UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE CIÊNCIAS EXATAS
CURSO DE BACHARELADO EM MATEMÁTICA

Túlio Joaquim Altoé

GRUPOS E CORPOS COM APLICAÇÕES EM GAP

VOLTA REDONDA-RJ
2017

Túlio Joaquim Altoé

GRUPOS E CORPOS COM APLICAÇÕES EM GAP

Trabalho de Conclusão de Curso na área de conhecimento Matemática Pura, apresentado ao Curso de Matemática, ICEx, da Universidade Federal Fluminense, como parte dos requisitos necessários à obtenção do título de Bacharel em Matemática.

Orientadora:
Profa. Dra. Rosemary Miguel Pires

Volta Redonda-RJ
2017

Ficha Catalográfica elaborada pela Biblioteca do Aterrado de Volta Redonda da UFF

A469 Altoé, Túlio Joaquim
Grupos e corpos com aplicações em GAP / Túlio Joaquim Altoé. – 2017.
74 f.

Orientador: Rosemary Miguel Pires
Trabalho de Conclusão de Curso (Bacharelado em Matemática) – Departamento de Matemática, Instituto de Ciências Exatas, Universidade Federal Fluminense, Volta Redonda, 2017.

1. Teoria dos grupos 2. Corpo finito (Álgebra). 3. GAP. I. Universidade Federal Fluminense. II. Pires, Rosemary Miguel, orientador. III. Título.

CDD 512

Túlio Joaquim Altoé

GRUPOS E CORPOS COM APLICAÇÕES EM GAP

Trabalho de Conclusão de Curso na área de conhecimento Matemática Pura, apresentado ao Curso de Matemática, ICEx, da Universidade Federal Fluminense, como parte dos requisitos necessários à obtenção do título de Bacharel em Matemática.

Trabalho aprovado em 09 de janeiro de 2017.

Profa. Dra. Rosemary Miguel Pires
Universidade Federal Fluminense

Prof. Dr. Rodrigo Lucas Rodrigues
Universidade Federal do Ceará

Prof. Ms. André Ebling Brondani
Universidade Federal Fluminense

Volta Redonda-RJ
2017

Este trabalho é dedicado à minha família.

Agradecimentos

Os agradecimentos principais são direcionados aos meus pais, à minha mãe, Maria Bernadethe Lopes Altoé e ao meu pai, José Egidio Altoé pelo apoio e motivação durante a faculdade.

Agradeço aos meus irmãos pela ajuda e incentivo, em especial ao meu irmão José Egidio Altoé Júnior, que participou das correções desta monagrafia.

Agradecimentos especiais são direcionados à minha orientadora professora Rosemary Miguel Pires, pela paciência, orientação e apoio, à Jessica Mitie Kotaira por sua ajuda e participação nas correções deste trabalho e ao Thiago Augusto Lucas da Silva pela colaboração com o Abstract.

Obrigado!

"A Álgebra é generosa:
frequentemente ela dá mais do que se lhe pediu."
(D'ALEMBERT)

Resumo

O trabalho em questão tem como objetivo apresentar alguns resultados clássicos da teoria de grupos e corpos finitos com aplicações no Sistema de Álgebra Computacional GAP. Em particular, trabalhamos com a classificação de grupos de ordem pequena, a menos de isomorfismos, e construção e caracterização de corpos finitos. Classificamos algebricamente os grupos de ordem 4, 6 e 8 e, desenvolvemos com GAP uma aplicação que exibe uma tabela de classificação de todos os grupos de ordem pequena, a menos de isomorfismos. Para construção de corpos finitos, utilizamos o Algoritmo de Rabin, que permite determinar um polinômio irredutível sobre um corpo finito, uma vez que polinômios irredutíveis são de fundamental importância para a construção de corpos. Apresentamos todos os códigos utilizados nas aplicações da teoria, tanto da classificação de grupos e fatoração sobre um domínio de integridade, quanto na construção de corpos.

Palavras-chave: Classificação de grupos. Construção e caracterização de corpos finitos. GAP.

Abstract

The following work has the goal to present some classic results of group theory and finite fields with applications in the computer algebra system GAP. In particular, we work with the classification of groups of small order up to isomorphisms, construction and characterization of finite fields. We classify algebraically groups of order 4, 6 and 8, and we developed using the GAP, an application that exposes the table of classification of groups of small order up to isomorphisms. To construct the finite fields, we use the Rabin's algorithm, it allows to determine a irreducible polynomial over a finite field, since irreducible polynomials are extremely important to the construction of fields. We present all the codes that were used in applications of the theory in both of the cases, in the classification of Groups and factorization over a domain of integrity and in the construction of fields.

Keywords: Classification of groups. Construction and characterization of finite fields. GAP.

Lista de ilustrações

Sumário

1 Introdução

O trabalho proposto consiste no estudo de alguns aspectos computacionais de álgebra, em particular Teoria de Grupos e Teoria de Corpos. A motivação desta pesquisa é devido à vasta aplicação na Matemática, como por exemplo, Criptografia, Teoria de Códigos, Teoria de Números, Geometria Finita e na Álgebra abstrata. Essas aplicações podem ser encontradas em [8].

Devido à magnitude do tema, focamos na classificação de grupos a menos de isomorfismos, fatoração de um polinômio sobre um domínio de integridade e caracterização e construção de corpos finitos. Entretanto, para realização de tal objetivo, torna-se de suma importância o estudo de definições e teoremas que fundamentam o trabalho.

Em cada capítulo, procuramos aplicar computacionalmente no GAP - Groups, Algorithms and Programming, os resultados mais importantes da teoria apresentada.

Sendo assim, o trabalho foi organizado da seguinte maneira:

No Capítulo 1, apresentamos a linguagem de programação que utilizamos para desenvolver os códigos presentes no trabalho. Nele apresentamos a sintaxe, como trabalhar com os recursos oferecidos pela linguagem e como construir funções. Encerramos o capítulo com exemplos da Teoria de Números, como por exemplo, calcular os primos de Wieferich e exemplificar com a conjectura de Collatz.

No Capítulo 2, focamos em teoria de grupos. Apresentamos os conceitos fundamentais para trabalhar com grupos e como representá-los no GAP. Investigamos a recíproca do Teorema de Lagrange e, concluímos que, em geral não é válida sua recíproca. O interessante é que chegamos a essa conclusão através da programação, pois desenvolvemos algoritmos que buscam um contraexemplo que mostra a não validade, em geral, da recíproca do Teorema de Lagrange. Ainda neste capítulo, classificamos algebricamente todos os grupos de ordem 4, 6 e 8. E, por fim, exibimos a tabela de classificação de grupos a menos de isomorfismo, uma aplicação que desenvolvemos com funções do GAP que exibe todos os grupos a menos de isomorfismos, disponibilizamos alguns recursos na aplicação que permite o usuário a trabalhar com um determinado grupo presente na tabela.

No Capítulo 3, introduzimos definições importantes para o Capítulo 4, tais como, anel, domínio e corpo. São apresentados também alguns teoremas importantes utilizados no capítulo 4. Finalizamos esse capítulo com a fatoração de um polinômio $p(x)$ sobre o conjunto dos números inteiros $\mathbb{Z}$, para tal, utilizamos o método de Kronecker.

Por fim, no Capítulo 4, contruímos e caracterizamos corpos finitos. Neste capítulo, também são apresentados alguns resultados importantes, como característica de um Corpo

Finito. Através de dois teoremas conseguimos construir corpos, seja por meio de um polinômio irredutível ou por meio de adjunção de raízes, comparamos também esses dois corpos e vimos que eles são isomorfos. Logo em seguida, caracterizamos os corpos finitos, garantindo assim a existência deles. Encerramos nosso trabalho com o Algoritmo de Rabin, que permite encontrar um polinômio irredutível sobre um corpo finito, pois é através dele que podemos construir novos corpos.

Nos apêndices são apresentados os códigos utilizados, como o Método de Kronecker e o Algoritmo de Rabin.

2 Introdução à Linguagem GAP e Aplicações

Nesta seção são abordados alguns recursos sobre a linguagem de programação científica GAP usada para exemplificar e estudar os demais problemas presentes neste trabalho, como por exemplo o Método de Kronecker para fatoração de um polinômio sobre um Domínio e o Algoritmo de Rabin que permite encontrar um polinômio irredutível sobre um corpo finito. Para o entendimento dos códigos usados, introduziremos a sintaxe de Programação do GAP.

2.1 GAP - Groups, Algorithms and Programming

GAP é um sistema de Álgebra Computacional com uma linguagem de programação científica voltada para o estudo/pesquisa de estruturas algébricas. É possível trabalhar com grupos e suas representações, anéis, espaços vetoriais, estruturas combinátorias, corpos, dentre outras possibilidades.

O sistema e sua fonte são distribuídos gratuitamente e nos permitem estudá-los e estendê-los de acordo com as nossas necessidades, podendo ser encontrado em [3].

2.2 Sintaxe

O GAP como qualquer outra linguagem de programação possui uma referência que permite ao programador conhecer toda sua linguagem, como por exemplo comandos básicos, funções predefinidas, estruturas de testes e repetições, dentre outros recursos. A seguir, serão apresentados os recursos da linguagem GAP, a fim de que o leitor tenha os subsídios necessários para entender os demais algoritmos na presente monografia. Exemplificamos como trabalhar com listas, conjuntos, matrizes, vetores e funções.

Introdução à Linguagem GAP

Como consideração inicial, cumpre dizer que, toda informação que vem depois do operador "#" é tratado como comentário, ou seja, o GAP ignora a informação. Para realizarmos uma atribuição a uma variável, utilizaremos o operador "**:=**". Todo comando que é informado no GAP tem que terminar com "**;**", caso contrário o GAP não interpreta o comando e não será possível prosseguir com o código enquanto o usuário não informar o operador. A cada comando informado ou código inserido o GAP irá informar o que foi interpretado, caso o leitor queira evitar isso, no final de cada comando deve-se utilizar o operador "**;;**".

1. Listas

As listas são usadas quando desejamos trabalhar com uma certa quantidade de elementos, ou ainda, quando queremos guardar inúmeras informações sobre um determinado problema. Para exemplificar, vejamos como construir uma lista de alguns números primos. Também serão apresentadas algumas funções já disponíveis na linguagem do GAP que nos permitem trabalhar com listas.

```
gap> primos:=[2,3,5,7,11,13,17,19,23,29];;
gap> primos;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

A variável "primos" é uma lista que possui alguns números primos. Exbiremos algumas funções que permitem trabalhar com listas:

```
gap> IsList(primos);
true
```

A função IsList("parâmetro") verifica se o que foi passado como parâmetro é uma lista e, se ela existe, retornando verdadeiro ou falso.

```
gap> Append(primos, [31,37]);
gap> primos;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37 ]
```

A função acima adiciona uma pequena lista a uma lista já existente de números primos. Para adicionar somente um valor, utilize a seguinte função:

```
gap> Add(primos, 41);
gap> primos;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41 ]
```

Entretanto, em alguns casos é necessário adicionar um elemento em uma determinada posição na lista, sendo assim, usamos o seguinte comando:

```
gap> Add(primos, 47, 15);
gap> primos;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,, 47 ]
```

A função Add(lista, num, pos) adiciona um novo elemento na sua lista já existente, onde "lista" é sua lista, "num" é o elemento que deseja adicionar e "pos" é a posição que você deseja inserir seu elemento.

```
gap> primos:=[2,3,5,7,11,13,17,19,23,29];;
gap> primos;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 ]
```

Para acessar os elementos da sua lista de números primos faça o seguinte:

```
gap> primos[3];
5
```

Dessa forma, podemos acessar os elementos da lista "lista[pos]" apenas colocando a posição que deseja em "pos". Além disso, podemos atribuir valores como no exemplo abaixo.

```
gap> primos[11]:= 31;
31
gap> primos;
[ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31 ]
```

A função Length("parâmetro") mostra o tamanho da lista.

```
gap> Length(primos);
11
```

Já a função Position(lista, num) retorna a posição de um elemento pertencente a lista, onde "num" é o elemento que deseja saber a posição e "lista" é a lista onde será feito a busca, caso o elemento não esteja na lista a função retorna "fail".

```
gap> Position(primos, 3);
2
gap> Position(primos, 4);
fail
gap> Position(primos, 29);
10
```

2. Conjuntos

Conjuntos são listas especiais, sem repetições e sem buracos. Vamos definir um conjunto e em seguida testaremos algumas funções do GAP.

