MÉTODOS NUMÉRICOS PARA INTERPOLAÇÃO POLINOMIAL

Evandro Pedro Alves de Mendonça[a]**, Marcelino José de Lima Andrade**[a].

[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:

$$F(x) = a + bx + cx^2 + \cdots + kx^n$$

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.

2 EXERCÍCIOS PROPOSTOS

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

2.1 1ª questão

Inicialmente, aplicamos os pontos:

$$\vec{x} = (0{,}15; 0{,}4; 0{,}5; 0{,}6; 0{,}75; 0{,}95)$$

Na função

$$f(x) = \sqrt{2x} sen(5\pi x)$$

E encontramos

$$\vec{y} = (0{,}3873; 0; 1; 0; -0{,}866025403784440; 0{,}974679434480898).$$

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:

$$\vec{x} = (0{,}4; 0{,}5; 0{,}6; 0{,}75; 0{,}95)$$
$$\vec{y} = (0; 1; 0; -0{,}866025403784440; 0{,}974679434480898)$$

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

$$\vec{x} = (x_0, x_1, x_2, x_3, x_4)$$
$$\vec{y} = (y_0, y_1, y_2, y_3, y_4)$$

1.1.1 − 1.a)

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

$$a + bx_0 + cx_o^2 + dx_o^3 + ex_o^4 = y_0$$
$$a + bx_1 + cx_1^2 + dx_1^3 + ex_1^4 = y_1$$
$$a + bx_2 + cx_2^2 + dx_2^3 + ex_2^4 = y_2$$
$$a + bx_3 + cx_3^2 + dx_3^3 + ex_3^4 = y_3$$
$$a + bx_4 + cx_4^2 + dx_4^3 + ex_4^4 = y_4$$

Essas equações formam um sistema linear.

$$\begin{bmatrix} 1 & x_0 & x_o^2 & x_o^3 & x_o^4 \\ 1 & x_1 & x_1^2 & x_1^3 & x_1^4 \\ 1 & x_2 & x_2^2 & x_2^3 & x_2^4 \\ 1 & x_3 & x_3^2 & x_3^3 & x_3^4 \\ 1 & x_4 & x_4^2 & x_4^3 & x_4^4 \end{bmatrix} \times \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

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

```
A =

   1.000000000000000   0.400000000000000   0.160000000000000   0.064000000000000   0.025600000000000
   1.000000000000000   0.500000000000000   0.250000000000000   0.125000000000000   0.062500000000000
   1.000000000000000   0.600000000000000   0.360000000000000   0.216000000000000   0.129600000000000
   1.000000000000000   0.750000000000000   0.562500000000000   0.421875000000000   0.316406250000000
   1.000000000000000   0.950000000000000   0.902500000000000   0.857375000000000   0.814506250000000

>> y

y =

  -0.000000000000000
   1.000000000000000
   0.000000000000000
  -0.866025403784440
   0.974679434480898

>> c = A\y

c =

   1.0e+03 *

  -0.109326484103271
   0.686505892839720
  -1.538590814430878
   1.465127878396599
  -0.502715896135111
```

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

Assim, temos o polinômio interpolador:

$$P_4(x) = -109{,}326484103271 + 686{,}505892839720x - 1538{,}590814430878x^2 + 1465{,}127878396599x^3 - 502{,}715896135111x^4$$

Fazendo de outra forma, pelo algoritmo criado 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:

```
>> PoliVandermonde(X,Y,0.6)
Coeficientes do polinômio interpolador
   1.0e+03 *

  -0.502715896135100
   1.465127878396573
  -1.538590814430856
   0.686505892839713
  -0.109326484103270

Polinômio interpolador:
                    4                    3                    2
  8843871572293997 x    6443700553903253 x    3383396981712143 x    1509642423428017 x    961645923963309
- ------------------- + ------------------- - ------------------- + ------------------- - ---------------
    17592186044416        4398046511104         2199023255552         2199023255552       8796093022208


ans =

   -1.563194018672220e-13
```

Figura 2: solução usando o método de Vandermonde para interpolação.

