

Universidade Federal do Paraná

Setor de Ciências Exatas

Departamento de Informática

**ALGORITMOS E ESTRUTURAS DE DADOS II**  
**(CI056)**

Primeiro Trabalho Prático

Hamer Iboshi

Professor - David Menotti

Curitiba

16 de outubro de 2015

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Considerações iniciais . . . . .	1
1.2	Especificação do problema . . . . .	1
<b>2</b>	<b>Algoritmo e estruturas de dados</b>	<b>2</b>
2.1	Algoritmos . . . . .	2
2.1.1	Criar Polinômio . . . . .	2
2.1.2	Obter Orde do Polinômio . . . . .	2
2.1.3	Obter Valor do Polinômio . . . . .	3
2.1.4	Adição de Polinômios . . . . .	3
2.1.5	Subtração de Polinômios . . . . .	4
2.1.6	Produto de Polinômios . . . . .	5
2.1.7	Método de Horner . . . . .	5
2.1.8	Cálculo da Derivada . . . . .	9
2.1.9	Cálculo dos Extremos . . . . .	10
2.1.10	Cálculo das raízes de polinômios com grau $n \geq 3$ . . . . .	13
2.2	Estruturas de dados . . . . .	13
2.2.1	TPolinômio . . . . .	13
2.2.2	TExtremo . . . . .	14
<b>3</b>	<b>Análise de complexidade dos algoritmos</b>	<b>15</b>
3.1	PObterValor . . . . .	15
3.2	PAdicao . . . . .	16
3.3	PSubtracao . . . . .	16
3.4	PProduto . . . . .	16
3.5	PDivisao . . . . .	17
3.6	PObterExtremos . . . . .	17
3.7	Complexidade do Completa do Programa . . . . .	17
<b>4</b>	<b>Testes</b>	<b>18</b>
4.1	PCriar . . . . .	20
4.2	PObterValor . . . . .	20
4.3	PObterOrdem . . . . .	20
4.4	PAdicao . . . . .	21
4.5	PSubtracao . . . . .	21
4.6	PProduto . . . . .	21
4.7	PDivisao . . . . .	21
4.8	ObterExtremos . . . . .	23
<b>5</b>	<b>Conclusão</b>	<b>24</b>

## Lista de Figuras

1	Método de Horner 1.1 . . . . .	6
2	Método de Horner 1.2 . . . . .	6
3	Método de Horner 1.3 . . . . .	7
4	Método de Horner 1.4 . . . . .	7

5	Método de Horner 1.5 . . . . .	7
6	Método de Horner 1.6 . . . . .	8
7	Método para calcular raízes de polinômios com Grau $n \geq 3$ . . . . .	14
8	TAD dos polinômios . . . . .	14
9	TAD dos extremos . . . . .	15
10	Gráfico do polinômio $x^3$ . . . . .	16
11	Gráfico do Teste 2 de Obter Extremos . . . . .	23
12	Gráfico do Teste 3 de Obter Extremos . . . . .	24

## Lista de Programas

1	PCriar . . . . .	2
2	PObterOrdem . . . . .	3
3	PObterValor . . . . .	3
4	PAdição . . . . .	3
5	PSubtração . . . . .	4
6	PProduto . . . . .	5
7	Divisão de Polinômios . . . . .	8
8	PDerivar . . . . .	9
9	Calculo de Extremos . . . . .	10
10	TPolinômio . . . . .	13
11	TExtremos . . . . .	15
12	Main . . . . .	18

# 1 Introdução

Este trabalho consiste em implementar um algoritmo que realize algumas operações com polinômios (ex.: soma, subtração, divisão, etc.) utilizando a linguagem C, e aplicando os conhecimentos vistos em aula, como, alocação dinâmica, tipos abstratos de dados, calculo ordem de complexidade, entre outras coisas.

## 1.1 Considerações iniciais

- Ambiente de desenvolvimento do código fonte: vim & gcc.
- Linguagem utilizada: Linguagem C.
- Ambiente de desenvolvimento da documentação: Tex Maker - Editor de L<sup>A</sup>T<sub>E</sub>X.
- Ferramenta para criação e verificação de testes e gráficos: WolframAlpha.

## 1.2 Especificação do problema

Você deverá implementar um TAD Polinômio para representar polinômios de ordem n qualquer ( $n > 0$ ), i.e.,

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_2 x^2 + a_1 x^1 + a_0 \quad (1)$$

em que  $a$ ,  $a_{n-1}$ ,  $a_{n-2}$ , ...,  $a_2$ ,  $a_1$  e  $a_0$  são os  $n + 1$  coeficientes do polinômio de ordem (ou grau)  $n$ . Considere que a ordem do polinômio será determinada em tempo de execução (alocação dinâmica). Este TAD deverá armazenar os coeficientes do polinômio e a sua ordem  $n$  e implementar procedimentos (ou funções quando for o caso) para:

1. **PCriar** - Criar (e ler) um polinômio.
2. **PObterValor(P,x)** - Obter o valor de  $P(x)$  dado um valor qualquer de  $x$ .
3. **PObterOrdem(P)** - Obter a ordem (grau) de  $P(x)$ , i.e.,  $n$ .
4. **PAdicao(P,Q)** - Realizar a soma/adição de dois polinômios  $P$  e  $Q$  quaisquer. Um novo polinômio deve ser criado e retornado neste caso.
5. **PSubtracao(P,Q)** - Realizar a subtração de dois polinômios  $P$  e  $Q$  quaisquer. Um novo polinômio deve ser criado e retornado neste caso.
6. **PProduto(P,Q)** - Realizar o produto de dois polinômios  $P$  e  $Q$  quaisquer. Um novo polinômio deve ser criado e retornado neste caso.
7. **PDivisao(P,Q)** - Realizar a divisão do polinômio  $P$  por  $Q$  (quando  $Q$  não for nulo, i.e.,  $n > 0$ ). Devem ser criados e retornados dois polinômios, um para o resto e outro para o quociente da divisão. A partir desta função, devem ser criadas duas funções, uma para calcular o resto e outra para calcular o quociente. A otimização de código destas três funções será objeto de avaliação em **ponto extra**.

