Construção de ferramentas para divisão de polinômios, implementadas com o *JavaScript*, no software *GeoGebra*[1]

Building tools for dividing polynomials, implemented with *JavaScript*, in *GeoGebra* software

Manuel Vinicius Ribeiro Lopes Lima[2]
Leandro Barbosa Paz[3]

RESUMO

Este artigo teve como objetivo a criação de ferramentas no software GeoGebra auxiliada do JavaScript, com a finalidade de realizar divisões entre polinômios mediante o dispositivo prático de Briot-Ruffini e o método da chave. Para a construção das ferramentas realizou-se primeiramente um estudo sobre a sintaxe básica do JavaScript e de como ocorre a troca de informações entre ele e o GeoGebra. Alguns processos de divisões foram realizados manualmente afim de analisar os padrões de funcionamento dos dois métodos e entender como o JavaScript poderia ser capaz de simula-los, a seguir o GeoGebra foi utilizado na construção das duas ferramentas, uma para o dispositivo de Briot-Ruffini e outra para o método da chave. A utilização de códigos em LaTeX foi essencial para uma boa exibição da ferramenta na janela de visualização do GeoGebra. As ferramentas se mostraram eficazes em todas as simulações realizadas no final da pesquisa, podendo ser utilizadas por professores e alunos, seja no estudo teórico dos polinômios ou na produção de outras ferramentas similares.

Palavras-chave: *Polinômios; JavaScript; GeoGebra.*

ABSTRACT

This article has the objective of creating tools in the GeoGebra software aided by JavaScript, with the purpose of making divisions between polynomials using the practical Briot-Ruffini device and the key method. For the construction of the tools, a study was first made of the basic syntax of JavaScript and how information exchange between it and GeoGebra occurs. Some processes of division were done manually in order to analyze the working patterns of the two methods and to understand how JavaScript could be able to simulate them, then GeoGebra was used in the construction of the two tools, one for the Briot-Ruffini device and another for the key method. The use of code in LaTeX was essential for a good view of the tool in the GeoGebra viewport. The tools were effective in all the simulations carried out at the end of the research, and could be used by teachers and students, either in the theoretical study of polynomials or in the production of other similar tools.

Key-words: *Polynomials; JavaScript; GeoGebra.*

[1] Apoio: IFCE – Campus Juazeiro do Norte – Edital PROBEPI/2017
[2] Instituto Federal do Ceará – manuelviniciuss@gmail.com
[3] Instituto Federal do Ceará – leandro.juazeiro@gmail.com

Introdução

O *software GeoGebra* é utilizado na maioria das vezes para fins geométricos, como ferramenta facilitadora no ensino de geometria plana e espacial, ou para analisar o comportamento de funções por meio de gráficos. Porém, o *GeoGebra* também dispõe de um recurso de programação pouco explorado e que torna possível a resolução de determinados problemas, além de ampliar os recursos disponíveis para o processo de criação de novas ferramentas.

Problemas matemáticos que necessitam de um certo número de repetições (ou iterações) para serem solucionados podem ser simulados através de códigos *JavaScript* inseridos na guia "Programação" na aba "*JavaScript* Global" do *GeoGebra.* Costuma-se simular repetições no *GeoGebra* com o comando *Sequência[]*, entretanto nesse comando é necessário saber quantas iterações precisariam ser feitas durante o processo, no *JavaScript* isso não é necessário, a condição de parada pode ser definida durante o próprio processo. Além disso, para utilizar o comando *Sequência[]* na geração de recorrências é essencial conhecer o termo geral, fato dispensável em implementações feitas no *JavaScript*.

Tem-se como objetivo principal a construção de ferramentas no *GeoGebra* capazes de calcular e exibir o processo de divisão entre polinômios, tanto pelo método de Briot-Ruffini como também pelo método da chave, por meio da inserção de códigos *JavaScript*.

Para atingir o objetivo principal, alguns objetivos específicos devem ser seguidos, tal como: Analisar como funciona a troca de informações entre *GeoGebra* e *JavaScript*; Estudar as estruturas e funcionalidades dos códigos *JavaScript* e dos códigos *LaTeX*; Entender o processo de divisão entre polinômios por meio do dispositivo de Briot-Ruffini e pelo método da chave; Programar e testar a implementação dos dois métodos de divisão no *GeoGebra*.

Para o desenvolvimento da pesquisa, foi realizado inicialmente um estudo bibliográfico sobre a sintaxe básica do *JavaScript*, dando ênfase nos laços de repetições, abordado em Flanagan (2013); em GeoGebra (2017) é tratado como acontece a troca de informações entre *JavaScript e GeoGebra*; já o estudo sobre a edição de textos em *LaTeX* foi realizado com base em Santos (2012); a conclusão do estudo bibliográfico é concretizado pela abordagem dos métodos de divisões polinomiais em Iezzi (1993).

