MÉTODOS NUMÉRICOS PARA INTERPOLAÇÃO POLINOMIAL

Evandro Pedro Alves de Mendonçaa, Marcelino José de Lima Andradea.

a Núcleo de Tecnologia (NTI), Universidade Federal de Pernambuco (UFPE), Campus Acadêmico do Agreste (CAA), Rodovia BR-104, km 59, S/N, Nova Caruaru, CEP. 55.014-900, Caruaru-PE, Brasil, <http://www.ufpe.br/caa>

**Palavras Chave:** Interpolação, polinômios, aproximações, curvas, dados discretos.

**Resumo**: Neste trabalho, serão abordados os diversos métodos de interpolação de funções, como também suas implementações. A partir da análise de cada método, pode-se discernir qual método se aplica melhor a cada situação. No processo de interpolação, estamos em busca de um polinômio que represente uma função mais complicada ou um conjunto de dados experimentais. É fácil entender por qual motivo os polinômios são facilmente computáveis: suas derivadas e integrais são novamente polinômios, suas raízes podem ser encontradas com relativa facilidade, etc. Portanto, é vantajoso substituir uma função complicada por um polinômio que a represente.

1. INTRODUção

Frequentemente necessitasse estimar alguns dados que são intermediários a dados precisos e discretos. Uma das formas de se fazer isso é pelo método da interpolação. Por meio da interpolação, visa-se obter-se uma equação que represente os pontos da seguinte forma:

Sabe-se que existe apenas um polinômio de ordem n que passa exatamente por todos os n+1 pontos necessários. Fica fácil de visualizar tal informação quando analisamos 2 pontos. Sabe-se que existe apenas uma reta que passa entre os dois pontos desejados. Da mesma forma, quando se tem 3 pontos, existe apenas uma parábola que passe exatamente pelos 3. A interpolação consiste em determinar esses polinômios que passam exatamente por esses pontos, polinômio que pode ser de grau n. Por determinar esse polinômio, podem-se estimar valores que estão entre os pontos exatos.

Dentre os vários métodos de interpolação, dois se destacam e são os mais utilizados e populares, são eles: O método das diferenças divididas de Newton e o método de Lagrange. Neste trabalho, esses métodos, além de outros, serão usados na resolução de problemas, como também serão implementados.

1. Exercícios Propostos

Segue, abaixo, a solução dos exercícios propostos sobre o tema.

* 1. 1ª questão

Inicialmente, aplicamos os pontos:

Na função

E encontramos

.

Como na questão foi pedido um polinômio de quarto grau, precisamos de 5 pontos de interpolação. Foram dados seis pontos de interpolação, então vamos ignorar um deles, que é o primeiro ponto, pois ele é o mais distante de seu vizinho imediato. Assim, temos os seguintes vetores:

Para não reescrever tantos dígitos, adotaremos a partir daqui a convenção de que:

* + 1. **─ 1.a)**

Pelo método do polinômio padrão, definiremos a matriz de Vandermonde referente a esses pontos.

Essas equações formam um sistema linear.

Aplicando os valores dos vetores x e y, encontramos os coeficientes do polinômio interpolador:

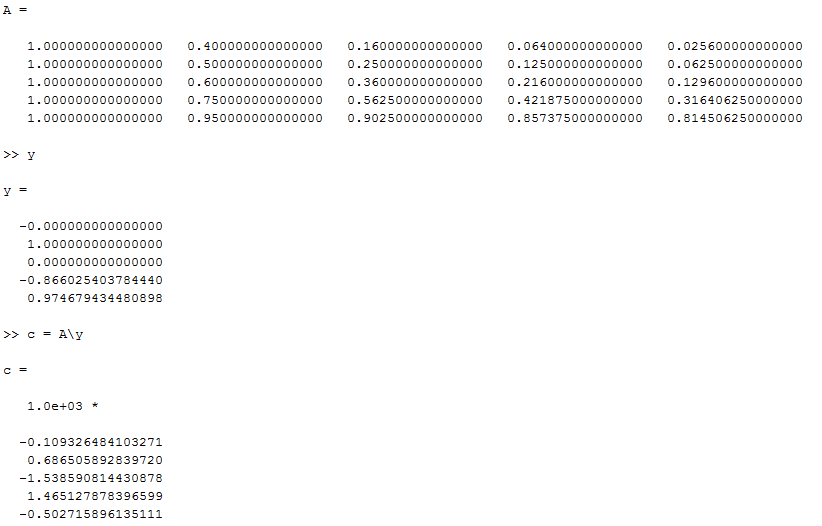


Figura 1: solução do sistema linear usando o MATLAB®.

Assim, temos o polinômio interpolador:

Fazendo de outra forma, pelo algoritmo ~~criado~~ (que consta) no Anexo 1, aplicando os valores dos vetores x e y, e um ponto qualquer para ser interpolado, encontramos os coeficientes ~~do polinômio interpolador~~ e o ~~seu~~ polinômio simbólico:

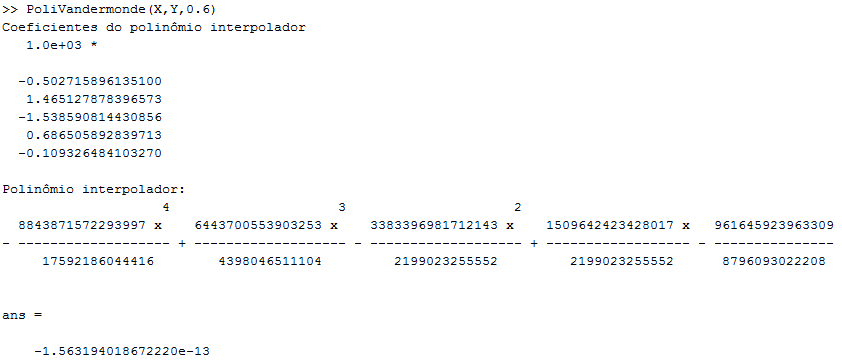


Figura 2: solução usando o método ~~de Vandermonde~~ (padrão) para interpolação.

