Trabalho Prático de Análise Numérica

Alisson Cordeiro e Gabriel Gaspar

1 Introdução

Este trabalho tem como objetivo apresentar o estudo de Interpolação Polinomial, para a matéria de Análise Númerica, do 3° semestre do curso de Ciência da Computação do Instituto Federal do Paraná - Campus Pinhais. A Interpolação Polinomial é uma técnica utilizada para encontrar um polinômio que passa por um conjunto de pontos dados, permitindo a estimativa de valores intermediários e a análise do comportamento da função.

2 Problemas conhecidos

Alguns problemas conhecidos de interpolação polinomial, são:

- **Fenômeno de Runge**
  O fenômeno de Runge ocorre quando usamos polinômios de grau alto para interpolar funções suaves, especialmente com pontos igualmente espaçados. Isso pode causar oscilações grandes nas extremidades da interpolação, resultando em aproximações ruins, mesmo com mais pontos.
- **Instabilidade numérica**
  Algoritmos como a eliminação de Gauss podem sofrer com instabilidade numérica, especialmente em presença de números muito grandes ou muito pequenos. Pequenos erros de arredondamento podem se propagar e gerar resultados imprecisos.
- **Complexidade**
  O algoritmo apresentado tem complexidade $O(n^3)$, o que significa que o tempo de execução cresce rapidamente com o aumento de n. Isso o torna pouco eficiente para grandes conjuntos de dados.

3 Código-fonte

Abaixo está o código-fonte do programa solicitado. De maneira resumida, usando interpolação polinomial, é inserido pontos de uma função específica. Com esses pontos, o programa irá encontrar a função polinomial que passe pelos pontos inseridos. Este programa

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

void pivoteamentoParcial(double** matriz, int n, int coluna) {
    int linhaMax = coluna;
    for (int i = coluna + 1; i < n; ++i) {
        if (abs(matriz[i][coluna]) > abs(matriz[linhaMax][coluna])) {
            linhaMax = i;
        }
    }
    if (linhaMax != coluna) {
        for (int j = 0; j <= n; ++j) {
            double temp = matriz[coluna][j];
            matriz[coluna][j] = matriz[linhaMax][j];
            matriz[linhaMax][j] = temp;
        }
    }
}

int main() {
    int n;
    cin >> n;
```

```
25
26      double** matriz = new double*[n];
27
28      for (int i = 0; i < n; ++i) {
29          double x, y;
30          cin >> x >> y;
31
32          matriz[i] = new double[n + 1];
33          for (int j = 0; j < n; ++j)
34              matriz[i][j] = pow(x, j);
35
36          matriz[i][n] = y;
37      }
38
39      for (int i = 0; i < n; ++i) {
40          pivoteamentoParcial(matriz, n, i);
41
42          for (int j = i + 1; j < n; ++j) {
43              double fator = matriz[j][i] / matriz[i][i];
44              for (int k = i; k <= n; ++k) {
45                  matriz[j][k] -= fator * matriz[i][k];
46              }
47          }
48      }
49
50      double* resultado = new double[n];
51      for (int i = n - 1; i >= 0; --i) {
52          resultado[i] = matriz[i][n];
53          for (int j = i + 1; j < n; ++j)
54              resultado[i] -= matriz[i][j] * resultado[j];
55          resultado[i] /= matriz[i][i];
56      }
57
58      for (int i = 0; i < n; ++i) {
59          cout<< fixed <<setprecision(16)<< resultado[i] << endl;
60      }
61
62      for (int i = 0; i < n; ++i)
63          delete[] matriz[i];
64      delete[] matriz;
65      delete[] resultado;
66
67      return 0;
68  }
```

4 Utilização do programa

Para compilar e utilizar o programa, foi usado o seguinte código:

```
1       g++ -o interpola interpola.cpp
```

Para uso do programa, foi usado um arquivo de entrada, contendo os valores necessários, sendo o primeiro valor, a quantidade de pontos conhecidos e o restante sendo os pares ordenados representando os pontos. Um exemplo de entrada está disponível abaixo, em Resultados.

