Manipulação de polinômios com coeficiente inteiros

Doyle Barboza, Gabriel Vaz, Luiz Mosmann, Nikolas Lacerda, Victor Aso

Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

Abstract. *This article presents the study of operations with polynomials, addition, subtraction, multiplication, division and calculation of their roots using proper algorithms for each case, to the Integrative Discipline I of the second semester of 2019, in the graduation of Computer Science from the Pontifícia Universidade Católica do Rio Grande do Sul.*

Resumo. *Este é um artigo que apresenta o estudo de operações com polinomios, soma, subtração, multiplicação, divisão e calculo de suas raízes utilizando algoritmos próprios para cada caso, para a Disciplina Integradora I do segundo semestre de 2019, na graduação de Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul.*

1. Introdução

Os polinômios são expressões muito importantes na álgebra e também na geometria, e calcular suas raízes nem sempre é simples, e ainda com dois polinômios, podemos realizar algumas operações entre eles, tal como, soma, subtração, multiplicação, divisão, portanto, temos como objetivo a pesquisa de algoritmos capazes de realizar essas operações, calcular suas raízes e desenvolver um software educacional com base nesses algoritmos a fim de realizar essas operações, além de avaliar o valor de um polinômio em qualquer ponto e construir o seu gráfico.

Na matemática, para calcular as raízes de um polinômio podemos encontrar um resultado aproximado ou exato, para aproximar um resultado utiliza-se computação numérica pois manipula apenas expressões numéricas por meio de uma aritmética de ponto flutuante e para um resultado exato computação simbólica pois é a área da computação que trata do desenvolvimento de operações tais como fatoração de polinômios, derivadas e integrais, operações e cálculo com matrizes, ou em soluções exatas de equações. As equações podem ser não-lineares, diferenciais ou a diferenças finitas.

Entrando-se um polinômio, iremos encontrar suas raízes utilizando computação numérica pois estamos interessados em um resultado aproximado. Utilizaremos uma estrutura de vetor descrita na seção 5 para trabalhar com a entrada de polinômios.

2. Problema Computacional

Para criação de um software para manipulação de polinômios com coeficiente inteiros, existem alguns importantes problemas computacionais para abordar, faremos uma breve descrição de cada um deles.

2.1. Soma de polinômios

A soma de polinômios consiste em somar dois polinômios quaisquer. O procedimento utilizado na adição de polinômios envolve técnicas de redução de termos semelhantes,

jogo de sinal, operações envolvendo sinais iguais e sinais diferentes.

Exemplo: $(x^2 - 3x - 1) + (3x^2 + 8x - 6) = x^2 - 3x - 1 + 3x^2 + 8x - 6$.
Reduzindo os termos semelhantes: $4x^2 + 5x - 7$, ou seja, $(x^2 - 3x - 1) + (3x^2 + 8x - 6) = 4x^2 + 5x - 7$

2.2. Subtração de polinômios

Assim como a soma de polinômios, a subtração de dois polinômios envolve as mesmas técnicas de redução de termos semelhantes, jogo de sinal, operações envolvendo sinais iguais e sinais diferentes.

Exemplo: $(x^2 - 3x - 1) - (3x^2 + 8x - 6) = x^2 - 3x - 1 - 3x^2 - 8x + 6$.
Reduzindo os termos semelhantes: $-2x^2 - 11x + 5$, ou seja, $(x^2 - 3x - 1) - (3x^2 + 8x - 6) = -2x^2 - 11x + 5$

2.3. Multiplicação de polinômios

As multiplicações de polinômios serão efetuadas utilizando as seguintes propriedades:

- Propriedade da base igual e expoente diferente: $a^n.a^m = a^{n+m}$
- Monômio multiplicado por monômio é o mesmo que multiplicar parte literal com parte literal e coeficiente com coeficiente.

Para realizarmos a multiplicação de polinômio com polinômio, utilizamos a propriedade distributiva, com isso multiplicamos cada termo (monômio) de um dos polinômios por cada um dos outros termos (monômio) do outro polinômio. Caso haja termos semelhantes, ou seja, com mesma parte literal devemos reduzi-los.

Exemplo: $(x^2 + 3x) * (2x^2 + 2) = 2x^4 + 2x^2 + 6x^3 + 6x$.
Neste caso não foi necessário reduzir os termos semelhantes.
Com isso chegamos ao resultado $(x^2 + 3x) * (2x^2 + 2) = 2x^4 + 2x^2 + 6x^3 + 6x$.

2.4. Divisão de polinômios

A divisão de polinômios consiste em dividir dois polinômios quaisquer.

O procedimento utilizado na divisão de polinômios é uma sequencia de passos: Dividir o monômio de maior grau do divisor pelo monômio de maior grau do dividendo, a divisão de monômios consciste em subtrair os expoentes e dividir o coeficientes

$$\begin{array}{r|l} 6x^3 - 2x^2 + \ x + 3 & x^2 - \ x + 1 \\ \hline & 6x \end{array}$$

Feito isso multiplicamos o resultado pelo monômio divisor

$$\begin{array}{rr|l} & 6x^3 - 2x^2 + \ x + 3 & x^2 - \ x + 1 \\ - & 6x^3 - 6x^2 + 6x & \hline & & 6x \\ \hline & & \end{array}$$

E então fazemos a subtração

$$
\begin{array}{rr|l}
- & 6x^3 - 2x^2 + \ \ x + 3 & x^2 - \ \ x + 1 \\
\cline{3-3}
 & 6x^3 - 6x^2 + 6x & 6x \\
\cline{1-2}
 & 4x^2 - 5x + 3 & \\
\cline{1-2}
\end{array}
$$