```
gap> beta:=[''c'',''b'',''a''];
[ ''c'', ''b'', ''a'' ]
gap> IsSet(beta);
false
```

A função acima verifica se o conjunto está ordenado. Para ordenar o seu conjunto basta realizar as seguintes operações:

```
 gap> beta;
[ ''c'', ''b'', ''a'' ]
gap> beta:= Set(beta);
[ ''a'', ''b'', ''c'' ]
```

Quando conveniente, precisamos testar se dado um elemento "a" ele pertence ao conjunto beta ou não. O operador "in" testa se o elemento está no conjunto.

```
gap> ''a'' in beta;
true
gap> ''d'' in beta;
false
```

Para adicionar um elemento, faça:

```
gap> AddSet(beta, ''d'');
gap> beta;
[ ''a'', ''b'', ''c'', ''d'' ]
```

A função acima já insere de forma ordenada. Se adicionarmos um elemento já existente, não alteraremos o conjunto. Vejamos um exemplo:

```
gap> AddSet(beta, ''d''); beta;
[ ''a'', ''b'', ''c'', ''d'' ]
```

Vamos atribuir elementos a outro conjunto alfa para exemplificar união e interseção de conjuntos:

```
gap> alfa:=[''d'', ''e'', ''f'', ''g''];
[ ''d'', ''e'', ''f'', ''g'' ]
gap> alfa; beta;
[ ''d'', ''e'', ''f'', ''g'' ]
[ ''a'', ''b'', ''c'', ''d'' ]
gap> Intersection(alfa, beta); Union(alfa, beta);
[ ''d'' ]
[ ''a'', ''b'', ''c'', ''d'', ''e'', ''f'', ''g'' ]
```

3. Vetores e Matrizes

O que utilizamos em lista se aplicam aos vetores. Nesta seção, vamos conhecer algumas funções que nos permitem trabalhar com vetores e matrizes.

```
gap> s:=[''3'', ''2'', ''1''];
[ ''3'', ''2'', ''1'' ]
gap> IsVector(s);
false
```

A função IsVector("parâmetro") verifica se o que foi passado como parâmetro é um vetor ou não.

```
gap> v:=[1,2,3];
```

```
[ 1, 2, 3 ]
gap> IsVector(v);
true
```

Definindo uma matriz:

```
gap> mat:=[[1,0,0],[0,1,0],[0,0,1]];
[ [ 1, 0, 0 ], [ 0, 1, 0 ], [ 0, 0, 1 ] ]
gap> PrintArray(mat);
[ [  1,  0,  0 ],
  [  0,  1,  0 ],
  [  0,  0,  1 ] ]
```

a função acima imprime no formato de matriz. Para acessar uma determinada linha da matriz, faça:

```
gap> mat[2];
[ 0, 1, 0 ]
```

Mas, se quiser explicitar uma determinada entrada da matriz, basta fazer o seguinte:

```
gap> mat[2][2];
1
```

E assim, terá o elemento na posição correspondente da matriz.

4. Comandos de Testes e Repetições

Os comandos de testes e repetições servem para repetir blocos de códigos evitando que o usuário tenha que digitá-los sempre, sendo muito utilizados para percorrer uma lista, testar condições para tormar determinadas ações, imprimir informações na tela, dentre outras possibilidades.

Para usar esses recursos basta seguir a sintaxe:

'For':	**'While':**
for variável in lista do	while Condições do
sentenças	sentenças
od;	od;
'If':	**'Repeat':**
if Condições then	repeat
afirmação ou sentença	instruções
fi;	until condição;

Seguem alguns exemplos da sintaxe acima:

Exemplo 1: Vamos usar a estrutura de repetição "for" para imprimir os elementos de uma determinada lista de números primos.

```
gap> primo:=[2,3,5,7,11,13];
[ 2, 3, 5, 7, 11, 13 ]
gap> for x in primo do
>       Print(x,''\n'');
> od;
2
3
5
7
11
13
```

Agora, para exemplificar a sintaxe do "for" e o "if", vamos imprimir os números pares de 1 a 10:

```
gap> for i in [1..10] do
>       if i mod 2 = 0 then
>               Print(i,''\n'');
>       fi;
> od;
2
4
6
8
10
```

No segundo exemplo, queremos imprimir uma mensagem usando o comando de repetição "While":

```
gap> x:=1;
1
gap> p:=6;
6
gap> while x<p do
> Print(''testando while\n'');
> x:=x+1;
> od;
testando while
testando while
```

```
testando while
testando while
testando while
```

A estrutura de repetição “repeat” é semelhante ao “while” só que executa a condição pelo menos uma vez antes de fazer o teste.

5. Funções

Funções são pedaços de códigos encapsulados que possibilitam realizar determinadas tarefas. O uso delas no GAP torna o trabalho/estudo do usuário mais eficiente e célere, pois evita que tenha que digitar blocos de códigos várias vezes.

Utilizamos a seguinte sintaxe para o uso de funções:

NomeDaFuncao:= function(parâmetros)

local variáveis locais;

instruções

end;

Podemos escrever nossa função em qualquer editor de texto e depois fazemos a leitura dela no GAP, possibilitando que o usuário use-a quantas vezes achar necessário.

Exemplo 2: Vamos criar uma função para retornar o fatorial de um determinado número.

Algoritmo 1: Fatorial

```
1 fac := function(n)
2         local i, f;
3         f:=1;
4         for i in [1 .. n] do
5                 f:=f*i;
6         od;
7         return f;
8 end;
```

Agora usaremos nossa função “fac(n)” para achar o fatorial de determinados números:

Primeiramente vamos efetuar a leitura da função que escrevemos, usando a seguinte função, passando o diretório de onde se encontra seu arquivo:

```
gap> Read(''/home/tulioaltoe/Documentos/Gap/funcoes/fac.g'');
```

A função "Read(diretório/nomedoarquivo.g);" buscará no diretório informado o algoritmo que escrevemos.

Vamos determinar o fatorial de alguns números usando a função que escrevemos:

```
gap> fac(5);
120
gap> fac(6);
720
```

Com o que foi proposto até então, já é suficiente para que se entenda os algoritmos presentes nesta monografia. Entretanto, para qualquer dúvida que por ventura surgir, é possível consultar o manual de referência[1] do GAP.

2.3 Aplicações do GAP na Teoria de Números

A título de ilustração da linguagem de Programação do GAP, nesta seção, trabalharemos com a Teoria de Números, utilizando o GAP para estudar os Primos de Wieferich e a famosa Conjectura de Collatz, onde recorremos ao auxílio computacional para elucidar seus resultados. Essas aplicações foram baseadas em [4].

Começaremos definindo e determinando os primos de Wieferich e na seção seguinte vamos investigar a conjectura de Collatz.

2.3.1 Primos de Wieferich

Os únicos primos de Wieferich conhecidos são 1093 e 3511. Foram encontrados por W. Meissner em 1913 e N.G.W.H. Beeger em 1922, respectivamente, e, caso existam outros, devem ser maiores que $1,25.10^{15}$, conforme [11].

O pequeno Teorema de Fermat afirma que: se p é um número primo, então, p divide $2^{p-1}-1$, assim, chegamos à definição do primo de Wieferich:

Definição 1 (Primo de Wieferich)**.** *Dizemos que um número p é primo de Wieferich se p^2 divide $2^{p-1}-1$.*

Com base no Teorema de Fermat e da definição de Primo de Wieferich, desenvolvemos um algoritmo no GAP para encontrar os Primos de Wieferich já conhecidos.

Para verificar que um dado primo p, ele é de Wieferich ou não, a função retorna verdadeiro se dado um número "n", passado por parâmetro, é primo de Wieferich e falso caso contrário. Segue a função:

[1] Disponível neste endereço: http://www.gap-system.org/Manuals/doc/ref/chap0.html

Algoritmo 2: Wieferich

```
1  Wieferich:= function(n)
2          local x,y;
3          if IsPrimeInt(n)= true then
4                  x:=(2^(n-1)) - 1;
5                  y:=n^2;
6                  if x mod y = 0 then
7                          return true;
8                  fi;
9          fi;
10         return false;
11 end;
```

Como já mencionado, apenas 2 primos de Wieferich são conhecidos, logo, com o auxílio da função "Wieferich", podemos encontrá-los:

Para achar os dois números de Wieferich, precisamos fazer a leitura da nossa função:

```
gap> Read("/home/tulioaltoe/Gap/Wieferich.g");
```

Precisamos de uma lista de primos, para testá-los e ver qual é primo de Wieferich.

```
gap> lista:= Filtered([1..6000], IsPrime);;
gap> cont:=0;
0
gap> while cont <= 2 do
> for i in lista do
> y:=Wieferich(i);
> if y = true then
> Print(i,"\n");
> cont:=cont+1;
> fi;
> od;
> od;
1093
3511
```

Assim, encontramos os dois primos conhecidos de Wieferich, ou seja, 1093 e 3511.

2.3.2 Conjectura de Collatz

Esta Conjectura nos diz o seguinte:

Conjectura 1 (Collatz)**.** *Dado um inteiro n positivo e definimos recursivamente uma sequência a_i da seguinte forma: Seja $a_1 = n$, e dado a_i, definimos a_{i+1} como $a_{i+1} = a_i/2$, se a_i for par; e $a_{i+1} = 3a_i + 1$ caso contrário. A conjetura de Collatz afirma que dado um inteiro positivo n qualquer, o número 1 sempre vai aparecer na sequência.*

A Conjectura de Collatz recebeu este nome em homenagem ao matemático alemão Lothar Collatz, pois foi o primeiro a propô-la, em 1937. Para mais detalhes desta conjectura consultar [9].

Para estudar a conjectura de Collatz, desenvolvemos alguns algoritmos no GAP, sendo que a primeira delas foi a função "ProxNum(ai)" que devolvea_{i+1} dado a_i, ou seja, retorna o próximo número da sequência.

Algoritmo 3: ProxNum

```
1  ProxNum:= function(ai)
2          local prox;
3          prox:=0;
4          if ai mod 2 = 0 then
5                  prox:= ai/2;
6                  return prox;
7          fi;
8          if ai mod 2 <> 0 then
9                  prox:=3*ai + 1;
10                 return prox;
11         fi;
12 end;
```

A função ProxNum(n) recebe como parâmetro o número inteiro "n", testa se ele é par ou ímpar e retorna o próximo número da sequência respeitando a condição da conjectura.

Em seguida, a função que calcula o número de passos que precisamos para chegar ao número 1.

Algoritmo 4: NumPassos

```
1  NumPassos:= function(n)
2          local cont;
3          cont:=0;
4          while n <> 1 do
```

```
5                   n:= ProxNum(n);
6                   cont:=cont +1;
7           od;
8           return cont;
9  end;
```

Para exemplificar, considerando uma lista de números inteiros de 1 até 20, vamos testar cada número desta lista e ver a quantidade de passos que cada número levou para atingir o número 1.

Primeiramente, precisamos realizar a leitura das nossas funções.

```
gap> Read("/home/tulioaltoe/Gap/funcoes/ProxNum.g");
gap> Read("/home/tulioaltoe/Gap/funcoes/NumPassos.g");

gap> x:=[1..20]; # lista de 1 ate 20.
[ 1 .. 20 ]
```

Vamos criar uma nova lista 't' com a quantidade de passos de cada número da lista 'x'.

```
gap> t:= List(x, y->NumPassos(y));
[ 0, 1, 7, 2, 5, 8, 16, 3, 19, 6, 14, 9, 9, 17, 17, 4, 12, 20, 20, 7 ]
```

Agora vamos achar os números com maior quantidade de passos.

