Manipulação de Polinômios

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

PUCRS

Problema Computacional

- Soma de Polinômios
- Subtração de Polinômios
- Multiplicação de Polinômios
- Divisão de Polinômios
- Raízes do Polinômio

Estrutura Computacional

Polinômio = $x^5 + 2x^4 + 3x^3 + 5x^2 - 1$

Entrada = x^5 + 3x^3 + 5x^2 + x - 1

Representação no vetor:

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

Soma de Polinômios

$2x^4 + 3x^3 + 5x^2 + x - 1 =$

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

+

$3x^3 + 2x^2 + x + 1 =$

x^0	x^1	x^2	x^3	x^4
1	1	2	3	0
0	1	2	3	4

=

$2x^4 + 6x^3 + 7x^2 + 2x =$

x^0	x^1	x^2	x^3	x^4
0	2	7	6	2
0	1	2	3	4

Algorithm 1 Soma de Polinômios

```
procedure POLYSUM(poly_1, poly_2)
    if poly_1 length > poly_2 length then
        poly_result = [poly_1 length]
        while poly_2 length < poly_1 length do
            poly_2.insert(0)
        end while
    else
        poly_result = [poly_2 length]
        while poly_1 length < poly_2 length do
            poly_1.insert(0)
        end while
    end if
    for i ← 0 to poly_result length do
        poly_result[i] = poly_1[i] + poly_2[i]
    end for
    return poly_result
end procedure
```

Subtração de Polinômios

$2x^4 + 3x^3 + 5x^2 + x - 1 =$

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

\-

$3x^3 + 2x^2 + x + 1 =$

x^0	x^1	x^2	x^3	x^4
1	1	2	3	0
0	1	2	3	4

=

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

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

Algorithm 2 Subtração de Polinômios

```
procedure POLYSUB(poly_1, poly_2)
    if poly_1 length > poly_2 length then
        poly_result = [poly_1 length]
        while poly_2 length < poly_1 length do
            poly_2.insert(0)
        end while
    else
        poly_result = [poly_2 length]
        while poly_1 length < poly_2 length do
            poly_1.insert(0)
        end while
    end if
    for i ← 0 to poly_result length do
        poly_result[i] = poly_1[i] − poly_2[i]
    end for
    return poly_result
end procedure
```

Multiplicação de Polinômios

$5x^2 + x =$

x^0	x^1	x^2
0	1	5
0	1	2

*

$x^3 + 1 =$

x^0	x^1	x^2	x^3
1	0	0	3
0	1	2	3

=

$5x^5 + x^4 + 5x^2 + x =$

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

Algorithm 3 Multiplicação de Polinômios

```
procedure POLYMULT(poly_1, poly_2)
    poly_result = [poly_1 length + poly_2 length]
    for i ← 0 to poly_1 length do
        for j ← 0 to poly_2 length do
            poly_result[i + j] = poly_result[i + j] + poly_1[i] * poly_2[j]
        end for
    end for
    return poly_result
end procedure
```

Divisão de Polinômios

$x^4 + x^2 =$

x^0	x^1	x^2	x^3
0	1	0	1
0	1	2	3

/

$x^2 + 1 =$

x^0	x^1	x^2	x^3
1	0	0	3
0	1	2	3

=

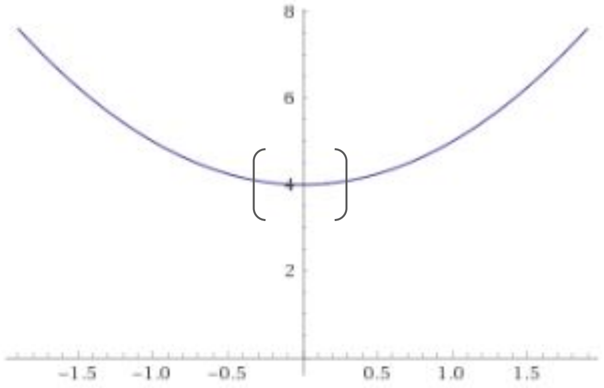
$x^2 =$

x^0	x^1	x^2
0	0	1
0	1	2

Algorithm 4 Divisão de Polinômios

```
procedure POLYDIV(poly_1, poly_2)
    poly_result = [poly_1 length − poly_2 length]
    poly_actual = poly_1
    while poly_actual length ≥ poly_2 length do
        aux = [poly_1 length − poly_2 length]
        dividendo = poly_actual[−1]
        divisor = poly_2[−1]
        aux[poly_actual length − poly_2 length] = dividendo//divisor
        poly_result[poly_1 length − poly_2 length] = dividendo//divisor
        poly_actual_aux = POLY_MULT(aux, poly_2)
        poly_actual = POLY_SUB(poly_actual, poly_actual2)
        while poly_actual[−1] == 0 do
            poly_actual.remove()
        end while
    end while
    return poly_result
end procedure
```

Regra da Lacuna e Huat



$x^2 + 4 = 0$

Algorithm 5 Regras de Huat e da Lacuna

```
procedure HUAT_LACUNA(poly_array)
    l = lengthpoly_array
    if poly_array[0] == 0 AND poly_array[1] == 0 then
        return True
    end if
    for i ← 1 to l − 2 do
        if power(poly_array[i], 2) ≤ poly_array[i − 1] ∗ poly_array[i + 1] then
            return True
        end if
        if poly_array[i] == 0 AND poly_array[i − 1] ∗ poly_array[i + 1] > 0 then
            return True
        end if
        if poly_array[i] == 0 AND poly_array[i + 1] == 0 then
            return True
        end if
    end for
    return False
end procedure
```