Após estes estudos preliminares, foi realizado manualmente com caneta e papel algumas simulações de divisões entre polinômios utilizando os dois métodos. Após entender o passo a passo, ficou evidenciado o processo de iteração que posteriormente foi implementado no *JavaScript*. As ferramentas foram construídas

por etapas, resolvendo problemas mais básicos e inserindo novos recursos até se chegar a uma ferramenta completa.

1. O dispositivo prático de Briot-Ruffini

O dispositivo de Briot-Ruffini é um método a qual se calcula a divisão de polinômios de uma variável com grau qualquer por um binômio de mesma variável do dividendo e de grau um, digamos do tipo $ax + b$ onde a e b são coeficiente reais com $a \neq 0$. Em síntese, o dispositivo utiliza apenas os coeficientes do polinômio dividendo e a raiz do polinômio divisor para determinar o quociente e o resto.

Definimos como $f(x)$ e $g(x)$ o polinômio dividendo e divisor respectivamente. Inicialmente é preciso encontrar a raiz do polinômio $g(x)$ e, em seguida, identificar os coeficientes do polinômio $f(x)$, incluindo aqueles que são nulos. Para montar o dispositivo, colocamos a raiz de $g(x)$ à esquerda e todos os coeficientes de $f(x)$ à direita em ordem decrescente do grau dos monômios, além de reescrever o primeiro coeficiente na segunda linha conforme a Figura 1.

O primeiro coeficiente reescrito na linha de baixo será multiplicado pela raiz de $g(x)$ e será somando com o segundo coeficiente de $f(x)$, o valor encontrado será escrito em baixo do segundo coeficiente de $f(x)$. Em seguida, esse valor será multiplicado pela raiz e somado com o terceiro coeficiente e o resultado será escrito na segunda linha em baixo do terceiro coeficiente. O processo será repetido até chegar no último coeficiente do polinômio dividendo, todos os valores obtidos através deste processo serão os coeficientes do quociente da divisão com exceção do último valor, que será o resto.

A Figura 1 abaixo ilustra um exemplo de divisão através do dispositivo de Briot-Ruffini e como funciona o processo de obtenção dos coeficientes do quociente e resto.

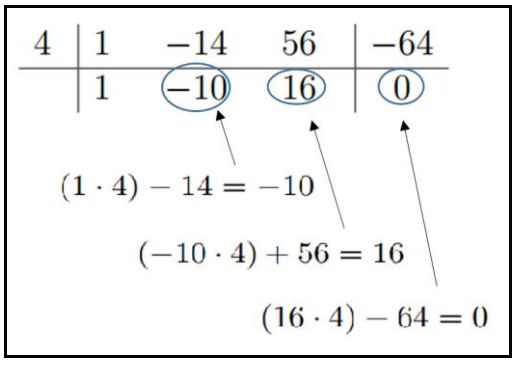

Figura 1. Divisão entre polinômios pelo método de Briot-Ruffini
Fonte: Elaborado pelo autor

No exemplo da figura 1, o polinômio dividendo é $f(x) = x^3 - 14x^2 + 56x - 64$ e o polinômio divisor é $g(x) = x - 4$. O quociente e resto desta divisão são $q(x) = x^2 - 10x + 16$ e $r(x) = 0$ respectivamente.

2. O método da chave

A divisão entre polinômios realizada pelo método da chave, diferentemente do método de Briot-Ruffini, pode ser aplicada entre polinômios de graus quaisquer e é um dos métodos mais utilizados para esse tipo de situação. O processo é semelhante ao método de divisão entre números inteiros e é finalizado quando o resto da divisão for o polinômio nulo ou quando seu grau for menor do que o grau do divisor.

O procedimento para realizar a divisão é separado em algumas etapas que vamos exemplificar a seguir com a divisão de $f(x) = 3x^5 - 6x^4 + 13x^3 - 9x^2 + 11x - 1$ por $g(x) = x^2 - 2x + 3$. Chamaremos o quociente e resto da divisão de q e r respectivamente.

Para encontrar o primeiro termo do quociente efetuamos a divisão do primeiro termo de $f(x)$ pelo primeiro termo de $g(x)$:

$$\frac{3x^5}{x^2} = 3x^3$$

Ao se obter esse resultado, iremos multiplica-lo por cada elemento de $g(x)$, obtendo como resultado o polinômio

$$(3x^3) \cdot g(x) = (3x^3) \cdot (x^2 - 2x + 3) = 3x^5 - 6x^4 + 9x^3$$

O "primeiro resto parcial" da divisão é obtido fazendo a subtração de $f(x)$ pelo resultado obtido na equação acima, que chamaremos de r_1, temos:

$$r_1 = f(x) - (3x^5 - 6x^4 + 9x^3) = 4x^3 - 9x^2 + 11x - 1$$

Observe que r_1 tem grau maior que $g(x)$, isso significa que a divisão ainda não está encerrada. O segundo termo do quociente é obtido da mesma forma como o primeiro, que desta vez será através da divisão do primeiro termo de r_1 pelo primeiro termo do divisor, exemplificado a seguir.