Feito isso repetimos os passos

$$
\begin{array}{rr|l}
- & 6x^3 - 2x^2 + \ \ x + 3 & x^2 - \ \ x + 1 \\
\cline{3-3}
 & 6x^3 - 6x^2 + 6x & 6x + 4 \\
\cline{1-2}
 & 4x^2 - 5x + 3 & \\
- & 4x^2 - 4x + 4 & \\
\cline{1-2}
 & -x - 1 & \\
\cline{1-2}
\end{array}
$$

A iteração acaba quando após uma subtração chegamos em um resultado que é um polinômio de grau menor que o divisor, como no exemplo anterior, não temos como dividir $-x$ por x^2, então isso impede a divisão desses termos, e $-x - 1$ acabam tornando-se o resto da divisão e o resultado é o quoeciente gerado, no caso, $6x + 4$

2.5. Cálculo de raízes

No estudo do valor numérico de um polinômio, notamos que para cada valor que atribuímos à variável x, encontramos um valor numérico para o polinômio.
A raiz de um polinômio é denotada pelo valor que a variável assume de modo que o valor numérico do polinômio seja igual a zero. Na linguagem matemática, seria assim:

α é raiz do polinômio $p(x)$ se e somente se, $p(\alpha) = 0$

Antes de compreendermos o conceito de raiz, vamos relembrar a forma geral de um polinômio de grau n.

$$p(x) = a_n X^n + a_{n-1} X^{n-1} + a_{n-2} X^{n-2} + ... + a_1 X^1 + a_0 X^0$$

O termo "raiz" é visto pela primeira vez como a solução de uma equação, entretanto você deve lembrar que aquela equação estava igual a zero, sendo o zero o valor numérico da equação.
As raízes polinomiais possuem grande importância para a construção de gráficos dos polinômios, afinal, com essas raízes podemos encontrar os pontos onde a função intersecta o eixo das abscissas (eixo x).
Problemas envolvendo raízes polinomiais podem aparecer, normalmente, de duas maneiras. Em uma verifica-se se o valor informado para a variável levará ao valor numérico zero, ou seja, se este valor é a raiz do polinômio; e na outra maneira deverá ser encontrada a raiz do polinômio.
Um fato importante a ser ressaltado é que a quantidade de raízes de um polinômio está diretamente relacionada ao grau deste polinômio. Por exemplo, um polinômio de grau 2 poderá ter no máximo duas raízes, sendo estes números complexos ou não. Por sua vez, o polinômio de grau 3 terá no máximo 3 raízes.

2.5.1. Método de Newton

O método de Newton (ou Método de Newton–Raphson), foi desenvolvido por Isaac Newton e Joseph Raphson. O método tem como objetivo estimar as raízes de uma função. É um método interativo que inicia ao escolhermos uma aproximação inicial. Após isso, calcula-se a equação da reta tangente (por meio da derivada) da função nesse ponto e a interseção dela com o eixo das abscissas , a fim de encontrar uma melhor aproximação para a raiz. Repetindo-se o processo para encontrarmos a raiz da função. Como lidamos com funções polinomiais o cálculo da derivada não é o maior dos problemas, o que torna o método de Newton um bom método para lidar com essas equações.

O método de Newton é dado pela seguinte sequência recursiva:

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n \in N$

onde x_0 é uma aproximação inicial dada, n indica a n-ésima iteração do algoritmo e $f'(x_n)$ é a derivada da função f no ponto x_n
O Método é executado até critério de parada ser atingido. Exatidão (DIGSE - digitos significativos exatos) ou número de iterações máximas.

Complexidade : (4n) + cálculo da derivada[1].
Classe : Classe P (para polinômios) [1].

2.5.2. Cota de Fujiwara

Cálculo que sustenta o Método de Newton. Resulta em um intervalo que contém todas as raízes do polinômio em questão. Possibilita busca limitada entre um número finito de inteiros como candidatos a próxima etapa.[3].

$$|\alpha| \leq 2 * max \left[\left| \frac{a_{n-1}}{a_n} \right|, \left| \frac{a_{n-2}}{a_n} \right|^{\frac{1}{2}}, \left| \frac{a_{n-3}}{a_n} \right|^{\frac{1}{3}}, \ldots, \left| \frac{a_1}{a_n} \right|^{\frac{1}{n-1}}, \left| \frac{a_0}{a_n} \right|^{\frac{1}{n}} \right]$$

2.5.3. Separação

O método de Newton, converge para a raíz real mais próxima do valor inicial desse método iterativo. Para calcular todas as raízes desejadas é imprescindível verificar valores próximos da solução. Para estipular onde estão as raízes dentro da cota, é necessário encontrar trocas de sinais entre dois valores f(a) e f(b), que indicam que há um número ímpar de raízes entre o intervalo $I = [a, b]$ pois a função f necessariamente cruza o eixo das abscissas ao menos 1 vez. Quanto menor a distância entre os valores a e b maior a qualidade do intervalo, mas menos eficiente será a busca pelas raízes. Pesquisaremos por intervalos de separação de acordo com o tamanho da cota encontrada.