O TAD deve implementar ainda uma função que calcule os extremos (máximos e/ou mínimos) locais de  $P(x)$  para polinômios de ordem 3 ou inferior (PObterExtremos). Caso a função calcule também os extremos para polinômios de ordem 4, um **ponto extra** (1% do total da disciplina) será atribuído. Caso essa funcionalidade seja implementada e a função também calcule os extremos para polinômios de ordem 5, um outro **ponto extra** será atribuído. Sugere-se desenvolver uma função PDerivar(P). Os valores extremos deverão ser armazenados em um outro TAD TExtremo que se comporta como um vetor. Além dos valores e da quantidade de extremos, o TAD deve ter um campo especial para indicar se cada extremo é de máximo ou de mínimo, ou ainda se é um ponto de inflexão. Dica: aplicação da derivada segunda. Implemente os TADs solicitados em arquivos separados do programa principal (sugestão: TPolinomio.c, TPolinomio.h, TExtremo.c, TExtremo.h e main.c). Apresente sucintamente na documentação a forma usada para compilação de todo o "projeto". Se necessário, você pode criar outras funções auxiliares nos TADs. Uma vez criado os TADs, você deverá também implementar um programa principal (main) para manipular estes TADS de forma que seja possível avaliar\testar a consistência das implementações.

## 2 Algoritmo e estruturas de dados

### 2.1 Algoritmos

Alguns algoritmos utilizados para implementar este programa não são muito triviais, e por isso serão detalhados nesta seção, e os mais triviais serão brevemente abordados na seção Análise de complexidade dos algoritmos.

#### 2.1.1 Criar Polinômio

Função que cria e aloca (com a função PAloca que aloca dinamicamente um vetor) o polinômio.

```

void PCriar (TPolinomio *P){ //Leitura do menor grau para o maior grau
    (0,1,2,...)
    int i;
    scanf( "%d",&P->quant );
    PAloca(&P->poli ,P->quant );
5    for ( i=0;i<P->quant ; i++){
        scanf( "%lf",&P->poli [ i ] );
    }
}

10 void PAloca( double **P, int n){
    (*P) = (double*) malloc( n*sizeof(double) );
}

```

Programa 1: PCriar

#### 2.1.2 Obter Orde do Polinômio

É simples notar que a ordem é calculada subtraindo 1 da quantidade de elementos do polinômio.

```

int PObterOrdem(TPolinomio *P){
    return P->quant -1;
}

```

Programa 2: PObterOrdem

### 2.1.3 Obter Valor do Polinômio

```

double potencia(double num, int pot){
    int i;
    double n=1;
    for (i=1;i<pot;i++){
5      n*=num;
    }
    return n;
}

10 double PObterValor(TPolinomio *P, double x){
    int i=0,n = P->quant;
    double valor = 0;
    valor+=P->poli[i];
    for (i=1;i<n;i++){
15      valor+=potencia(P->poli[i], i+1)*x;
    }
    return valor;
}

```

Programa 3: PObterValor

### 2.1.4 Adição de Polinômios

A soma é operação simples de realizar e sua complexidade e de outros algoritmos triviais será detalhada na seção Análise de complexidade dos algoritmos, veja abaixo o código da soma de polinômios 4, que basicamente soma os polinômios com índices iguais.

```

void PAdicao(TPolinomio *P1, TPolinomio *P2, TPolinomio *Sum){
    int i,l,l1 = P1->quant,l2 = P2->quant;
    if (P1->quant == P2->quant){
5      l = P1->quant;
      Sum->quant = l;
      PAloca(&Sum->poli,Sum->quant);
      for (i = 0; i<l;i++){
        Sum->poli[i] = P1->poli[i] + P2->poli[i];
      }
10    } else{
      if (P1->quant > P2->quant){
        l = l1;
        Sum->quant = l;
        PAloca(&Sum->poli,Sum->quant);
15      for (i = 0; i<l;i++){
        if (i<l2){
          Sum->poli[i] = P1->poli[i] + P2->poli[i];
        } else{
          Sum->poli[i] = P1->poli[i];
        }
      }
    }
}

```

```

20     }
    }
    }else{
        l = l2;
        Sum->quant = l;
25     PAloca(&Sum->poli, Sum->quant);
        for (i = 0; i < l; i++){
            if (i < l1){
                Sum->poli[i] = P1->poli[i] + P2->poli[i];
            }else{
30                Sum->poli[i] = P2->poli[i];
            }
        }
    }
}
35 }

```

Programa 4: PAdição

### 2.1.5 Subtração de Polinômios

A subtração é também uma operação simples de realizar que também subtrai os polinômios com índices iguais (ou seja, mesmo grau) do Polinômio 1, pelo Polinômio 2 e seu código 5 pode ser visualizado abaixo:

```

void PSubtracao(TPolinomio *P1, TPolinomio *P2, TPolinomio *Sub){
int i, l, l1 = P1->quant, l2 = P2->quant; //l1 eh limite(o tamanho maximo
da do polinomio) de P1, l2 eh o limite para P2
    if (P1->quant == P2->quant){
        l = P1->quant;
5        Sub->quant = l;
        PAloca(&Sub->poli, Sub->quant);
        for (i = 0; i < l; i++){
            Sub->poli[i] = P1->poli[i] - P2->poli[i];
        }
10    }else{
        if (P1->quant > P2->quant){
            l = l1;
            Sub->quant = l;
            PAloca(&Sub->poli, Sub->quant);
15            for (i = 0; i < l; i++){
                if (i < l2){
                    Sub->poli[i] = P1->poli[i] - P2->poli[i];
                }else{
                    Sub->poli[i] = P1->poli[i];
20                }
            }
        }
        else{
            l = l2;
            Sub->quant = l;
25            PAloca(&Sub->poli, Sub->quant);
            for (i = 0; i < l; i++){
                if (i < l1){
                    Sub->poli[i] = P1->poli[i] - P2->poli[i];
                }else{

```

```

30         Sub->poli[i] = -(P2->poli[i]); //Como o polinomio dois eh o
        que vai subtrair e nesse grau nao existe um polnomio P1,
        nega-se o polinomio p2
        }
    }
}
35

```

Programa 5: PSubtração

### 2.1.6 Produto de Polinômios

```

void PProduto(TPolinomio *P1,TPolinomio *P2,TPolinomio *Prod){
    int i,j,tam,l1 = P1->quant,l2 = P2->quant;
    tam=l1+l2; //tamanho maximo = a soma dos maiores indices dos
    polinomios
    Prod->quant=tam;
    5   Prod->poli = (double*) calloc(Prod->quant,sizeof(double));
    for(i=0;i<l1;i++){
        for(j=0;j<l2;j++){
            Prod->poli[i+j]+=P1->poli[i]*P2->poli[j];
        }
    }
    10 }
}