$$\frac{4\mathrm{x}^3}{x^2} = 4\mathrm{x}$$

Da mesma forma como realizado anteriormente, iremos multiplicar $4x$ por $g(x)$, obtendo o seguinte resultado:

$$4x \cdot g(x) = 4x \cdot (x^2 - 2x + 3) = 4x^3 - 8x^2 + 12x$$

Agora, para obter o "segundo resto parcial" r_2, faremos a operação:

$$r_2 = r_1 - 4x \cdot g(x) = (4x^3 - 9x^2 + 11x - 1) - (4x^3 - 8x^2 + 12x)$$
$$= -x^2 - x - 1$$

O grau de r_2 ainda não é menor do que o grau de $g(x)$, realizamos mais uma divisão, obtendo o terceiro termo do quociente. A operação é idêntica à realizada anteriormente. Encontramos o terceiro termo de q através da operação $\frac{-x^2}{x^2} = -1$.

Ao multiplicar -1 por $g(x)$ encontramos como resultado $-x^2 + 2x - 3$, que será usado na obtenção do r_3.

Construímos o "terceiro resto parcial" da mesma maneira como feito anteriormente na obtenção de r_1 e r_2. Assim,

$$r_3 = r_2 - [-1 \cdot g(x)] = (-x^2 - x - 1) - (-x^2 + 2x - 3) = -3x + 2$$

Perceba que r_3 tem grau menor que o divisor $g(x)$, isso indica que a divisão está encerrada com $q = 3x^3 + 4x - 1$ e $r = -3x + 2$.

A representação desse processo pode ser visualizada na Figura 2 abaixo.

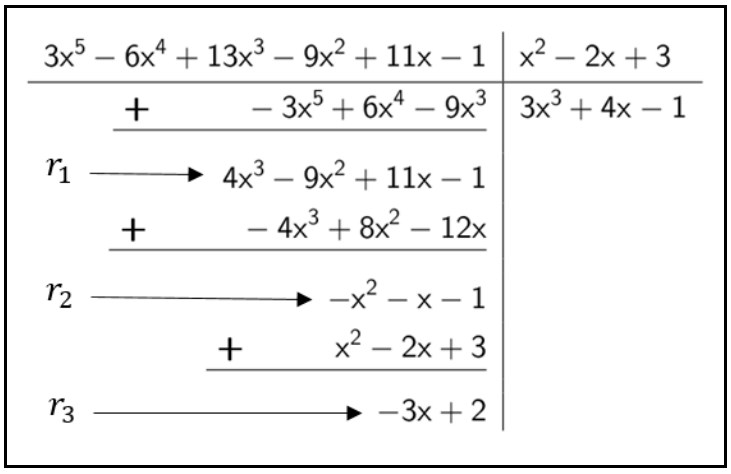

Figura 2. Divisão utilizando o método da chave
Fonte: Elaborada pelo autor

Outra versão da estruturação detalhada desse processo é exibida na imagem a seguir, na qual trabalhamos somente com os coeficientes dos polinômios para a resolução da divisão. Armazenar os coeficientes em vetores facilitou a realização das operações entre polinômios necessárias para a construção da ferramenta.

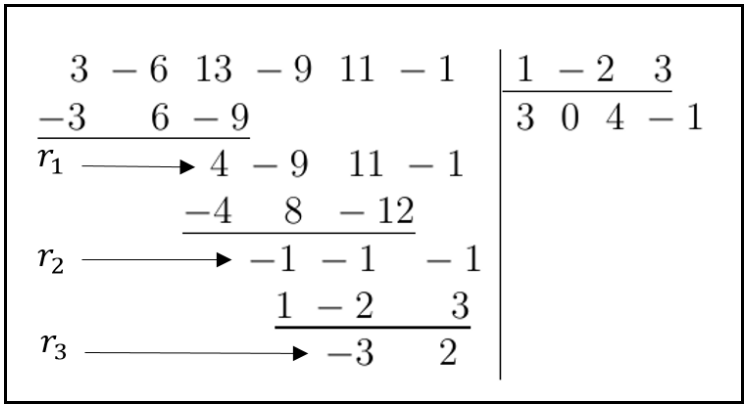

Figura 3. Outra estruturação da divisão utilizando o método da chave
Fonte: Elaborada pelo autor

3. JavaScript

O *JavaScript* é um tipo de linguagem de programação aplicado usualmente no lado cliente de aplicações web evitando chamadas desnecessárias ao servidor, entretanto a utilização do *JavaScript* no *GeoGebra* possui outros fins. O fato do *GeoGebra* aceitar a inclusão de códigos *JavaScript* na aba de programação, permite unir todos os recursos do *GeoGebra* com a possibilidade de implementar novas funções através da linguagem do *JavaScript*. No quadro a seguir encontra-se listado os comandos *JavaScript* utilizados na pesquisa.

