Análise de Complexidade dos Métodos de Interpolação Implementados em MATLAB

Sinayra Pascoal Cotts Moreira
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade de Brasília
Email: sinayra@hotmail.com

***Resumo*—Este artigo analisa uma aplicação onde, dada uma tabela (x, y) descrita pelo usuário, retorna diferentes polinômios baseados nos métodos de Interpolação Polinomial. A análise implica em comparar a complexidade e tempo de execução de cada método implementado e concluir quais são os menos custosos computacionalmente.**

I. Introdução

Este trabalho visa a comparação da complexidade dos métodos *Lagrange, Lagrange igualmente espaçado, Newton/Diferença Dividida, Diferença Finita Progressiva, Diferença Finita Regressiva, Spline Linear, Spline Cúbico Natural* e *Spline Cúbico Extrapolado* [2], [4]. Para esta análise, foram considerados três funções distintas, com n pontos, $n \in [4, 170]$, com intervalo igualmente espaçado $h = 1$, anotando o tempo de execução de cada método para realizar todos os casos de testes. Os casos de testes são feitos criando uma nova tabela (x, y) aplicando o polinômio calculado para cada valor de x na tabela.

II. Objetivos

O objetivo deste trabalho foi desenvolver uma implementação dos métodos de Interpolação Polinomial para consolidar os conhecimentos do curso Cálculo Numérico, relacionando esta implementação com a área de estudo do aluno. A área escolhida foi de Projeto e Análise de Algoritmo e a relação contruída foi de análise de complexidade de algoritmo. Além disso, o trabalho também teve como objetivo o estudo de linguagem de programação MATLAB, voltada para realização de cálculos numéricos com escrita próxima de expressões algébricas [5], diferenciando-se de linguagens imperativas como C.

III. Metodologia

Sejam dados $n+1$ pontos $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, sendo x_i distintos tais que $f(x_i) = y_i$, deseja-se construir um polinômio de grau não superior a n que possua nos pontos x_i os mesmos valores de $f(x_i)$. Uma vez que este polinômio é único [1], os métodos apresentados nesta Seção, apesar de apresentarem polinômios diferentes, são equivalentes.

Seja X o vetor que possui os valores x_i e Y é o vetor que possui os valores de $f(x_i)$.

A. Lagrange

O método de Lagrange é definido por

$$L_n(x) = \sum_{i=0}^{n} c_i l_i(x) \quad c_i = \frac{f(x_i)}{P_i(x_i)} \quad l_i(x) = \prod_{j=0, j\neq i}^{n} (x - x_j)$$

Listagem 1: Resolução de $l_i(x)$. Complexidade $O(n^2)$

```
for i = 1:n
    produto = 1; %valor neutro
    for j = 1:n
        if(i ~= j)
            produto = double(produto) * (double(X(i)) - double(X(j)));
        end
    end
    l(i) = 1.0/produto;
end
```

Listagem 2: Resolução de $L_n(x)$ e de c_i. Complexidade $O(n^2)$

```
function [y] = calculaMetodo2(X, Y, l, n, x)
    y = 0;
    for i = 1:n
        aux = Y(i) * l(i);
        for j = 1:n
            if(j ~= i)
                aux = aux * (x - X(j));
            end
        end
        y = y + aux;
    end
end
```

Analisando a complexidade da Listagem 1 e Listagem 2, a complexidade do método de Lagrange é $O(n^2)$.

B. Lagrange igualmente espaçado

O método de Lagrange igualmente espaçado é definido por

$$\bar{L_n}(u) = \sum_{i=0}^{n} c_i \bar{l_i}(u) \quad c_i = \frac{f(u_i)}{P_i(u_i)} \quad l_i\bar{(u)} = \prod_{j=0, j\neq i}^{n} (x - x_j)$$
$$u = \frac{x - x_0}{h}$$

Listagem 3: Resolução de $\bar{l_i}(u)$. Complexidade $O(n^2)$

```
for i = 0:n-1
    produto = 1; %valor neutro
    for j = 0:n-1
```

```
        if(i ~= j)
            prod = double(produto) * double((i - j));
        end
    end
    l(i+1) = 1.0/produto;
end
```

Listagem 4: Resolução de $\bar{L_n}(x)$ e de c_i. Complexidade $O(n^2)$

```
function [y, u] = calculaMetodo3(X, Y, l, n, h, x)
    u = (x - X(1,1))/h;
    y = 0;
    for i = 1:n
        aux = Y(i) * l(i);
        for j = 0:n-1
            if(j ~= i-1)
                aux = aux * (u - j);
            end
        end
        y = y + aux;
    end
end
```

Analisando a complexidade da Listagem 3 e Listagem 4, a complexidade do método de Lagrange igualmente espaçado é $O(n^2)$.