2.5.4. Regra de Descartes

Utilizado para encontrar o número de raízes positivas e negativas de um polinômio [2]. Para encontra-las devemos começar contando as mudanças de sinais no polinômio, para assim, encontrarmos o total de possíveis raízes deste polinômio. Utilizaremos o polinômio $x^3 - 2x^2 - x + 2$ como exemplo.
Neste polinomio teremos o total de 3 possiveis respostas, pois é o total de mudança de sinais em $+x^3 - 2x^2$, $-2x^2 - x$ e $-x + 2$
Após encontrarmos o total de sinais, podemos encontrar as raízes positivas, negativas e imaginárias.

positivas	negativas	imaginárias	total
			3
			3

As raízes positivas se dá pelo número de trocas de sinais entre positivo(+) e negativo(-), neste caso, teremos duas raízes positivas pois apenas contamos a troca entre $+x^3 - 2x^2$ e $-x + 2$ e as linhas abaixo de cada raíz positiva ou negativa será uma subtração da raíz da célula acima com dois, desde que o número seja par, caso contrário repete-se o número.

positivas	negativas	imaginárias	total
2			3
0			3

Para as raízes negativas, trocaremos todos os x do nosso polinômio por -x, logo teremos, $+x^3 - 2x^2$ e $-x + 2$, e o número de troca de sinais nos dá as raízes negativas, neste caso teremos apenas uma raíz negativa pois só há mudança em $-2x^2 + x$

positivas	negativas	imaginárias	total
2	1		3
0	0		3

Encontra-se as raízes imaginarias sendo o total de raízes menos a soma das positivas e negativas. Assim teremos as possíveis raízes deste polinômio sendo:

positivas	negativas	imaginárias	total
2	1	0	3
0	1	2	3

Assim, utilizaremos essas informações para complementar as informações que serão obtidas na etapa de separação e do método de newton, pois será possivel em alguns casos afirmar, por exemplo, sobre multiplicidade de raízes.

2.5.5. Regra de Huat

A regra de Huat [4] é utilizada para verificar se existem ou não raízes complexas de um polinomio.
Para um polinomio, por exemplo, $p(x) = 3x^4 + x^3 + 2x^2 + x + 3$ deve-se ser satisfeitas as seguintes premissas: Se $p(0) \neq 0$ e para algum $k, 0 < k < n$, tivermos $(ak)^2 \leq (ak-1).(ak+1)$, onde ak é um coeficiente do polinomio, então p(x) terá raízes complexas.

Neste caso:
$p(0) = 10 \neq 0$
$a0 = 3$
$a1 = 1$
$a2 = 2$
$a3 = 1$
$a4 = 3$
$(a1)^2 \leq (a0).(a2)$
$(1)^2 \leq 3.2$
$1 \leq 6$
Portanto, $p(x)$ possui raízes complexas.
Assumindo um polinomio $p(x) = 2x^3 - 3x^2 + 4x - 2$, neste caso teremos:
$p(0) = -2 \neq 0$
$a0 = 2$
$a1 = -3$
$a2 = 4$
$a3 = -2$
$(a1)^2 \leq (a0).(a2)$
$(-3)^2 > 2.4$
$9 > 8$
e como não satisfaz as premissas, nada podemos afirmar ao utilizarmos esta regra.

2.5.6. Regra de Lacuna

A regra de Lacuna [4] é utilizada para verificar se um polinomio tem ou não raízes complexas. Para o polinomio, por exemplo, $p(x) = -3x^5 - 2x^3 + x^2 - x + 1$ devem ser satisfeitas as seguintes premissas:
Se $p(0) \neq 0$ e para algum $k, 0 < k < n$, tivermos $ak = 0$ e $ak - 1.ak + 1 > 0$, então $p(x)$ terá raízes complexas ou se $p(0) \neq 0$ e existirem dois ou mais coeficientes nulos sucessivos, então $p(x)$ terá raízes complexas.
Neste caso:
$p(0) \neq 0$
$a0 = -3$ (coeficiente de x^5)
$a1 = 0$ (coeficiente de x^4)
$a2 = -2$ (coeficiente de x^3)
$a3 = 1$ (coeficiente de x^2)
$a4 = -1$ (coeficiente de x^1)
$a5 = 1$ (coeficiente de x^0)
$(a1) = 0$ e $(a0).(a2) > 0$, pois $0 = 0$ e $(-3).(-2) > 0$
portanto, $p(x)$ possui raízes complexas.

3. Aplicabilidades em problemas

Encontrar raízes de uma função é um problema que muitas vezes não pode ser resolvido por métodos diretos. Mesmo assim essa é uma tarefa necessária em diversas áreas que precisam desse tipo de resposta.

Para realizar tal tarefa podemos usar métodos iterativos que aproximam os zeros

de dada função. Exitem métodos como Bisseção, Falsa-Posição, Secante, e o Método de Newton, que será utilizado para nossa aplicação.

4. Estruturas e modelos discretos que sustentam a solução

Identificaremos os coeficientes de cada termo de cada polinômio de entrada em texto os mapeando para vetores de inteiros, ordenados de forma que as primeiras posições de cada vetor, representam os termos independetes (x^0). A segunda posição o coeficente do termo x^1, a terceira o x^2, e assim sucessivamente. O numero de posições necessárias é calculado conforme o coeficiente com mair grau da entrada.

Dado um polinômio como entrada, iremos armazenar seus coeficientes em um vetor e dividi-lo por operações, ou seja, com a string de entrada do tipo $ax^2 + bx + c$, iremos procurar o maior termo desse polinomio onde o tamanho do nosso vetor será esse termo mais um, após, vamos separar a entrada por espaços em branco e armazenar cada coeficiente na posição onde o índice do vetor é igual ao termo.

Por exemplo, entrando-se $x^6 + 2x^4 + x^2 - 1$, vamos encontrar o maior termo como 6, logo teremos um vetor de tamanho 7.