DESCRIÇÃO	EXEMPLO
for: Laço de repetição. Pode ser executado por um determinado números de vezes.	`for (i = 0; i < 5; i++) { texto = "O número é " + i; }`
function: É usado para criar funções.	`function produto(x1, x2) { var P = x1 * x2 return P }` \\ A função retorna o produto de x1 com x2
ggbApplet.evalCommand: Comando que funciona no ambiente JavaScript e serve para simular a digitação de comandos no campo de entrada do *Geogebra*.	`ggbApplet.evalCommand("r = FractionText["+3.29+"]")`
ggbApplet.getValue: Transfere valores de objetos do *Geogebra* para variáveis do JavaScript.	`a = ggbApplet.getValue("b") ;` Em que "a" é uma variável do JavaScript e "b" é um objeto do *Geogebra*.
ggbApplet.getValueString: Transfere valores de objetos do *Geogebra* para variáveis textuais do JavaScript.	`l1=ggbApplet.getValueString("lista1") ;` Em que "l1" é uma variável do JavaScript e "lista1" é um objeto do *Geogebra*.

if e **else**: São expressões condicionais. O if é executado se as instruções forem verdadeiras, o else é executado se a condição do *if* for falsa.	`if (hora < 13) { cumprimento = "Bom dia";} else {cumprimento = "Boa tarde";}`
indexOf: Em uma string, ele retorna a primeira ocorrência de um elemento procurado. No array, ele retorna a posição do elemento procurado.	`var str = "Olá mundo"; var n = str.indexOf("l"); O resultado retornado de n é 1.`
length: Retorna a quantidade de caracteres em uma string e o número de elementos em um array.	`var str = "Olá Planeta Terra"; var n = str.length; O resultado retornado de n é 17.`
parseFloat: Converte de string para float, fazendo se necessário alguns ajustes.	`parseFloat("3.14 metros") Retorna 3.14 parseFloat("0.1") Retorna 0.1`
push: Adiciona elementos no final de uma array.	`var frutas=["Banana", "Laranja", "Maçã", "Manga"]; frutas.push(limão); O resultado de frutas será: ["Banana", "Laranja", "Maçã", "Manga", limão]`
return: Define qual será a imagem da função construída.	`function Celsius(fahrenheit) { var c = (5/9) * (fahrenheit-32) return c; } //Retorna o valor da temperatura em Celsius.`
split: Converte uma lista formada por uma única string em um array.	`var str = "Como vai você hoje?"; var res = str.split(" "); O resultado de res é:[Como,vai,você,hoje?]`
substring: Extrai os caracteres de uma string, entre dois índices especificados e retorna a nova substring.	`var str = "Olá mundo!"; var res = str.substring(1, 5); O valor de res é: "lá m"`
while: Laço de repetição. Executa um código enquanto uma determinada condição é verdadeira.	`i=0 while (i < 10) { texto = "O número é " + i; i++;}`

Quadro 1. Comandos ou sintaxe do *JavaScript*

4. Implementação do dispositivo de Briot-Ruffini

O dispositivo, como detalhado anteriormente, serve para realizar divisões entre polinômios com a condição de que o divisor tenha sempre grau um. Para criar um código no *JavaScript* que simulasse esse processo foi utilizado somente os coeficientes do polinômio dividendo e a raiz do polinômio divisor, como proposto pelo dispositivo.

Embora o dispositivo de Briot-Ruffini possa ser implementado usando o comando *Sequência[]* (cujo termo geral pode ser obtido resolvendo a recorrência linear de primeira ordem), sem necessariamente usar *JavaScript*, implementaremos inicialmente este algoritmo por se tratar de um problema de menor complexidade

do que a divisão pelo método da chave. Como o objetivo neste primeiro problema é exemplificar de forma sucinta a utilização do *JavaScript*, não foram adicionados alguns tratamentos de erros com mensagens sobre a exigência do grau do divisor ser um.