C. *Newton/Diferença dividida*

O método de Newton é definido por:

$$\begin{aligned} N(x) = \quad & f[y_0]+ \\ & +f[y_0, y_1](x-x_0)+ \\ & +\cdots+ \\ & +f[y_0,\cdots,y_k](x-x_0)(x-x_1)\cdots(x-x_{k-1}) \end{aligned}$$

$$\begin{aligned} f[y_i] \quad & = y_i \\ & i \in \{0\cdots k\} \end{aligned}$$

$$\begin{aligned} f[y_i,\cdots,y_{i+j}] \quad & = \tfrac{f[y_{i+1},\cdots,y_{i-j}]-f[y_i,\cdots,y_{i+j-1}]}{x_{i+j}-x_i} \\ & i \in \{0\cdots k-j\} \\ & j \in \{1\cdots k\} \end{aligned}$$

Listagem 5: Resolução de $f[y_0,\cdots,y_k]$. Complexidade $O(n^2)$

```
function [D] = f(X, Y, n, total)
    D = [];
    if(n ~= 0)
        ordem = zeros(1, n-1);
        for i = 1:n-1
            aux = (double(Y(i+1)) - double(Y(i)))/(double(X(i+1+(total-n))) - double(X(i)));
            ordem(i) = aux;
        end
        D = f(X, ordem, n-1, total); %chama f para próxima ordem, que terá n-1 elementos
        mzero = zeros(1, total - n); %ajusta para matrizes terem o mesmo tamanho
        D = [D ; Y mzero]; %concatena na resposta tudo o que encontrou
    end
end
```

Listagem 6: Resolução de $N(x)$. Complexidade $O(n^2)$

```
function [y] = calculaMetodo4(D, X, n, x)
    y = 0;
    for i = 1:n
        aux = D(n-(i-1), 1);
        for j = 1:i-1
            aux = aux * (x - X(j));
        end
        y = y + aux;
    end
end
```

Analisando a complexidade da Listagem 5 e Listagem 6, a complexidade do método de Newton é $O(n^2)$.

D. *Diferença Finita Progressiva*

O método de Diferença Finita Progressiva é definido por

$$N_k(x) = \sum_{i=0}^{k} \frac{\triangle^i}{i!h^i} \prod_{j=0}^{i-1}(x-x_j) \tag{1}$$

Listagem 7: Resolução de $\triangle^j$. Complexidade $O(n^2)$

```
function [D] = f(Y, n, total)
    D = [];
    if(n ~= 0)
        ordem = zeros(1, n-1);
        for i = 1:n-1
            aux = double(Y(i+1)) - double(Y(i));
            ordem(i) = aux;
        end
        D = f(ordem, n-1, total); %chama f para próxima ordem, que terá n-1 elementos
        mzero = zeros(1, total - n); %ajusta para matrizes terem o mesmo tamanho
        D = [D ; Y mzero]; %concatena na resposta tudo o que encontrou
    end
end
```

Listagem 8: Resolução de $N_k(x)$. Complexidade $O(n^2)$

```
function [y] = calculaMetodo5(D, X, h, n, x)
    y = 0;
    for i = 1:n
        d = D(n-(i-1), 1) * 1/( ( fatorial(i-1) * power(h, (i-1) ) ) );

        for j = 1:i-1
            d = d * (x - X(j));
        end
        y = y + d;
    end
end
```

Analisando a complexidade da Listagem 7 e Listagem 8, a complexidade do método de Diferença Finita Progressiva é $O(n^2)$.