0	1	2	3	4	5	6

Criado o vetor, adicionaremos os coeficientes do polinômio em seus respectivos índices e onde não há o termo na expressão colocaremos um '0':

x^0	x^1	x^2	x^3	x^4	x^5	x^6
-1	0	1	0	2	0	1
0	1	2	3	4	5	6

5. Algoritmos que sustentam a solução

A seguir, faremos uma apresentação dos problemas computacionais relacionados aos polinômios e os respectivos pseudocódigos dos algoritmos para solucionar tais problemas apresentados.

5.1. Exemplo do Funcionamento da Soma de Polinômios

Dado dois polinômios quaisquer estes estão representados computacionalmente como vetores:

$2x^4 + x^2 + 6x - 1 =$

x^0	x^1	x^2	x^3	x^4	x^5	x^6
-1	6	1	0	2	0	0
0	1	2	3	4	5	6

$-2x^6 + 4x^5 + x^3 + 2x^2 + 5 =$

x^0	x^1	x^2	x^3	x^4	x^5	x^6
5	0	2	1	0	4	-2
0	1	2	3	4	5	6

Com os polinômios representados como vetores, cada índice do vetor representa o monômio de grau equivalente ao índice, então somamos as posições iguais entre o vetor do polinômio um e o vetor do polinômio dois, assim somando os monômios de grau igual entre os polinômios e então temos o resultado da soma entre esses polinômios:

$-2x^6 + 4x^5 + 2x^4 + x^3 + 3x^2 + 6x + 4 =$

x^0	x^1	x^2	x^3	x^4	x^5	x^6
4	6	3	1	2	4	-2
0	1	2	3	4	5	6

5.1.1. Algoritmo da Soma de Polinômios

A seguir, fizemos em pseudocódigo o algoritmo para somar polinômios.

Algorithm 1 Soma de Polinômios

```
1:  procedure POLYSUM(poly_1, poly_2)
2:      if poly_1 length > poly_2 length then
3:          poly_result = [poly_1 length]
4:          while poly_2 length < poly_1 length do
5:              poly_2.insert(0)
6:          end while
7:      else
8:          poly_result = [poly_2 length]
9:          while poly_1 length < poly_2 length do
10:             poly_1.insert(0)
11:         end while
12:     end if
13:     for i ← 0 to poly_result length do
14:         poly_result[i] = poly_1[i] + poly_2[i]
15:     end for
16:     return poly_result
17: end procedure
```

5.2. Exemplo do Funcionamento da Subtração de Polinômios

Dado dois polinômios quaisquer estes estão representados computacionalmente como vetores:

$$5x^5 - 2x^4 + x^2 + 2x + 1 = \begin{array}{|c|c|c|c|c|c|} \hline x^0 & x^1 & x^2 & x^3 & x^4 & x^5 \\ \hline 1 & 2 & 1 & 0 & 2 & 5 \\ \hline 0 & 1 & 2 & 3 & 4 & 5 \end{array}$$

$$4x^4 + x^3 + x^2 + 5 = \begin{array}{|c|c|c|c|c|c|} \hline x^0 & x^1 & x^2 & x^3 & x^4 & x^5 \\ \hline 5 & 0 & 1 & 1 & 4 & 0 \\ \hline 0 & 1 & 2 & 3 & 4 & 5 \end{array}$$

Com os polinômios representados como vetores, cada índice do vetor representa o monômio de grau equivalente ao índice, então subtraimos as posições iguais entre o vetor do polinômio um e o vetor do polinômio dois, assim subtraindo os monômios de grau igual entre os polinômios e então temos o resultado da subtração entre esses polinômios:

$$5x^5 - 2x^4 - x^3 + 2x - 4 = \begin{array}{|c|c|c|c|c|c|} \hline x^0 & x^1 & x^2 & x^3 & x^4 & x^5 \\ \hline \text{-4} & 2 & 0 & \text{-1} & \text{-2} & 5 \\ \hline 0 & 1 & 2 & 3 & 4 & 5 \end{array}$$

5.2.1. Algoritmo da Subtração de Polinômios

A seguir, fizemos em pseudocódigo o algoritmo para subtrair polinômios.

Algorithm 2 Subtração de Polinômios

```
1:  procedure PolySub(poly_1, poly_2)
2:      if poly_1 length > poly_2 length then
3:          poly_result = [poly_1 length]
4:          while poly_2 length < poly_1 length do
5:              poly_2.insert(0)
6:          end while
7:      else
8:          poly_result = [poly_2 length]
9:          while poly_1 length < poly_2 length do
10:             poly_1.insert(0)
11:         end while
12:     end if
13:     for i ← 0 to poly_result length do
14:         poly_result[i] = poly_1[i] − poly_2[i]
15:     end for
16:     return poly_result
17: end procedure
```

5.3. Exemplo do Funcionamento da Multiplicação de Polinômios

Dado dois polinômios quaisquer estes estão representados computacionalmente como vetores:

$$x^2 + 2x + 1 = \begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline \end{array} \quad \begin{matrix} x^0 & x^1 & x^2 \\ 0 & 1 & 2 \end{matrix}$$

$$2x^2 + 5 = \begin{array}{|c|c|c|} \hline 5 & 0 & 2 \\ \hline \end{array} \quad \begin{matrix} x^0 & x^1 & x^2 \\ 0 & 1 & 2 \end{matrix}$$