* + 1. **─ 1.b)**

Utilizando o código do Anexo 2, encontramos os mesmos coeficientes obtidos no item anterior, ou seja, temos o mesmo polinômio interpolador. Isso evidencia a consistência do método e a implementação correta da função. A seguir, é mostrada a resposta da função após fornecermos os parâmetros da questão.

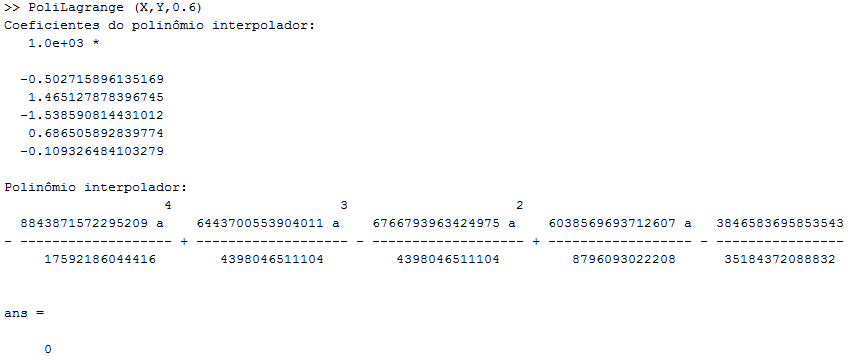


Figura 3: solução usando o método de Lagrange para interpolação.

* + 1. **─ 1.c)**

Utilizando o código do Anexo 3, encontramos novamente os mesmos coeficientes, que nos remetem ao mesmo polinômio interpolador.

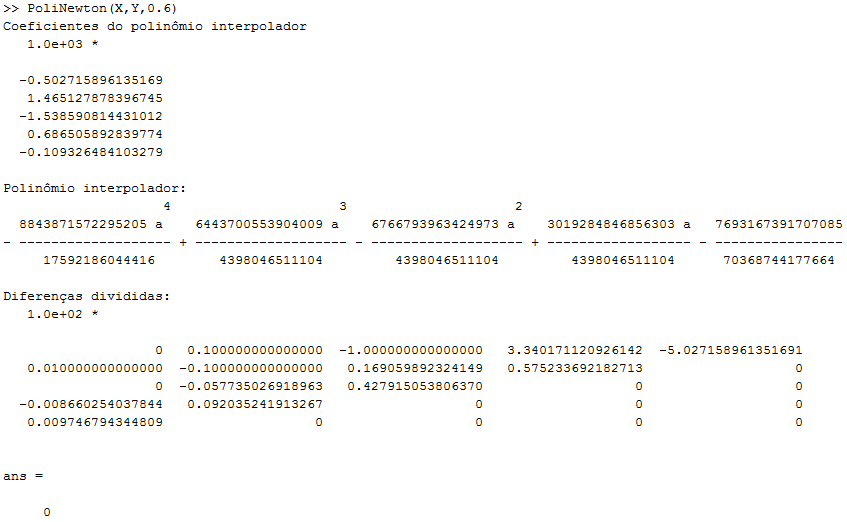


Figura 4: solução usando o método de Newton para interpolação.

* + 1. **─ 1.d)**

Executando a seguinte sequência de comandos no MATLAB®, encontramos o erro:

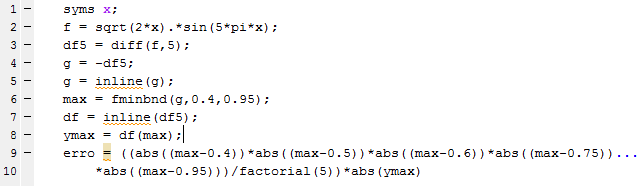


Figura 5: sequência de passos executados para encontrar o erro de truncamento do polinômio interpolador.

* + 1. **─ 1.e)**

Como os polinômios são quase idênticos, consideramos que têm resultados equivalentes. O que diferencia uma função de outra é o tempo de processamento, o que faz com que o método de Newton seja mais eficaz que os outros.

A seguir, é mostrado o gráfico do polinômio interpolador referente aos dados dessa questão.



Figura 6: gráfico do polinômio interpolador.

* 1. 2ª questão
     1. ─ 2.a)

Aplicando o algoritmo do Anexo 3, temos:

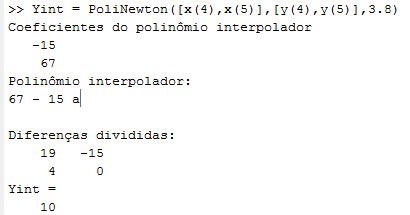


Figura 5: polinômio interpolador de primeiro grau de Newton e interpolação no ponto 3,8.



Figura 6: gráfico do polinômio ~~interpolado~~ (interpolador) de Newton do primeiro grau.