E. *Diferença Finita Regressiva*

O método de Diferença Finita Regressiva é definido por

$$N_k(x) = \sum_{i=0}^{k} \frac{\nabla^i}{i!h^i} \prod_{j=0}^{i-1}(x-x_j) \tag{2}$$

Listagem 9: Resolução de $\bigtriangledown^j$. Complexidade $O(n^2)$

```
function [D] = f(Y, n, total)
    D = [];
    if(n ~= 0)
        ordem = zeros(1, n-1);
        for i = 2:n
            aux = double(Y(i)) - double(Y(i-1));
            ordem(i-1) = aux;
        end
        D = f(ordem, n-1, total); %chama f para próxima ordem, que terá n-1 elementos
        mzero = zeros(1, total - n); %ajusta para matrizes terem o mesmo tamanho
        D = [D ; Y mzero]; %concatena na resposta tudo o que encontrou
    end
end
```

Listagem 10: Resolução de $N_k(x)$. Complexidade $O(n^2)$

```
function [y] = calculaMetodo6(D, X, h, n, x)
    y = 0;
    for i = 1:n
        d = D(n-(i-1), 1) * 1/( ( fatorial(i-1) * power(h, (i-1) ) ) );

        for j = 1:i-1
            d = d * (x - X(j));
        end
        y = y + d;
    end
end
```

Analisando a complexidade da Listagem 9 e Listagem 10, a complexidade do método de Diferença Finita Regressiva é $O(n^2)$.

F. Spline Linear

O método Spline Linear, $S_1(x)$, tem grau 1 e pode ser escrito em cada subintervalo $[x_{i-1}, xi], i \in \{0 \cdots k\}$, de modo que

$$s_i(x) = f(x_{i-1})\frac{x_i - x}{x_i - x_{i-1}} + f(x_i)\frac{x - x_{i-1}}{x_i - x_{i-1}}, \forall x \in [x_{i-1}, x_i]$$

Listagem 11: Limitação dos intervalos. Complexidade $O(n)$

```
for i = 1:n-1
    intervalo(i, 1) = Xi(i);
    intervalo(i, 2) = Xi(i+1);
    xs(i) = 1/(Xi(i) - Xi(i+1));
end
```

Listagem 12: Resolução de s_i. Complexidade $O(n)$

```
function [Yf, I] = calculaMetodo7(X, Y, intervalo, xs, n, x)
    Yf = [];
    I = [];
    for i = 1:n-1
        if(x >= intervalo(i, 1) && x <= intervalo(i, 2))
            y = Y(i) * (x - X(i+1)) - Y(i+1)*(x - X(i));
            y = y * xs(i);
            Yf = [Yf y]; %y dentro do intervalo
            I = [I i]; %intervalo
        end
    end
end
```

Analisando a complexidade da Listagem 11 e Listagem 12, a complexidade do método de Spline Linear é $O(n)$.

G. Spline Cúbico Natural

O método Spline Cúbico Natural, $S_3(x)$, tem grau 3 e pode ser escrito em cada subintervalo $[x_{i-1}, xi], i \in \{0 \cdots k\}$, de modo que

$$s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

$$a_i = \frac{g_i - g_{i-1}}{6h_i} \quad b_i = \frac{g_i}{2} \quad c_i = \frac{h_i}{6}(g_{i-1} + 2g_i) + f[x_{i-1}, x_i] \quad d_i = y_i$$

Onde $A.\bar{g} = \bar{y}$

$$A = \begin{pmatrix} h_1 & 2(h_1 + h_2) & h_2 & \cdots & 0 & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & \cdots & h_{k-1} & 2(h_{k-1} + h_k) & h_k \end{pmatrix}$$

$$\bar{y} = 6 \begin{pmatrix} f[x_1, x_2] - f[x_0, x_1] \\ \vdots \\ f[x_{k-2}, x_{k-1}] - f[x_{k-1}, x_k] \end{pmatrix} \quad \bar{g} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_k \end{pmatrix}$$

Listagem 13: Limitação dos intervalos e resolvendo matriz $\bar{y}$. Complexidade $O(n)$

```
for i = 1:n-1

    intervalo(i, 1) = Xi(i);
    intervalo(i, 2) = Xi(i+1);
    hs(i) = Xi(i+1) - Xi(i);  %h
    f(i) = (Yi(i+1) - Yi(i))/hs(i); %newton
end
for i = 1:n-2
    y(i) = f(i+1) - f(i);
end
y = 6 * y; %y
```