```

Programa 6: PProduto

### 2.1.7 Método de Horner

Para solucionar o problema da divisão de polinômios[3] o método escolhido foi o método de Horner[2] que é um excelente método para está função, embora também possua outras aplicações. Logo abaixo há um exemplo que demonstra o método dele.

$$P(x) = 12x^5 + 6x^4 + 24x + 36 \quad (2)$$

$$Q(x) = 6x + 3 \quad (3)$$

No exemplo abaixo Figura 1 é possível ver que o método de Horner pega o elemento de maior grau do dividendo e coloca-o na diagonal esquerda superior (destacado na figura acima) e os seguintes elementos do dividendo vão nas linhas abaixo e na mesma coluna (coluna 1) com os sinais invertidos (o 3 do polinômio divisor é invertido e fica -3) e com a mesma ordem decrescente de grau. Na primeira linha ao lado do polinômio de maior grau do divisor vão os elementos do dividendo, que são colocados também em ordem decrescente, ou seja, do maior grau para o menor grau. A última divisória de coluna é utilizada pra gerar o resto no final da execução do método, e ela é colocada contando da direita pra esquerda o número de coeficientes do divisor que ficaram abaixo da primeira linha (que neste caso foi somente o '-3').

Figura 1: Método de Horner 1.1  
P(x) Dividendo

Divisor	6	12	6	0	0	24	36
	-3						
		Quociente					Resto

Figura 2: Método de Horner 1.2

6	12	6	0	0	24	36
-3						
	2					

Na Figura 2 ocorre a primeira operação que por ser a primeira ela simplesmente pega o elemento na diagonal superior esquerda (elemento de maior grau do divisor) e divide pelo primeiro elemento da linha dos dividendos (elemento de maior grau dos dividendos):

$$12 / 6 = 2$$

Na Figura 3 o coeficiente já calculado é utilizado para encontrar o próximo, multiplicando-o pelo primeiro divisor, e colocando na primeira linha abaixo do seu respectivo grau do dividendo (caso existisse mais divisores, os próximos seriam multiplicados por esse mesmo coeficiente e seriam colocados na mesma linha ao lado do já calculado de acordo com o respectivo grau do dividendo), e depois soma todos os elementos da coluna e divide pelo divisor de maior grau (o elemento 6):

$$2 * (-3) = -6 \text{ [Segunda seta (vertical) da Figura 3.]}$$

$$6 + (-6) = 0 / 6 = 0 \text{ [Primeira seta (horizontal) da Figura 3.]}$$

É trivial que as mesmas operações serão realizadas para obtenção dos seguintes coeficientes como é possível visualizar nas seguintes Figuras.

Figura 3: Método de Horner 1.3

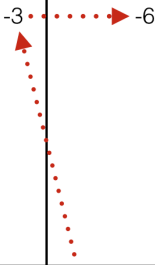
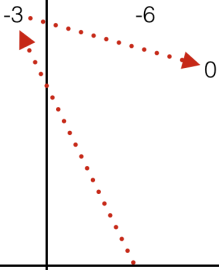
6	12	6	0	0	24	36
-3						
	2	0				

Figura 4: Método de Horner 1.4

6	12	6	0	0	24	36
-3						
	2	0	0			

$$0 * (-3) = 0 \text{ [Figura 4.]}$$

$$0 / 6 = 0 \text{ [Figura 4.]}$$

Figura 5: Método de Horner 1.5

6	12	6	0	0	24	36
-3	-6	0	0	24		
		0	0	4		
	2	0	0	0	4	

$$0 * (-3) = 0 \text{ [Figura 5.]}$$

$$(0 + 24) / 6 = 4 \text{ [Figura 5.]}$$

Figura 6: Método de Horner 1.6

6	12	6	0	0	24	36
-3	-6	0	0	24	-12	
	2	0	0	0	4	24

O calculo do resto demonstrado na Figura 6 só se diferencia do calculo do coeficiente por conta de não ser necessário a divisão por 6, só a multiplicação do coeficiente anterior pelos divisores válidos (neste caso '-3') e a soma das colunas.

$$4 * (-3) = -12 \text{ [Figura 6.]}$$

$$(-12) + 36 = 24 \text{ [Figura 6]}$$

Então após aplicar o método é possível ver o coeficiente D(x) resultante em ordem decrescente de grau (do maior grau para o menor, da esquerda para direita). E o resto R(x) que caso tivesse mais elementos também seria representado em ordem decrescente.

$$D(x) = 2x^4 + 4 \quad (4)$$

$$R(x) = 24 \quad (5)$$

```

//METODO DE HORNER
void PDivisao(TPolinomio P, TPolinomio Q, TPolinomio *D, TPolinomio *R)
{ //P e Q nao foi passado por referencia
  int i, j, lP = P.ordem, lQ = Q.ordem, lD, div, Pi, Qi, Di, Ri, Ai, parametro;
  //Pi, Qi, Di, Ri, Ai sao usados para controlar os indices durante a
  divisao;
  double *aux; //Div eh o Parametro principal para realizar a
  divisao pelo metodo de Horner
  aux = (double*) calloc(lP, sizeof(double));
  parametro = lP - (lQ - 1); //Parametro para o numero de vezes que o
  metodo sera implementado

```

```

D->ordem = parametro;
R->ordem=lQ;
D->poli = (double*) calloc (D->ordem, sizeof(double));
10 R->poli = (double*) calloc (R->ordem, sizeof(double));
lD=D->ordem-1;
Di=lD-1;
lQ--;
Ri=lQ;
15 Pi = lP-2; //indice auxiliar fixo para manipular a divisao e
            utilizar o aux, diferente do Ai ele so diminui no final do for
            externo
div = Q.poli[lQ];
//Atribuir P para AUX
for (i=0;i<lP;i++)
    aux[i]=P.poli[i];
20 D->poli[Di+1]=P.poli[Pi+1]/div;
for (i=0;i<parametro;i++){
    Ai=Pi;
    Qi=lQ-1;
    for (j=lQ-1 ;j>=0;j--){
25     aux[Ai]+=D->poli[Di+1]*Q.poli[Qi]*-1; //Pegando o Quociente
        da da divisao e invertendo os cocientes atraves do
        metodo de horner
    Ai--;
    if (Qi>0)
        Qi--;
    }
30 if (Di>=0)
    D->poli[Di]+=aux[Pi]/div;
    Pi--;
    Di--;
}
35 //Calcula resto
for (i=0;i<Ri; i++) {
    R->poli[i]=aux[i];
}
free(aux);
40 }