Com os polinômios representados como vetores, cada índice do vetor representa o monômio de grau equivalente ao índice, então multiplicamos cada elemento do primeiro vetor com todos os elementos do segundo vetor, assim criando um novo vetor com o resultado:

$$x^2 * (2x^2 + 5) = \begin{array}{|c|c|c|c|c|} \hline 0 & 0 & 5 & 0 & 2 \\ \hline \end{array} \quad \begin{matrix} x^0 & x^1 & x^2 & x^3 & x^4 \\ 0 & 1 & 2 & 3 & 4 \end{matrix}$$

ao multiplicar o próximo elemento do primeiro vetor com todos os elementos do segundo vetor consideramos o mesmo vetor resultado, com os resultados calculados anteriormente, então em caso de termos semelhantes, fazemos a soma naquela determinada posição:

$$2x * (2x^2 + 5) = \begin{array}{|c|c|c|c|c|} \hline 0 & 0\text{+}10 & 5 & 0\text{+}4 & 2 \\ \hline \end{array} \quad \begin{matrix} x^0 & x^1 & x^2 & x^3 & x^4 \\ 0 & 1 & 2 & 3 & 4 \end{matrix} = \begin{array}{|c|c|c|c|c|} \hline 0 & 10 & 5 & 4 & 2 \\ \hline \end{array} \quad \begin{matrix} x^0 & x^1 & x^2 & x^3 & x^4 \\ 0 & 1 & 2 & 3 & 4 \end{matrix}$$

e o mesmo acontece para os demais elementos:

$$1 * (2x^2 + 5) = \begin{array}{|c|c|c|c|c|} x^0 & x^1 & x^2 & x^3 & x^4 \\ \hline 0+5 & 10 & 5+2 & 4 & 2 \\ \hline 0 & 1 & 2 & 3 & 4 \end{array} = \begin{array}{|c|c|c|c|c|} x^0 & x^1 & x^2 & x^3 & x^4 \\ \hline 5 & 10 & 7 & 4 & 2 \\ \hline 0 & 1 & 2 & 3 & 4 \end{array}$$

chegando assim no resultado: $2x^4 + 4x^3 + 7x^2 + 10x + 5$

5.3.1. Algoritmo da Multiplicação de Polinômios

A seguir, fizemos em pseudocódigo o algoritmo para multiplicar polinômios.

Algorithm 3 Multiplicação de Polinômios

```
1: procedure POLYMULT(poly_1, poly_2)
2:     poly_result = [poly_1 length + poly_2 length]
3:     for i ← 0 to poly_1 length do
4:         for j ← 0 to poly_2 length do
5:             poly_result[i + j] = poly_result[i + j] + poly_1[i] * poly_2[j]
6:         end for
7:     end for
8:     return poly_result
9: end procedure
```

5.4. Exemplo do Funcionamento da Divisão de polinômios

Dado dois polinômios quaisquer estes estão representados computacionalmente como vetores:

$$x^4 + x + 1 = \begin{array}{|c|c|c|c|c|} x^0 & x^1 & x^2 & x^3 & x^4 \\ \hline 1 & 1 & 0 & 0 & 4 \\ \hline 0 & 1 & 2 & & \end{array}$$

$$x^2 + 5 = \begin{array}{|c|c|c|} x^0 & x^1 & x^2 \\ \hline 5 & 0 & 2 \\ \hline 0 & 1 & 2 \end{array}$$

Com os polinômios representados como vetores, fazemos os seguintes passos:

Pegamos o monômio de maior grau do polinômio divisor e o monomio de maior grau do polinômio dividendo, subtraimos os seus expoentes e dividimos os seus coeficientes, ficando da seguinte forma:

$$x^4/x^2 = \begin{array}{|c|} x^4 \\ \hline 4 \\ \hline 4 \end{array} / \begin{array}{|c|} x^2 \\ \hline 2 \\ \hline 2 \end{array} = \begin{array}{|c|} x^2 \\ \hline 2 \\ \hline 2 \end{array} = x^2$$

Esse monômio resultante será inserido em um novo vetor, chamado vetor resultado, e será inserido como o seu maior grau:

$$x^2{+}?{+}? = \begin{array}{|c|c|c|} x^0 & x^1 & x^2 \\ \hline ? & ? & 2 \\ \hline 0 & 1 & 2 \end{array}$$

Pegamos então esse monômio e multiplicamos pelo dividendo:

$$x^2 * (x^2 + 5) = \begin{array}{|c|c|c|c|c|} \hline x^0 & x^1 & x^2 & x^3 & x^4 \\ \hline 0 & 0 & 5 & 0 & 1 \\ \hline 0 & 1 & 2 & 1 & 2 \\ \end{array}$$

E por fim, fazemos a subtração do dividendo pela resultado da multiplicação feita anteriormente:

$$x^4 + x + 1 - (x^4 + 5x^2) = \begin{array}{|c|c|c|c|c|} \hline x^0 & x^1 & x^2 & x^3 & x^4 \\ \hline 1 & 1 & \text{-5} & 0 & 0 \\ \hline 0 & 1 & 2 & 1 & 2 \\ \end{array}$$

Com isso conseguimos eliminar um monômio do nosso divisor. Esse algoritmo é iterativo, iremos repetir os passos anteriores, apenas mudando o nosso dividendo, que agora é o resultado da subtração anterior, ou seja, $-5x^2 + x + 1$, e acrescentaremos os próximos valores resultados no mesmo vetor resultado já criado (onde constam os '?'), assim ao fim da iteração o vetor resultado será o vetor relativo ao resultado da divisão dos dois polinômios. A iteração acaba quando o grau do vetor dividendo é menor do que o grau do vetor divisor.