Listagem 14: Resolvendo matriz A. Complexidade $O(n)$

```
j = -1;
for i = 1:n-2
    coluna = 1;

    if(j + coluna >= 1 && j + coluna <= (n-2))
        A(i, coluna+j) = hs(i);
    end

    coluna = coluna + 1;
    if(j + coluna >= 1 && j + coluna <= (n-2))
        A(i, coluna+j) = 2 * (hs(i) + hs(i+1));
    end

    coluna = coluna + 1;
    if(j + coluna >= 1 && j + coluna <= (n-2))
        A(i, coluna+j) = hs(i+1);
    end

    j = j + 1;
```

```
end
```

Listagem 15: Resolvendo matriz $\bar{g}$. Complexidade $O(n^2)$

```
g(n) = A(n,(n+1))/A(n,n);
i = n-1;         %De trás para frente
while(i >= 1)
    j = n;
    while(j > i)
        g(i) = g(i) - g(j)*A(i,j); %subtrai todos os a's que possuem valor com seus respectivos x's
        j = j - 1;
    end
    g(i) = g(i) + A(i, n+1); %soma com respectivo yi que foi triangulizado
    g(i) = g(i) / A(i,i); %divide com xi respectivo
    i = i - 1;
end
```

Listagem 16: Resolução de s_i. Complexidade $O(n)$

```
function [Yf, I] = calculaMetodo8(X, Y, intervalo, g, h, f, n, x)
    Yf = [];
    I = [];
    for i = 2:n
        if(x >= intervalo(i-1, 1) && x <= intervalo(i-1, 2))
            a = (g(i) - g(i-1))/(6*h(i-1));
            b = g(i)/2;
            c = f(i-1) + (2*h(i-1)*g(i) + g(i-1)*h(i-1))/6;
            d = Y(i);
            y = a * power(x - X(i), 3) + b * power(x - X(i), 2) + c * (x - X(i)) + d;
            Yf = [Yf y];
            I = [I i-1];

        end
    end
end
```

Analisando a complexidade da Listagem 13, Listagem 14, Listagem 15 e Listagem 16, a complexidade do método de Spline Cúbico Natural é $O(n^2)$.

H. Spline Cúbico Extrapolado

O método Spline Cúbico Extrapolado, $S_3(x)$, tem grau 3 e pode ser escrito em cada subintervalo $[x_{i-1}, xi], i \in \{0 \cdots k\}$, de modo que

$$s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

$$a_i = \frac{g_i - g_{i-1}}{6h_i} \quad b_i = \frac{g_i}{2} \quad c_i = \frac{h_i}{6}(g_{i-1} + 2g_i) + f[x_{i-1}, x_i] \quad d_i = y_i$$

Onde $A.\bar{g} = \bar{y}$

$$A = \begin{pmatrix} 2_h1 & h_1 & 0 & 0 & \cdots & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & \cdots & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & \cdots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \cdots & h_k & 2h_k \end{pmatrix}$$

$$\bar{y} = 6 \begin{pmatrix} f[x_0, x_1] - y_0{}' \\ f[x_1, x_2] - f[x_0, x_1] \\ \vdots \\ y_k{}' - f[x_{k-1}, x_k] \end{pmatrix} \quad \bar{g} = \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_k \end{pmatrix}$$

Listagem 17: Limitação dos intervalos e resolvendo matriz $\bar{y}$. Complexidade $O(n)$.

```
%ajustando h e calculando f(newton)
f(1) = y0; %derivada de f(x) usando x0
for i = 1:n-1
    intervalo(i, 1) = Xi(i);
    intervalo(i, 2) = Xi(i+1);
    h(i) = Xi(i+1) - Xi(i);
    f(i+1) = (Yi(i+1) - Yi(i))/h(i);

end
hs = [0 h 0];
f(n+1) = yn; %derivada de f(x) usando xn
%ajustando y
for i = 1:n
    y(i) = f(i+1) - f(i);
end
y = 6 * y;
```

Listagem 18: Resolvendo matriz A. Complexidade $O(n)$.

```
%ajustando A
j = -1;
for i = 1:n
    coluna = 1;

    if(j + coluna >= 1 && j + coluna <= (n))
        A(i, coluna+j) = hs(i);
    end

    coluna = coluna + 1;
    if(j + coluna >= 1 && j + coluna <= (n))
        A(i, coluna+j) = 2 * (hs(i) + hs(i+1));
    end

    coluna = coluna + 1;
    if(j + coluna >= 1 && j + coluna <= (n))
        A(i, coluna+j) = hs(i+1);
    end

    j = j + 1;
end
```