Segue passo-a-passo para a construção da ferramenta:

- Digitar no campo de entrada do *GeoGebra* duas funções quaisquer $f(x)$ e $g(x)$, com $g(x)$ sendo um polinômio do tipo $ax + b$;
- Com o comando *Coeficientes[ ]* do *GeoGebra*, construir para cada polinômio uma lista com os seus respectivos coeficientes, denominando-as de *lista1* e *lista2*, a primeira sendo para o dividendo e a segunda para o divisor;
- Na janela de visualização, construir dois campos de entradas, vinculando uma função para cada campo, para que se possa atualizar os polinômios na própria janela de visualização do *GeoGebra*;
- Na aba "*JavaScript* Global" crie a função BriotRuffini (lista1, lista2) digitando todo o código exibido no Quadro 2;
- Para que a função seja chamada ao se atualizar os polinômios nos campos de entrada, abra as propriedades de um dos campos e na aba programação, na opção "ao atualizar", altere o tipo de código para "*JavaScript*" e digite o código:

  lista1=ggbApplet.getValueString("lista1");

  lista2=ggbApplet.getValueString("lista2");

  BriotRuffini(lista1,lista2);

- Repita o processo do item anterior para o segundo campo de entrada.

No código do Quadro 2, nas primeiras linhas, é passado uma estrutura que consegue retirar somente os valores numéricos das listas 1 e 2 e transforma-las em vetores para que assim seja possível realizar operações com os números, pois as mesmas são repassadas inicialmente para o *JavaScript* em uma única variável textual.

A variável "*R*" foi criada com a função de receber o valor da raiz do polinômio divisor. A variável "*j*" recebe o valor do primeiro coeficiente do polinômio dividendo. Com todos esses valores armazenados, já podemos construir um laço de repetição para calcular os valores dos coeficientes do quociente da divisão, conforme consta no Quadro 2.

```
function BriotRuffini(lista1,lista2) {
      asp = String.fromCharCode(34) //Inserir aspas

      lista1=lista1.substring(lista1.indexOf("=")+3,lista1.length()-1)
      //Coeficientes do       polinômio dividendo
      lista1=lista1.split(",")

      lista2=lista2.substring(lista2.indexOf("=")+3,lista2.length()-1)
      //Coeficientes do polinômio divisor
      lista2=lista2.split(",")

      R=(-parseFloat(lista2[1])/parseFloat(lista2[0]))   //Raiz   do   polinômio
      divisor

      inicio="\\begin{array}{c|c"
      q="} "+asp+"FractionText["+R+"]"+asp
      d="\\\\ \\hline & "+asp+"FractionText["+parseFloat(lista1[0])+"]"+asp
      fim="&  \\\\ \\end{array}"

      pol_div= parseFloat(lista1[0]) +"x^"+parseInt(lista1.length-2)

      for(i=0; i<lista1.length; i++) { q=q+" &
      "+asp+"FractionText["+parseFloat(lista1[i])+"]"+asp }

      j=parseFloat(lista1[0])

      //Laço para obtenção dos coeficiente de q
      for(i=1; i<lista1.length; i++){
      k=(j*(R))+parseFloat(lista1[i])

      if(i<lista1.length-1){pol_div=pol_div+"+"+k
      +"x^"+parseInt(lista1.length-i-2)}
      else{
      pol_rest=k
      }

      d=d+" & "+asp+"FractionText["+k+"]"+asp
      j=k
      }
      tabela=inicio+q+d+fim

      //Retorna informações ao GeoGebra
      ggbApplet.evalCommand("tabela=LaTeX["+asp+tabela+asp+"]")
      ggbApplet.evalCommand("q(x)=Simplify["+pol_div+"]")
      ggbApplet.evalCommand("r(x)=FractionText["+k+"]")
}
```

Quadro 2. Código para implementação do dispositivo de Briot-Ruffini

Os textos editorados em *LaTeX* permitem uma boa estruturação visual do processo na janela de visualização do *GeoGebra*. O principal ambiente TeX utilizado foi o array responsável pela construção de tabelas contendo os valores obtidos durante o processo.

Ao tentar utilizar códigos do *LaTeX* no *JavaScript* encontrou-se um tipo de problema com a utilização de barras (\), pois as mesmas servem tanto para iniciar comandos do *JavaScript* como também no *LaTeX*. De acordo com Paz e Alves (2016), para inserirmos uma barra (\) em variáveis textuais do *JavaScript*, teremos que colocar duas barras (\\). Se, por exemplo, fossemos adicionar uma quebra de linha no texto *LaTeX* usamos normalmente duas barras (\\), já no *JavaScript*

usamos quatro barras (\\\\). Esse tipo de observação pode ser visualizada em alguns locais do código inserido no Quadro 2.

A Figura 4 mostra como ficou a ferramenta final exibida na janela se visualização, exibindo o dispositivo, o quociente e resto da divisão.