```
gap> Maximum(t);
20
gap> Positions(t,20);
[ 18, 19 ]
```

Assim os números que precisam do maior número de passos para atingir o número 1 são: 18 e 19.

Uma proposta da demonstração desta conjectura foi sugerida pelo matemático alemão Gerhard Opfer em maio de 2011, disponível em [9]. Entretanto, em 17 de julho de 2011, o autor publicou uma nota em seu artigo, onde reconhecia que haviam inconsistências em suas afirmações, e que, não garantia a prova da conjectura.

3 Grupos

Neste capítulo, inicialmente, apresentamos uma introdução à Teoria de Grupos com a finalidade de classificar algebricamente alguns grupos de ordem pequena. Por fim, apresentamos uma aplicação que desenvolvemos usando o GAP para o estudo da classificação de Grupos. Em particular, apresentamos uma tabela de classificação de alguns grupos de ordem pequena gerada a partir das funções pré-definidas do GAP.

3.1 Preliminares da Teoria de Grupos

Seja G um conjunto não vazio onde está definida uma operação entre dois elementos x e y de G, denotada por:

$$\begin{aligned} * : G \times G &\to G \\ (x, y) &\mapsto x * y \end{aligned}$$

Dizemos que o par $(G, *)$ é um **grupo** se são válidas as seguintes propriedades:

1. $a * (b * c) = (a * b) * c$, para todo $a, b, c \in G$. (associatividade)
2. Existe $e \in G$ tal que $a * e = e * a = a$, para todo $a \in G$. (elemento neutro)
3. Para todo $a \in G$, existe $b \in G$ tal que $a * b = b * a = e$. (elemento inverso)

   Se em um grupo $(G, *)$ verifica-se a propriedade:

4. $a * b = b * a$, para todo $a, b \in G$ dizemos que o grupo $(G, *)$ é um **grupo abeliano**.

Usamos a seguinte notação para representar o produto $a * b$: $a.b$ ou ab.

Dado um grupo $(G, *)$ e H um subconjunto de G. Dizemos que H é um **subgrupo** de G se:

1. $e \in H$.
2. $\forall\, a, b \in H$ tem-se $ab \in H$.
3. $\forall\, a \in H$ tem-se $a^{-1} \in H$.

Se G é um grupo finito, a **ordem** de G, denotada por $|G|$, é a cardinalidade de G.

Vejamos agora alguns exemplos de Grupos:

Exemplo 1: $\mathbb{Z}$ é um grupo aditivo com infinitos elementos.

Exemplo 2: Dado n≥1, tal que $n \in \mathbb{N}$ então o conjunto $\mathbb{Z}_n$ dos inteiros módulo n, munido das operações:

$$\begin{aligned} + : \mathbb{Z}_n \times \mathbb{Z}_n &\to \mathbb{Z}_n \qquad & \cdot : \mathbb{Z}_n \times \mathbb{Z}_n &\to \mathbb{Z}_n \\ (x, y) &\mapsto \overline{x + y} & (x, y) &\mapsto \overline{x \cdot y} \end{aligned}$$

é um grupo aditivo contendo n elementos.

Exemplo 3: Considere agora um conjunto não vazio S e seja:

$$G = \{f : S \to S; f \text{ é } bijetora\}.$$

A operação que iremos definir para esse conjunto será a composição de funções, ou seja,

$$\begin{aligned} * : G \times G &\to G \\ (g, f) &\mapsto g \circ f \end{aligned}$$

Temos que $(G, *)$ é um grupo, onde seu elemento neutro é:

$$\begin{aligned} I_S : S &\to S \\ x &\mapsto x \end{aligned}$$

Esse grupo que acabamos de definir é chamado grupo das permutações do conjunto S. Se $S = \{1, 2, ..., n\}$, denotamos o grupo formado a partir de S por S_n. . O grupo S_n possui exatamente $n!$ elementos.

Usualmente denotamos os elementos do grupo S_n da seguinte forma:

$$f = \begin{pmatrix} 1 & 2 & ... & n \\ f(1) & f(2) & ... & f(n) \end{pmatrix}$$

O GAP faz uso da notação cíclica dos elementos desse conjunto. Para entender essa notação vamos considerar o seguinte exemplo:

$$f = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix}$$

Temos que f(1) = 2, f(2) = 3 e f(3) = 1, portanto, com a notação cíclica temos (1 2 3), onde o elemento seguinte é a imagem do anterior. Assim o grupo S_3 é formado pelos seguintes elementos:

$$e = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} = (1) \qquad f_3 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} = (1\ 3)$$

$$f_1 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} = (1\ 2) \qquad f_4 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} = (1\ 2\ 3)$$

$$f_2 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} = (2\ 3) \qquad f_5 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix} = (1\ 3\ 2)$$

Exemplo 4: Para cada $n \geq 3$, existe um grupo não abeliano com $2n$ elementos conhecido como grupo Diedral e denotado por D_n.

D_n é subgrupo do S_n e é formado pelas simetrias de um polígono regular de n lados.

Para ver sua construção, consultar [2].

Nesta monografia investigamos a não validade em geral da recíproca do Teorema de Lagrange, o que nos faz necessário tomar conhecimento deste resultado.

Teorema 2 (Lagrange)**.** *Se G é um grupo finito e H é um subgrupo de G, então $|H|$ é um divisor de $|G|$ (isto é, a ordem de H é um divisor da ordem de G).*

Demonstração: Disponível em [1].

Para aplicação que desenvolvemos com o GAP, é necessário definir subgrupo normal, p-subgrupos de Sylow e enunciar os teoremas de Sylow.

Definição 3 (Subgrupo Normal)**.** *Seja G um grupo e seja H um subgrupo de G. Se $g \in G$ definimos a função ψ_g (conjugação pelo elemento $g \in G$) por*

$$\begin{aligned} \psi_g : G &\longrightarrow G \\ x &\longmapsto \psi_g(x) = x^g = g^{-1}xg \end{aligned}$$

Observe que $\psi_g(H) = \{\psi_g(h) : h \in H\} = \{h^g = g^{-1}hg : h \in H\}$ que vamos denotar por H^g ou $g^{-1}Hg$. H^g é subgrupo de G, pois

1. $e = e^g \in H^g$
2. $h_1^g,\ h_2^g \in H^g \Rightarrow h_1^g \cdot h_2^g = (h_1h_2)^g \in H^g$
3. $h^g \in H^g \Rightarrow (h^g)^{-1} = (h^{-1})^g \in H^g$.

Dizemos que um subgrupo $H \leq G$ é normal em G se $\psi_g(H) = H^g \subseteq H$ para todo $g \in G$

Notação: $H \triangleleft G$.

Definiremos a noção de Homomorfismo e Isomorfismos de Grupos, para tal, considere a seguinte definição:

Definição 4. *Sejam G e G' dois grupos e $\varphi : G \to G'$ uma função de G em G'. Dizemos que φ é um homomorfismo de grupos se $\varphi(x.y) = \varphi(x).\varphi(y)\ \forall\, x, y \in G$. Quando $\varphi : G \to G'$ for bijetiva dizemos que φ é um isomorfismo de Grupos e neste caso dizemos que G é Isomorfo a G' e denotamos $G \simeq G'$.*

Vamos fixar uma notação, se S é um subconjunto não vazio do grupo G, o conjunto $\{a_1a_2...a_n | n \in \mathbb{N}, a_i \in S\}$ será denotado por $< S >$.

Exemplo 5: Primeiramente, definiremos Grupo Cíclico C_n. Seja C_n um grupo, dizemos que C_n é cíclico quando ele pode ser gerado por apenas um elemento, ou seja, $C_n =< a >$, onde $a^n = e$, para algum $a \in G$. Desta forma, os elementos deste grupo são potências de a.

$$e, a, a^2, ..., a^{n-1}$$

A operação entre seus elementos é o produto definido da seguinte maneira:

$$a^i.a^j = \begin{cases} a^{i+j} & se\, i+j < n \\ a^{i+j-n} & se\, i+j \geq n \end{cases}$$

Nesta monografia já definimos o grupo dos Inteiros módulo n $(\mathbb{Z}_n, +)$.Considerando a seguite função:

$$\begin{aligned} \psi : \mathbb{Z}_n &\to C_n \\ \bar{i} &\mapsto a^i \end{aligned}$$

temos que ψ é um Isomorfismo de Grupos, e então, $\mathbb{Z}_n \simeq C_n$.

Usualmente o GAP trabalha com o grupo cíclico C_n, mas podemos relacionar facilmente suas aplicações com $\mathbb{Z}_n$ devido a este isomorfismo.

Agora, veremos um resultado muito importante na Álgebra presente na Classificação de Grupos de Ordem Pequena.

Teorema 5 (1º Teorema de Sylow)**.** *Seja G um grupo finito de ordem $p^m b$ com p primo e $mdc\{p, b\} = 1$. Então, para cada n, $0 \leq n \leq m$, existe um subgrupo H de G tal que $|H| = p^n$.*

Demonstração: Ver [1].

Definição 6 (p-subgrupos de Sylow)**.** *Seja G um grupo finito, p um primo e p^m a maior potência de p que divide $|G|$. Os subgrupos de G que têm ordem p^m, são chamados p-subgrupos de Sylow de G.*

3.2 Apresentação no GAP dos Grupos de Permutações S_n e Diedral D_n

O GAP já fornece uma biblioteca com alguns grupos existentes. Na Teoria de Grupos os mais usados em exemplos e fixação da teoria são: Grupo Diedral e o Grupo das Permutações. A linguagem de programação do GAP possui algumas funções para trabalhar com determinados grupos, mostraremos algumas no exemplo que se segue:

Começaremos com o Grupo Diedral de ordem 12. Para tal, vamos atribuir a variável G1 o grupo Diedral de ordem 12, utilizando a seguinte função:

```
gap> G1:= SmallGroup(12,4);
<pc group of size 12 with 3 generators>
```

A função acima acessa uma biblioteca com alguns Grupos já estabelecidos. A próxima função nos dá a descrição do grupo G1.

```
gap> StructureDescription(G1);
"D12"
```

Agora, vamos acessar o Grupo das Permutações de ordem 24.

```
gap> G2:= SmallGroup(24,12);
<pc group of size 24 with 4 generators>
gap> StructureDescription(G2);
"S4"
```

Uma vez que já realizamos as atribuições necessárias, já podemos trabalhar com os demais grupos escolhidos, como por exemplo, listar seus elementos;

```
gap> List(G1); # Listando os elementos do Grupo D12.
[ <identity> of ..., f3, f3^2, f2, f2*f3, f2*f3^2, f1, f1*f3,
f1*f3^2, f1*f2, f1*f2*f3, f1*f2*f3^2 ]
gap> List(G2); # Listando os elementos do Grupo S4.
[ <identity> of ..., f4, f3, f3*f4, f2, f2*f4, f2*f3, f2*f3*f4,
f2^2, f2^2*f4, f2^2*f3, f2^2*f3*f4, f1, f1*f4, f1*f3, f1*f3*f4, f1*f2,
f1*f2*f4, f1*f2*f3, f1*f2*f3*f4, f1*f2^2, f1*f2^2*f4, f1*f2^2*f3,
f1*f2^2*f3*f4 ]
```

Podemos também calcular seus subgrupos, bastando utilizar a seguinte função:

```
gap> Subgroups(G1); # Calculando os subgrupos de D12.
[ Group([ ]), Group([ f1 ]), Group([ f1*f3^2 ]), Group([ f1*f3 ]),
Group([ f2 ]), Group([ f1*f2 ]), Group([ f1*f2*f3^2 ]),
Group([ f1*f2*f3 ]), Group([ f3 ]), Group([ f1, f2 ]),
Group([ f1*f3^2, f2 ]), Group([ f1*f3, f2 ]),
Group([ f3, f1 ]), Group([ f3, f2 ]), Group([ f3, f1*f2 ]),
Group([ f3, f1, f2 ])]
gap> Subgroups(G2); # Calculando os subgrupos de S4.
[ Group([ ]), Group([ f4 ]), Group([ f3*f4 ]), Group([ f3 ]),
Group([ f1 ]), Group([ f1*f2^2 ]), Group([ f1*f3*f4 ]),
Group([ f1*f2 ]), Group([ f1*f2*f4 ]), Group([ f1*f2^2*f3 ]),
Group([ f2 ]), Group([ f2*f3*f4 ]), Group([ f2*f3 ]),
Group([ f2*f4 ]), Group([ f3, f4 ]), Group([ f1, f3*f4 ]),
Group([ f1*f2^2, f3 ]), Group([ f1*f2, f4 ]), Group([ f1*f4, f3*f4 ]),
Group([ f1*f2^2*f3*f4, f3 ]), Group([ f1*f2*f3, f4 ]),
Group([ f1, f2 ]), Group([ f1*f3*f4, f2*f3*f4 ]),
Group([ f1*f3*f4, f2*f3 ]), Group([ f1, f2*f4 ]), Group([ f3, f4, f1 ]),
Group([ f1*f2^2, f3*f4, f4 ]), Group([ f1*f2, f3, f4 ]),
Group([ f3, f4, f2 ]), Group([ f3, f4, f2, f1 ]) ]
```

A justificativa dessa seção foi para ter conhecimento de como trabalhar com Grupos no GAP, pois a partir de agora usaremos muitas das funções e Grupos citados aqui.

3.3 Teorema de Lagrange

Sabemos que o Teorema de Lagrange é muito importante na Álgebra. Foi observado que, em geral, não vale a recíproca deste Teorema. A recíproca seria: Se G é um grupo finito, então para cada divisor "d" da ordem de G, existe um subgrupo H de G tal que $|H| = d$.

Para mostrar que a afirmação acima, em geral, não é verdadeira, criamos duas funções, a primeira função retorna uma lista de ordens dos Subgrupos de um determinado grupo G, já a segunda retorna verdadeiro se, para todo divisor d de $|G|$, G possui um subgrupo de ordem 'd', caso contrário retorna falso.

Primeira função:

Algoritmo 5: Ordens