1.1.2 — 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.

```
>> PoliLagrange (X,Y,0.6)
Coeficientes do polinômio interpolador:
   1.0e+03 *

  -0.502715896135169
   1.465127878396745
  -1.538590814431012
   0.686505892839774
  -0.109326484103279

Polinômio interpolador:
                    4                    3                    2
  8843871572295209 a    6443700553904011 a    6766793963424975 a    6038569693712607 a    3846583695853543
- ------------------- + ------------------- - ------------------- + ------------------- - ----------------
    17592186044416        4398046511104         4398046511104         8796093022208       35184372088832


ans =

     0
```

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

1.1.3 — 1.c)

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

```
>> PoliNewton(X,Y,0.6)
Coeficientes do polinômio interpolador
   1.0e+03 *

  -0.502715896135169
   1.465127878396745
  -1.538590814431012
   0.686505892839774
  -0.109326484103279

Polinômio interpolador:
                      4                     3                     2
  8843871572295205 a    6443700553904009 a    6766793963424973 a    3019284846856303 a    7693167391707085
- ------------------- + ------------------- - ------------------- + ------------------ - ----------------
    17592186044416        4398046511104         4398046511104         4398046511104       70368744177664

Diferenças divididas:
   1.0e+02 *

                  0   0.100000000000000  -1.000000000000000   3.340171120926142  -5.027158961351691
   0.010000000000000  -0.100000000000000   0.169059892324149   0.575233692182713                   0
                  0  -0.057735026918963   0.427915053806370                   0                   0
  -0.008660254037844   0.092035241913267                   0                   0                   0
   0.009746794344809                   0                   0                   0                   0


ans =

     0
```

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

1.1.4 — 1.d)

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

$$|E_4(x)| \leq 2{,}550193195120122$$

```
syms x;
f = sqrt(2*x).*sin(5*pi*x);
df5 = diff(f,5);
g = -df5;
g = inline(g);
max = fminbnd(g,0.4,0.95);
df = inline(df5);
ymax = df(max);
erro = ((abs((max-0.4))*abs((max-0.5))*abs((max-0.6))*abs((max-0.75))...
    *abs((max-0.95)))/factorial(5))*abs(ymax)
```

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

1.1.5 — 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.

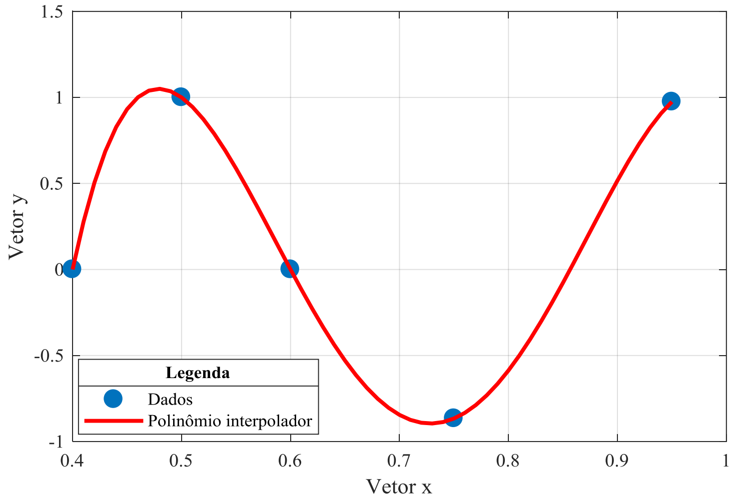

Figura 6: gráfico do polinômio interpolador.

2.2 2ª questão

Parâmetros no MATLAB:

```
>> x = [1.4,2,2.7,3.2,4.2,4.9]

x =

    1.4000    2.0000    2.7000    3.2000    4.2000    4.9000

>> y = [2,8,14,19,4,-5]

y =

     2     8    14    19     4    -5
```

Figura 7: Parâmetros da questão 2.