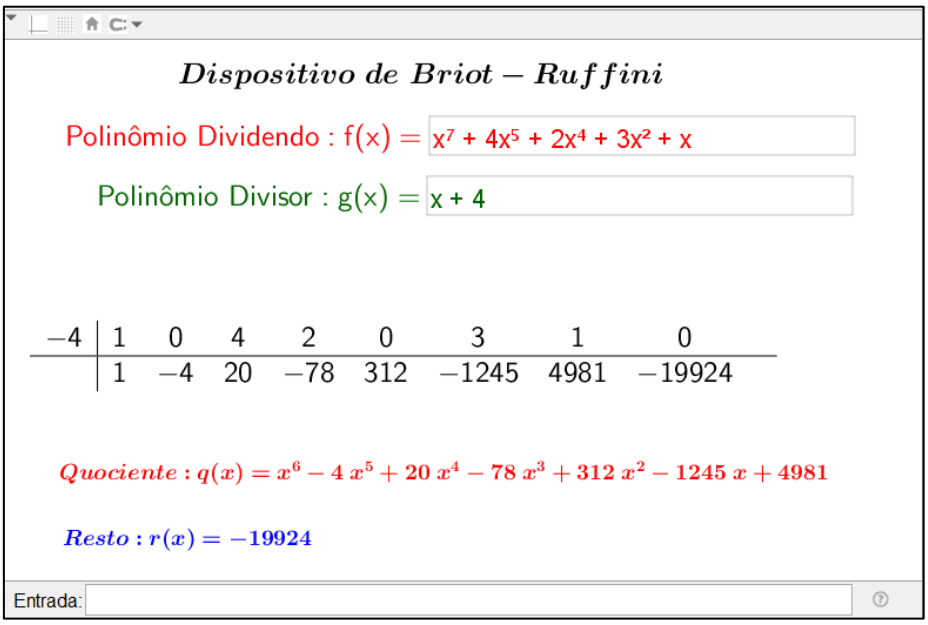

Figura 4. Tela final da ferramenta
Fonte: Elaborada pelo autor e disponível em: https://goo.gl/yKvjVK

5. Implementação do método da chave

Para a criação desta ferramenta, seguimos a mesma ideia de construção da ferramenta anterior, trabalhando somente com os coeficientes dos polinômios. É claro que a operação é totalmente diferente, além da divisão ser realizada entre polinômios quaisquer. Com isso o código se torna um pouco mais complexo e alguns casos particulares exigem uma análise com muita atenção, como por exemplo: ao se obter o primeiro termo do quociente, o primeiro resto parcial poderá diminuir dois graus de uma vez e assim torna o procedimento diferente caso diminuísse somente um grau, pois as operações são trabalhadas com vetores e cada coeficiente dos polinômios deve ser armazenado na posição específica do vetor criado.

Para melhorar a organização o código foi dividido em três funções, cada uma com papéis diferentes. Uma função foi feita para calcular somente o primeiro termo do quociente e o primeiro resto parcial, retornando os valores obtidos em um mesmo vetor. Para determinar os restos parciais juntamente com os termos do quociente da divisão, a primeira função será chamada na segunda, na qual é

repassado os coeficientes dos polinômios digitados no *GeoGebra*. Após determinar o primeiro resto parcial e o primeiro termo do quociente, eles serão separados e armazenados em outros vetores através de um laço de repetição. Será chamado novamente a primeira função para que se possa obter os próximos restos e termos do quociente da divisão a partir dos anteriores. A condição de parada é feita quando o último resto parcial obtiver grau menor do que o grau do divisor ou ele for zero.

A última função construída serve para adicionar as variáveis "x", juntamente com o seu respectivo expoente, aos coeficientes de cada polinômio obtido durante a divisão. Ela também é chamada na segunda função.

Os três primeiros passos para a construção da ferramenta são idênticos ao da ferramenta do dispositivo de Briot-Ruffini construído anteriormente, com a exceção de que a função $g(x)$ poderá ter um grau qualquer, desde que seja menor ou igual ao grau de $f(x)$. Segue o que deve ser feito após esses três primeiros passos:

- Na aba "*JavaScript* Global" crie a função divisao(l1, l2) digitando todo o código exibido no Quadro 3;
- Para que a função seja chamada ao se atualizar os polinômios nos campos de entrada, abra as propriedades da "lista1" de coeficientes e na aba programação, na opção "ao atualizar", altere o tipo de código para "*JavaScript*" e digite o código:

  l1=ggbApplet.getValueString("lista1")

  l2=ggbApplet.getValueString("lista2")

  divisao(l1,l2)

- Repita o processo do item anterior para a segunda lista de coeficientes.