```
1       ./interpola < entrada.txt
```

A saída do programa está de acordo com as especificações do trabalho, com os valores sendo exibidos em ordem crescente do grau da variável, sendo o primeiro valor referente a x^0, seguido por x^1, x^2, até x^{n-1}.

5 Resultados

Usando a tabela abaixo como entrada, o número 5 representa a quantidade de pontos e o resto são os pontos em si. Com isso, o programa retorna a função abaixo da tabela.

5

1	2
1.3	4.4
1.7	4.56
1.9	5
2.3	7

$$f(x) = -10{,}561661x^4 + 77{,}710623x^3 - 207{,}739316x^2 + 241{,}080037x - 98{,}489683$$

A figura abaixo mostra o gráfico da função interpoladora, com os cinco pontos previamente dados plotados sobre ele para verificar o ajuste:

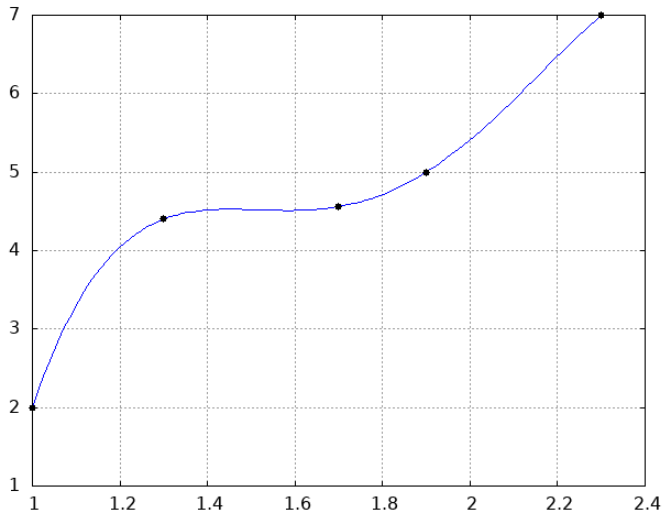


Figura 1: Gráfico da função com os pontos dados previamente

6 Exemplos de teste

Abaixo estão os três exemplos de teste especificados no documento do trabalho prático.

6.1 Exemplo 1

6.1.1 Entrada do programa

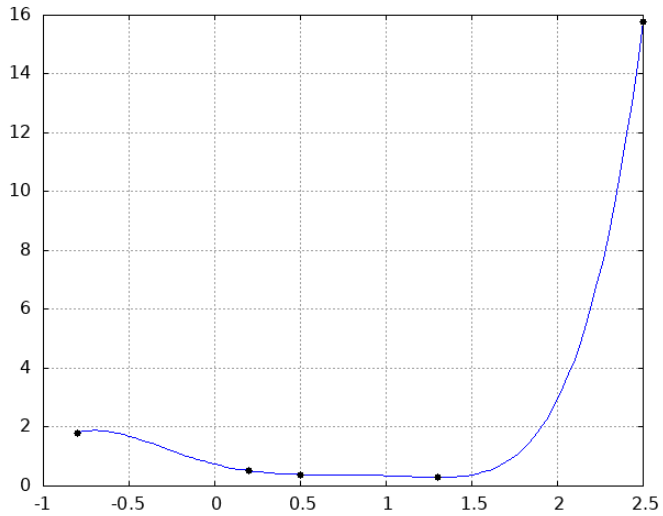
6

-1	1
-0.8	1.808704
0.2	0.514624
0.5	0.390625