```

Programa 7: Divisão de Polinômios

### 2.1.8 Cálculo da Derivada

```

TPolinomio PDerivar(TPolinomio *P){
    int i,tam;
    TPolinomio aux;
    aux.quant = P->quant-1;
5    tam = aux.quant;
    aux.poli = (double*) calloc (aux.quant, sizeof(double));
    for (i=0;i<tam;i++){
        aux.poli[i]=(i+1)*P->poli[i+1];
    }
10 return aux;
}

```

Programa 8: PDerivar

## 2.1.9 Cálculo dos Extremos

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

5 #define notE -1 //Quando nao eh nenhuma das opcoes abaixo
#define infE 0
#define maxE 1
#define minE 2

10 TPolinomio PDerivar(TPolinomio *P){
    int i,tam;
    TPolinomio aux;
    aux.quant = P->quant-1;
    tam = aux.quant;
15 aux.poli = (double*) calloc(aux.quant,sizeof(double));
    for(i=0;i<tam;i++){
        aux.poli[i]=(i+1)*P->poli[i+1];
    }
    return aux;
20 }

void ExibeExtremos(TExtremo *Ext, int aloca, TPolinomio *Deri1,
    TPolinomio *Deri2){
    int i;
    if(aloca){
25     if(aloca == 1){
        printf("O Polinomio nao Possui Maximos e Minimos Locais, pois eh
            uma reta\n");
    }else if(aloca == 2){
        ExibeE(Ext);
        free(Ext->x);
        free(Ext->y);
30     free(Ext->tipo);
        free(Deri1->poli);
        free(Deri2->poli);
    }else if(aloca == 3){
35     ExibeE(Ext);
        free(Ext->x);
        free(Ext->y);
        free(Ext->tipo);
        free(Deri1->poli);
40     }else if(aloca == -1){
        printf("O Polinomio nao Possui Raizes Reais\n");
    }
    }else{
45     printf("O Polinomio nao Possui Maximos e Minimos Locais\n");
    }
}

void ExibeE(TExtremo *Ex){
    int i;
50     if (Ex->quant == 1){
        if(*Ex->tipo == 0)
            printf("P(X,Y) = (%.2lf,%.2lf) Ponto de Inflexão\n",Ex->x[0],Ex->y[0]);
        else if(*Ex->tipo == 1)
```

```

    printf( "P(X,Y) = (%.2lf,%.2lf)  Maximo Local\n",*Ex->x,*Ex->y);
55  else
    printf( "P(X,Y) = (%.2lf,%.2lf)  Minimo Local\n",*Ex->x,*Ex->y);
} else{
    for(i=0;i<Ex->quant;i++){
        if(Ex->tipo[i] == 0)
60         printf( "P(X,Y) = (%.2lf,%.2lf)  Ponto de Inflexão\n",Ex->x[i],
            Ex->y[i]);
        else if(Ex->tipo[i] == 1)
            printf( "P(X,Y) = (%.2lf,%.2lf)  Maximo Local\n",Ex->x[i],Ex->y[i]);
        else if(Ex->tipo[i] == 2)
            printf( "P(X,Y) = (%.2lf,%.2lf)  Minimo Local\n",Ex->x[i],Ex->y[i]);
65     }
}
}
//Sera utilizado para encontrar raizes de GRAU > 3
void EBriotRuffunni(){
70
}

void ObterExtremos(TPolinomio *P){
75  int i, teste = 1, aloca=0, pass; //aloca usado para verificar se um
    malloc foi usado, para nao dar free quando nao utilizar nada
    double delta;
    TPolinomio Deri1, Deri2;
    TExtremo Ext;
    switch(P->quant-1){//Grau/Ordem do Polinomio
80     case 0:
        aloca = 0;
        break;
        case 1:
            aloca = 1;
85     break;
        case 2:
            aloca = 3;
            Deri1 = PDerivar(P);
            Ext.tipo = (int*) malloc(sizeof(int));
90     Ext.x = (double*) calloc(1, sizeof(double));
            Ext.y = (double*) calloc(1, sizeof(double));
            *Ext.x = ((-1*Deri1.poli[0])/Deri1.poli[1]);
            *Ext.y = PObterValor(P,*Ext.x);
            //printf("%.2lf %.2lf +1 %.2lf %.2lf\n",*Ext.x,*Ext.y,
                PObterValor(P,(*Ext.x)+1),PObterValor(P,(*Ext.x)-1));
95     if ((PObterValor(P,(*Ext.x)-1) > *Ext.y)&&(PObterValor(P,(*Ext.x)
        +1)>*Ext.y)){
        *Ext.tipo = minE;
        Ext.quant = 1;
    } else if ((PObterValor(P,(*Ext.x)-1)<*Ext.y)&&(PObterValor(P,(*Ext
        .x)+1)<*Ext.y)){
        *Ext.tipo = maxE;
100    Ext.quant = 1;
    }
    break;
    case 3:
        Deri1 = PDerivar(P);

```