* + 1. **─ 2.b)**

Aplicando o algoritmo do Anexo 3, temos:

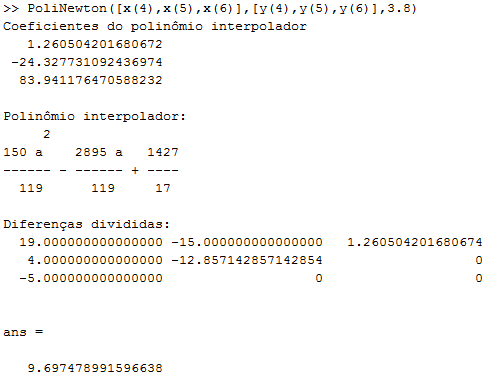


Figura 7: polinômio interpolador de segundo grau de Newton e interpolação no ponto 3,8.



Figura 8: gráfico do polinômio interpolador de Newton do segundo grau (parábola aberta).

* + 1. **─ 2.c)**

Aplicando o algoritmo do Anexo 3, temos:

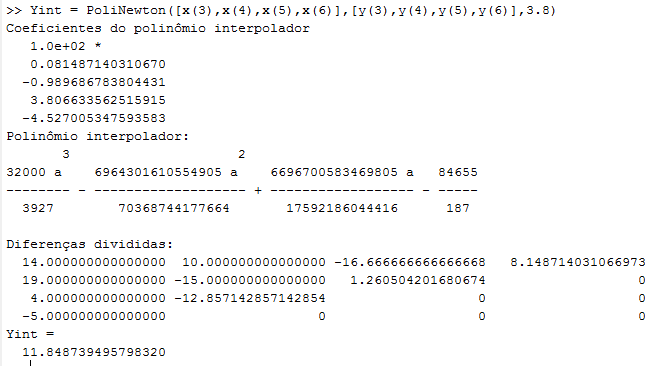


Figura 9: polinômio interpolador de terceiro grau de Newton e interpolação no ponto 3,8.



Figura 10: gráfico do polinômio interpolador de Newton do terceiro grau.

* + 1. **─ 2.d)**

Aplicando o algoritmo do Anexo 5, temos:

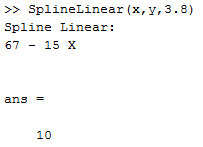
****

Figura 11: Spline interpolador linear e interpolação no ponto 3,8.

****

Figura 12: gráfico do spline interpolador linear.

* + 1. **─ 2.e)**

Aplicando o algoritmo do Anexo 6, temos:

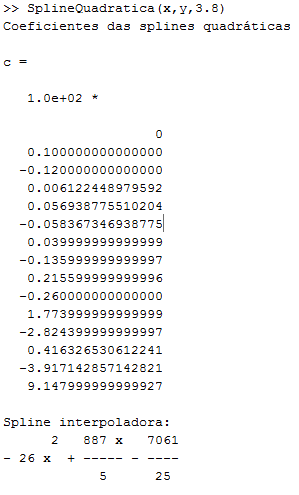
****

Figura 13: Spline interpolador quadrático e interpolação no ponto 3,8.

****

Figura 14: gráfico do spline interpolador quadrático.

* + 1. **─ 2.f)**

Aplicando o algoritmo do Anexo 7, temos:

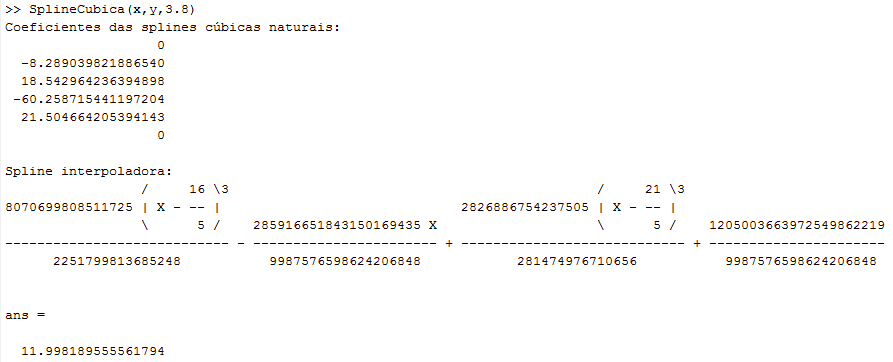
****

Figura 15: Spline interpolador cúbico e interpolação no ponto 3,8.

****

Figura 16: gráfico do spline interpolador cúbico.

* + 1. **─ 2.g)**

Executando a seguinte sequência de comandos no MATLAB®, encontramos o erro:

* + 1. **─ 2.h)**

Os gráficos estão em cada letra respectiva. Em relação ao melhor método, pelo tempo de processamento podemos analisar que o melhor método é o de spline quadrática como mostrado na tabela a seguir:

|  |  |
| --- | --- |
| **Método:** | **Tempo de processamento (segundos):** |
| Newton primeiro grau | 0.502978 |
| Newton segundo grau | 0.668421 |
| Newton terceiro grau | 0.477585 |
| Spline linear | 0.514903 |
| *Spline quadrática* | *0.240139* |
| Spline Cúbica | 0.369546 |

* 1. 3ª questão

A resposta para esta questão consta no Anexo 2.

* 1. 4ª questão

A resposta para esta questão consta no Anexo 3.

* 1. 5ª questão

Para mais detalhes da 5ª questão, vide Anexo 8.