2.2.1 — 2.a)

Aplicando o algoritmo do Anexo 3, temos:

```
>> Yint = PoliNewton([x(4),x(5)],[y(4),y(5)],3.8)
Coeficientes do polinômio interpolador
   -15
    67
Polinômio interpolador:
67 - 15 a

Diferenças divididas:
    19   -15
     4     0
Yint =
    10
```

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

2.2.2— 2.b)

Aplicando o algoritmo do Anexo 3, temos:

```
>> PoliNewton([x(4),x(5),x(6)],[y(4),y(5),y(6)],3.8)
Coeficientes do polinômio interpolador
   1.260504201680672
 -24.327731092436974
  83.941176470588232

Polinômio interpolador:
     2
150 a    2895 a   1427
------ - ------ + ----
  119      119     17

Diferenças divididas:
  19.000000000000000 -15.000000000000000   1.260504201680674
   4.000000000000000 -12.857142857142854                   0
  -5.000000000000000                   0                   0


ans =

   9.697478991596638
```

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

2.2.3— 2.c)

Aplicando o algoritmo do Anexo 3, temos:

```
>> Yint = PoliNewton([x(3),x(4),x(5),x(6)],[y(3),y(4),y(5),y(6)],3.8)
Coeficientes do polinômio interpolador
   1.0e+02 *
   0.081487140310670
  -0.989686783804431
   3.806633562515915
  -4.527005347593583
Polinômio interpolador:
       3                     2
32000 a    6964301610554905 a    6696700583469805 a    84655
-------- - ------------------- + ------------------ - -----
  3927        70368744177664       17592186044416      187

Diferenças divididas:
  14.000000000000000  10.000000000000000 -16.666666666666668   8.148714031066973
  19.000000000000000 -15.000000000000000   1.260504201680674                   0
   4.000000000000000 -12.857142857142854                   0                   0
  -5.000000000000000                   0                   0                   0
Yint =
  11.848739495798320
```

Figura 10: polinômio interpolador de terceiro grau de Newton e interpolação no ponto 3,8.

2.2.4— 2.d)

Aplicando o algoritmo do Anexo 5, temos:

```
>> SplineLinear(x,y,3.8)
Spline Linear:
67 - 15 X


ans =

    10
```

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

2.2.5— 2.e)

Aplicando o algoritmo do Anexo 6, temos:

```
>> SplineQuadratica(x,y,3.8)
Coeficientes das splines quadráticas

c =

   1.0e+02 *

                  0
   0.100000000000000
  -0.120000000000000
   0.006122448979592
   0.056938775510204
  -0.058367346938775
   0.039999999999999
  -0.135999999999997
   0.215599999999996
  -0.260000000000000
   1.773999999999999
  -2.824399999999997
   0.416326530612241
  -3.917142857142821
   9.147999999999927

Spline interpoladora:
      2   887 x   7061
- 26 x  + ----- - ----
            5      25

ans =

  16.240000000000009
```

Figura 12: Spline interpolador quadrático e interpolação no ponto 3,8.

2.2.6— 2.f)

Aplicando o algoritmo do Anexo 7, temos:

```
>> SplineCubica(x,y,3.8)
Coeficientes das splines cúbicas naturais:
                   0
  -8.289039821886540
  18.542964236394898
 -60.258715441197204
  21.504664205394143
                   0

Spline interpoladora:
                 /     16 \3                                                  /     21 \3
8070699808511725 | X - -- |                              2826886754237505 | X - -- |
                 \      5 /    285916651843150169435 X                         \      5 /    1205003663972549862219
---------------------------- - ------------------------ + ---------------------------- + ----------------------
      2251799813685248          9987576598624206848          281474976710656              9987576598624206848


ans =

  11.998189555561794
```

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

2.2.7— 2.g)

Erro:

Newton primeiro grau:

$$R_1 = (x - x_0) \times (x - x_1) \times f[x_2, x_1, x_0]$$
$$= (x - 3{,}2) \times (x - 4{,}2) \times (1{,}260504201680674)$$

$$= (3{,}8 - 3{,}2) \times (3{,}8 - 4{,}2) \times (1{,}260504201680674)$$
$$= \mathbf{-0{,}302521008}$$