```

105 Deri2 = PDerivar(&Deri1);
    aloca = 2;
    pass = 1;
    delta = (Deri1.poli[1]*Deri1.poli[1]) - 4*Deri1.poli[2]*Deri1.poli
        [0];
    if(delta < 0){
110     aloca = -1;
    } else if((int)delta == 0){
        Ext.tipo = (int*) malloc(2*sizeof(int));
        Ext.x = (double*) calloc(2, sizeof(double));
        Ext.y = (double*) calloc(2, sizeof(double));
115     Ext.x[0] = (-1*Deri1.poli[1]) / (Deri1.poli[2]*2);
        Ext.y[0] = PObterValor(P, *Ext.x);
        if ((PObterValor(P, (*Ext.x)-1) > *Ext.y) && (PObterValor(P, (*Ext.x)
            +1) > *Ext.y)) {
            Ext.tipo[0] = minE;
        } else if ((PObterValor(P, (*Ext.x)-1) < *Ext.y) && (PObterValor(P, (*
            Ext.x)+1) < *Ext.y)) {
120         Ext.tipo[0] = maxE;
        } else {
            Ext.x[0] = (-1*Deri2.poli[0]) / Deri2.poli[1];
            Ext.y[0] = PObterValor(P, Ext.x[1]);
            Ext.tipo[0] = infE;
125         Ext.quant = 1;
            pass=0;
        }
        aloca = 2;
        if (pass){
130         Ext.x[1] = (-1*Deri2.poli[0]) / Deri2.poli[1];
            Ext.y[1] = PObterValor(P, Ext.x[1]);
            Ext.tipo[1] = infE;
            Ext.quant = 2;
        }
135     } else {
        Ext.tipo = (int*) malloc(3*sizeof(int));
        Ext.x = (double*) calloc(3, sizeof(double));
        Ext.y = (double*) calloc(3, sizeof(double));
        //Para Primeira Raiz de X
140     Ext.x[0] = (-1*Deri1.poli[1] + sqrt(delta)) / (Deri1.poli[2]*2);
        Ext.y[0] = PObterValor(P, Ext.x[0]);
        if ((PObterValor(P, Ext.x[0]-1) > Ext.y[0]) && (PObterValor(P, Ext.x
            [0]+1) > Ext.y[0])) {
            Ext.tipo[0] = minE;
        } else if ((PObterValor(P, Ext.x[0]-1) < Ext.y[0]) && (PObterValor(P,
            Ext.x[0]+1) < Ext.y[0])) {
145         Ext.tipo[0] = maxE;
        } else {
            Ext.tipo[0] = notE;
        }
        //Para segunda Raiz de X
150     Ext.x[1] = (-1*Deri1.poli[1] - sqrt(delta)) / (Deri1.poli[2]*2);
        Ext.y[1] = PObterValor(P, Ext.x[1]);
        if ((PObterValor(P, Ext.x[1]-1) > Ext.y[1]) && (PObterValor(P, Ext.x
            [1]+1) > Ext.y[1])) {
            Ext.tipo[1] = minE;
        } else if ((PObterValor(P, Ext.x[1]-1) < Ext.y[1]) && (PObterValor(P,
            Ext.x[1]+1) < Ext.y[1])) {
155         Ext.tipo[1] = maxE;
    }

```

```

    }else{
        Ext.tipo[1] = notE;
    }
    //Inflexao
160 Ext.x[2] = (-1*Deri2.poli[0])/Deri2.poli[1];
    Ext.y[2] = PObterValor(P,Ext.x[2]);
    Ext.tipo[2] = infE;
    Ext.quant = 3;
}
165 break;
case 4:
    //Para polinomios de grau 4
    aloca=0;
    break;
170 case 5:
    //Para polinomios de grau 5
    aloca=0;
    break;
}
175 ExibeExtremos(&Ext, aloca, &Deri1, &Deri2);
}

```

Programa 9: Calculo de Extremos

### 2.1.10 Cálculo das raízes de polinômios com grau $n \geq 3$

O algoritmo[1] que resolve raízes de polinômios com grau  $n \geq 3$  é utilizado para Obter os Extremos (máximos, mínimos e pontos de inflexão) de polinômios com um grau  $n \geq 4$ . Os passos deste método são encontrar as possíveis raízes, testar-las, reduzir o grau utilizando o método de Briot-ruffini, até chegar em um polinômio de Grau 2 para aplicar Bhaskara e que podem ser visualizadas a seguir:

A implementação desta função é relativamente complexa, pois varia de polinômio para polinômio (sendo de grau  $n \geq 3$ ) e por conta disso para casos médios de números não inteiros e os piores casos (por exemplo, os números grandes e não inteiros) a função fica extremamente custosa e ainda os resultados não são mais tão precisos, e o valor retornado é aproximado.

## 2.2 Estruturas de dados

### 2.2.1 TPolinômio

Ao lado direito da Figura 8 está a representação do polinômio como TAD<sup>1</sup>sendo "quant" a quantidade de elementos do polinômio e o ponteiro "poli" representando o polinômio em si que assume o formato de um vetor dinâmico, alocando somente o necessário de acordo com as entradas, e ao lado esquerdo um exemplo de como ele funciona, sendo N o tamanho de um polinômio, os índices vão de 0 até n-1.

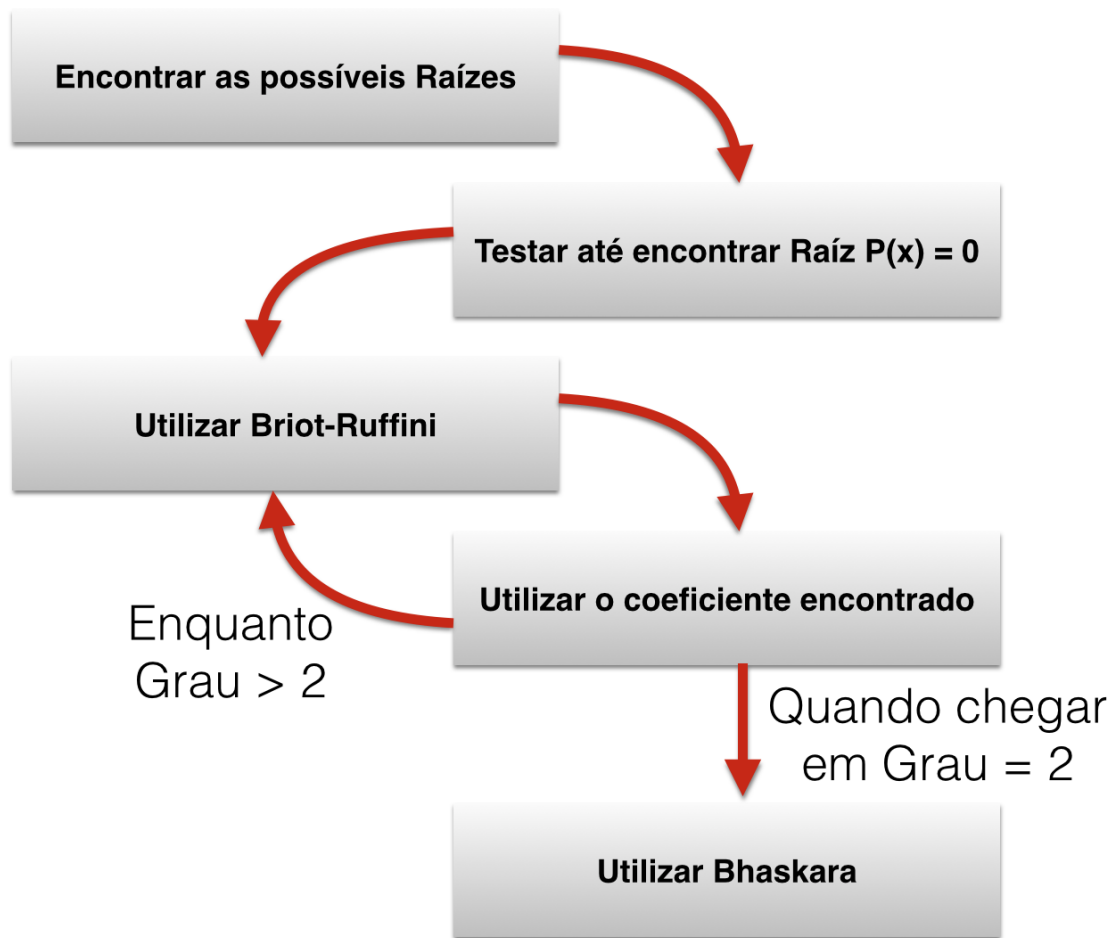
```