* + 1. **– 5.a)**

Utilizando o código do Anexo 2, os seguintes resultados foram obtidos:

* Para 🡪 .
* Para 🡪 .
  + 1. **– 5.b)**

Utilizando o código do Anexo 3, os seguintes resultados foram obtidos:

* Para 🡪 .
* Para 🡪 .

Como esperado, o mesmo resultado foi obtido, o que mostra a consistência dos métodos.

* + 1. **– 5.c)**

Utilizando a função *interp1* do MATLAB®, os resultados obtidos foram:

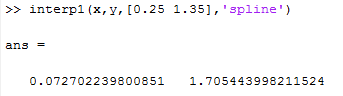


Figura 13: resultados das interpolações dos pontos fornecidos utilizando função nativa do MATLAB® que implementa, dentre outros, o método das splines.

Percebe-se que os resultados obtidos por esse método diferiram um pouco dos resultados encontrados anteriormente, mas ainda assim são muito próximos (com erro na ordem de 10-3). Com relação ao tempo de processamento, temos a seguinte tabela:

|  |  |
| --- | --- |
| Função | Tempo de processamento (s) |
| PoliLagrange | 0,506942 |
| PoliNewton | 0,295237 |
| Interp1 | 0,003055 |

Tabela 1: tempos de processamento de cada função, medidos através do comando “tic, toc” do MATLAB®.

Vemos que a função *interp1* tem melhor tempo de processamento, e que a função *PoliNewton* executou em menor tempo que a função *PoliLagrange*, porém também deve-se levar em consideração que a função *interp1* exibe apenas o resultado da interpolação, enquanto que as demais funções também exibem outros detalhes e também o gráfico. Assim, já era esperado que estas levassem mais tempo para executar que aquela.

* 1. 6ª questão
     1. **– 6.a)**

Utilizando o algoritmo que consta no Anexo 3, encontramos a seguinte solução.

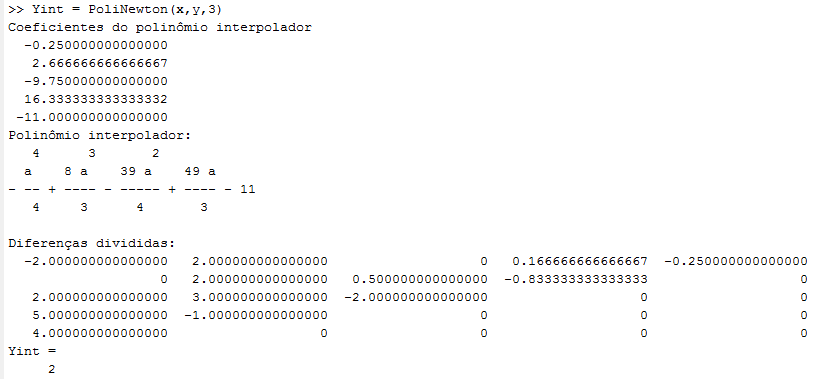


Figura 14: solução utilizando a função PoliNewton.

Portanto, o polinômio interpolador é:

* + 1. **– 6.b)**

Utilizando o algoritmo que consta no Anexo 4, temos:

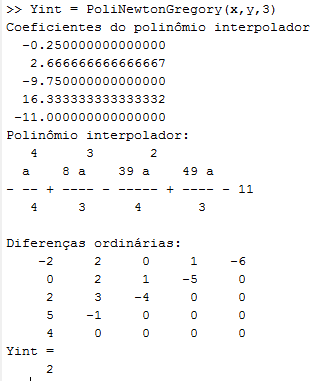


Figura 15: solução utilizando a função PoliNewtonGregory.

Percebe-se que o polinômio encontrado é o mesmo, como já era esperado. A vantagem do método de Newton-Gregory sobre o de Newton é que são feitas muito menos operações de divisão e, assim, o custo computacional é muito menor.

O gráfico do polinômio interpolador é mostrado a seguir.



Figura 16: gráfico do polinômio interpolador.

1. conclusão

Ao longo do trabalho, algumas formas de interpolação foram empregadas para realizar o ajuste de curvas. O ajuste de curvas tem como principal método desenvolvido, o método dos mínimos quadrados, porém, nesse segundo caso (interpolação) consideramos que não existem erros nos dados e podemos exigir que a curva passe pelos pontos dados.

Portanto, O ajuste de curvas por interpolação, se mostra uma ferramenta muito útil para fornecer o valor exato dos pontos pertencentes a um conjunto de dados e um valor estimado entre esses pontos, através de uma fórmula matemática.

REFERÊNCiaS

Chapra, S. C., e Canale, R. P. *Métodos Numéricos para Engenharia*. 5ª edição. Porto Alegre: AMGH, 2011.