5.4.1. Algoritmo da Divisão de polinômios

Algorithm 4 Divisão de Polinômios

```
1:  procedure PolyDiv(poly_1, poly_2)
2:      poly_result = [poly_1 length − poly_2 length]
3:      poly_actual = poly_1
4:      while poly_actual length ≥ poly_2 length do
5:          aux = [poly_1 length − poly_2 length]
6:          dividendo = poly_actual[−1]
7:          divisor = poly_2[−1]
8:          aux[poly_actual length − poly_2 length] = dividendo//divisor
9:          poly_result[poly_1 length − poly_2 length] = dividendo//divisor
10:         poly_actual_aux = POLY_MULT(aux, poly_2)
11:         poly_actual = POLY_SUB(poly_actual, poly_actual2)
12:         while poly_actual[−1] == 0 do
13:             poly_actual.remove()
14:         end while
15:     end while
16:     return poly_result
17: end procedure
```

5.5. Algoritmo para Regra de Huat e da Lacuna

A seguir, fizemos em pseudocódigo o algoritmo para as regras para indicar raízes complexas.

5.6. Algoritmo para Regra de Descartes

A seguir, fizemos em pseudocódigo o algoritmo para a regra dos sinais de descartes.

Algorithm 5 Regras de Huat e da Lacuna

```
1:  procedure HUAT_LACUNA(poly_array)
2:      t = lengthpoly_array
3:      if poly_array[0] == 0 AND poly_array[1] == 0 then
4:          return True
5:      end if
6:      for i ← 1 to t − 2 do
7:          if power(poly_array[i], 2) ≤ poly_array[i − 1] ∗ poly_array[i + 1] then
8:              return True
9:          end if
10:         if poly_array[i] == 0 AND poly_array[i − 1] ∗ poly_array[i + 1] > 0 then
11:             return True
12:         end if
13:         if poly_array[i] == 0 AND poly_array[i + 1] == 0 then
14:             return True
15:         end if
16:     end for
17:     return False
18: end procedure
```

5.7. Algoritmo para Cota de Fujiwara

A seguir, fizemos em pseudocódigo o algoritmo para o calculo da cota de Fujiwara.

5.8. Algoritimo para Separação de raízes

A seguir, fizemos em pseudocódigo o algoritmo para separar valores iniciais para o método de Newton.

5.9. Algoritimo para o Método de Newton

A seguir, fizemos em pseudocódigo o algoritmo para o Método de Newton.

6. Implementação

Essa seção tem como objetivo apresentar e expor o processo para desenvolvimento do software educacional para manipulação de polinômios

6.1. Tecnologias

O software educacinal foi pensado para ser usado via web, por uma questão de acessibilidade, então toda a parte visual do aplicativo e a interação com o usuário encontram-se implementadas no front-end com o uso das seguintes tecnologias:

- HTML
- CSS
- Javascript
- Flask

Algorithm 6 Regra dos Sinais de Descartes

procedure DESCARTES($poly\_array$)
  $array\_aux = poly\_array\ without\ zeros$
  $pos\_alt\_signs = 0$
  $i = 0$
  while $i \leq length\ array\_aux - 2$ **do**
    $aux1 = array\_aux[i]$
    $aux2 = array\_aux[i + 1]$
    if $(aux1 > 0\ AND\ aux2 < 0)\ OR\ (aux1 < 0\ AND\ aux2 > 0)$ **then**
      $pos\_alt\_signs = pos\_alt\_signs + 1$
    end if
    $i = i + 1$
  end while
  $array\_aux = poly\_array$
  $neg\_alt\_signs = 0$
  $i = 0$
  for $i \leftarrow bound1$ to $bound2$ **do**
    if $i\%2 \neq 0$ **then**
      $array\_aux[i] = -array\_aux[i]$
    end if
  end for
  $i = 0$
  while $i \leq length\ array\_aux - 2$ **do**
    $aux1 = array\_aux[i]$
    $aux2 = array\_aux[i + 1]$
    if $(aux1 > 0\ AND\ aux2 < 0)\ OR\ (aux1 < 0\ AND\ aux2 > 0)$ **then**
      $neg\_alt\_signs = neg\_alt\_signs + 1$
    end if
    $i = i + 1$
  end while
  $possible\_pos\_roots = pos\_alt\_signs - 2n$
  $possible\_neg\_roots = neg\_alt\_signs - 2n$ ▷ N is integer and result is ≥ 0
end procedure

Algorithm 7 Cota de Fujiwara

procedure FUJIWARA($poly\_array$)
  $t = lengthpoly\_array$
  $n = t - 1$
  $d = poly\_array[n]$
  $partial\_results = []$
  for $i \leftarrow 0$ to n **do**
    $partial\_results[i] = power(absolute(poly\_array[i]/d), (1/(n-1)))$
  end for
  $return\ 2 * max(partial\_results)$
end procedure

Algorithm 8 Separação de Raízes

```
1: procedure SEPARAÇÃO(poly_array, bound1, bound2)
2:     results = []
3:     step = bound1 * 2/1000
4:     previous_evall = POLY_EVALUATE(poly_array, bound1)
5:     if previous_eval == 0 then
6:         results.append(bound1)
7:     end if
8:     for x ← bound1 to bound2, absolute(step) do
9:         current_eval = POLY_EVALUATE(poly_array, x)
10:        if (previous_eval > 0 AND current_eval < 0) OR (previous_eval < 0
   AND current_eval > 0) then
11:            results.append(x)
12:        end if
13:        if current_eval == 0 then
14:            results.append(x)
15:        end if
16:        previous_eval = current_eval
17:    end for
18:    return results
19: end procedure
```