```
1
2 Ordens:= function(g)
3         local subgruposdeg, classesdeconj,
```

```
4               listadosrepresentantes,listadasordens;
5           subgruposdeg:=LatticeSubgroups(g);
6           classesdeconj:=ConjugacyClassesSubgroups(subgruposdeg);
7           listadosrepresentantes:=List(classesdeconj,
8               x->Representative(x));
9           listadasordens:=List(listadosrepresentantes, y-> Order(y));
10          return listadasordens;
11  end;
```

Segunda função:

Algoritmo 6:ContemSubgrupoTodaOrdem

```
1   ContemSubgrupoTodaOrdem:= function(G)
2           local ordemg, listadivisores,listadeordenssubgrupos,
3                   listaordenada,tamanho,x,cont;
4           cont:=0;
5           ordemg:= Order(G);
6           listadivisores:= DivisorsInt(ordemg);
7           listadeordenssubgrupos:= Ordens(G);
8           listaordenada:= Set(listadeordenssubgrupos);
9           for x in listadivisores do
10                  if x in listaordenada then
11                          cont:= cont+1;
12                  fi;
13          od;
14          tamanho:= Length(listadivisores);
15          if tamanho=cont then
16                  return true;
17          fi;
18  return false;
19  end;
```

Vamos investigar a recíproca do Teorema de Lagrange com o seguinte algoritmo. Primeiramente, vamos realizar a leitura das funções que escrevemos acima:

```
gap> Read("/home/tulio/Gap/Ordens.g");
gap> Read("/home/tulio/Gap/ContemSubgrupoTodaOrdem.g");
```

Para mostrar a não validade, em geral, da recíproca do Teorema de Lagrange, vamos utilizar o grupo Alternado A_n, onde $A_n = \{\alpha \in S_n | \alpha$ é *permutação par*$\}$. A_n é um subgrupo de S_n denominado grupo alternado ou grupo das permutações pares.

```
gap> G:= AlternatingGroup(4);
```

```
Alt( [ 1 .. 4 ] )
gap> Order(G); # Mostra a ordem do Grupo G
12
```

Vamos calcular os divisores inteiros da ordem de G;

```
gap> divisores:= DivisorsInt(Order(G));
[ 1, 2, 3, 4, 6, 12 ]
```

Ao executar nossa função, vamos verificar se para cada divisor "d" da ordem de G, G possui um subgrupo de ordem "d".

```
gap> ContemSubgrupoTodaOrdem(G);
false
```

Logo, existe algum divisor "d", tal que G não possui um subgrupo de ordem "d". Para confirmar quem é esse divisor vamos chamar a função que calcula a ordem dos subgrupos de G.