Listagem 19: Resolvendo matriz $\bar{g}$. Complexidade $O(n^2)$

```
g(n) = A(n,(n+1))/A(n,n);
i = n-1;         %De trás para frente
while(i >= 1)
    j = n;
    while(j > i)
        g(i) = g(i) - g(j)*A(i,j); %subtrai todos os a's que possuem valor com seus respectivos x's
        j = j - 1;
    end
    g(i) = g(i) + A(i, n+1); %soma com respectivo yi que foi triangulizado
    g(i) = g(i) / A(i,i); %divide com xi respectivo
    i = i - 1;
end
```

Listagem 20: Resolução de s_i. Complexidade $O(n)$

```
function [Yf, I] = calculaMetodo9(X, Y, intervalo, g, h, f, n, x)
    Yf = [];
    I = [];
    for i = 2:n
        if(x >= intervalo(i-1, 1) && x <= intervalo(i-1, 2))
            a = (g(i) - g(i-1))/(6*h(i-1));
            b = g(i)/2;
            c = f(i) + (2*h(i-1)*g(i) + g(i-1)*h(i-1))/6;
            d = Y(i);
            y = a * power(x - X(i), 3) + b * power(x - X(i), 2) + c * (x - X(i)) + d;
            Yf = [Yf y];
            I = [I i-1];
        end
    end
end
```

Analisando a complexidade da Listagem 17, Listagem 18, Listagem 19 e Listagem 20, a complexidade do método de Spline Cúbico Extrapolado é $O(n^2)$.

IV. Desenvolvimento

Ao iniciar a aplicação, o programa realiza uma série de perguntas ao usuário para montar a tabela (x, y). Dentre as perguntas, estão se o usuário irá escrever uma função (ou se irá apenas digitar os valores na tabela), se será igualmente espaçada e se deseja ser informado sobre o progresso dos cálculos que serão realizados durante a execução da aplicação. Após isso, o programa inseres os valores caso não estejam na tabela, verifica se os valores de x estão ordenados e se os pares (x, y) caracterizam uma função. Em seguida, o programa oferece uma lista de métodos (explicados na Seção III) que podem ser aplicados. Com o método escolhido pelo usuário, o programa imprime na tela o polinômio correspondente e oferece a opção de testar ou interpolar o polinômio. Cada método foi desenvolvido em um arquivo separado do programa principal.

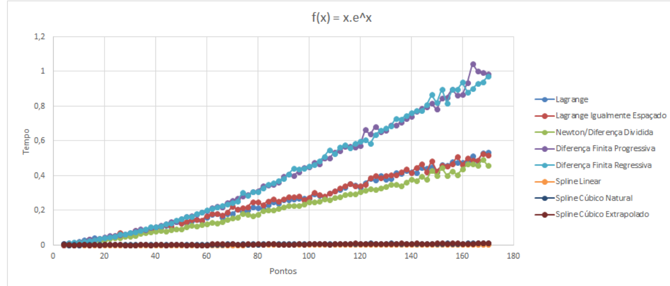
No caso da análise de complexidade de cada método, foi criado uma versão automatizada do mesmo programa, sendo que este não realiza operações de entrada e saída na tela. O tempo analisado foi somente do tempo total de realização dos testes de cada polinômio, não sendo considerado os erros acumulados de cada teste.

Na escolha de quantidade de pontos, foi considerado a maior quantidade de pontos que não causasse *overflow* nem *underflow* em nenhum dos métodos analisados. Uma vez que os métodos de Diferenças Finitas III-D III-E utilizam a operação *fatorial* e o fatorial do MATLAB possui fator de saturação para dados do tipo *double* de *171* [3], o intervalo escolhido para análise foi de $x \in \{4 \cdots 171\}$.