Algorithm 9 Método de Newton

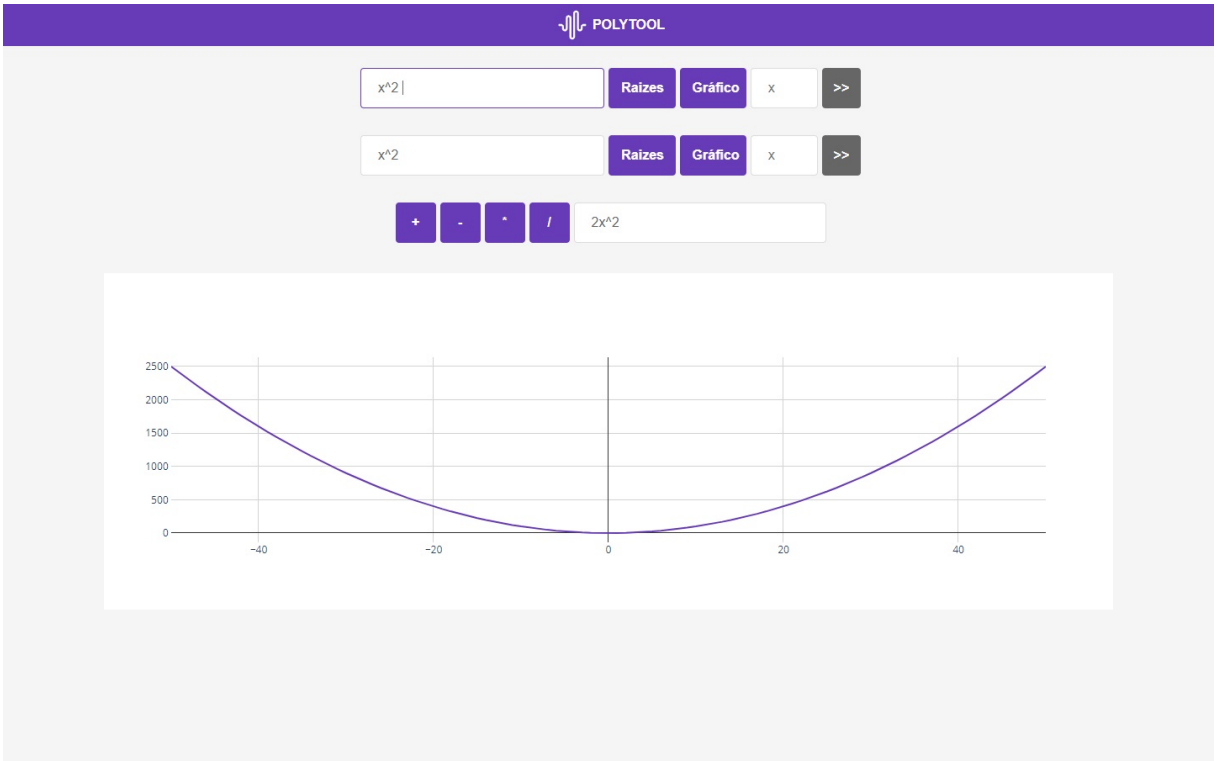
```
1: procedure NEWTON(poly_array, derivate_array, x0, eps, max_iter)
2:     curr_x = x0
3:     new_x = 0
4:     iter_count = 0
5:     while iter_count < max_iter do
6:         func_eval = POLY_EVALUATE(poly_array, curr_x)
7:         der_eval = POLY_EVALUATE(derivate_array, curr_x)
8:         if func_eval == 0 then
9:             return curr_x
10:        end if
11:        if der_eval ≠ 0 then
12:            new_x = curr_x − (func_eval/der_eval)
13:            if absolute(new_x − curr_x) ≤ eps then
14:                return new_x
15:            end if
16:        end if
17:        curr_x = new_x
18:        iter_count = iter_count + 1
19:    end while
20:    return curr_x
21: end procedure
```

A parte dos algoritmos está implementada em Python sem o uso de bibliotecas externas e o gráfico é criado a partir da biblioteca Pyplot <https://matplotlib.org/3.1.1/tutorials/\\introductory/pyplot.html/>, o uso dos algoritmos e do gráfico pela aplicação web é feito via consumo de API.

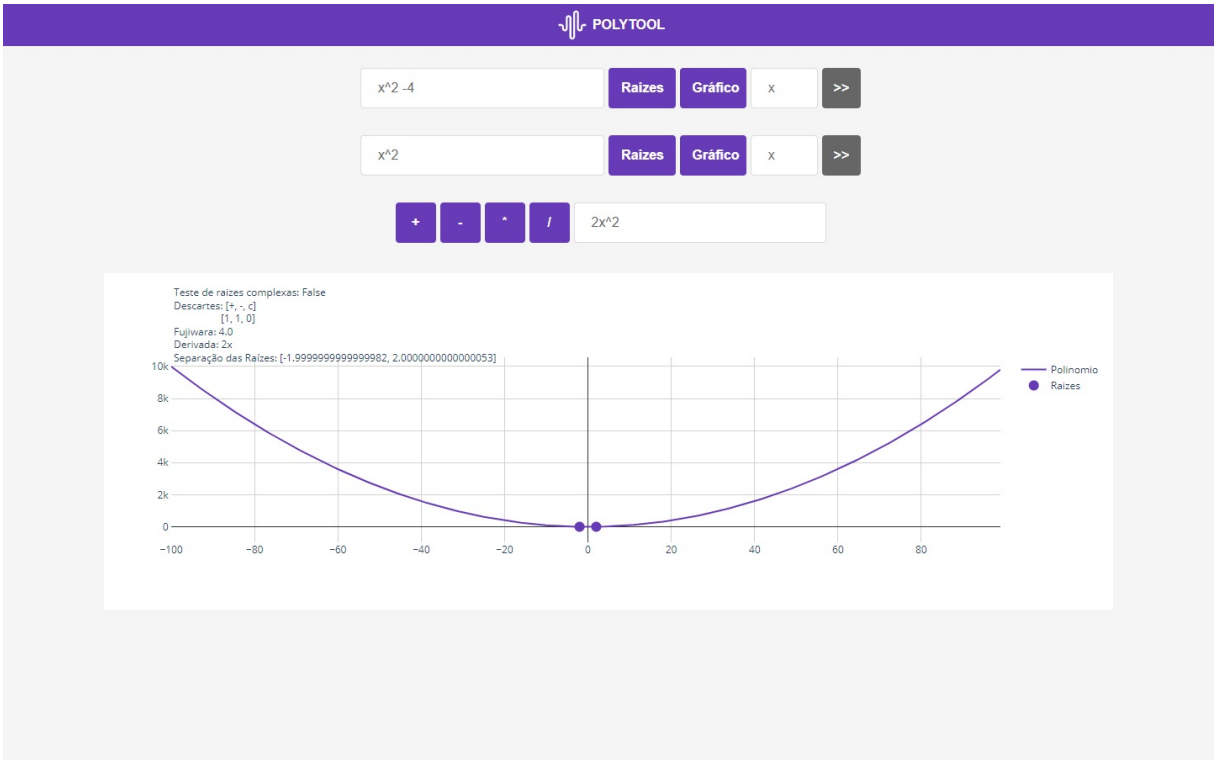
A aplicação está hospedada no heroku <http://www.https://www.heroku.com/> e pode ser acessada em: https://polytool.herokuapp.com/