Gilat, A., e Subramaniam, V. *Métodos Numéricos para Engenheiros e Cientistas: uma introdução com aplicações usando o MATLAB*. Porto Alegre: Bookman, 2008.

anexo 1



Figura 17: função em MATLAB® que implementa o método padrão para interpolação.

anexo 2

function Ylag = PoliLagrange (x,y,Xlag)  
%função que realiza interpolação usando o método de lagrange  
%Parâmetros: Ylag = PoliLagrange (x,y,Xlag)  
% x e y são os pontos que pertencem ao conjuto de dados  
% Xlag é o ponto a ser interpolado  
% Ylag é a saída: valor interpolado  
  
 %validação  
 if isnumeric(x)==false||isnumeric(y)==false|| ...  
 isnumeric(Xlag)==false %um dos parâmetros não é numérico  
 disp('Erro. Todos os parâmetros devem ser numéricos.');  
 Ylag = 'erro';  
 return;  
 end  
 if(((size(x,1)>1)&&(size(x,2)>1))||((size(y,1)>1)&&(size(y,2)>1)))  
 disp('Erro! X ou Y não é um vetor.');  
 Ylag = 'erro';  
 return;  
 elseif(length(x)~=length(y))%x e y devem ter mesmo número de elementos  
 disp('Erro! Dimensões de X e Y não são equivalentes.');  
 Ylag = 'erro';  
 return;  
 end  
  
 syms a;%variável simbólica usada no cálculo do polinômio interpolador  
 soma = 0; % variável que guarda o resultado do produtório '(Li)\*f(xi)'  
  
 for i = 1:length(x) %laço que soma os produtórios  
 produto = 1; %variável que soma os produtórios '(Li)\*f(xi)'  
 for j = 1:length(x) %laço que realiza o produtório  
 if i~=j %se i=j, x(i) não entra no denominador do produtório  
 produto = produto\*((a - x(j))/(x(i) - x(j))); %produtório  
 end  
 end  
 produto = produto \* y(i); % multiplicação pela 'imagem' da função  
 soma = soma + produto; % somatório dos produtos  
 end  
 %soma agora é simbólica e contém o polinômio interpolador  
 format long;  
  
 disp('Coeficientes do polinômio interpolador:');  
 c = sym2poly(soma);  
 disp(c'); %imprime coeficientes  
  
 %imprimindo polinômio interpolador com variável simbólica  
 poli = 0;  
 j = 0;  
 for i = length(c):-1:1  
 poli = poli + c(i)\*a^(j);  
 j = j + 1;  
 end  
 disp('Polinômio interpolador:');  
 pretty(poli);  
  
 Ylag = sym2poly(subs(soma,a,Xlag)); %interpolação  
  
 %plotando gráfico  
 poli = inline(poli);  
 plot(x,y,'o'), grid on, hold on;  
 x1 = x(1):0.01:x(length(x));  
 y1 = poli(x1);  
 plot(x1,y1), hold off;  
end

Figura 18: função em MATLAB® que implementa o método de Lagrange para interpolação.

anexo 3





Figura 19: função em MATLAB® que implementa o método de Newton para interpolação.

anexo 4

function Yint = PoliNewtonGregory(x,y,Xint)  
%Função que realiza interpolação pelo método de Newton-Gregory  
%Parâmetros: Yint = PoliNewton (x,y,Xint)  
% x e y são os pontos que pertencem ao conjuto de dados  
% Xint é o ponto a ser interpolado  
% Yint é a saída: valor interpolado  
  
 %validação  
 if isnumeric(x)==false||isnumeric(y)==false|| ...  
 isnumeric(Xint)==false %um dos parâmetros não é numérico  
 disp('Erro. Todos os parâmetros devem ser numéricos.');  
 Yint = 'erro';  
 return;  
 end  
 if(((size(x,1)>1)&&(size(x,2)>1))||((size(y,1)>1)&&(size(y,2)>1)))  
 disp('Erro! X ou Y não é um vetor.');  
 Yint = 'erro';  
 return;  
 elseif(length(x)~=length(y))%x e y devem ter mesmo número de elementos  
 disp('Erro! Dimensões de X e Y não são equivalentes.');  
 Yint = 'erro';  
 return;  
 end  
 if length(x)>2  
 anterior = abs(x(2)-x(1));  
 for i = 2:length(x)-1  
 atual = abs(x(i+1)-x(i));  
 if atual~=anterior  
 disp('Erro! Vetor x deve ser igualmente espaçado.');  
 Yint = 'erro';  
 return;  
 end  
 end  
 end  
 syms a;%variável simbólica usada no cálculo do polinômio interpolador  
 diford = zeros(length(y));%matriz que armazena as diferenças ordinárias  
 diford(:,1) = y; %1ª coluna da matriz é o próprio vetor y  
 for j = 2:length(y) %colunas  
 for i = 1:(length(y)+1-j) %linhas  
 diford(i,j) = diford(i+1,j-1)-diford(i,j-1);  
 end  
 end  
  
 soma = diford(1,1); %polinômio interpolador com valor inicial  
 %#ok<\*NASGU>  
 produtorio = 1; %variável que guarda os produtórios  
 h = x(2)-x(1); %tamanho do espaçamento  
 for i = 2:length(x) %laço que constrói o polinômio interpolador  
 produtorio = produtorio \* (a - x(i-1));  
 soma = soma + (diford(1,i)/(factorial(i-1)\*h^(i-1)))\*produtorio;  
 end  
 %soma agora é simbólica e contém o polinômio interpolador  
 format long;  
  
 disp('Coeficientes do polinômio interpolador');  
 %disp('(do menor ao maior grau):');  
 c = sym2poly(soma);  
 disp(c'); %imprime coeficientes  
  
 %imprimindo polinômio interpolador com variável simbólica  
 poli = 0;  
 j = 0;  
 for i = length(c):-1:1  
 poli = poli + c(i)\*a^(j);  
 j = j + 1;  
 end  
 disp('Polinômio interpolador:');  
 pretty(poli);  
  
 disp('Diferenças ordinárias:');  
 disp(diford);  
  
 Yint = sym2poly(subs(soma,a,Xint)); %interpolação  
  
 %plotando gráfico  
 poli = inline(poli);  
 plot(x,y,'o'), grid on, hold on;  
 x1 = x(1):0.01:x(length(x));  
 y1 = poli(x1);  
 plot(x1,y1), hold off;  
end

Figura 20: função em MATLAB® que implementa o método de Newton-Gregory para interpolação.

anexo 5

function [ Yint ] = SplineLinear(x,y,Xint)  
%SplineLinear calcula a interpolação usando splines lineares.  
%Parâmetros: [ Yint ] = SplineLinear(x,y,Xint)  
%x: Vetor com as coordenadas x dos pontos dados.  
%y: Vetor com as coordenadas y dos pontos dados.  
%Xint: Coordenada x do ponto a ser interpolado.  
%Yint: O valor interpolado de Xint.  
  