Newton segundo grau:

$$R_2 = (x - x_0) \times (x - x_1) \times (x - x_2) \times f[x_3, x_2, x_1, x_0]$$
$$= (x - 3{,}2) \times (x - 4{,}2) \times (x - 4{,}9) \times (8{,}148714031066973)$$
$$= (3{,}8 - 3{,}2) \times (3{,}8 - 4{,}2) \times (3{,}8 - 4{,}9) \times (8{,}148714031066973)$$
$$= \mathbf{2{,}151260504}$$

Newton terceiro grau:

Para este caso, precisaremos de mais um ponto para utilizar a sua última diferença dividida, nos cálculos dos erros anteriores esses valões de diferenças divididas já estavam calculados nas letras (b) e (c) da questão 2. Então:

```
>> PoliNewton([x(2),x(3),x(4),x(5),x(6)],[y(2),y(3),y(4),y(5),y(6)],3.8)
Coeficientes do polinômio interpolador
   1.0e+02 *

   0.056088266027414
  -0.759836850100542
   3.660591352528472
  -7.413599703204166
   5.446116540660151

Polinômio interpolador:
       4                      3                     2
91250 a    5346876492148705 a    402486275664133 a    3260535630939873 a    4790454770143233
-------- - ------------------- + ------------------ - ------------------ + ----------------
  16269       70368744177664        1099511627776        4398046511104       8796093022208

Diferenças divididas:
   8.000000000000000   8.571428571428569   1.190476190476192  -8.116883116883118   5.608826602741409
  14.000000000000000  10.000000000000000 -16.666666666666668   8.148714031066973                   0
  19.000000000000000 -15.000000000000000   1.260504201680674                   0                   0
   4.000000000000000 -12.857142857142854                   0                   0                   0
  -5.000000000000000                   0                   0                   0                   0


ans =

  13.477542741234425
```

$$R_2 = (x - x_0) \times (x - x_1) \times (x - x_2) \times (x - x_2) \times f[x_4, x_3, x_2, x_1, x_0]$$
$$= (x - 2{,}7) \times (x - 3{,}2) \times (x - 4{,}2) \times (x - 4{,}9)$$
$$\times (5{,}608826602741409)$$
$$= (3{,}8 - 2{,}7) \times (3{,}8 - 3{,}2) \times (3{,}8 - 4{,}2) \times (3{,}8 - 4{,}9)$$
$$\times (5{,}608826602741409) = \mathbf{1{,}628803245}$$

Analisando os erros da letra referente ao método de Newton, podemos ver que o menor erro é do método de Newton do primeiro grau, *dentre os graus de Newton, apresentou a melhor aproximação.*

2.2.8 — 2.h)

Os gráficos estão presentes no Anexo 8. Analisando-os, o melhor método é o de Spline Cúbica, pois leva em consideração todos os pontos do conjunto de dados, enquanto que os polinômios de Newton só levam em consideração alguns dos pontos para calcular a tendência geral dos dados.

Pelo tempo de processamento dos algoritmos implementados, podemos analisar que o melhor método em questão de “processamento” é o da spline quadrática como mostrado na tabela a seguir. Porém, para um conjunto de dados maior, as Splines quadráticas podem não ser vantajosas devido ao fato de terem que resolver sistemas lineares com 3n-3 equações, enquanto que as Splines Cúbicas precisam resolver um sistema linear de apenas n-2 equações.

Método:	*Tempo de processamento (s):*
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

Tabela 1: tempos de processamento de cada método, obtidos pelo comando "*tic*, *toc*" do MATLAB®.

2.3 3ª questão

A resposta para esta questão consta no Anexo 2.

2.4 4ª questão

A resposta para esta questão consta no Anexo 3.

2.5 5ª questão

Para mais detalhes da 5ª questão, vide Anexo 9.

2.5.1– 5.a)

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

- Para $x = 0{,}25 \rightarrow y = 0{,}072647477348189$.
- Para $x = 1{,}35 \rightarrow y = 1{,}705441497414939$.

2.5.2– 5.b)

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

- Para $x = 0{,}25 \rightarrow y = 0{,}072647477348189$.
- Para $x = 1{,}35 \rightarrow y = 1{,}705441497414939$.