1.3	0.297649
2.5	15.765625

6.1.2 Polinômio interpolador encontrado

$$f(x) = 0{,}6999999999999992x^5 - 1{,}7099999999999984x^4 + 0{,}4370000000000012x^3 + 1{,}6460000000000008x^2 - 1{,}4610000000000005x + 0{,}7399999999999977$$

6.1.3 Gráfico dos pontos dados e da função interpoladora



6.2 Exemplo 2

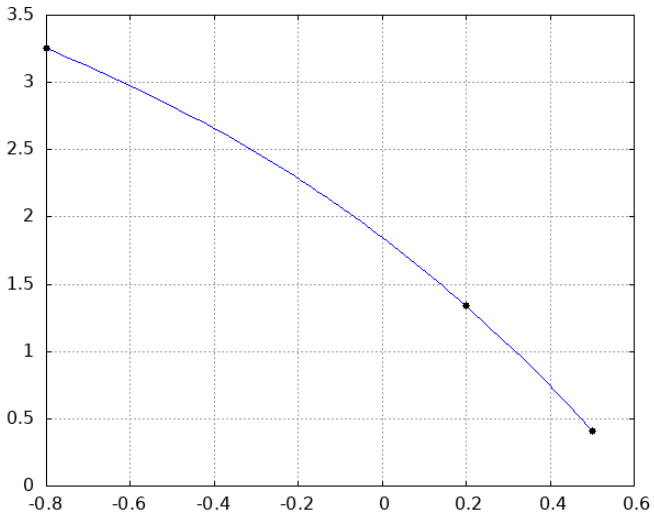
6.2.1 Entrada do programa

	4
-1	3.5
-0.8	3.25312
0.2	1.33712
0.5	0.40625

6.2.2 Polinômio interpolador encontrado

$f(x) = -0{,}22999999999999993x^3 - 0{,}9359999999999992x^2 - 2{,}3580000000000001x + 1{,}8479999999999996$

6.2.3 Gráfico dos pontos dados e da função interpoladora



6.3 Exemplo 3

6.3.1 Entrada do programa

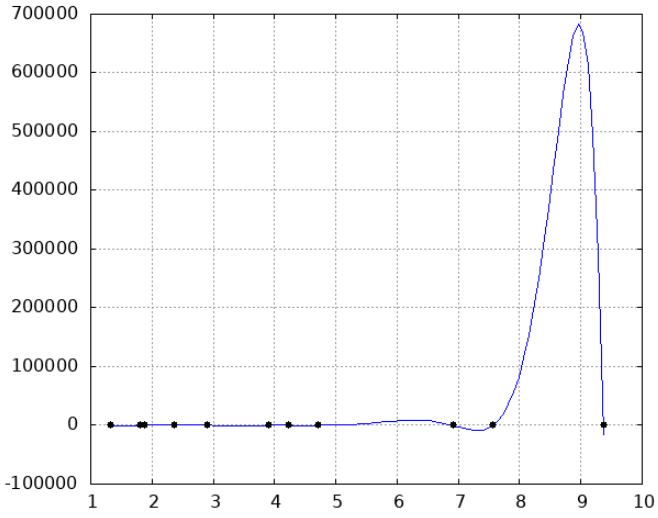
12

2.096564322979868	9.745443487783383
6.922977157751151	1.242846887386261
4.220441410259928	3.789449024006479
1.802878236405755	0.8368959365195927
1.324377887241789	5.936317566366855
1.877280316779064	9.774407080764629
7.557760165820275	6.712616033725468
4.720106905544402	1.230350867632133
2.356347202074704	8.896870895536257
3.905720613358983	3.180695896049662