```
gap> Ordens(G);
[ 1, 2, 3, 4, 12 ]
```

Portanto 6 divide $|G|$, mas G não possui subgrupo de ordem 6.

Conforme visto no código, com o grupo Alternado de 12 elementos mostramos que não é válida a recíproca do Teorema de Lagrange.

3.4 Classificação de Grupos de Ordem Pequena

Sabemos que para a classificação de um grupo abeliano G é necessário, primeiramente, fazer a decomposição de G em soma direta de p-subgrupos de Sylow de G, e depois fazer a decomposição de cada p-grupo como soma direta de subgrupos cíclicos de G, de forma que se obtenha a decomposição de G em soma direta de grupos cíclicos. Estas duas decomposições determinam o Teorema Fundamental dos Grupos Abelianos Finitos, conforme [12]. Por outro lado, não existe um modo semelhante para a classificação dos grupos não abelianos. Em [12], podemos encontrar os principais Teoremas de classificação.

A título de ilustração, mostramos algebricamente como classificar os grupos de ordens, 4, 6 e 8. Para ordens maiores consultar [10].

3.4.1 Grupos de Ordem 4

Proposição 1. *Seja ϕ um homomorfismo de grupos. Se $o(a) < \infty$, então, $o(\phi(a))$ divide $o(a)$*

De acordo com exemplos apresentados neste trabalho, podemos informar dois Grupos com respeito à adição que possuem 4 elementos, são eles:

$$\mathbb{Z}_4 = \{\bar{0}, \bar{1}, \bar{2}, \bar{3}\} \qquad e \qquad \mathbb{Z}_2 \times \mathbb{Z}_2 = \{(\bar{0}, \bar{0}), (\bar{1}, \bar{0}), (\bar{0}, \bar{1}), (\bar{1}, \bar{1})\}$$

Vejamos que $\mathbb{Z}_4$ e $\mathbb{Z}_2 \times \mathbb{Z}_2$ não são isomorfos. Se o fossem, existiria um isomorfismo $f : \mathbb{Z}_4 \longrightarrow \mathbb{Z}_2 \times \mathbb{Z}_2$. Desse modo, $f(\bar{1}) = (\bar{a}, \bar{b})$, para alguns $\bar{a}, \bar{b} \in \mathbb{Z}_2 \times \mathbb{Z}_2$. Em particular, como f seria homomorfismo, teríamos $f(\bar{2}) = f(\bar{1}) + f(\bar{1}) = (\bar{a}, \bar{b}) + (\bar{a}, \bar{b}) = (\bar{0}, \bar{0}) = f(\bar{0})$. Como f seria injetora, teríamos $\bar{2} = \bar{0}$, o que é uma contradição, pois $\bar{2} \neq \bar{0}$ em $\mathbb{Z}_4$.

Portanto $\mathbb{Z}_4$ e $\mathbb{Z}_2 \times \mathbb{Z}_2$ não são Isomorfos.

Agora mostraremos que qualquer outro grupo G com 4 elementos é isomorfo a $\mathbb{Z}_4$ ou a $\mathbb{Z}_2 \times \mathbb{Z}_2$. Vamos considerar os casos:

1. Seja G um grupo de 4 elementos, se um de seus elementos possui ordem 4, então G é cíclico. Considere $G =< a >= \{e, a, a^2, a^3\}$, e seja

$$\begin{aligned} \theta : \mathbb{Z}_4 &\to G \\ \bar{i} &\mapsto a^i \end{aligned}$$

   Como já visto θ é um isomorfismo, logo $\mathbb{Z}_4 \simeq G$.

2. Se G não possuir nenhum elemento de ordem 4, pelo Teorema de Lagrange, seus elementos, com exceção do elemento neutro, possuem ordem 2, então, G é Abeliano.

   De fato,

   Sejam $a, b \in G$, como G é grupo temos que $ab, ba \in G$ e sabemos que $o(a) = o(b) = o(ab) = o(ba) = 2$.

   Assim,

$$ab = e(ab)e = b^2(ab)a^2 = b(ba)(ba)a = b(ba)^2a = bea = ba$$

   Como a e b são arbitrários, temos que G é abeliano. Seja $G = \{e, a, b, c\}$, onde a, b e c são distintos. Vamos determinar o resultado da multiplicação ab.

   a) $ab \neq a$, pois senão,$b = e$, oque é um absurdo.

b) $ab \neq b$, pois caso contrário, teríamos $a = e$, o que também é um absurdo.

c) $ab \neq e$, caso contrário teríamos $b = a$, pois b tem ordem 2, o que é uma contradição.

Logo, $ab = c$ e $ba = c$, pois como mostrado aqui G é abeliano. Assim,

$$ac = b = ca \qquad e \qquad bc = a = cb$$

E, estabelecendo a seguinte relação:

$$(\bar{0}, \bar{0}) \longmapsto e$$
$$(\bar{1}, \bar{0}) \longmapsto a$$
$$(\bar{0}, \bar{1}) \longmapsto b$$
$$(\bar{1}, \bar{1}) \longmapsto c$$

temos que G é isomorfo a $\mathbb{Z}_2 \times \mathbb{Z}_2$. Portanto, a menos de isomorfismos $\mathbb{Z}_4$ e $\mathbb{Z}_2 \times \mathbb{Z}_2$ são os únicos grupos de ordem 4.

3.4.2 Grupos de Ordem 6

Ao decorrer deste trabalho conhecemos dois grupos de ordem 6, os quais são:

$$\mathbb{Z}_6 \ e \ S_3.$$

Os grupos $\mathbb{Z}_6$ *e* S_3 não são isomorfos, e uma maneira simples de observar este fato é notar que $\mathbb{Z}_6$ é abeliano enquanto S_3 o não é.

Vamos provar que $\mathbb{Z}_6$ e S_3 são os únicos grupos com 6 elementos a menos de isomorfismo.

Seja G um grupo qualquer com 6 elementos. Vamos provar alguns fatos:

1. G possui um elemento b de ordem 3;

   a) Se G for cíclico, então, G é gerado por um único elemento a. Assim, tomando $b = a^2$, note que $o(b) = 3$, pois $b^3 = a^6 = e$, $b^2 = a^4 \neq e$, e $b = a^2 \neq e$, $G =< a >$.

   b) Agora, se G não é cíclico, vamos supor, por absurdo, que G não possui elementos de ordem 3. Pelo Teorema de Lagrange, todo elemento diferente do neutro tem ordem 2, e como já demonstrado anteriormente, G é abeliano, e ainda, se tomarmos $H = \{e, a, b, ab\}$, temos que H é subgrupo de G de ordem 4, o que é um absurdo, pois 4 não divide 6, o que contradiz o Teorema de Lagrange.

2. G possui um elemento c de ordem 2 e ainda $G =< b, c >$.

   a) Se G é cíclico, logo é gerado por um elemento a, ou seja, $G =< a >$, tome $c = a^3$, tal que $o(c) = 2$, pois $c = a^3 \neq e$ e $c^2 = a^6 = e$.

   b) Se G não é cíclico, seja $c \notin < b >= \{e, b, b^2\}$. Vamos verificar que os elementos e, b, b^2, c, bc, b^2c são distintos. De fato,

   Observe que G possui um elemento c de ordem 2 e, assim, como $\{e, b, b^2, c, bc, b^2c\}$ está contido em G e estes dois conjuntos possuem a mesma cardinalidade, temos $G = \{e, b, b^2, c, bc, b^2c\} =< b, c >$.

   Vamos analisar todos os casos de igualdade:

   i. $bc = e \Rightarrow c = b^{-1}$, o que não é verdade pois $o(c) = 2$, assim $c = c^{-1}$.

   ii. $bc = b \Rightarrow c = e$, o que também não é verdade, pois, $c \notin < b >$.

   iii. $bc = b^2 \Rightarrow c = b$, o que não ocorre.

   iv. $bc = c \Rightarrow b = e$, o que é uma contradição.

   v. $bc = b^2c \Rightarrow b = b^2$, o que sabemos que não é válido.

   vi. $b^2c = e \Rightarrow bc = b^{-1} = b^2 \Rightarrow c = b$, absurdo.

   vii. $b^2c = b \Rightarrow bc = e$, contradição, já analisamos este caso.

   viii. $b^2c = b^2 \Rightarrow c = e$, o que não ocorre.

   ix. $b^2c = c \Rightarrow b^2 = e$, o que não é verdade.

   Portanto $G =< b, c >= \{e, b, b^2, c, bc, b^2c\}$, o que caracteriza G da seguinte forma:

$$\begin{cases} |G| = 6; \\ G =< b, c >; \\ b^3 = e; \\ c^2 = e; \end{cases}$$

   Mas, não analisamos o produto cb, vamos aos casos:

   i. $cb = e \Rightarrow c = b^{-1} = b^2$. Absurdo, pois $c \notin < b >$.

   ii. $cb = b \Rightarrow c = e$. Absurdo.

   iii. $cb = b^2 \Rightarrow c = b$. Absurdo.

   iv. $cb = c \Rightarrow b = e$. Absurdo.

   Assim, nos deparamos com as duas possibilidades:

$$cb = bc \; ou \; cb = b^2c.$$

   Desse modo, temos os seguintes grupos:

$$\textbf{Grupo 1:}\begin{cases} |G| = 6 = 3.2; \\ G =< b, c >; \\ b^3 = e; \\ c^2 = e; \\ cb = bc. \end{cases} \qquad \textbf{Grupo 2:}\begin{cases} |G| = 6 = 3.2; \\ G =< b, c >; \\ b^3 = e; \\ c^2 = e; \\ cb = b^2c. \end{cases}$$

Para observar o isomorfismo precisamos do seguinte resultado:

Teorema 7. *Sejam $n, m, s, u \in \mathbb{Z}^+$.*

1. *Se G é um grupo de ordem mn, tais que, para $b, c \in G$ temos:*

$$G =< b, c >,\ b^n = e,\ c^m = b^u,\ cb = b^s c. \tag{3.1}$$

   Então, $s^m \equiv 1$ (mod n) e $u(s-1) \equiv 0$ (mod n). Reciprocamente, se $s^m \equiv 1$ (mod n) e $u(s-1) \equiv 0$ (mod n), então, existe um grupo G de ordem $n.m$ que satisfaz a relação (3.1) acima.

2. *Quando existir um grupo G satisfazendo a relação (3.1) ele é único a menos de isomorfismo.*

Demonstração: Ver [10]

Sendo assim, de acordo com o Teorema 7, temos que o grupo 1 é isomorfo a $\mathbb{Z}_6$ e o grupo 2 é isomorfo a S_3. Portanto, a menos de isomorfismos, $\mathbb{Z}_6$ *e* S_3 são os únicos grupos de 6 elementos.

3.4.3 Grupos de Ordem 8

A menos de Isomorfismo só existem 5 grupos de ordem 8. Já conhecemos os seguintes grupos de ordem 8:

$$\mathbb{Z}_8, \mathbb{Z}_4 \times \mathbb{Z}_2, \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2 \, e \, D_4.$$

Observe que esses grupos não são isomorfos, pois D_4 não é abeliano, $\mathbb{Z}_8$ é cíclico, $\mathbb{Z}_8 =< \bar{1} >$, sendo assim possui elemento de ordem 8, mas $\mathbb{Z}_4 \times \mathbb{Z}_2$ e $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$ não possuem elementos de ordem 8, pois $\mathbb{Z}_4 \times \mathbb{Z}_2$ possui elementos de ordem no máximo 4 e $\mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$ possui elementos de ordem no máximo 2. Se f é um homomorfismo injetor de grupos multiplicativos, então, $o(f(a)) = o(a)$, para todo $a \in G$.

Vamos provar que existe a menos de isomorfismo apenas mais um grupo G com 8 elementos, conhecido como grupo dos Quatérnios e denotado po Q_8, cujos elementos são:

$$Q_8 = \left\{ \begin{array}{l} e = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, M = \begin{pmatrix} -i & 0 \\ 0 & -i \end{pmatrix}, N = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, MN = \begin{pmatrix} 0 & i \\ i & 0 \end{pmatrix}, \\ M^2 = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, M^3 = \begin{pmatrix} i & 0 \\ 0 & i \end{pmatrix}, M^2N = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, M^3N = \begin{pmatrix} 0 & -i \\ -i & 0 \end{pmatrix} \end{array} \right\}$$

onde $i \in \mathbb{C}$. Temos que Q_8 é um grupo com relação à operação multiplicação de matrizes, e tomando $M, N \in Q_8$, podemos verificar que:

$$Q_8 \begin{cases} |Q_8| = 8; \\ Q_8 =< M, N >; \\ M^4 = e; \\ N^2 = M^2; \\ NM = M^3N. \end{cases}$$

Pelo Teorema 7, temos que Q_8 não é isomorfo a $\mathbb{Z}_8, \mathbb{Z}_4 \times \mathbb{Z}_2, \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$ pois o produto de suas matrizes não é comutativo.

Vamos verificar que D_4 e Q_8 não são isomorfos,

$$D_4 = \{e, a, a^2, a^3, b, ab, a^2b, a^3b\}$$

As ordens dos elementos de D_4 são:

$$o(e) = 1 \ , \ o(a) = 4$$
$$o(a^2) = 2 \ , \ o(a^3) = 4$$
$$o(b) = 2 \ , \ o(ab) = 2$$
$$o(a^2b) = 2 \ , \ o(a^3b) = 2$$

e os elementos de Q_8 possuem as seguintes ordens:

$$o(e) = 1 \ , \ o(M) = 4$$
$$o(M^2) = 2 \ , \ o(M^3) = 4$$
$$o(N) = 4 \ , \ o(MN) = 4$$
$$o(M^2N) = 4 \ , \ o(M^3N) = 4$$

Note que Q_8 possui apenas um elemento de ordem 2, enquanto D_4 tem 5 elementos de ordem 2, o que nos permite concluir que Q_8 e D_4 não são isomorfos.

Agora vamos verificar que qualquer outro grupo G com 8 elementos é isomorfo a alguns desses cinco grupos citados aqui.

Seja G um grupo de 8 elementos, Pelo Teorema de Lagrange as únicas possíveis ordens para seus elementos, diferentes do elemento neutro, são 2, 4 e 8. Sendo assim, vamos aos casos:

1. Se G possui um elemento a de ordem 8, então G é cíclico, e como já vimos, $G \simeq \mathbb{Z}_8$.

2. Se G não possui elementos de ordem 8, então, de acordo com o Teorema de Lagrange, as ordens de seus elementos, diferentes do neutro, são 2 e 4.

   a) Suponha que G não possui elemento de ordem 4, logo todos os seus elementos possuem ordem 2, e como já observado, neste caso G é abeliano. Temos também que:

      i. Se $a \in G$, $a \neq e$, então, $H_1 = \{e, a\}$ é subgrupo de G;

      ii. Se $b \in G \backslash H_1$, então, $H_2 = \{e, a, b, ab\}$ é subgrupo de G;

      iii. se $c \in G \backslash H_2$, então, $\{e, a, b, ab, c, ac, bc, abc\}$ é um subgrupo de G. Portanto, como $\{e, a, b, ab, c, ac, bc, abc\}$ está contido em G e $|G| = 8$, temos $G = \{e, a, b, ab, c, ac, bc, abc\}$.

   Observe que $ac \neq ab$, $ac \neq bc$ e que $ab \neq bc$, pois senão, $b = c$,$a = b$ ou $a = c$, o que é um absurdo.

   Vamos provar que $abc \notin \{e, a, b, c, ab, ac, bc\}$, já sabemos que $o(a) = o(b) = o(c) = 2$, logo:

      i. $abc = e \Rightarrow ab = c^{-1} \Rightarrow ab = c$, absurdo.

      ii. $abc = a \Rightarrow bc = e \Rightarrow b = c^{-1} = c$, o que não é verdade.

      iii. $abc = b \Rightarrow ac = e \Rightarrow a = c^{-1} = c$, contradição.

      iv. $abc = c \Rightarrow ab = e \Rightarrow a = b^{-1} = b$, absurdo.

      v. $abc = ab \Rightarrow c = e$, o que não é verdade.

      vi. $abc = ac \Rightarrow b = e$, o que não ocorre.

      vii. $abc = bc \Rightarrow a = e$, absurdo.

   Portanto, $G = \{e, a, b, c, ab, ac, bc, abc\} = \{a^i b^j c^k; i, j, k \in \{0, 1\}\}$.

   Considerando a função:

$$\begin{aligned} \theta : \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2 &\to G \\ (\bar{i}, \bar{j}, \bar{k}) &\mapsto a^i b^j c^k \end{aligned}$$

   temos que θ é um isomorfismo de grupos e, neste caso, $G \simeq \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$.

   b) Suponha, agora, que G possui um elemento a de ordem 4. Temos que $H_1 =< a >$ é um subgrupo de G. Tomando $b \in G \backslash H_1$, então $H_2 =< a, b >$ possui $|H_2| > 4$. Pelo Teorema de Lagrange, $|H_2||8$, o que implica,

$$H_2 = G = \{e, a, a^2, a^3, b, ab, a^2b, a^3b\}.$$

Observe que $b^2 \notin G \backslash H_1 = \{b, ab, a^2b, a^3b\}$, de fato:

i. $b^2 = b \Rightarrow b = e$, absurdo.

ii. $b^2 = ab \Rightarrow b = a$, o que não é verdade.

iii. $b^2 = a^2b \Rightarrow b = a^2$, contradição.

iv. $b^2 = a^3b \Rightarrow b = a^3$, o que também não é verdade.

Portanto $b^2 \in H_1$. Observe também que:

i. $ba = e \Rightarrow a = b^{-1}$, o que não ocorre.

ii. $ba = a \Rightarrow b = e$, absurdo.

iii. $ba = a^2 \Rightarrow b = a$, o que não é verdade.

iv. $ba = a^3 \Rightarrow b = a^2$, que é uma contradição.

v. $ba = b \Rightarrow a = e$, absurdo.

Portanto $b^2 \notin \{e, a, a^2, a^3, b\}$.

Assim, G é cara caracterizado por:

$$\begin{cases} |G| = 8; \\ G =< a, b >; \\ a^4 = e; \\ b^2 = a^u, \ \ u \in \{0, 1, 2, 3\}; \\ ba = a^s b, \ \ s \in \{1, 2, 3\}; \end{cases}$$

Antes de analisarmos as possibilidades para s e u, observe que:

$$(bab^{-1})^n = (bab^{-1})(bab^{-1})...(bab^{-1}) = (ba)(b^{-1}b)(ab^{-1})...(b^{-1}b)(ab^{-1}) =$$
$$= (ba)e(ab^{-1})...e(ab^{-1}) = ba^nb^{-1}.$$

Se $s = 2$, temos que $ba = a^2b \Rightarrow bab^{-1} = a^2 \Rightarrow (bab^{-1})^2 = a^4 = e$, o que é um absurdo. Assim, $s = 1$ ou $s = 3$. Além disso, como $b^2 \notin \{a, a^3\}$, temos que $u = 0$ ou $u = 2$.

Portanto, temos quatro possibilidades de grupos:

Grupo 1: $\begin{cases} |G| = 8; \\ G =< a, b >; \\ a^4 = e; \\ b^2 = e; \\ ba = ab. \end{cases}$ **Grupo 2:** $\begin{cases} |G| = 8; \\ G =< a, b >; \\ a^4 = e; \\ b^2 = e; \\ ba = a^3b. \end{cases}$

$$\textbf{Grupo 3:}\begin{cases} |G| = 8; \\ G =< a, b >; \\ a^4 = e; \\ b^2 = a^2; \\ ba = ab. \end{cases} \qquad \textbf{Grupo 4:}\begin{cases} |G| = 8; \\ G =< a, b >; \\ a^4 = e; \\ b^2 = a^2; \\ ba = a^3b. \end{cases}$$

Nestes casos, temos que o grupo 1 e o grupo 3 são isomorfos a $\mathbb{Z}_4 \times \mathbb{Z}_2$, pois G é abeliano. O grupo 2 é isomorfo a D_4 e por fim, o grupo 4 é isomorfo a Q_8. Portanto, $\mathbb{Z}_8, \mathbb{Z}_4 \times \mathbb{Z}_2, \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2, D_4$ e Q_8 são os únicos grupos de ordem 8 a menos de isomorfismos.

Na seção seguinte, vamos conhecer a aplicação que desenvolvemos que exibe todos os grupos a menos de Isomorfismo.

3.5 Aplicação do GAP para Classificação de Grupos Finitos

Nesta seção, apresentamos alguns algoritmos que foram implementados com o sistema computacional GAP que auxiliam na construção da Tabela de Classificação dos Grupos de ordem pequena. Com a linguagem de programação GAP, vários conteúdos de Álgebra serão melhores estudados e aprofundados, tendo em vista que vários dos resultados algébricos nem sempre são fáceis de serem ilustrados e visualizados manualmente.

Por fim, utilizando os recursos fornecidos pela linguagem do GAP, construímos uma aplicação que classifica os Grupos a menos de Isomorfismo, e além disso, disponibilizamos uma série de possibilidades para que o usuário trabalhe com um determinado Grupo pertencente à Tabela, como por exemplo, calcular subgrupos, subgrupos normais, subgrupos de Sylow e tabela de multiplicação de grupos.

O GAP possui uma biblioteca com vários grupos, entretanto, o usuário precisa ter um conhecimento prévio de programação para acessá-los. Sendo assim, a aplicação possibilita apresentar os grupos, a menos de isomorfismo, e exibir uma série de recursos para que o usuário trabalhe com qualquer grupo apresentado na tabela. A diferença da aplicação desenvolvida em relação ao que o próprio GAP oferece é que o usuário precisa ter apenas um mínimo de conhecimento em programação para executar as mesmas operações introduzidas no algoritmo.

3.5.1 Tabela de Classificação de Grupos de Ordem Pequena

O intuito é exibir a construção da Tabela de Classificação de Grupos de Ordem Pequena, sendo assim, para o entendimento dos algoritmos, vamos, inicialmente, ilustrar alguns comandos disponíveis na biblioteca do GAP que são utilizados para a determinação de grupos de ordem n. Todas estas funções podem ser encontradas no manual de referência[1] do GAP.

A função *AllSmallGroups(n)* nos apresenta uma lista de todos os grupos de ordem "n", já a função "*SmallGroup(ordem,pos)*" contém uma lista de vários grupos que são organizados numa certa ordem sequencial preestabelecida pela própria linguagem; esta função retorna o grupo com a "ordem" escolhida e o apresenta na posição "pos" correspondente na sequência. Por sua vez, a função "*StructureDescription(grupo)*" retorna uma string que descreve a representação do grupo G no GAP.

Para exemplificar como estas funções são executadas no GAP, relembremos que existem dois grupos a menos de isomorfismo com ordem igual a 4, são eles: $G_1 = \mathbb{Z}_4$ e $G_2 = \mathbb{Z}_2 \times \mathbb{Z}_2$; existem dois grupos a menos de isomorfismo com ordem igual a 6, são eles: $G_1 = S_3$ e $G_2 = \mathbb{Z}_6$; e existem cinco grupos a menos de isomorfismo com ordem igual a 8, são eles: $G_1 = \mathbb{Z}_8$, $G_2 = \mathbb{Z}_4 \times \mathbb{Z}_2$, $G_3 = D_8$, $G_4 = Q_8$ e $G_5 = \mathbb{Z}_2 \times \mathbb{Z}_2 \times \mathbb{Z}_2$. Vejamos a representação no GAP de alguns destes grupos usando as funções descritas acima:

```
gap> StructureDescription(SmallGroup(4,2));
''C2 x C2''
gap> StructureDescription(SmallGroup(8,3));
''D8''
```

Apresentaremos os algoritmos criados a partir das funções introduzidas acima com o objetivo de determinar a classificação dos grupos de ordem n. Para a apresentação da tabela, são necessários quatro algoritmos: o primeiro, serve para carregar a logo da aplicação e o pacote Sonata[2] que auxilia nos rercusos disponíveis no "*Menu*"; o segundo, "*Grupo*", informa todos os grupos existentes a menos de Isomorfismo de uma determinada ordem n que foi passado como parâmetro; o terceiro algoritmo,"*TabelaClassGrupos*", é responsável pela exibição da tabela de Classificação de Grupos de Ordem Pequena, que, no nosso exemplo, exibe até a ordem 15; Por fim, o quarto e último algoritmo, "*Menu*" é responsável por chamar os recursos que permitem que o usuário trabalhe com um determinado Grupo exibido na Tabela. A função "*Menu*" carrega subalgoritmos que possibilitam saber a Tabela de multiplicação dos elementos do Grupo, calcular subgrupos, calcular p-subgrupos de Sylow, subgrupos normais e verificar se um determinado Grupo é abeliano. O algoritmo que carrega a logo será omitido nesta monografia. Os algoritmos são:

[1] Disponível neste endereço: http://www.gap-system.org/Manuals/doc/ref/chap0.html
[2] Disponível em: http://www.gap-system.org/Packages/sonata.html

Segundo Algoritmo:

Algoritmo 7: Grupo

```
1  Grupo:=function(ordem)
2          local lista,apresent,i;
3          lista:=AllSmallGroups(ordem);
4          apresent:=List(lista,x->StructureDescription(x));
5          Print(``\t'',ordem,``\t'');
6          for i in [1..Size(apresent)] do
7                  Print(apresent[i],``;'');
8          od;
9          Print(``\n'');
10 end;
```

Terceiro Algoritmo:

Algoritmo 8: TabelaClassGrupos

```
1  TabelaClassGrupos:= function(ordemmax)
2          local i,ord;
3          ord:=ordemmax;
4          Print(``\n\t\t Ordem \t Grupo \n'');
5          for i in [1..ord] do
6                  Print(``\t''); Grupo(i);
7          od;
8  end;
```

Com base nos algoritmos citados, podemos fazer um arquivo que carregue a aplicação. Assim, ao executar o arquivo no GAP, todas as funções serão carregadas e teremos o seguinte resultado:

Ao executar a função "*Menu*", serão exibidos recursos para que o usuário trabalhe com um determinado grupo escolhido na Tabela, os recursos são apresentados conforme a Figura 2.

Como observado, as funcionalidades no "*Menu*" são autoexplicativas, basta seguir os passos e será possível realizar as operações desejadas. Sendo assim, o usuário em questão não necessita de conhecimentos em programação para usar a Tabela, o que abrange um número maior de matemáticos e estudantes que queiram estudar a Teoria de Grupos. Convém observar que, a aplicação não é restrita a Grupos de ordem até 15 somente, basta executar a função "*TabelaClassGrupos(ordemMAX)*" e terá uma maior quantidade de

```
_____________      _____
     ||           |     \\           ||     Versão: 1.0
     ||           |      ||  _____   ||                      Rosemary Miguel Pires
     ||    /\\    |_____//  |        ||            /\\       Túlio Joaquim Altoé
     ||   /  \\   |     \\  |___     ||           /  \\
     ||  /____\\  |      || |        ||          /____\\     rosemarypires@id.uff.br
     || /      \\ |_____//  |_____   ||_______ /      \\    tuliojaltoe@gmail.com

                de Classificação dos Grupos de Ordem Pequena.

               ORDEM    GRUPO
                1       1;
                2       C2;
                3       C3;
                4       C4; C2 x C2;
                5       C5;
                6       S3; C6;
                7       C7;
                8       C8; C4 x C2; D8; Q8; C2 x C2 x C2;
                9       C9; C3 x C3;
                10      D10; C10;
                11      C11;
                12      C3 : C4; C12; A4; D12; C6 x C2;
                13      C13;
                14      D14; C14;
                15      C15;

         Para acessar os recursos chame 'Menu()'.

gap>
```

Figura 1 – Tabela de Classificação de Grupos de Ordem Pequena

Grupos para se trabalhar, onde “ordemMAX” é a ordem máxima de grupos que serão exibidos.

```
gap> Menu();

        1) PARA SABER A TABELA DE MULTIPLICAÇÃO, FAÇA:
               -Escolha um Grupo.
               -Pegue a sua posição ('grupo1'; 'grupo2'; ...).
               -Observe a ordem do Grupo escolhido.
               -Utilize a função: 'TabelaMultiplicacao(ordem, posição);'.


        2) PARA SABER OS SUBGRUPOS, FAÇA:
               -Escolha um Grupo.
               -Pegue a sua posição ('grupo1'; 'grupo2'; ...).
               -Observe a ordem do Grupo escolhido.
               -Utilize a função: 'Subgrupos(ordem, posição);'.


        3) PARA SABER SE UM GRUPO É ABELIANO E ACHAR OS SUBGRUPOS NORMAIS, FAÇA:
               -Escolha um Grupo.
               -Pegue a sua posição ('grupo1'; 'grupo2'; ...).
               -Observe a ordem do Grupo escolhido.
               -Utilize a função: 'SubgruposNormais(ordem, posição);'.


        4) PARA SABER OS SUBGRUPOS DE SYLOW, FAÇA:
               -Escolha um Grupo.
               -Pegue a sua posição ('grupo1'; 'grupo2'; ...).
               -Observe a ordem do Grupo escolhido.
               -Escolha um número primo.
               -Utilize a função: 'SubgruposDeSylow(ordem, posição, nº primo);'.

gap>
```

Figura 2 – Recursos Disponíveis para trabalhar com um determinado Grupo presente na Tabela

Os subalgoritmos que compõem a função "*Menu*" são:

1. **Cálculo de Subgrupos:**

Algoritmo 9: Subgrupos

```
1  Subgrupos:= function(ordem, pos)
2          local g,h,lista;
3          g:= SmallGroup(ordem,pos);
4          h:= Subgroups(g);
5          Print(``\n\t Subgrupos \n\n'');
6          Print(``\t --> '',h);
7          lista:= List(h, x-> StructureDescription(x));
8          Print(``\n\n\t Subgrupos \n\n'');
9          Print(``\t --> '',lista,``\n\n'');
10 end;
```

2. **Cálculo dos Subgrupos Normais:**

Algoritmo 10: Subgrupos Normais

```
1  SubgruposNormais:= function (ordem,pos)
2          local g,h,i,listanormais,normais;
3          listanormais:=[];
4          g:= SmallGroup(ordem,pos);
5          h:= Subgroups(g);
6          for i in [1..Size(h)] do
7                  if IsNormal( g, h[i] )= true then
8                          Add(listanormais,h[i]);
9                  fi;
10         od;
11         normais:= List(listanormais,
12                 x -> StructureDescription(x));
13         Print(``\n\t'', normais, ``\n\n'');
14 end;
```

3. **Cálculo dos Subgrupos de Sylow:**

Algoritmo 11: Subgrupos de Sylow

```
1  SubgruposDeSylow:= function (ordem,pos,primo)
2          local g,h,lista;
3          g:= SmallGroup(ordem,pos);
4          h:= SylowSubgroup(g,primo);
5          Print(``\n \t Grupo: '',
6                          StructureDescription(g),``\n'');
7          Print(``\n\n\t '',primo,``-
8                          Subgrupo de Sylow \n\n'');
9          Print(``\t  --> '',
10                         StructureDescription(h),``\n\n'');
11 end;
```

4. **Cálculo da Tabela de Multiplicação:**

Algoritmo 12: Tabela de Multiplicação

```
1  TabelaMultiplicacao:= function(ordem,pos)
2          local g;
3          g:= SmallGroup(ordem,pos);
4          Print(``\n \t Grupo: '',
5                          StructureDescription(g),``\n\n'');
6          PrintTable(g);
```

```
7           Print(``\n\n'');
8  end;
```

5. **Para determinar se o grupo é abeliano:**

Algoritmo 13: Abeliano

```
1  Abeliano:= function (ordem)
2           local lista,i;
3           lista:= AllSmallGroups(ordem);
4           Print(``\n\t\t Grupo \t\t Abeliano\n'');
5           for i in [1..Size(lista)] do
6                   Print(``\n\t\t'',
7                           StructureDescription(lista[i]);
8                   Print(``\t\t'', IsAbelian(lista[i]),``\n'');
9           od;
10 end;
```

Como exemplo, vejamos a tabela de multiplicação do grupo cíclico de ordem 4:

Algoritmo 12: Exemplo Tabela