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

2.5.3– 5.c)

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

```
>> interp1(x,y,[0.25 1.35],'spline')

ans =

   0.072702239800851   1.705443998211524
```

Figura 14: 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 2: tempos de processamento de cada função, obtidos pelo 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.

2.6 6ª questão

2.6.1 – 6.a)

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

```
>> Yint = PoliNewton(x,y,3)
Coeficientes do polinômio interpolador
  -0.250000000000000
   2.666666666666667
  -9.750000000000000
  16.333333333333332
 -11.000000000000000
Polinômio interpolador:
   4      3       2
  a    8 a    39 a    49 a
- -- + ---- - ----- + ---- - 11
   4     3      4       3

Diferenças divididas:
  -2.000000000000000   2.000000000000000                   0   0.166666666666667  -0.250000000000000
                   0   2.000000000000000   0.500000000000000  -0.833333333333333                   0
   2.000000000000000   3.000000000000000  -2.000000000000000                   0                   0
   5.000000000000000  -1.000000000000000                   0                   0                   0
   4.000000000000000                   0                   0                   0                   0
Yint =
     2
```

Figura 15: solução utilizando a função PoliNewton.

Portanto, o polinômio interpolador é:

$$\boxed{P_5(x) = -11 + 16{,}33x - 9{,}75x^2 + 2{,}66x^3 - 0{,}25x^4}$$

2.6.2 – 6.b)

Utilizando o algoritmo que consta no Anexo 4, temos:

```
>> Yint = PoliNewtonGregory(x,y,3)
Coeficientes do polinômio interpolador
  -0.250000000000000
   2.666666666666667
  -9.750000000000000
  16.333333333333332
 -11.000000000000000
Polinômio interpolador:
   4      3       2
  a    8 a    39 a    49 a
- -- + ---- - ----- + ---- - 11
   4     3      4       3

Diferenças ordinárias:
    -2     2     0     1    -6
     0     2     1    -5     0
     2     3    -4     0     0
     5    -1     0     0     0
     4     0     0     0     0
Yint =
     2
```

Figura 16: 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.

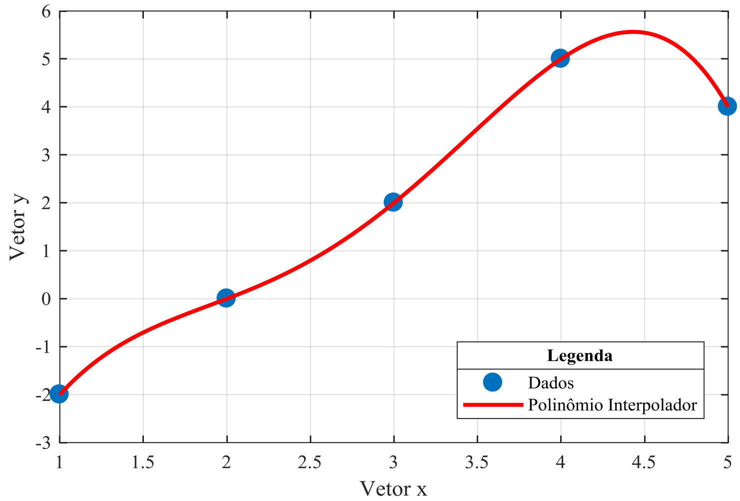

Figura 17: gráfico do polinômio interpolador.

3 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. A técnica de interpolação é usada para estimar um valor entre dois pontos do conjunto de dados, para um valor que esteja fora do conjunto de dados, uma técnica semelhante pode ser empregada, que é a extrapolação. Os polinômios interpoladores têm bastantes aplicações, sendo uma delas, de grande importância, que é a integração numérica.