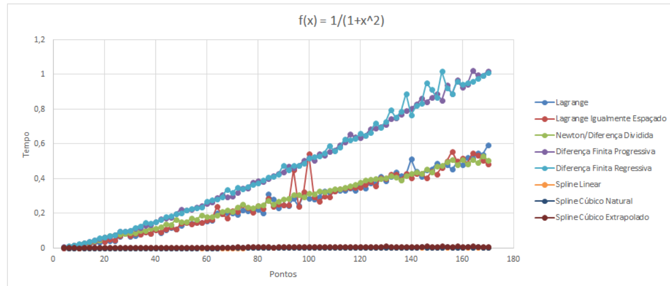
A. *Especificações*

Software MATLAB R2015b

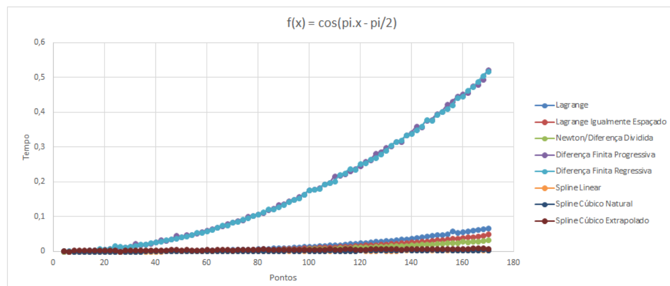
Workspace
- Windows 10;
- 12GB de memória RAM;
- Processador Intel i3 3.5GHz;



(a) $f(x) = xe^x$



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



(c) $f(x) = \cos(\pi x - \frac{\pi}{2})$

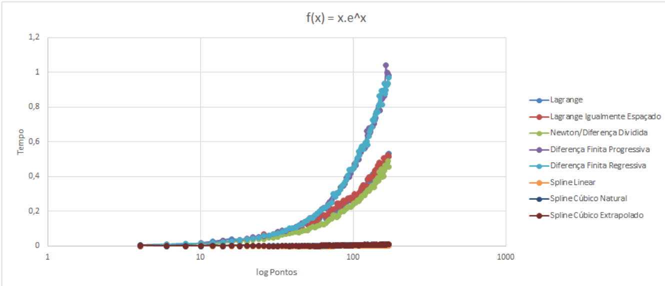
Figura 1: Tempo absoluto de execução dos testes

V. Resultados

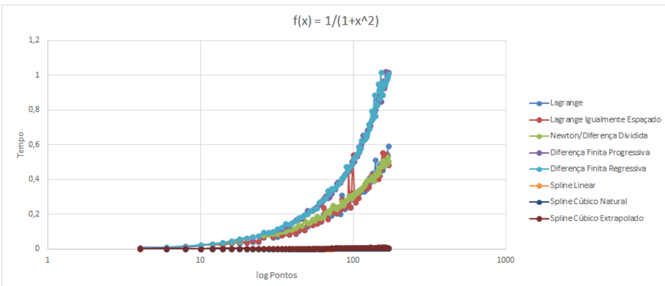
Na Figura 1 é apresentado os tempos absolutos por quantidade de pontos, na Figura 2 é aplicada escala logarítima no tempo de execução dos testes e na Figura 3 é aplicada escala logarítima em ambos os eixos.

Podemos notar, na Figura 1, que independente da função de entrada, os métodos de Diferença Finita (III-D e III-E) possuem o maior tempo de execução comparado com os outros métodos, e isso pode ser explicado devido a operação de *fatorial*. Mesmo que os métodos explicados na Seção III possuem complexidade $O(n^2)$, com exceção do método Spline Linear III-F, podemos reparar a diferença que operações de soma, multiplicação e recursão causam no processamento do algoritmo, sendo que estas diferenças seriam consideradas irrelevantes caso fosse possível uma quantidade infinita de pontos para serem analisados.

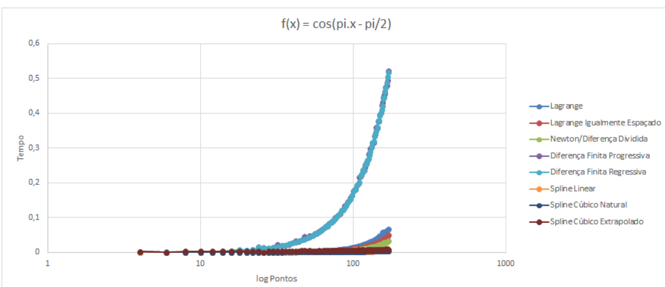
Na Figura 2 fica mais explícito que os métodos estão seguindo sua complexidade descrita. Entretanto, nos métodos de Spline Cúbicos (III-G e III-H) não fica tão