```
1  gap> TabelaMultiplicacao(4,1);
2  Let:
3  g0 := <identity> of ...
4  g1 := f1
5  g2 := f2
6  g3 := f1*f2
7
8     *  | g0  g1  g2  g3
9    ------------------------
10    g0  | g0  g1  g2  g3
11    g1  | g1  g2  g3  g0
12    g2  | g2  g3  g0  g1
13    g3  | g3  g0  g1  g2
```

Neste capítulo usando o sistema computacional GAP criamos algoritmos com as próprias funções do sistema que podem ser aplicados para ajudar na exemplificação e visualização de vários dos teoremas clássicos relacionados à Classificação de grupos. Determinamos uma tabela de Classificação de Grupos de ordem pequena, a menos de isomorfismo, e apresentamos aplicações para serem executadas no GAP que facilitam o acesso aos grupos de determinada ordem e que auxiliam no estudo de alguns conceitos da Teoria de Grupos.

4 Anéis e Corpos

Este capítulo está destinado às definições que serão de fundamental importância para fundamentar o Capítulo 5. Aqui, serão apresentados conceitos fundamentais da Álgebra, tais como, anéis, anel comutativo, domínio de integridade, corpos, ideais, ideais maximais e polinômios irredutíveis.

Relacionamos a teoria com o GAP, através do Método de Kronecker, método este que permite fatorar um polinômio $p(x)$ sobre $\mathbb{Z}$.

4.1 Anel, Domínio e Corpo

Todas as definições aqui apresentadas, podem ser consultadas em [1] e [2].

Um **Anel** associativo é uma terna $(A, +, .)$, constituido por um conjunto não vazio A e que possui duas operações fechadas em A, as quais chamaremos de soma $(+)$ e produto $(\cdot)$, definidas da seguinte maneira:

$$\begin{aligned} + : A \times A \to A \qquad & \cdot : A \times A \to A \\ (x, y) \mapsto x + y \qquad & (x, y) \mapsto x.y \end{aligned}$$

e que satisfazem as propriedades abaixo:

1. Para todo $x, y, z \in A, (x + y) + z = x + (y + z)$ (Associatividade da Soma);
2. Existe $0 \in A$ tal que, para quaisquer $x \in A, 0 + x = x + 0 = x$ (Elemento Neutro da Soma);
3. Para todo $x \in A$; existe $y \in A$ tal que, $y + x = 0$ *e* $x + y = 0$ (Inverso Aditivo);
4. Para quaisquer $x, y \in A, y + x = x + y$ (Comutatividade da Soma);
5. Para todo $x, y, z \in A, (x.y).z = x.(y.z)$ (Associatividade do Produto);
6. Para quaisquer $x, y, z \in A, (x + y).z = x.z + y.z$ (Distributividade à esquerda);
7. Para quaisquer $x, y, z \in A, z.(x + y) = z.x + z.y$ (Distributividade à direita);

Por questão de simplicidade, a partir de agora, denotaremos o anel $(A, +, .)$ por A. Se o anel A satisfaz as seguintes propriedades ele é classificado de modo diferenciado:

1. Para todo $x, y \in A,$ *temos* $y.x = x.y$;

   Neste caso, A é um **Anel Comutativo**.

2. Para todo, $x \in A$, existe $1 \in A,$ *tal que,* $1.x = x.1 = x$;

   Neste caso, A é um **Anel com unidade**.

3. Para todo $x, y \in A, temos\ x.y = 0 \Rightarrow x = 0\ ou\ y = 0.$

   Neste caso, A é um **Anel sem Divisores de Zero**.

   Caso A satisfaça 1,2 e 3 dizemos que A é um **Domínio de Integridade**.

4. Para todo $x \in A$, $x \neq 0$, existe $y \in A,$ *tal que,* $y.x = x.y = 1.$

   Neste caso, A é chamado de **Corpo**.

Seja A um anel comutativo. Dizemos que um subconjunto $I \subset A$, com I $\neq \emptyset$, é um **ideal** de A se satisfazer as seguintes propriedades:

1. $\forall\, x, y \in I$ *temos* $x - y \in I$
2. $\forall\, a \in A$ *e* $\forall\ x \in I$ *temos* $a \cdot x \in I$

Um ideal M de uma anel A com $M \neq A$ é chamado de **ideal maximal** se, para todo ideal I de A, tal que, $M \subset I \subset A$, temos $I = A$ ou $I = M$.

Vamos definir agora, homomorfismo e isomorfismo de anéis e, logo em seguida, enunciaremos o Teorema dos Isomorfismos.

Definição 8. *Sejam A e B dois anéis. Uma aplicação $f : A \longrightarrow B$ é chamada de homomorfismo se satisfaz as seguintes propriedades:*

1. $f(x + y) = f(x) + f(y)$, $\forall\ x, y \in A$;
2. $f(x.y) = f(x).f(y)$, $\forall\ x, y \in A$;

Se f é um homomorfismo bijetor, dizemos que f é um isomorfismo de anéis, e neste caso, dizemos que A e B são isomorfos.

Teorema 9. *(Teorema dos Isomorfismos)*

Seja f: $A \longrightarrow B$ um homomorfismo de anéis. Então, a aplicação $\bar{f}$, definida como:

$$\bar{f} : A/Ker(f) \to Im(f)$$
$$\bar{a} \mapsto f(a)$$

é um isomorfismo de anéis.

Demonstração: Pode ser encontrada em [1]

Considere agora um anel K, chamamos de $K[x]$ o conjunto de todos os polinômios com coeficientes em K, em uma indeterminada x, ou seja,

$$K[x] = \{a_0 + a_1x + a_2x^2 + \ldots + a_nx^n + \ldots ; a_i \in K, i \in \mathbb{N}\}$$

Vamos definir as operações de soma e produto no conjunto $K[x]$. Sejam $p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ e $q(x) = b_0 + b_1x + b_2x^2 + \ldots + b_nx^n$, elementos de $K[x]$.

Definimos,

$p(x) + q(x) = c_0 + c_1x + c_2x^2 + \ldots + c_nx^n + \ldots$, onde $c_i = (a_i + b_i) \in K, i \in \mathbb{N}$

e,

$p(x) \cdot q(x) = c_0 + c_1x + c_2x^2 + \ldots + c_nx^n + \ldots$, onde $c_0 = a_0b_0, c_1 = a_0b_1 + a_1b_0, ..., c_k = a_0b_k + a_1b_{k-1} + ... + a_{k-1}b_1 + a_kb_0$, $k \in \mathbb{N}$.

Quando K é um domínio de integridade temos que $K[x]$ também é um domínio de integridade.

Seja K um anel e $f(x) \in K[x]$, tal que, $f(x) = a_nx^n + ... + a_0$. Dizemos que n é o grau do polinômio $f(x)$ e denotamos por $\partial(f(x)) = n$, quando $a_n \neq 0$. Dizemos que um polinômio $f(x)$ é redutível sobre K se é possível expressar $f(x)$ como $f(x) = g(x)h(x)$, onde $\partial(g(x)) \geq 1$ e $\partial(h(x)) \geq 1$, com $g(x), h(x) \in K[x]$. Caso contrário, ele é chamado de irredutível.

Na seção seguinte mostraremos um método que permite fatorar um polinômio $p(x)$ sobre o Domínio de Integridade $\mathbb{Z}[x]$.

4.2 Fatoração de um polinômio sobre um Domínio de Integridade

Nesta seção, nos limitaremos ao estudo da fatoração de polinômios sobre um anel. Este processo, muitas das vezes, é árduo para ser feito manualmente. Por isto, torna-se necessário a utilização de ferramentas computacionais que auxiliem no processo dos cálculos e também, na ilustração e simulação dos teoremas matemáticos. Vamos implementar o método de Kronecker, mais detalhes em [13], que permite decompor um polinômio $p(x) \in \mathbb{Z}[x]$.

Se um polinômio $f(x)$ é redutível sobre um anel K, podemos decompor $f(x)$ como produto de dois outros polinômios irredutíveis $g(x)$, $h(x) \in K[x]$. Em geral, este processo não é fácil de ser executado, dependendo muito do Domínio de Integridade $K[x]$ em que $f(x)$ pertence.

Considerando $K = \mathbb{Z}$, existe um método que nos possibilita fatorar um polinômio em $\mathbb{Z}[x]$. Tal método, conhecido como **Método de Kronecker** não é viável algebricamente, pois depende de muitos cálculos, mas quando implementado computacionalmente podemos executá-lo de maneira bem eficiente.

Este método consiste na busca dos polinômios divisores de$p(x)$, onde $p(x) \in \mathbb{Z}[x]$. O método será compreendido em etapas.

Dado um polinômio $p(x)$ de grau n, analisaremos separadamente seus possíveis divisores.

• Divisores de $p(x)$ de grau 1.

Suponhamos inicialmente que $q(x) = a.x + b$, com $q(x) \in \mathbb{Z}[x]$ seja um fator de $p(x)$, ou seja, $p(x) = q(x).h(x)$

Dado $\beta \in \mathbb{Z}$ temos que $p(\beta) = q(\beta).h(\beta) = (a.\beta + b)h(\beta)$, e assim $(a.\beta + b)|p(\beta)$. Assim nossa fatoração se reduz à busca dos valores de "a" e "b" de modo que esta fatoração seja possível.

Como β é qualquer, tomando dois inteiros β e ϕ com $\beta \neq \phi$, tais que $p(\beta) \neq 0$ e $p(\phi) \neq 0$, obtemos duas igualdades que dependem de 'a' e 'b' e que nos fornece um sistema como:

$$\begin{cases} a.\beta + b = d_1 \\ a.\phi + b = d_2 \end{cases}$$

onde d_1 é um divisor de $p(\beta)$, e d_2 é um divisor de $p(\phi)$. Assim, obtemos todos os possíveis candidatos a divisores da forma $ax + b$ de $p(x)$. A escolha de β e ϕ, utilizados acima, deve levar em conta que quanto menor for o número de divisores de $p(\beta)$ e $p(\phi)$, menor será o número de sistemas de equações que teremos que resolver.

• Divisores de $p(x)$ de grau 2.

Para determinar os divisores de grau 2 de $p(x)$, da forma $ax^2 + bx + c$ de $p(x) \in \mathbb{Z}$, devemos tomar três inteiros β, ϕ, γ, distintos dois a dois, tais que nenhum deles seja raiz de $p(x)$. Assim temos o seginte sistema:

$$\begin{cases} a.\beta^2 + b\beta + c = d_1 \\ a.\phi^2 + b\phi + c = d_2 \\ a.\gamma^2 + b\gamma + c = d_3 \end{cases}$$

onde d_1 é um divisor de $p(\beta)$, d_2 é um divisor de $p(\phi)$ e d_3 é um divisor de $p(\gamma)$. Assim, resolvendo esse sistema de equações obtemos os possíveis candidatos a divisores quadráticos de $p(x)$, lembrando que devemos escolher com cuidado os valores β, ϕ, γ.

• Para determinar os fatores de $p(x)$ com grau maior que dois, o processo é análogo.

Para exemplificar o Método de Kronecker, vamos considerar o seguinte problema:

4.2.1 O Problema

O problema consiste em dado um polinômio $p(x) \in \mathbb{Z}[x]$ determinar os fatores quadráticos de $p(x)$. Para isso vamos considerar o polinômio $p(x) = x^4 + 2x^3 + x^2 - 1$, $\beta = 0$, $\phi = 1$ e $\gamma = -1$ e ver se $p(x)$ se decompõe em fatores do tipo $ax^2 + bx + c$.

Algebricamente esse processo é cansativo, uma vez que, teríamos que resolver 16 sistemas lineares para determinar todos os possíveis candidatos a fatores de $p(x)$.

Para esse tipo de problema, percebe-se que o lado esquerdo das equações não mudam, o que é alterado nesse sistema de equações são apenas os divisores d_1, d_2 e d_3. Para solução desses sistemas lineares usaremos o Método de Decomposição L.U, método mais clássico utilizado para resolver sistemas de equações algébricas, que é ideal para esse tipo de problema. Podemos fixar nossa matriz de coeficientes provenientes das equações do sistema e variar o vetor de divisores do lado direito da equação.

Implementamos esse problema usando o GAP. O código se encontra no apêndice deste trabalho, mas mostraremos agora a execução do nosso algoritmo.

Primeiramente vamos definir a indeterminada na qual usaremos para trabalhar com polinômios:

```
gap> x:= X(Integers, ``x'');
x
```

Uma vez definida a indeterminada, agora podemos trabalhar com polinômios com coeficientes inteiros na indeterminada x. Sendo assim, vamos atribuir a p(x) o polinômio no qual queremos fatorar.