A função que será chamada dentre as três construídas ao se atualizar os polinômios será a "*divisao (l1, l2)*". Segue no quadro abaixo o código para implementação do método da chave para divisão de polinômios.

```
function divisao(l1,l2){
   asp = String.fromCharCode(34)
   dividendo=l1.substring(l1.indexOf("=")+3,l1.length()-1)
   //lista1=Coeficientes do polinomio dividendo
   dividendo=dividendo.split(",")

   divisor=l2.substring(l2.indexOf("=")+3,l2.length()-1) //lista2 =
   Coeficientes do polinômio divisor
   divisor=divisor.split(",")

   poli_dividendo=poli(dividendo)
   poli_divisor=poli(divisor)

   if (divisor[0] !=0){
```

```
if (divisor.length <= dividendo.length){
// inicio da criação do variável temporária
var mono_x_divisor = [];
    for (i=0; i<dividendo.length; i++){ mono_x_divisor.push(0) }

q= dividendo[0]/divisor[0]

for (i=0; i<divisor.length; i++)
    {
    mono_x_divisor[i]=parseFloat(-q)*parseFloat(divisor[i])
    }
mono_x_divisor=poli(mono_x_divisor)

inicio="\\begin{array}{r|l"
linha1="} "+poli_dividendo+" & "+poli_divisor
linha2=" \\\\ \\hline "+"\\underline{"+"\\textbf{+} \\hspace{2.5cm}
"+mono_x_divisor+"}"
linha3=""
fim="\\\\  \\end{array}"

quociente=[]
temp = etapa(dividendo, divisor)

quociente.push(temp[0])// adicionar primeiro caractere de temp

resto_i=[]
for(i=1; i<temp.length; i++){ resto_i.push(temp[i]) } //eliminar o 1°
elemento de temp

// inicio do laço
etapa3=resto_i
j=0

while (j<dividendo.length-divisor.length){

var temp2 = []

for(i=1; i<resto_i.length; i++){ temp2.push(resto_i[i]) } //eliminar o " 0
"


if (temp2[0]!=0){

resto_parcial=poli(temp2)

//Variável temporária
var mono_x_divisor = [];
for (i=0; i<temp2.length; i++){ mono_x_divisor.push(0) }

q= temp2[0]/divisor[0]

for (i=0; i<divisor.length; i++)
    {
    mono_x_divisor[i]=parseFloat(-q)*parseFloat(divisor[i])
    }


var_sem_zero=mono_x_divisor
for( k=0; k<mono_x_divisor.length; k++){
    if (var_sem_zero[0]==0 & var_sem_zero.length>1)
            {
            var_sem_zero =var_sem_zero.slice(1, var_sem_zero.length)
            }
}

var_sem_zero=poli(var_sem_zero)

linha3=linha3+" \\\\  "+" \\  & \\ \\\\"+resto_parcial+" & \\  \\\\
"+"\\underline{"+"\\textbf{+} \\hspace{2.5cm} "+var_sem_zero+"}"+"& \\  "
```

```
    }

    temp=etapa(temp2, divisor)

    quociente.push(temp[0]) //adcionar q


    etapa3=[]

    //Eliminar o 1° elemento de temp novamente
    for(i=1; i<temp.length; i++){ etapa3.push(temp[i]) }

    resto_i=etapa3


    j=j+1

    }
    //fim do laço
    resto_sem_zero=etapa3
    for( k=0; k<etapa3.length; k++){
        if (resto_sem_zero[0]==0 & resto_sem_zero.length>1)
                {
                resto_sem_zero =resto_sem_zero.slice(1, resto_sem_zero.length)
                }
    }
    resto_sem_zero=poli(resto_sem_zero)
    quociente_final = poli(quociente)

    linha3=linha3+"\\\\"+" \\  & \\ \\\\"+resto_sem_zero+" & \\ "
    linha2=linha2+" & "+quociente_final
    tabela=inicio+linha1+linha2+linha3+fim

    ggbApplet.evalCommand("tabela=LaTeX[Simplify["+asp+tabela+asp+"]]")

    }
    else{alert("O grau do polinômio dividendo dever ser maior ou igual ao grau
    do poliômio divisor")}

    }else{alert("Divisor não pode ser o polinômio nulo ")}
}

function etapa(dividendo, divisor){
    var temp = [];

    for (i=0; i<dividendo.length; i++){ temp.push(0) }

    var pre_resto = dividendo

    q= dividendo[0]/divisor[0]

    for (i=0; i<divisor.length; i++)
        {
        temp[i]=parseFloat(-q)*parseFloat(divisor[i])
        pre_resto[i]=parseFloat(dividendo[i])+parseFloat(temp[i])
        }

    var resposta = []
    resposta.push(parseFloat(q))

    for(i=0; i<pre_resto.length; i++){ resposta.push(parseFloat(pre_resto[i]))
    }

    return resposta;

}

    //função que adiciona as variáveis "x"
```

```
function poli(l1){

    if(l1.length>0){

    p=asp+"Simplify["+parseFloat(l1[0]) +"x^"+parseInt(l1.length-1)+"]"+asp

    for(i=1;i<l1.length;i++){
    if(l1[i] != 0){
        p=p+"+"+asp+"Simplify["+parseFloat(l1[i])+"x^"+parseInt(l1.length-i-
    1)+"]"+asp
        }
    }
    } else {p=[]}
    return p

}
```

Quadro 3. Código para implementação do método da chave

A Figura 5 mostra como ficou a ferramenta final no *GeoGebra*.