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

```
function [ Yint ] = PoliVandermonde(x,y,Xint)
%Função que realiza interpolação pelo método padrão, utilizando a matriz de
%Vandermonde.
%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

    %processamento
    A = ones(length(x));% matriz dos coeficientes inicialmente unitária
    for i = 2:length(x) %criando matriz de Vandermonde
        A(:,i) = x.^(i-1);
    end

    %resultados
    format long;
    disp('Coeficientes do polinômio interpolador');
    d = A\y'; %solução do sistema linear
    c = d; %criando um vetor do tamanho de d
    j = length(x); %variável auxiliar
    for i = 1:j %invertendo vetor de coeficientes
        c(i) = d(j);
        j = j - 1;
    end
    disp(c);
    %imprimindo polinômio interpolador com variável simbólica
    poli = poly2sym(c);
    disp('Polinômio interpolador:');
    pretty(poli);
    poli = inline(poli);

    Yint = poli(Xint); %interpolação

    %plotando gráfico
    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 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 19: função em MATLAB® que implementa o método de Lagrange para interpolação.

ANEXO 3

```
function Ylag = PoliNewton(x,y,Xlag)
%Função que realiza interpolação pelo método de Newton
%Parâmetros: Ylag = PoliNewton (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
    difdiv = zeros(length(y)); %matriz que armazena as diferenças divididas
    difdiv(:,1) = y; %1ª coluna da matriz é o próprio vetor y
    aux = 1; %variável auxiliar
    for j = 2:length(y) %colunas
        for i = 1:(length(y)+1-j) %linhas
            difdiv(i,j) = (difdiv(i,j-1)-difdiv(i+1,j-1))/(x(i)-x(i+aux));
        end
        aux = aux + 1;
    end

    soma = difdiv(1,1); %polinômio interpolador com valor inicial
    %#ok<*NASGU>
    produtorio = 1; %variável que guarda os produtórios
    for i = 2:length(x) %laço que constrói o polinômio interpolador
        produtorio = produtorio * (a - x(i-1));
        soma = soma + difdiv(1,i)*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 divididas:');
    disp(difdiv);

    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 20: 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 21: 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

    %INTERPOLAÇÃO
    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 22: 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 23: 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 24: função em MATLAB® que implementa o método das *Splines* Quadráticas para interpolação.

ANEXO 8

Detalhes da 2ª questão letra (h) - GRÁFICOS

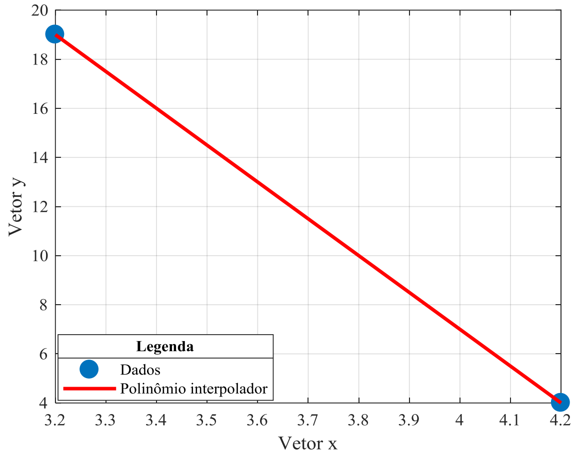

Figura 25: gráfico do polinômio interpolado de Newton do primeiro grau.

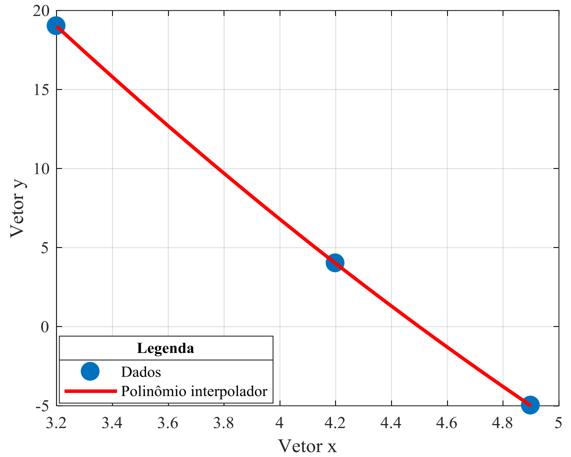

Figura 26: gráfico do polinômio interpolador de Newton do segundo grau (parábola aberta).