typedef struct{
    int quant;
    double *poli;
}TPolinomio;

```

<sup>1</sup>Tipo Abstrato de Dado.

Figura 7: Método para calcular raízes de polinômios com Grau  $n \geq 3$



TPOLINÔMIO	
int	quant
double	*Poli

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \dots + a_2 x^2 + a_1 x^1 + a_0$$

índices	[n-1]	[n-2]	[n-3]	...	1	0
Elementos	a	a	a	...	a	a

Figura 8: TAD dos polinômios

Programa 10: TPolinômio

### 2.2.2 TExtremo

```
typedef struct {
```

Figura 9: TAD dos extremos

TEXTREMO		$P(x)=x^3$	
int	*Tipo	Tipo	Inflexão
double	*X	*x[0]	0
double	*Y	*x[0]	0
int	quant	Quantidade	1

```

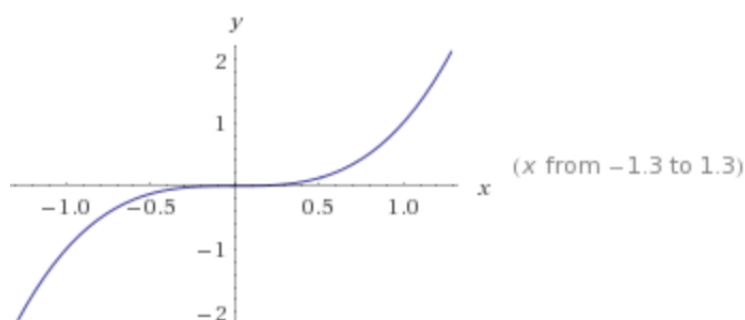
5  int *tipo;
    double *x;
    double *y;
    int quant;
} TExtremo;

```

Programa 11: TExtremos

Na Figura 9 à direita está representado o TAD que armazena os extremos de determinado polinômio, sendo o "quant" o que armazena a quantidade de extremos locais (e/ou pontos de inflexão), o 'x' e 'y' sendo as coordenadas dos pontos, e o tipo que indica se o ponto encontrado é um máximo, mínimo ou ponto de inflexão (que na implementação do programa em C, foi definido cada tipo sendo uma constante), sendo todos (com exceção da quantidade de elementos) alocados dinamicamente de acordo com a quantidade necessária. Na esquerda da figura é possível visualizar um exemplo da representação de um TExtremo, que nesse caso só possui um ponto de inflexão na coordenada  $P(x,y) = (0,0)$ . Para complementar abaixo na Figura 3 há um gráfico<sup>2</sup> que ilustra melhor o ponto de inflexão mostrado acima.

Figura 10: Gráfico do polinômio  $x^3$



)

<sup>2</sup>Gráfico gerado pelo WolframAlfa.

## 3 Análise de complexidade dos algoritmos

### 3.1 PObterValor

O código do Programa 3 torna simples ao considerar a função em si, e função potência. Sendo a ordem de complexidade da função PObtemValor:

$$PObterValor = \sum_{i=1}^{n-1} \left( \sum_{j=1}^i 1 \right) = \sum_{i=1}^{n-1} i = O\left(\frac{n * (n + 1)}{2}\right) \quad (6)$$

### 3.2 PADicao

A soma possui a complexidade relativa ao seu N, que é o tamanho do polinômio de maior grau:

$$O(n) \quad (7)$$

A complexidade de espaço dessa função é igual a N, que é a quantidade de elementos do polinômio de maior grau, que é alocado de forma dinâmica sendo do tipo TPolinomio.

### 3.3 PSubtracao

A subtração é semelhante a soma e possui a complexidade relativa ao seu N, que é o tamanho do polinômio de maior grau:

$$O(n) \quad (8)$$

A complexidade de espaço dessa função é idêntica a soma, ou seja, igual a N, que é a quantidade de elementos do polinômio de maior grau, que é alocado de forma dinâmica sendo do tipo TPolinomio.

### 3.4 PProduto

É fácil de visualizar no Programa 6 que a multiplicação possui ordem de complexidade:

$$O(n^2) = O(n * n) \quad (9)$$

A complexidade de espaço da função é igual a soma da quantidade de elementos dos dois polinômios passados por referência, que é alocado de forma dinâmica sendo do tipo TPolinomio. Por exemplo, supondo que os polinômios tenham n1 e n2 elementos, o polinômio resultante da operações terá n1+n2 elementos.

### 3.5 PDivisao

Mesmo com dois laços que são utilizados para manipular o auxiliar, passando os elementos do Dividendo para o auxiliar, e depois passando o resto do auxiliar para o Resto em si, há dois laços atrelados que podem ser visualizadas no Programa 7, e por conta disso a ordem de complexidade dessa função é:

$$O(n^2) = O(n * n) \quad (10)$$

A complexidade de espaço varia conforme os dois TPolinomios passados por referência, e aloca outros duas variáveis do tipo TPolinomio representando o Quociente e o Resto dinamicamente, e também um vetor auxiliar que é utilizado para realizar a divisão, que também é alocado dinamicamente e possui o mesmo tamanho do Dividendo (que é recebido por referência). O Quociente possui o tamanho do Dividendo, menos um, e menos o tamanho do Divisor, e o Resto vai ter o tamanho do Divisor menos 1, é possível compreender todo o processo melhor observando com atenção os processos representados a partir da Figura 1 e no Programa 7.

$$\begin{aligned} \text{TamanhoQuociente} &= \text{TamanhoDividendo} - \text{TamanhodoDivisor} - 1 \\ \text{TamanhoResto} &= \text{TamanhodoDivisor} - 1 \\ \text{Tamanhodovetorauxiliar} &= \text{TamanhoDividendo} \end{aligned}$$

### 3.6 POberExtremos

O custo de espaço desta função varia de acordo com o grau do polinômio, mas considerando o pior caso para um  $\text{Grau} \leq 3$  a função aloca 3 vetores dinâmicos de tamanho de acordo com a necessidade de pontos (máximo, mínimo e/ou ponto de inflexão) pertencentes ao tipo TExtremo (sendo tipo, x e y). A ordem de complexidade dessa função é calculada com base na operação mais custosa da função POberExtremos (Programa 9) que é a PDerivar que pode ser visualizada com mais detalhes no Programa 8, pois a que obtém extremos em si só utiliza algumas comparações e cálculos para determinar, enfim a ordem de complexidade pode ser visualizada abaixo:

$$O(2n) = O(n) \quad (11)$$

### 3.7 Complexidade do Completa do Programa

A seguir a o calculo da ordem de complexidade geral do programa (aproximadamente e considerando o :

$$f(2n + n^2 + n^2 + n + n + n^2) = f(4n + 3n^2) = O(n^2) \quad (12)$$

O custo de espaço geral do programa (considerando que os TPolinomio base da maioria operações são os mesmos P e Q, e no caso da função PDivisao o caso mais comum e mais caro para polinômios de  $Grau \leq 3$ ) pode ser visualizado a abaixo e o tamanho dos polinômios das operações variam de acordo com os tamanhos de P e Q, como na descrição de alocação de espaço de cada função que foi detalhada acima como no exemplo do PProduto, PSoma e PSubtracao que alocam mais um TPolinomio para suas respectivas operações e principalmente considerando as alocações de espaço mais relevantes:

$$Quantidade \ TPolinomio = 9 = P + Q + D + R + Sum + Sub + Prod + Deri1 + Deri2$$

$$Quantidade \ TExtremo = 1 = Ext$$

$$Vetor \ Auxiliar = 1 = aux(PDivisao)$$

$$TOTAL \ (APROXIMADO) = 11$$

## 4 Testes

Neste capítulo são abordados as entradas e saídas do programa, e principalmente os testes realizados. Abaixo está o Programa 12 Main utilizado para realizar os testes.