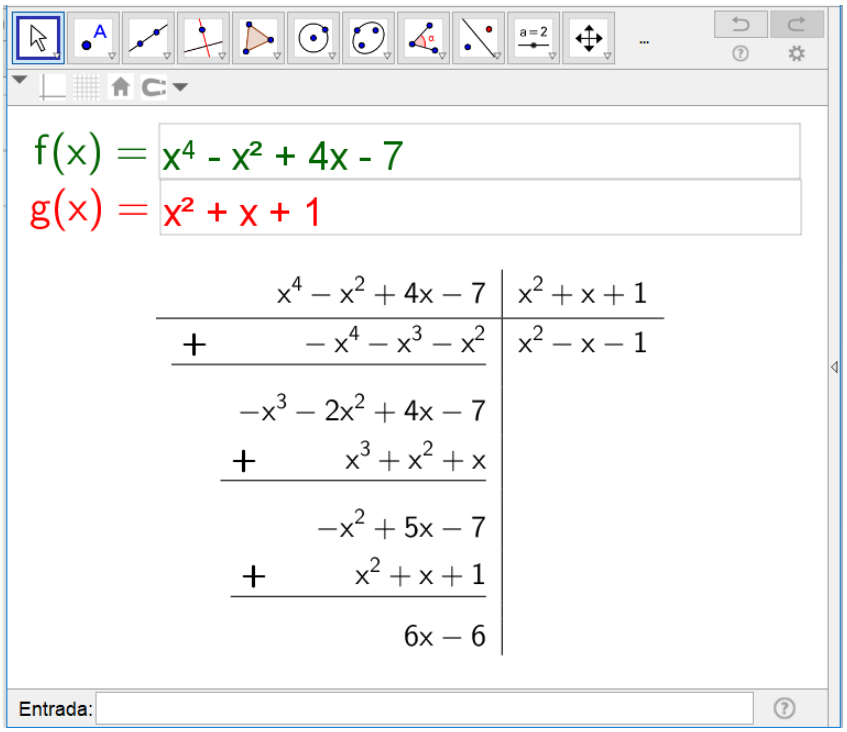

Figura 5. Ferramenta final exibida na janela de visualização do *GeoGebra*.
Fonte: Elaborada pelo autor e disponível em: https://goo.gl/FP4Ffs

Considerações finais

O *GeoGebra* é utilizado comumente para fins geométricos, porém a pesquisa mostrou que ferramentas envolvendo tópicos de álgebra e interações também podem ser construídas no software. A utilização do *JavaScript* dentro do *software GeoGebra* foi um recurso que tornou possível a construção das ferramentas, a

possibilidade de inserção de textos em *LaTeX* foi um ponto essencial que deixou mais harmoniosa a exibição dos resultados processados.

A forma como ambas as ferramentas foram implementadas, só trabalham corretamente se os coeficientes forem números decimais, onde a quantidade de casas decimais é a informada no menu "opções" sub menu "arredondamento". Como pesquisas futuras sugerimos aprimorar a ferramenta para usar o *JavaScript* apenas nas iterações e realizar todas as operações matemáticas com o recurso CAS, possibilitando inclusive a realização de divisão entre polinômios com coeficientes complexos. Desta forma será possível, por exemplo, obter que o resto da divisão de $x^2 + 2$ por $x - \sqrt{2}$ é igual a zero, nas ferramentas construídas e com as configurações padrões do *GeoGebra* é interpretado que $\sqrt{2}$ é $1{,}41$.

As ferramentas construídas podem ser utilizadas por professores e alunos para possíveis consultas sobre resolução de exercícios relacionados a divisões de polinômios. As ideias utilizadas nas implementações podem ser úteis para que outras novas ferramentas possam ser desenvolvidas.

Referências

FLANAGAN, David. *JavaScript:* o guia definitivo. 6ª. ed. Porto Alegre: Bookman, 2013. 1094 p.

GEOGEBRA (2017). *Reference: JavaScript*. Disponível em: <https://wiki.geogebra.org/en/Reference:JavaScript>. Acesso em: 26 set. 2017.

IEZZI, Gelson. *Fundamentos de Matemática Elementar*. 6ª. ed. São Paulo: Atual, 1993. 240 p. v. 6.

PAZ, Leandro Barbosa.; ALVES, F.R.V. *Implementação do Algoritmo de Euclides no Software GeoGebra*. In: I Congresso Brasileiro do Geogebra, 2016, Natal. GeoGebra: Múltiplos olhares para o ensino e a aprendizagem de conceitos, 2016.

SANTOS, R.J (2002). *Introdução ao LaTeX*. Disponível em: <https://regijs.github.io/apostilas.html>. Acesso em: 26 set. 2017.