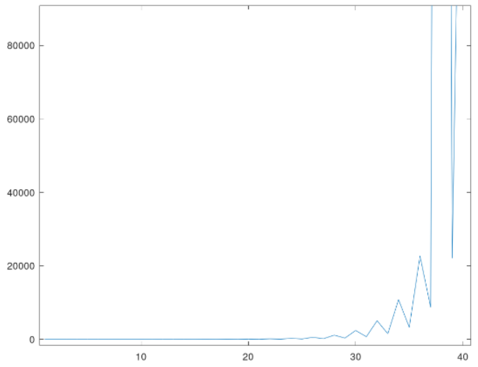

*Figura 3 Gráfico com aplicação de zoom percebe-se o pico em k = 38. Gráfico de e_k por k gerado pelo **Octave***

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 e_k calculados formatado condicionalmente por cor.

Implementação:

```
function [max_ek]=max_ek(k)
    x=-1:0.0001:1;          %Cria-se o domínio do tempo "contínuo"
    f=1./(1+25*x.*x);       %Define o polinômio original

        %Cria-se o vetor com k+1 pontos igualmente espaçados no intervalo entre 1 e 1 (xi = -1 + 2i/k, com i pertencente a {0,...,k})
    xi=linspace(-1,1,k+1);


        %Define os valores de y a partir dos pontos de xi definidos acima
    yi=1./(1+25*xi.*xi);


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


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


    %Retorna o valor máximo da função ek definida acima
    max_ek = max(abs(f-polyval(pk,x)));

    %{
    figure(1);
    subplot(4,1,1);
    plot(x,f);
    xlabel('x');
    ylabel('y');
    title('Equação original');
    grid on;

    subplot(4,1,2);
    plot(xi,yi);
```

```
    xlabel('x');
    ylabel('y');
    title('Pontos selecionados');
    grid on;

    subplot(4,1,3);
    plot(x, polyval(pk,x));
    xlabel('x');
    ylabel('y');
    title('Polinômio pk encontrado');
    grid on;

    subplot(4,1,4);
    plot(x, ek);
    xlabel('x');
    ylabel('y');
    title('Erros do polinômio');
    grid on;
    %}
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 p_k.

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 p_k 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:

```
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)];
k_domain_1 = 1:1:40;
plot(k_domain_1,ek_results_1);
```

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

3. 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.

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

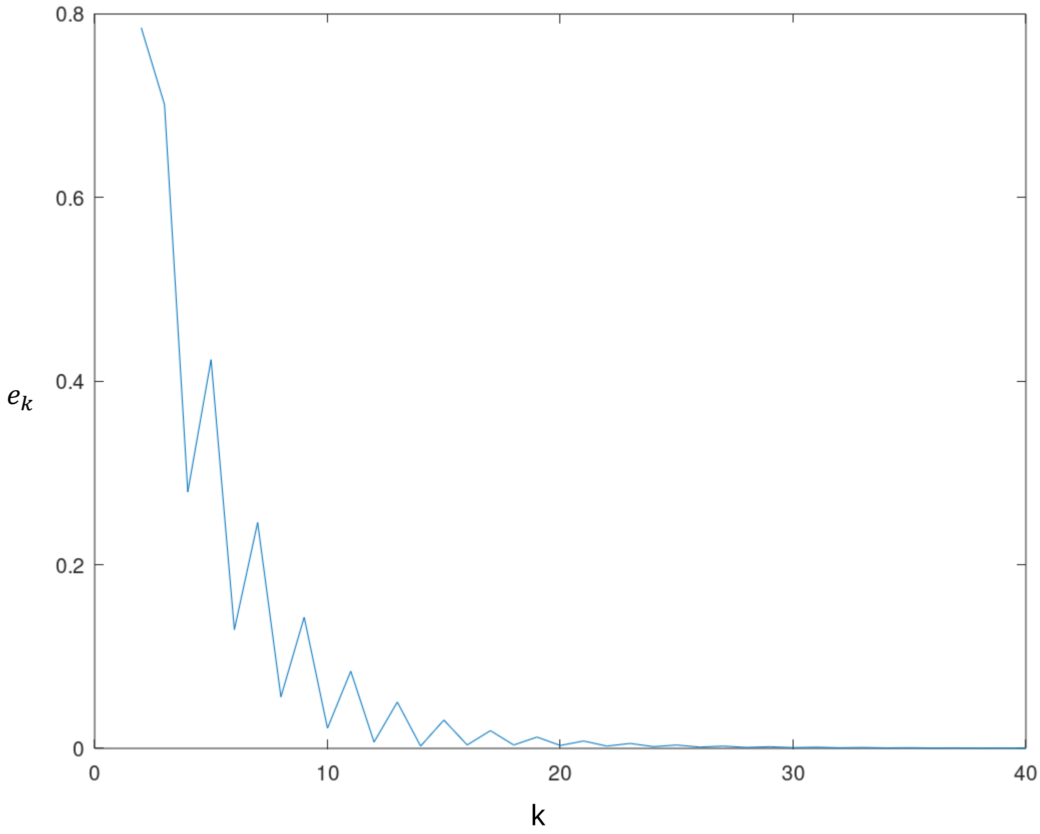

Figura 4 Gráfico de e_k por k gerado no ***Octave***

Implementação:

```
function [max_ek]=spline(k)
    x=-1:0.0001:1;
    f=1./(1+25*x.*x);

    xi=linspace(-1,1,k+1);
    yi=1./(1+25*xi.*xi);

    s = csape(xi, yi,'variational')
    s_function = fnval(s,x);
         ek = abs(f-s_function);
         max_ek = max(ek);

         %{
         figure(1);
         subplot(4,1,1);
         plot(x,f);
         xlabel('x');
         ylabel('y');
         title('Equação original');
         grid on;

         subplot(4,1,2);
         plot(xi,yi);
         xlabel('x');
         ylabel('y');
         title('Pontos selecionados');
```

```
        grid on;

        subplot(4,1,3);
        plot(x,s_function,'k-',xi,yi,'ro')
        xlabel('x');
        ylabel('y');
        title('Polinômio s encontrado');
        grid on;

        subplot(4,1,4);
        plot(x, ek);
        xlabel('x');
        ylabel('y');
        title('Erros do polinômio');
        grid on;
        %}
    end
```

Como podemos ver o valor de e_k 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 e_k obtidos e supondo que ele seja da forma Ch^q 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, $h = \frac{1}{k} \rightarrow \log(h) = -\log(k)$:

$$e_k \approx Ch^q$$
$$\log(e_k) \approx \log(Ch^q)$$
$$\log(e_k) \approx \log(C) + log(h^q)$$
$$\log(e_k) \approx \log(C) + qlog(h)$$

$$\log(e_k) \approx \log(C) - qlog(k)$$

Dessa maneira armazena-se os valores de $\log(e_k)$ em um array y. Já em uma matriz X, preenche-se a primeira coluna com 1 e a segunda coluna com os valores de - $log(k)$. 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

4. 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 $q \neq 0$ e $q > 1$**,** tal velocidade é de fato exponencial e a precisão aumenta ao iterar por valores altos de k.