 %VALIDAÇÃO  
 if isnumeric(x)==false||isnumeric(y)==false|| ...  
 isnumeric(Xint)==false %um dos parâmetros não é numérico  
 disp('Erro. Todos os parâmetros devem ser numéricos.');  
 Yint = 'erro';  
 return;  
 end  
 if(((size(x,1)>1)&&(size(x,2)>1))||((size(y,1)>1)&&(size(y,2)>1)))  
 disp('Erro! X ou Y não é um vetor.');  
 Yint = 'erro';  
 return;  
 elseif(length(x)~=length(y))%x e y devem ter mesmo número de elementos  
 disp('Erro! Dimensões de X e Y não são equivalentes.');  
 Yint = 'erro';  
 return;  
 end  
  
 %PROCESSAMENTO  
 n = length(x);  
 for i = 2:n %i contém o valor final do intervalo de interpolação  
 if (Xint<x(i))  
 break;  
 end  
 end  
  
 %CRIANDO SPLINE E IMPRIMINDO EXPRESSÃO  
 syms X;%declarando variável simbólica  
 spline = (X - x(i))\*y(i-1)/(x(i-1)-x(i)) + ...  
 (X - x(i-1))\*y(i)/(x(i)-x(i-1));%declarando spline simbólica  
 disp('Spline Linear:');  
 pretty(spline);%imprimindo expressão da spline  
  
 %INTERPOLAÇÂO  
 spline = inline(spline); %transformando spline em função inline  
 Yint = spline(Xint); %realizando interpolação  
  
 %PLOTANDO GRÁFICO  
 k = i-1; %ponto inicial da spline  
 plot(x,y,'o'), grid on, hold on;%plotando dados  
 x1 = x(k):0.005:x(i);%intervalo em x  
 y1 = spline(x1);%cálculo das imagens no intervalo  
 plot(x1,y1), hold off;%plotando spline  
end

Figura 21: função em MATLAB® que implementa o método das Splines Lineares para interpolação.

anexo 6

function [ Yint ] = SplineQuadratica(x,y,Xint)  
%SplineQuadratica calcula a interpolação usando splines quadráticas.  
%Parâmetros: [ Yint ] = SplineQuadratica(x,y,Xint)  
%x: Vetor com as coordenadas x dos pontos dados.  
%y: Vetor com as coordenadas y dos pontos dados.  
%Xint: Coordenada x do ponto a ser interpolado.  
%Yint: O valor interpolado de Xint.  
  
 %validação  
 if isnumeric(x)==false||isnumeric(y)==false|| ...  
 isnumeric(Xint)==false %um dos parâmetros não é numérico  
 disp('Erro. Todos os parâmetros devem ser numéricos.');  
 Yint = 'erro';  
 return;  
 end  
 if(((size(x,1)>1)&&(size(x,2)>1))||((size(y,1)>1)&&(size(y,2)>1)))  
 disp('Erro! X ou Y não é um vetor.');  
 Yint = 'erro';  
 return;  
 elseif(length(x)~=length(y))%x e y devem ter mesmo número de elementos  
 disp('Erro! Dimensões de X e Y não são equivalentes.');  
 Yint = 'erro';  
 return;  
 end  
 %processamento  
 n = length(x); %número de pontos  
 ns = n-1; %número de splines  
 nc = 3\*ns-1; %número de coeficientes a encontrar (a1 = 0)  
 %CRIANDO VETOR RESPOSTA  
 b = ones(nc,1); %criando vetor com 3n-1 elementos  
 j = 1; %variável auxiliar  
 for i = 1:2\*ns  
 b(i) = y(j);  
 if mod(i,2)~=0  
 j = j + 1;  
 end  
 end  
 %CRIANDO MATRIZ DOS COEFICIENTES  
 %PRIMEIRA ETAPA: EQUAÇÕES DOS EXTREMOS DAS SPLINES  
 A = zeros(nc);  
 expoente = 2; %variável auxiliar  
 i = 1; %variável auxiliar para linhas  
 k = 1; %variável auxiliar para índice de x  
 for j = 1:nc  
 expoente = expoente - 1;  
 A(i,j) = x(k)^expoente;  
 A(i+1,j) = x(k+1)^expoente;  
 if expoente == 0  
 expoente = 3;  
 i = i + 2; %i pula 2 linhas  
 k = k + 1; %k incrementa  
 end  
 end  
 %SEGUNDA ETAPA: EQUAÇÕES DAS DERIVADAS  
 j = 1; %variável auxiliar para colunas  
 k = 2; %variável auxiliar para índice de x (nós internos)  
 for i = 2\*ns+1:nc  
 if i == 2\*ns+1  
 A(i,j) = 1;  
 A(i,j+2) = -2\*x(k+1);  
 A(i,j+3) = -1;  
 j = j + 2; %j pula 2 colunas  
 else  
 A(i,j) = 2\*x(k);  
 A(i,j+1) = 1;  
 A(i,j+3) = -2\*x(k);  
 A(i,j+4) = -1;  
 j = j + 3; %j pula 3 colunas  
 end  
 k = k + 1; %k incrementa  
 end  
 format long;  
 c = A\b; %resolvendo sistema linear  
 %agora c contém todos os coeficientes exceto a1  
 disp('Coeficientes das splines quadráticas');  
 c = [0;c] %adicionando a1 em c  
 %encontrando o intervalo de interpolação  
 for k = 2:n %k contém o valor final do intervalo de interpolação  
 if (Xint<x(k))  
 break;  
 end  
 end  
 k = k-1; %k agora é o índice da spline que será usada  
 %declarando a spline índice k como um polinômio quadrático simbólico  
 spline = poly2sym([c(3\*k-2),c(3\*k-1),c(3\*k)]);  
 disp('Spline interpoladora:');  
 pretty(spline);%imprimindo spline interpoladora  
 %transformando spline em uma função do tipo inline  
 spline = inline(spline);  
 %avaliando a spline no ponto Xint  
 Yint = spline(Xint); %interpolação  
  