Regra de Descartes

$x^3 - 2x^2 - x + 2$

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

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 ≤ 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 ← bound1 to bound2 do
        if i%2 ≠ 0 then
            array_aux[i] = −array_aux[i]
        end if
    end for
    i = 0
    while i ≤ 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
```

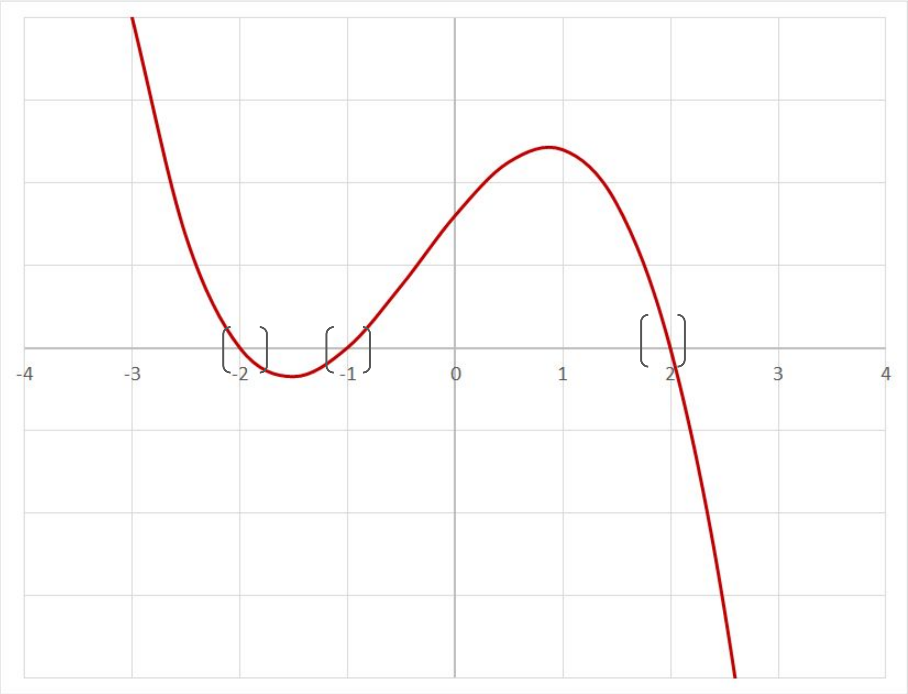
Cota de Fujiwara

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

Algorithm 7 Cota de fujiwara

```
procedure FUJIWARA(poly_array)
    t = lengthpoly_array
    n = t − 1
    d = poly_array[n]
    partial_results = []
    for i ← 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
```

Separação de Raízes

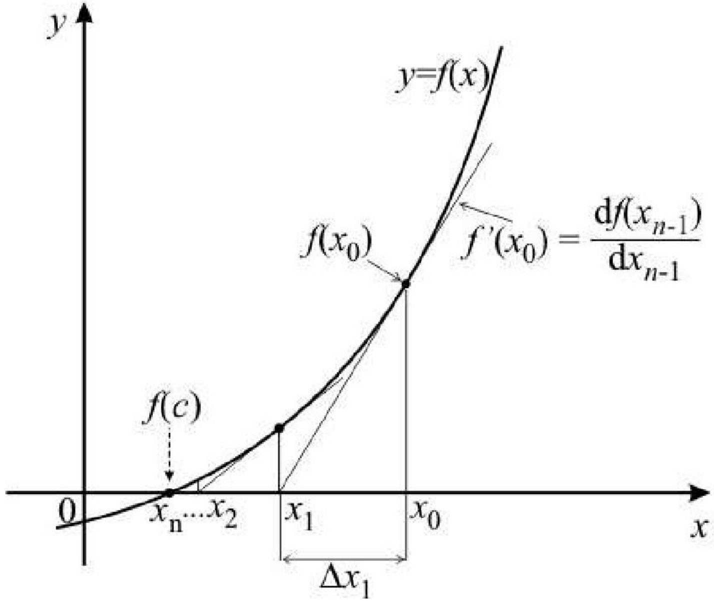


Algorithm 8 Separação de Raízes

```
procedure SEPARAÇÃO(poly_array, bound1, bound2)
    results = []
    step = bound1 * 2/1000
    previous_evall = POLY_EVALUATE(poly_array, bound1)
    if previous_eval == 0 then
        results.append(bound1)
    end if
    for x ← bound1 to bound2, absolute(step) do
        current_eval = POLY_EVALUATE(poly_array, x)
        if (previous_eval > 0 AND current_eval < 0) OR (previous_eval < 0
AND current_eval > 0) then
            results.append(x)
        end if
        if current_eval == 0 then
            results.append(x)
        end if
        previous_eval = current_eval
    end for
    return results
end procedure
```

Método de Newton

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



Algorithm 9 Método de Newton

```
procedure NEWTON(poly_array, derivate_array, x0, eps, max_iter)
    curr_x = x0
    new_x = 0
    iter_count = 0
    while iter_count < max_iter do
        func_eval = POLY_EVALUATE(poly_array, curr_x)
        der_eval = POLY_EVALUATE(derivate_array, curr_x)
        if func_eval == 0 then
            return curr_x
        end if
        if der_eval ≠ 0 then
            new_x = curr_x − (func_eval/der_eval)
            if absolute(new_x − curr_x) ≤ eps then
                return new_x
            end if
        end if
        curr_x = new_x
        iter_count = iter_count + 1
    end while
    return curr_x
end procedure
```

Implementação

Foi criada a ferramenta **Polytool** com o intuito de explorar a manipulação de polinômios, como o cálculo de somas, subtrações, multiplicações, divisões e raízes, assim como o respectivo gráfico do polinômio e o cálculo de pontos.

Frontend

Backend

Outros

Software

https://polytool.herokuapp.com/

Conclusões

Obrigado!

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

PUCRS