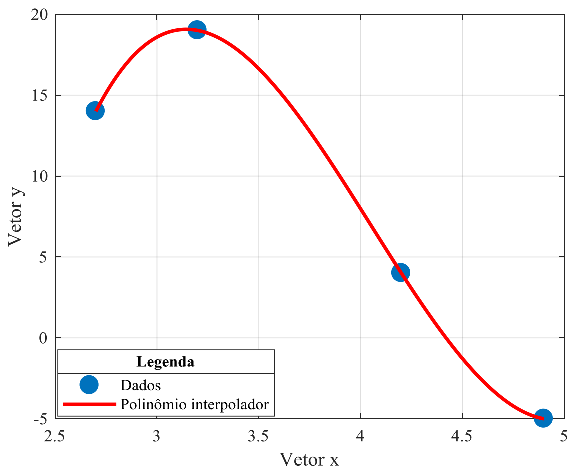

Figura 27: gráfico do polinômio interpolador de Newton do terceiro grau.

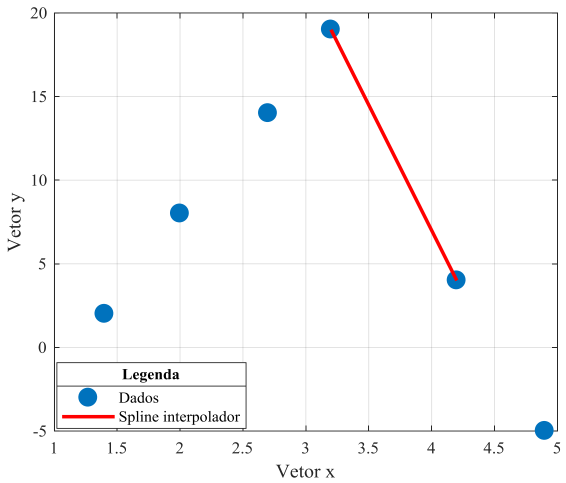

Figura 28: gráfico do spline interpolador linear.

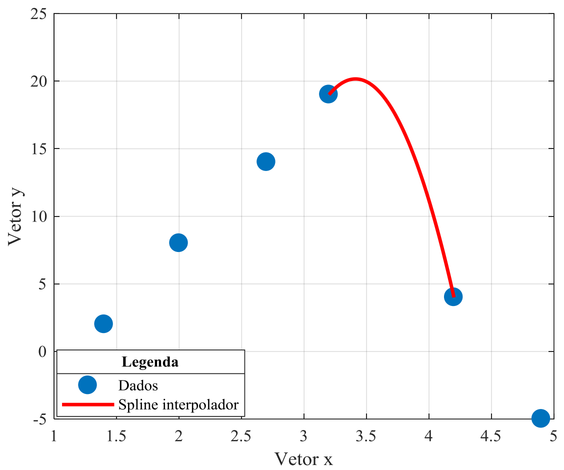

Figura 29: gráfico do spline interpolador quadrático.

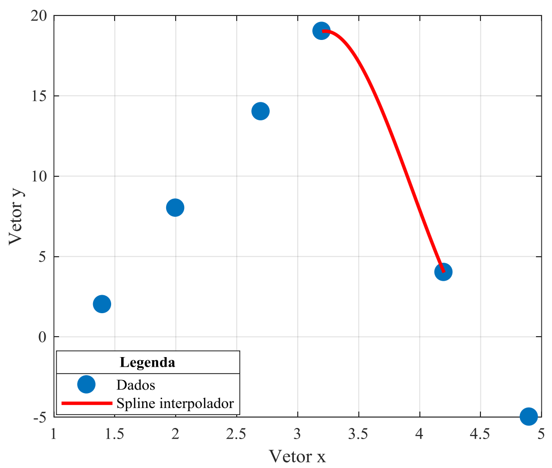

Figura 30: gráfico do spline interpolador cúbico.