2.893813967551298	9.832444876731104
9.364300979225938	0.7215618713442351

6.3.2 Polinômio interpolador encontrado

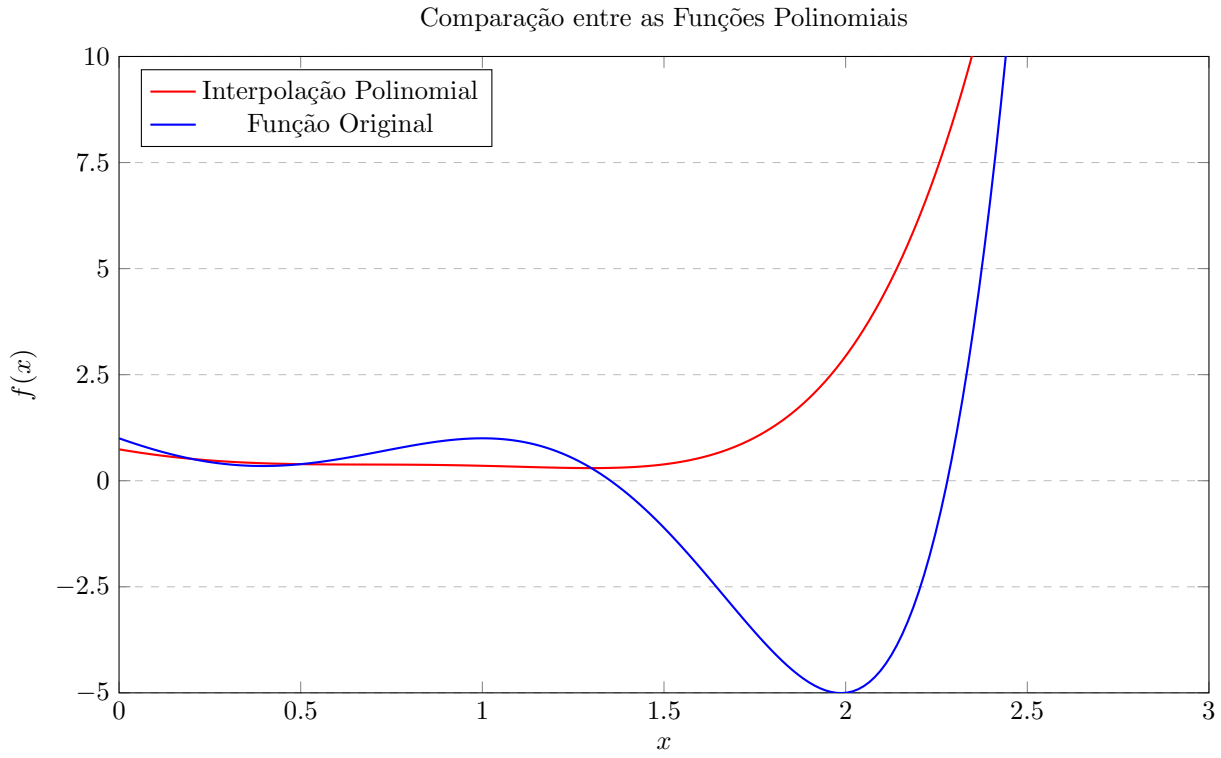
$$\begin{aligned} f(x) = {} & 553846{,}2166198785416782 - 2003336{,}3049126006662846x + 3181893{,}9942112155258656x^2 \\ & - 2929693{,}9608184378594160x^3 + 1737428{,}8216584778856486x^4 - 696749{,}3978601371636614x^5 \\ & + 192772{,}1912586242251564x^6 - 36792{,}6141283220349578x^7 + 4747{,}0648406829632222x^8 \\ & - 394{,}3311612651109499x^9 + 18{,}9837925153810367x^{10} - 0{,}4013836068413766x^{11} \end{aligned}$$

6.3.3 Gráfico dos pontos dados e da função interpoladora

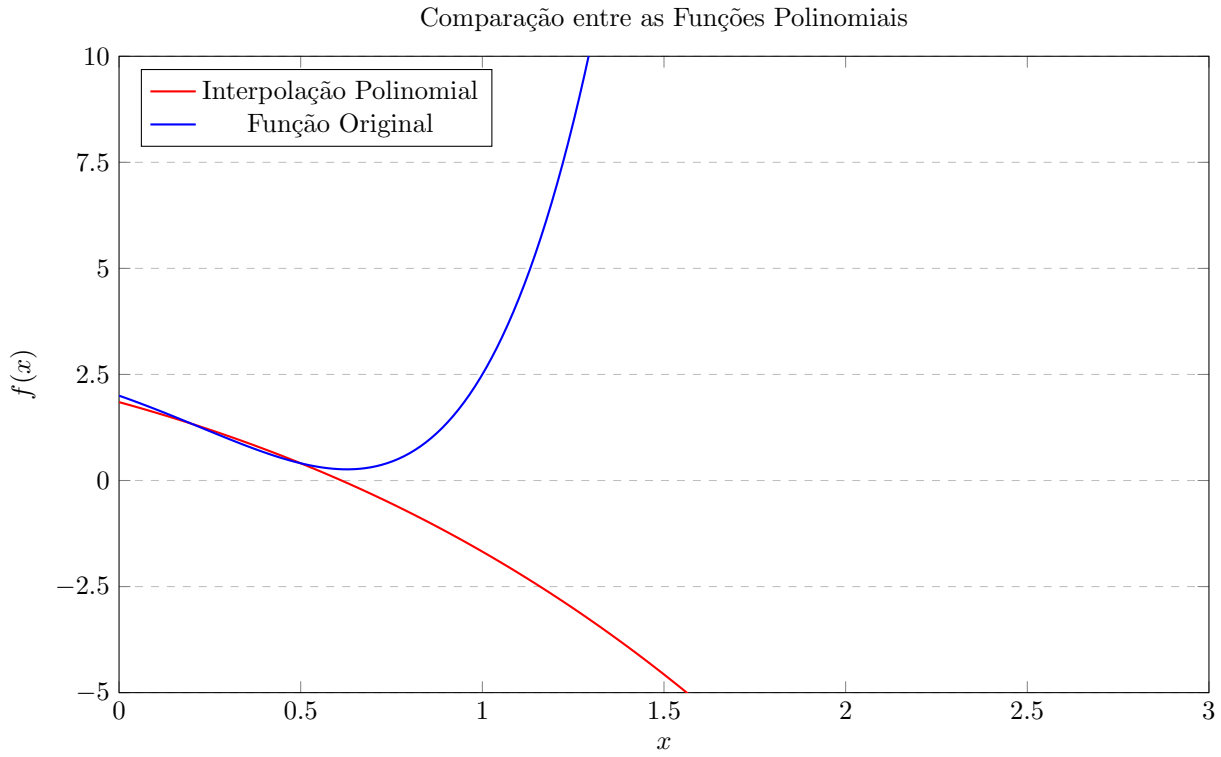


7 Comparação entre as funções dadas e as funções encontradas

7.1 Função do 6.1



7.2 Função do 6.2



8 Código-fonte do gerador de gráficos

Abaixo está o código-fonte do programa usado para gerar os gráficos. Basicamente, ajustamos o programa original para automaticamente exportar um gráfico com os pontos de entrada e o gráfico da função interpoladora encontrada.