 %plotando gráfico  
 plot(x,y,'o'), grid on, hold on;%plotando dados  
 x1 = x(k):0.005:x(k+1);%intervalo em x  
 y1 = spline(x1);%cálculo das imagens no intervalo  
 plot(x1,y1), hold off;%plotando spline  
end

Figura 22: função em MATLAB® que implementa o método das Splines Quadráticas para interpolação.

anexo 7

function [ Yint ] = SplineCubica(x,y,Xint)  
%SplineCubica calcula a interpolação usando splines cúbicas.  
%Parâmetros: [ Yint ] = SplineCubica(x,y,Xint)  
%x: Vetor com as coordenadas x dos pontos dados.  
%y: Vetor com as coordenadas y dos pontos dados.  
%Xint: Coordenada x do ponto a ser interpolado.  
%Yint: O valor interpolado de Xint.  
  
 %%%%%VALIDAÇÃO%%%%%  
 if isnumeric(x)==false||isnumeric(y)==false|| ...  
 isnumeric(Xint)==false %um dos parâmetros não é numérico  
 disp('Erro. Todos os parâmetros devem ser numéricos.');  
 Yint = 'erro';  
 return;  
 end  
 if(((size(x,1)>1)&&(size(x,2)>1))||((size(y,1)>1)&&(size(y,2)>1)))  
 disp('Erro! X ou Y não é um vetor.');  
 Yint = 'erro';  
 return;  
 elseif(length(x)~=length(y))%x e y devem ter mesmo número de elementos  
 disp('Erro! Dimensões de X e Y não são equivalentes.');  
 Yint = 'erro';  
 return;  
 end  
  
 %%%%%PROCESSAMENTO%%%%%  
 n = length(x); %número de pontos  
 ns = n-1; %número de splines  
 h = zeros(1,ns); %vetor de tamanhos de intervalos  
 for i = 1:ns  
 h(i) = x(i+1)-x(i);%calculando tamanhos de intervalos  
 end  
 %MONTAGEM DO SISTEMA LINEAR PARA ENCONTRAR COEFICIENTES  
 %Como a spline cúbica é natural, a(1)=a(n)=0  
 b = zeros(n-2,1); %declarando vetor coluna inicialmente nulo  
 for i = 1:n-2 %montando vetor resposta do sistema linear  
 b(i) = 6\*(((y(i+2)-y(i+1))/h(i+1))-((y(i+1)-y(i))/h(i)));  
 end  
 A = zeros(n-2); %declaração da matriz dos coeficientes (nula)  
 j = 1; %variável auxiliar para colunas  
 for i = 1:n-2 %calculando elementos da matriz A  
 if i == 1 %primeira linha  
 A(i,j) = 2\*(h(i)+h(i+1)); %coeficiente de a(2)  
 A(i,j+1) = h(i+1); %coeficiente de a(3)  
 elseif i == n-2 %última linha  
 A(i,j) = h(i); %coeficiente de a(n-3)  
 A(i,j+1) = 2\*(h(i)+h(i+1)); %coeficiente de a(n-2)  
 else %linhas do meio  
 A(i,j) = h(i); %coeficiente de a(i)  
 A(i,j+1) = 2\*(h(i)+h(i+1)); %coeficiente de a(i+1)  
 A(i,j+2) = h(i+1); %coeficiente de a(i+2)  
 j = j + 1; %incremento de j  
 end  
 end  
 %SOLUÇÃO DO SISTEMA LINEAR  
 format long;  
 a = A\b;%vetor a contém os coeficientes de a(2) até a(n-1)  
 a = [0;a;0];%adicionando os demais coeficientes  
  