ANEXO 9

Detalhes da 5ª questão.

```
>> Yint = PoliLagrange(x,y,0.25)
Coeficientes do polinômio interpolador:
  -0.026405142476571
   0.173063246277532
  -0.474413800485228
   0.735559696256124
  -0.456655998465819
   0.845867751772037
   0.104733012356691
  -0.001717885714286
Polinômio interpolador:
                    7                    6                    5                    4                   3
  3805382074180803 a    1558815142894041 a    4273139630169391 a    6625332747935669 a    514148946131791 a
- ------------------ + ------------------ - ------------------ + ------------------ - ------------------
   14411518807585587 2    9007199254740992      9007199254740992      9007199254740992     1125899906842624

                       2
     3809449691685265 a    943351110845967 a     3961174764913829
   + ------------------ + ---------------- - ------------------
       4503599627370496     9007199254740992   2305843009213693952

Yint =
   0.072647477348189
```

Figura 31: interpolação usando função que implementa o método de Lagrange para o primeiro ponto fornecido.

```
Yint =

    1.705441497414939
```

Figura 32: 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).

```
>> Yint = PoliNewton(x,y,0.25)
Coeficientes do polinômio interpolador
  -0.026405142476580
   0.173063246277559
  -0.474413800485235
   0.735559696256063
  -0.456655998465734
   0.845867751771991
   0.104733012356702
  -0.001717885714287
Polinômio interpolador:
                    7                   6                    5                    4                    3
  3805382074182077 a    779407571447145 a    8546279260338909 a    1656333186983781 a    2056595784526783 a
- ------------------ + ---------------- - ------------------ + ------------------ - ------------------
   14411518807585587 2    4503599627370496    18014398509481984      2251799813685248      4503599627370496

                       2
     7618899383370119 a    7546808886768503 a     7922349529831731
   + ------------------ + ---------------- - ------------------
       9007199254740992    72057594037927936   4611686018427387904
```

Figura 33: interpolação usando função que implementa o método de Newton para o primeiro ponto fornecido (parte 1).

```
Diferenças divididas:
  Columns 1 through 5
   0.050446000000000   0.479800000000000   0.753366666666666   0.189523809523809   0.059464285714285
   0.098426000000000   0.781146666666667   0.886033333333333   0.243041666666665   0.033077380952392
   0.332770000000000   1.312766666666666   1.080466666666665   0.276119047619057   0.032976190476181
   0.726600000000000   1.852999999999999   1.273750000000005   0.302500000000002   0.010714285714212
   1.097200000000000   2.362500000000001   1.425000000000006   0.309999999999950                   0
   1.569700000000000   2.790000000000003   1.579999999999981                   0                   0
   1.848700000000000   3.263999999999997                   0                   0                   0
   2.501500000000000                   0                   0                   0                   0
  Columns 6 through 8
  -0.023988095238085   0.019913419913396  -0.026405142476580
  -0.000091991342009  -0.017053779553816                   0
  -0.022261904761970                   0                   0
                   0                   0                   0
                   0                   0                   0
                   0                   0                   0
                   0                   0                   0
                   0                   0                   0
Yint =
   0.072647477348189
```

Figura 34: interpolação usando função que implementa o método de Newton para o primeiro ponto fornecido (parte 2).

```
Yint =

   1.705441497414939
```

Figura 35: 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).

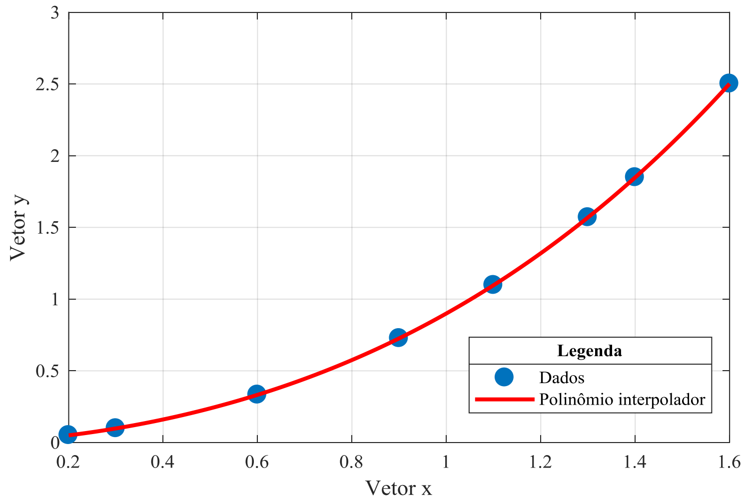

Figura 36: gráfico do polinômio interpolador.