```

#include <stdio.h>
#include <stdlib.h>
#include "TPolinomios.h"
#include "TPolinomios.c"
5  #include "TExtremos.h"
#include "TExtremos.c"

int main() {
    int p;
    double x;
10  TPolinomio P, Q, D, R, Sum, Sub, Prod; //D = Quociente, R = Resto, usado
    na PDivisao
    int escolha;
    scanf("%d",&escolha);
    switch(escolha){
15  case 1: //Obter Valor de P(x) dado um X;
        PCriar(&P);
        scanf("%lf",&x);
        printf("Valor = %.2lf\n",PObterValor(&P,x));
        break;
20  case 2: //Obter Obter Ordem/grau
        PCriar(&P);
        printf("Ordem = %d\n",PObterOrdem(&P));
        break;
    case 3: //Adicao
25  PCriar(&P);
        PCriar(&Q);
        PAdicao(&P,&Q,&Sum);
        PExibe(&Sum);
        PLibera(&P.poli);
30  PLibera(&Q.poli);
        PLibera(&Sum.poli);

```

```

break;
case 4: //Subtracao
    PCriar(&P);
    PCriar(&Q);
    PSubtracao(&P,&Q,&Sub);
    PExibe(&Sub); // Exibe Subtracao
    PLibera(&P.poli);
    PLibera(&Q.poli);
    PLibera(&Sub.poli);
break;
case 5: // Produto
    PCriar(&P);
    PCriar(&Q);
    PProduto(&P,&Q,&Prod);
    PExibe(&Prod);
    PLibera(&P.poli);
    PLibera(&Q.poli);
    PLibera(&Prod.poli);
break;
case 6: //Divisao
    PCriar(&P);
    PCriar(&Q);
    PDivisao(P,Q,&D,&R);
    printf(" \nQuociente \n");
    PExibe(&D);
    printf(" Resto \n");
    PExibe(&R);
    PLibera(&D.poli);
    PLibera(&R.poli);
    PLibera(&P.poli);
    PLibera(&Q.poli);
break;
case 7:
    PCriar(&P);
    ObterExtremos(&P);
break;
default:
    PCriar(&P);
    PCriar(&Q);
    PExibe(&P); //Polinomios digitados
    PExibe(&Q);
    PAdicao(&P,&Q,&Sum);
    PSubtracao(&P,&Q,&Sub);
    PProduto(&P,&Q,&Prod);
    printf("Informe o X: \n");
    scanf("%lf",&x);
    PExibe(&Sum); // Exibe Soma
    PExibe(&Sub); // Exibe Subtracao
    PExibe(&Prod); // Exibe Produto
    printf("Ordem %d \n",PObterOrdem(&P));
    printf("Valor %.2lf \n",PObterValor(&P,x));
    //Divisao
    PDivisao(P,Q,&D,&R);
    printf(" \nQuociente \n");
    PExibe(&D);
    printf(" Resto \n");
    PExibe(&R);
    PLibera(&D.poli);

```

```

90      PLibera(&R.poli);
      //Procurar Extremos
      ObterExtremos(&P);
      PLibera(&P.poli);
      PLibera(&Q.poli);
95      PLibera(&Sum.poli);
      PLibera(&Sub.poli);
      PLibera(&Prod.poli);
      break;
    }
100   return 0;
}

```

Programa 12: Main

## 4.1 PCriar

A função PCriar (Programa 1) é utilizada para ler da entrada e alocar o polinômio que será utilizado nas funções e o formato de entrada que ele recebe é descrito a seguir neste exemplo:

$$1 + 2x + 3x^2 + 4x^3 + 5x^4 \quad (13)$$

```

5
1 2 3 4 5

```

Contendo na primeira linha a quantidade de elementos que serão lidos da entrada, e na segunda linha os próprios elementos em ordem crescente de grau, como demonstrado no exemplo acima.

E a representação de saída do programa de um polinômio pode ser vista a seguir:

```

1 + 2x + 3x^2 + 4x^3 + 5x^4

```

## 4.2 PObterValor

Entrada:

```

5
1 2 3 4 5
10 // valor do X

```

Saída:

```

Valor = 54321.00

```

## 4.3 PObterOrdem

Entrada:

```

7
7 8 6 4 5 2 12

```

Saída:

6

## 4.4 PAdicao

Entrada:

6  
13 4 8 7 4 26  
7  
21 3 1 0 9 18 2

Saída:

34.0 + 7.0 X^1 + 9.0 X^2 + 7.0 X^3 + 13.0 X^4 + 44.0 X^5 + 2.0 X^6

## 4.5 PSubtracao

Entrada:

8  
8 7 6 4 3 1 2 9  
7  
2 9 3 1 2 6 4

Saída:

6.0 -2.0 X^1 + 3.0 X^2 + 3.0 X^3 + 1.0 X^4 -5.0 X^5 -2.0 X^6 + 9.0 X^7

## 4.6 PProduto

Entrada:

5  
4 7 2 3 9  
3  
2 3 4

Saída:

8.0 + 26.0 X^1 + 41.0 X^2 + 40.0 X^3 + 35.0 X^4 + 39.0 X^5 + 36.0 X^6

## 4.7 PDivisao

Teste 1

Entrada:

5  
27 0 0 -65 38  
3  
3 -5 2

Saída:

Quociente  
9.0 + 15.0 X<sup>1</sup> + 19.0 X<sup>2</sup>  
Resto

### Teste 2

Entrada:

6  
12 6 0 0 24 36  
1  
6

Saída:

Quociente  
2.0 + 1.0 X<sup>1</sup> + 4.0 X<sup>4</sup> + 6.0 X<sup>5</sup>  
Resto

### Teste 3

Entrada:

6  
12 6 0 0 24 36  
1  
6

Saída:

Quociente  
2.0 + 1.0 X<sup>1</sup> + 4.0 X<sup>4</sup> + 6.0 X<sup>5</sup>  
Resto

### Teste 4

Entrada:

6  
36 24 0 0 6 12  
2  
3 6

Saída:

Quociente  
4.0 + 2.0 X<sup>4</sup>  
Resto  
24.0

### Teste 5

Entrada:

6  
5 0 3 0 -1 12  
4  
-1 0 2 3

Saída:

Quociente  
2.0 - 3.0 X<sup>1</sup> + 4.0 X<sup>2</sup>  
Resto  
7.0 - 3.0 X<sup>1</sup> + 3.0 X<sup>2</sup>

## 4.8 ObterExtremos

### Teste 1

Entrada:

```
4
0 0 0 3
1
```

Saída:

$P(X,Y) = (0.00, 0.00)$  Ponto de Inflexão

O gráfico da função deste polinômio pode ser visualizado na Figura 10. **Teste 2**

Entrada:

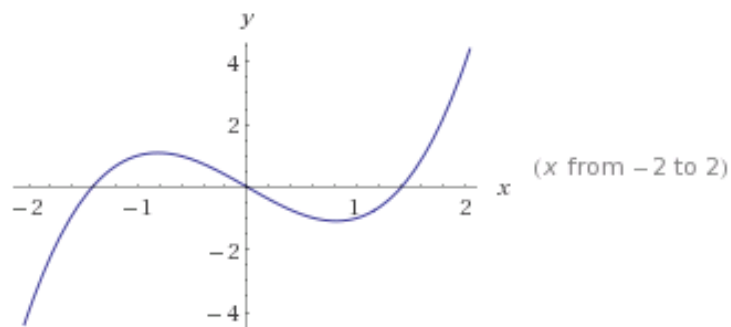
```
4
0 -2 0 1
3
```

Saída:

$P(X,Y) = (0.82, -1.09)$  Minimo Local  
 $P(X,Y) = (-0.82, 1.09)$  Maximo Local  
 $P(X,Y) = (0.00, 0.00)$  Ponto de Inflexão

O gráfico<sup>3</sup> da função do polinômio acima pode ser visto logo abaixo:

Figura 11: Gráfico do Teste 2 de Obter Extremos



### Teste 3 Entrada:

```
4
1 -90 10 9
```

Saída:

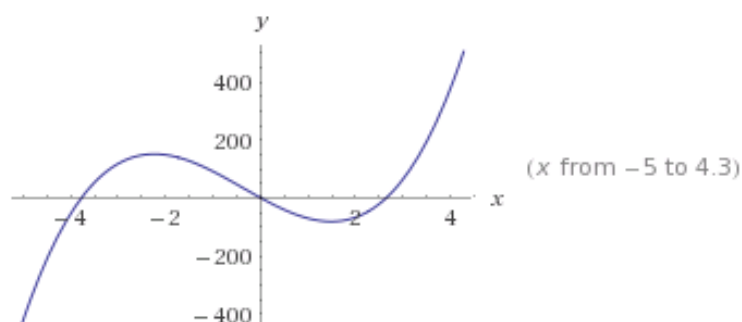
$P(X,Y) = (1.49, -81.13)$  Minimo Local  
 $P(X,Y) = (-2.23, 151.62)$  Maximo Local  
 $P(X,Y) = (-0.37, 35.25)$  Ponto de Inflexão

O gráfico<sup>3</sup> da função do polinômio acima pode ser visto logo abaixo:

---

<sup>3</sup>Gráficos gerado pelo WolframAlfa.

Figura 12: Gráfico do Teste 3 de Obter Extremos



## 5 Conclusão

O trabalho por completo foi desenvolvido visando tentar otimizar ao máximo todas as funções, e também garantir a precisão dos resultados, como pode ser visto nos testes. O desenvolvimento do trabalho foi muito interessante, pois pude revisar conceitos vistos em sala (Alocação dinâmica, Tipos Abstrato de dados, entre outras coisas importantes), e também conteúdos vistos no semestre passado de Cálculo Diferencial e Integral I. A parte razoavelmente difícil foi fazer esta documentação em  $\text{\LaTeX}$  por ser a primeira vez.

## Referências

- [1] Como encontrar as raízes de um polinômio, 2013. Disponível em: <https://www.youtube.com/watch?v=KQ4Nx58MzUM>.
- [2] William George Horner. A new method of solving numerical equations of all orders, by continuous approximation, 1819. Disponível em: [https://pt.wikipedia.org/wiki/Esquema\\_de\\_Horner](https://pt.wikipedia.org/wiki/Esquema_de_Horner).
- [3] Donald E Knuth. The art of computer programming (volume 2), 1981.