(a) $f(x) = xe^x$

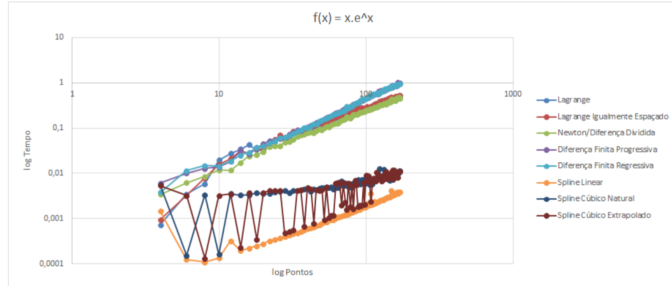


(b) $f(x) = \frac{1}{1+x^2}$

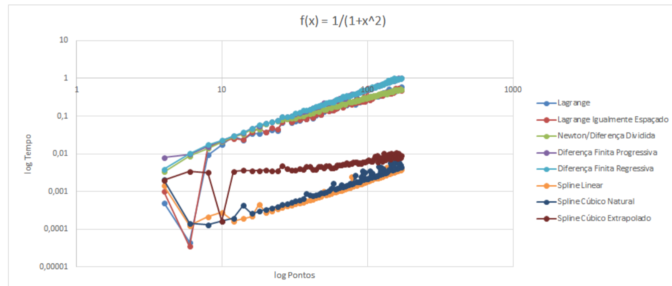


(c) $f(x) = \cos(\pi x - \frac{\pi}{2})$

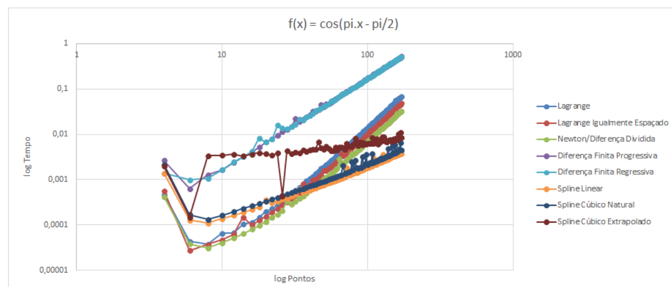
Figura 2: Escala logarítima no tempo de execução



(a) $f(x) = xe^x$



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



(c) $f(x) = \cos(\pi x - \frac{\pi}{2})$

Figura 3: Escala logarítima em ambos os eixos

explícito sua complexidade devido a limitação de pontos imposta pelos outros métodos. Caso fossem analisados separadamente, a quantidade de pontos máximas poderia ser ampliada para *700* pontos sem causar *overflow*.

Na Figura 3 é possível visualizar o custo computacional de todos os métodos, concluindo que o método Spline Linear é o menos custoso computacionalmente se levarmos a quantidade de pontos para infinito.

VI. Trabalhos futuros

Devido a limitação computacional do *workspace* onde foi realizado os testes e do próprio *software*, poderia ser analisado o desenvolvimento da mesma aplicação em outras linguagens matemáticas, como a linguagem R, e que os testes sejam aplicados em uma quantidade maior de computadores, preferencialmente com melhores especificações. Além disso, para que operações que não sejam *loop* não interfiram na análise do algoritmo, é necessário uma otimização das operações realizadas por cada método, como o próprio *fatorial* ou *exponenciação*, com o objetivo de ser possível uma análise de maiores quantidade de pontos.

Referências

[1] Carlos J. S. Alves. Interpolação polinomial. Disponível em http://www.math.tecnico.ulisboa.pt/~calves/cursos/interp/capiii11.html. Acessado em 17 de novembro de 2015.

[2] Frederico Ferreira Campos Filho. *Algoritmos Numéricos*. LTC Editora, 2th edition, 2007.

[3] MathWorks. Factorial. Disponível em http://www.mathworks.com/help/matlab/ref/factorial.html. Acessado em 26 de novembro de 2015.

[4] Paulo Xavier Pamplona. *Apostila de Cálculo Numérico*. 1th edition, 2012.

[5] Reginaldo J. Santos. Introdução ao matlab. Disponível em http://www.mat.ufmg.br/~regi/topicos/intmatl.pdf, Agosto 2005. Acessado em 17 de novembro de 2015.