6.2. Software

O software em si possui uma única tela com todas as suas funcionalidades inclusas, a tela possui:



- Dois campos inputs referentes a entrada de polinômios pelo usuário
- Botões referentes a cada campo input
  - Um botão 'raiz' cuja funcionalidade é calcular as raízes do polinômio e plotar o gráfico do polinômio com suas respectivas raízes
  - Um botão 'gráfico' cuja funcionalidade é plotar o gráfico do polinômio
- Botões referentes a operações aritméticas
  - Um botão '+' cuja funcionalidade é somar os polinômios que foram inseridos pelo usuário
  - Um botão '-' cuja funcionalidade é subtrair os polinômios que foram inseridos pelo usuário
  - Um botão '*' cuja funcionalidade é multiplicar os polinômios que foram inseridos pelo usuário
  - Um botão '/' cuja funcionalidade é dividir os polinômios que foram inseridos pelo usuário



O software libera as funcionalidades conforme as entradas do usuário, por exemplo, não é possível plotar um gráfico enquando não tiver digitado um polinômio nem somar dos polinômios caso apenas um foi informado.

O gráfico é gerado na parte inferior da tela, assim que o usuário solicita a sua criação, é possível interagir com o gráfico, utilizando a navegação, zoom e navegando com o mouse entre os pontos para ver os valores da função em x.

7. Resultados

7.1. Caso de teste 1 (soma)

Considerando como entrada os polinômios $x^5+4x^3+2x^2-3x+1$ e $4x^5+2x^3+3x^2-x+2$, obtivemos pelo algoritmo de soma o seguinte resultado: $5x^5 + 6x^3 + 5x^2 - 2x^+3$

7.2. Caso de teste 2 (subtração)

Considerando como entrada os polinômios $x^5+4x^3+2x^2-3x+1$ e $4x^5+2x^3+3x^2-x+2$, obtivemos pelo algoritmo de subtração o seguinte resultado: $-3x^5 + 2x^3 - x^2 - 4x^-1$

7.3. Caso de teste 3 (multiplicação)

Considerando como entrada os polinômios $x^5+4x^3+2x^2-3x+1$ e $4x^5+2x^3+3x^2-x+2$, obtivemos pelo algoritmo de multiplicação o seguinte resultado: $4x^10 + 18x^8 + 11x^7 - 3x^6 + 22x^5 + 4x^4 + 3x^3 + 4x^2 - 5x^+2$

7.4. Caso de teste 4 (divisão)

Considerando como entrada os polinômios $4x^5+2x^3+3x^2-x+2$ e $x^5+4x^3+2x^2-3x+1$, obtivemos pelo algoritmo de divisão o seguinte resultado: 2

7.5. Caso de teste 5 (raízes)

Considerando como entrada o polinômio $x^5 + 4x^3 + 2x^2 - 3x + 1$, obtivemos o seguinte resultado: $-1,084755e - 6,661338x^{-16}$

8. Considerações finais

Conseguimos estudar e aplicar regras auxiliares e um dos métodos numéricos iterativos para avaliar polinômios com coeficientes inteiros. Apesar desse domínio representar apenas parte das funções, os algoritmos que aplicamos tem grande utilidade.

Trabalhos posteriores poderiam expandir este de modo a testar de forma organizada todas as funções além de expandir as informações para o usuário, mostrando o passo a passo por trás do site, tornando o software mais didático.

Referências

[1] Frederico Ferreira Campos Filho. *Algoritmos numéricos*. LTC, 2007.

[2] Dalcidio Moraes Cláudio and Jussara Maria Marins. *Cálculo numérico computacional: teoria e prática*, volume 2. Atlas, 1989.

[3] Neide Bertoldi Franco. *Cálculo numérico*. Pearson, 2006.

[4] Carlos Alberto Alonso Sanches and Juliana de Melo Bezerra. Matemática computacional. `http://www.comp.ita.br/~alonso/ensino/CCI22/cci22-cap4.pdf`, Nov 2011. Accessed on 2019-11-21.