 %%%%%SAÍDA%%%%%  
 disp('Coeficientes das splines cúbicas naturais:');  
 disp(a);  
 %encontrando o intervalo de interpolação  
 for k = 2:n %k contém o ponto final do intervalo de interpolação  
 if (Xint<x(k))  
 break;  
 end  
 end  
 i = k-1; %i é o ponto inicial do intervalo de interpolação  
 %criando spline  
 syms X;  
 spline = (a(i)/(6\*h(i)))\*((x(i+1)-X)^3);%1ª parte  
 spline = spline + (a(i+1)/(6\*h(i)))\*((X-x(i))^3);%2ª parte  
 spline = spline + ((y(i)/h(i))-((a(i)\*h(i))/6))\*(x(i+1)-X);%3ª parte  
 spline = spline + ((y(i+1)/h(i))-((a(i+1)\*h(i))/6))\*(X-x(i));%4ª parte  
 disp('Spline interpoladora:');  
 pretty(spline);%imprimindo spline interpoladora  
 %transformando spline em uma função do tipo inline  
 spline = inline(spline);  
 %avaliando a spline no ponto Xint  
 Yint = spline(Xint); %INTERPOLAÇÃO  
  
 %plotando gráfico  
 plot(x,y,'o'), grid on, hold on;%plotando dados  
 x1 = x(i):0.005:x(k);%intervalo em x  
 y1 = spline(x1);%cálculo das imagens no intervalo  
 plot(x1,y1), hold off;%plotando spline  
end

Figura 23: função em MATLAB® que implementa o método das Splines Quadráticas para interpolação.

anexo 8

Detalhes da 5ª questão.

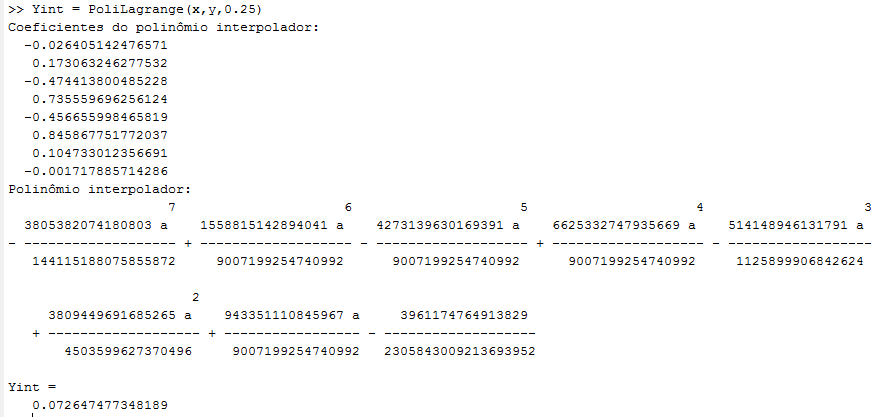


Figura 24: interpolação usando função que implementa o método de Lagrange para o primeiro ponto fornecido.

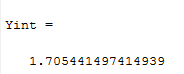


Figura 25: interpolação usando função que implementa o método de Lagrange para o segundo ponto fornecido (obs.: os coeficientes e o polinômio interpolador são os mesmos da figura acima).

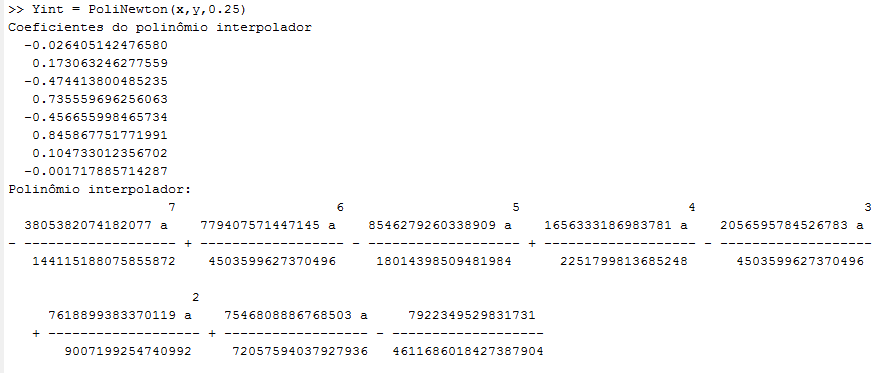


Figura 26: interpolação usando função que implementa o método de Newton para o primeiro ponto fornecido (parte 1).

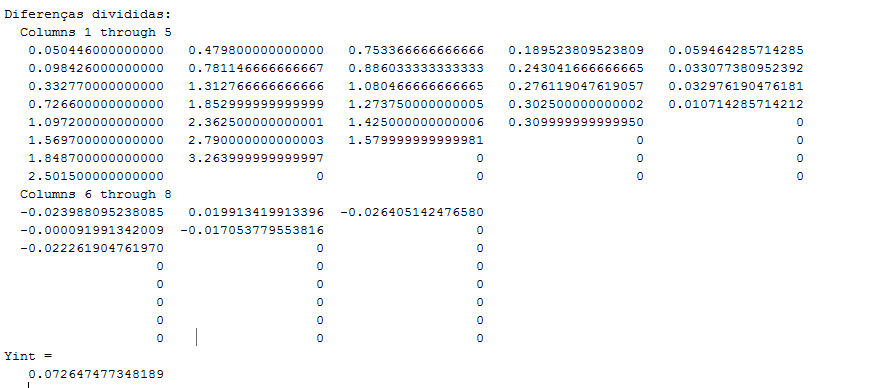


Figura 27: interpolação usando função que implementa o método de Newton para o primeiro ponto fornecido (parte 2).

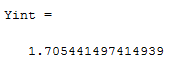


Figura 28: interpolação usando função que implementa o método de Newton para o segundo ponto fornecido (obs.: os coeficientes e o polinômio interpolador são os mesmos da figura acima).



Figura 29: gráfico do polinômio interpolador.