```
#include <cmath>
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <string>
using namespace std;

void pivoteamentoParcial(double **matriz, int n, int coluna) {
  int linhaMax = coluna;
  for (int i = coluna + 1; i < n; ++i) {
    if (abs(matriz[i][coluna]) > abs(matriz[linhaMax][coluna])) {
      linhaMax = i;
    }
  }
  if (linhaMax != coluna) {
    for (int j = 0; j <= n;
         ++j) { // precisa ir ate n, incluindo termo independente
      swap(matriz[coluna][j], matriz[linhaMax][j]);
    }
  }
}

int main() {
  int n;
  cin >> n;

  double **matriz = new double *[n];

  for (int i = 0; i < n; ++i) {
    double x, y;
    cin >> x >> y;

    matriz[i] = new double[n + 1];
    for (int j = 0; j < n; ++j)
      matriz[i][j] = pow(x, j);

    matriz[i][n] = y; // termo independente
  }

  // Eliminacao de Gauss com pivoteamento parcial
  for (int i = 0; i < n; ++i) {
    pivoteamentoParcial(matriz, n, i);

    for (int j = i + 1; j < n; ++j) {
      double fator = matriz[j][i] / matriz[i][i];
      for (int k = i; k <= n; ++k) {
        matriz[j][k] -= fator * matriz[i][k];
      }
    }
  }

  // Substituicao regressiva
  double *resultado = new double[n];
  for (int i = n - 1; i >= 0; --i) {
    resultado[i] = matriz[i][n];
    for (int j = i + 1; j < n; ++j)
      resultado[i] -= matriz[i][j] * resultado[j];
    resultado[i] /= matriz[i][i];
  }

  string saida = "";

  for (int i = 0; i < n; ++i) {
    cout<<resultado[i]<<endl;
    if (resultado[i] < 0)
      saida = saida + " " +to_string(resultado[i]) + "*x**" + to_string(i);
    else
      saida = saida + " + " + to_string(resultado[i]) + "*x**" + to_string(i);
  }
```

```
70
71    saida = "gnuplot -e \"set terminal png; set grid; unset key; set output "
72            "'saida.png'; "
73            "plot " +
74            saida + " with lines dashtype 2 lc 'blue', 'entrada' skip 2 with points pt 7 lc 'black' \"";
75
76    system(saida.c_str());
77    system("xdg-open saida.png");
78    // Liberacao de memoria
79    for (int i = 0; i < n; ++i)
80      delete[] matriz[i];
81    delete[] matriz;
82    delete[] resultado;
83
84    return 0;
85  }
```