```
gap> p:= x^4 + 2*x^3 + x^2 -1;
x^4+2*x^3+x^2-1
```

Vamos realizar agora a leitura do algoritmo presente no apêndice deste trabalho:

```
gap> Read(``/home/tulioaltoe/kroneckerGAP.g'');
```

Basta agora executar o algoritmo, passando como parâmetro as condições iniciais do problema, e teremos o seguinte resultado:

```
gap> KroneckerGAP(p, 0, 1, -1);

 A matriz A:
[ [    1,   -1,    1 ],
```

```
[    1,    1,    1 ],
[    0,    0,    1 ] ]

Divisores:

d1=-1
d2=3
d3=-1
```

Divisores de d1, d2 e d3:

```
divD1=[ -1, 1 ]
divD2=[ -3, -1, 1, 3 ]
divD3=[ -1, 1 ]

Tamanho da Tabela:16
```

	D3	D2	D1	a	b	c
1	−1	−3	−1	−1	−1	−1
2	1	−3	−1	0	−2	−1
3	−1	−1	−1	0	0	−1
4	1	−1	−1	1	−1	−1
5	−1	1	−1	1	1	−1
6	1	1	−1	2	0	−1
7	−1	3	−1	2	2	−1
8	1	3	−1	3	1	−1
9	−1	−3	1	−3	−1	1
10	1	−3	1	−2	−2	1
11	−1	−1	1	−2	0	1

12	1	−1	1	−1	−1	1
13	−1	1	1	−1	1	1
14	1	1	1	0	0	1
15	−1	3	1	0	2	1
16	1	3	1	1	1	1

O algoritmo então vai percorrer todas as possibilidades e informar os possíveis divisores de p(x):

[x^2−x−1, x^2+x−1, x^2+x+1]

E por fim, ele encontra a fatoração:

x^4+2∗x^3+x^2−1=(x^2+x−1)(x^2+x+1)

Obtendo assim o resultado desejado. O exemplo em questão é simples de ser verificado, mas a medida que aumentamos o grau do polinômio ou ainda chutamos os valores iniciais que geram uma grande quantidade de sistemas a serem resolvidos, tornamos, algebricamente a fatoração de $p(x)$ impossível, e ainda quanto maior o grau do polinômio ou maior a quantidade de sistemas a serem resolvidos também caímos em um custo computacional enorme, onde precisamos recorrer à computação de alto desempenho. Como o objetivo aqui é exemplificar, esses casos mais complexos não serão tratados nesta monografia.

4.3 Polinômios Irredutíveis

Esta seção está voltada para os resultados envolvendo polinômios irredutíveis sobre corpos. Veremos mais adiante que polinômios irredutíveis são de fundamental importância para construção de corpos.

Quando K é um corpo, podemos verificar facilmente que $K[x]$ munido das operações soma e produto definidas na seção 4.1 é um Domínio de Integridade, onde o polinômio $p(x) = 0$ é o elemento neutro e $q(x) = 1$ é a unidade de $K[x]$.

Seja K um corpo e $K[x]$ o domínio de integridade dos polinômios sobre K na indeterminada x. Vamos agora enunciar alguns resultados importantes envolvendo polinômios. O primeiro deles é o algoritmo da Divisão para polinômios.

Todas as demonstrações dos resultados aqui citados, podem ser encontradas em [1], [2] e [8].

Teorema 10. *(Algoritmo da Divisão)*

Sejam $f(x), g(x) \in K[x]$ e $g(x) \neq 0$. Então, existem únicos $q(x), r(x) \in K[x]$ tais que $f(x) = q(x)g(x) + r(x)$, onde $r(x) = 0$ ou $\partial(r(x)) < \partial(g(x))$.

Proposição 2. *Seja K um corpo e seja $f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ um polinômio não nulo em $K[x]$ de grau n. Então, o número de raízes de $f(x)$ em K é no máximo igual a $\partial(f(x)) = n$.*

Corolário 11. *Seja $f(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ um polinômio não nulo de grau n em $K[x]$. Então, $f(x)$ possui no máximo n raízes em qualquer extensão L de K.*

Um **ideal principal** de $K[x]$ é o conjunto dos múltiplos de um elemento $p(x) \in K[x]$, ou seja,

$$J = K[x] \cdot p(x) = \{f(x)p(x) : f(x) \in K[x]\}.$$

Todo ideal de $K[x]$ é principal, conforme [2]. Vamos relembrar o que seria polinômios irredutíveis sobre um corpo K.

Seja um polinômio não constante $f(x)$, dizemos que $f(x)$ é **irredutível** em $K[x]$, se não é possível expressar $f(x)$ como produto de outros dois polinômios não constantes, ou seja, $f(x) = g(x).h(x)$, com $g(x)$,$h(x) \in K[x]$, onde $\partial g(x), \partial h(x) \geq 1$. Caso contrário dizemos que $f(x)$ é redutível sobre K.

Exemplos:

• Considerando o corpo dos números complexos $(\mathbb{C}, +, \cdot)$, temos que o polinômio $p(x) = x^2 + 1$ pode ser fatorado da seguinte forma:

$$x^2 + 1 = (x + i)(x - i)$$

Entretanto sobre o corpo dos números reais $(\mathbb{R}, +, \cdot)$ esse polinômio é irredutível.

• Consirere o corpo dos números inteiros módulo 2 $(\mathbb{Z}_2, +, \cdot)$, temos que o polinômio $p(x) = x^2 + x + 1$ é irredutível sobre $\mathbb{Z}_2$.

5 Construção e Caracterização de Corpos Finitos

Neste capítulo, são apresentados alguns conceitos importantes, como característica de um Corpo Finito. Veremos que é possível construir um corpo, e para tal, utilizamos dois teoremas importantes, um que permite construir um corpo através de um polinômio irredutível, e outro, que permite construir corpo por adjunção de raízes. Também caracterizamos os corpos finitos, garantindo assim a existência deles. Vimos que de maneira geral para se construir um corpo é necessário um polinômio irredutível, sendo assim, utilizamos o algoritmo de Rabin para encontrar polinômios irredutíveis sobre corpos finitos.

5.1 Característica de Corpo Finito

Vamos estudar a característica de um corpo finito. Muitos dos resultados presentes nesta seção são usados para caracterização de corpos finitos.

Seja A um anel. Dizemos que a característica de um anel A é a ordem do elemento 1 no grupo aditivo de A.

Sendo assim, A possui característica p, se $p \cdot 1 = 0$ e $m \cdot 1 \neq 0$ para $1 \leq m < p$. Se a característica de A não é finita, dizemos que A possui característica zero.

Exemplos: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ e $\mathbb{C}$ possuem característica 0 e $\mathbb{Z}_p$ tem característica p.

Seja $\mathbb{F}_p$ um corpo com p elementos. Dizemos que $\mathbb{F}_p$ é um corpo primo se ele não contêm subcorpos próprios.

Vejamos dois resultados importantes envolvendo característica de um corpo, onde as demonstrações podem ser encontradas em [8]:

Teorema 12. *Todo corpo finito possui característica p, onde p é um número primo.*

Teorema 13. *Todo subcorpo primo K de um corpo F é isomorfo a $\mathbb{F}_p$ ou a $\mathbb{Q}$, e assim, a característica de K é p ou 0, onde p é um número primo.*

5.2 Construção de Corpos Finitos

Veremos agora dois resultados importantes que permitem construir corpos. O primeiro nos permite construir um corpo através de um polinômio irredutível.

Teorema 14. *Sejam K um corpo e $p(x) \in K[x]$. Então, as seguintes afirmações são equivalentes:*

1. *$p(x)$ é irredutível sobre K;*
2. *$J = K[x] \cdot p(x)$ é um ideal maximal em $K[x]$;*
3. *$K[x]/J$ é um corpo, onde $J = K[x] \cdot p(x)$.*

Antes da demonstração do teorema, observe que, claramente temos o seguinte resultado: $f(x) \in J$ se, e somente se, $f(x) + J = J$.

Demonstração:

Primeiramente vamos verificar a primeira equivalência (1) $\Leftrightarrow$ (2).

(1) $\Rightarrow$ (2)

Seja $p(x) \in K[x]$, tal que $p(x)$ é irredutível sobre K e considere o conjunto:

$$J = K[x] \cdot p(x) = \{g(x)p(x) | g(x) \in K[x]\}$$

Como $\partial(p(x)) \geq 1$, temos que $J \neq K[x]$. Seja I um ideal de $K[x]$, tal que $J \subset I$, como todo ideal de $K[x]$ é principal, temos que

$$I = K[x] \cdot h(x)$$

Observe que $p(x) \in J$ e $J \subset I$, portanto $p(x) = q(x)h(x)$, para algum $q(x) \in K[x]$. Entretanto, $p(x)$ é irredutível, logo $q(x) = a$ ou $h(x) = b$, onde $a, b \in K - \{0\}$.

• Se $q(x) = a$, temos que $p(x) = ah(x) \Rightarrow h(x) = a^{-1}p(x)$ e assim, $I = K[x] \cdot h(x) \subset K[x] \cdot p(x) = J \Rightarrow I \subset J \Rightarrow I = J$.

• Se $h(x) = b$, temos $I = K[x] \cdot h(x) = K[x]$.

(2) $\Rightarrow$ (1)

Seja $J = K[x] \cdot p(x)$ um ideal maximal em $K[x]$. Por definição de ideal maximal, temos que $J \neq K[x]$, daí $\partial(p(x)) \geq 1$.

Suponha que $p(x)$ é da seguinte forma:

$$p(x) = g(x)h(x),$$

para algum $h(x) \in K[x]$.

Tomando $I = K[x] \cdot h(x)$, temos que $p(x) \in I$ e portanto $J \subset I$. Como J é maximal, temos que $I = J$ ou $I = K[x]$.

• Se $I = J$, então $h(x) \in J$, logo $h(x) = f(x)p(x)$ para algum $f(x) \in K[x]$. Portanto,

$$p(x) = g(x)f(x)p(x)$$

Observe que $p(x) \neq 0$ e $K[x]$ é domínio de integridade, logo,

$$1 = g(x)f(x)$$

Portanto $g(x) \in K[x]$ é um polinômio invertível em $K[x]$, assim $g(x) = a \neq 0$, logo $p(x)$ é irredutível sobre K.

• Se $I = K[x]$ segue de imediato que $h(x) = b \neq 0$, ou seja, $p(x)$ é irredutível sobre K

Agora é suficiente mostrar (2) $\Leftrightarrow$ (3).

(2) $\Rightarrow$ (3)

Suponha que $J = K[x] \cdot p(x)$ seja um ideal maximal de $K[x]$. E seja $\overline{f(x)} \in K[x]/J$, onde $\overline{f(x)} \neq \bar{0}$. Vamos provar que existe um $g(x) \in K[x]/J$, tal que, $\overline{f(x)}\,\overline{g(x)} = \bar{1}$.

Considere $I = K[x] \cdot f(x)$ um ideal de $K[x]$, sendo assim, $J + I$ também é um ideal de $K[x]$. Observe que $\overline{f(x)} \neq \bar{0}$, se, e somente se, $f(x) \notin J$ e ainda que, $f(x) = 1f(x) \in I$, portanto $J \subset J + I$ e $J \neq J + I$. Como J é ideal maximal, temos o seguinte, $K[x] = J + I$, logo $1 \in J + I$, ou seja, existe $r(x) \in J$ e $s(x) \in I$, tal que, $1 = r(x) + s(x)$. Como $s(x) \in I$, temos $s(x) = g(x)f(x)$, para algum $g(x) \in K[x]$. Daí, $1 = r(x) + g(x)f(x)$, olhando para classe de congruência módulo J temos:

$$\bar{1} = \overline{r(x) + g(x)f(x)} = \overline{r(x)} + \overline{g(x)f(x)} = \bar{0} + \overline{g(x)}\,\overline{f(x)} \Rightarrow \bar{1} = \overline{g(x)}\,\overline{f(x)}$$

(3) $\Rightarrow$ (2)

Seja $K[x]/J$ um corpo. Assim, $\bar{0}$, $\bar{1} \in K[x]/J$ e, ainda, $J \neq K[x]$.

Considere $I \neq J$ um ideal de $K[x]$, tal que, $J \subset I \subset K[x]$. Tomemos $f(x) \in I$ tal que $f(x) \notin J$, ou seja, $f(x) \in K[x]/J$ e $\overline{f(x)} \neq \bar{0}$. Como $K[x]/J$ é um corpo, existe $\overline{g(x)} \in K[x]/J$, tal que, $\overline{f(x)}\,\overline{g(x)} = \bar{1}$, ou ainda,

$f(x)g(x) \equiv 1 \pmod{J} \Rightarrow f(x)g(x) - 1 \in J \Rightarrow \exists h(x) \in J$ tal que $f(x)g(x) - 1 = h(x) \Rightarrow 1 = f(x)g(x) - h(x)$.

Observe que, se $f(x) \in I$ então $f(x)g(x) \in I$ e se, $h(x) \in J \subset I$, logo, $h(x) \in I$. Daí, $1 = f(x)g(x) - h(x) \in I \Rightarrow 1 \in I \Leftrightarrow I = K[x]$.

Portanto, J é um ideal maximal de $K[x]$.

■

Para exemplificar vamos construir corpos de 4 e 8 elementos:

Corpo com 4 elementos:

Seja $K = \mathbb{Z}_2 = \{\bar{0}, \bar{1}\}$ um corpo e seja ainda, $p(x) = x^2 + x + 1$, $p(x) \in K[x]$.

Observe que $p(x)$ é irredutível sobre K, de fato:

$$p(0) = 1 \; e \; p(1) = 3 = 1$$

• $p(x)$ é irredutível sobre K.

De acordo com o Teorema anterior, temos o seguinte:

• $J = K[x] \cdot p(x)$ é um ideal maximal de $K[x]$;

• $L = K[x]/J = \{f(x) + J | f(x) \in K[x]\}$ é um corpo.

Vamos exibir os elementos de L.

Seja $f(x) \in K[x]$ um representante da classe $f(x) + J$, vamos dividir $f(x)$ por $p(x)$, sendo assim, pelo algoritmo da divisão, existem $q(x), r(x) \in K[x]$, tais que:

$$f(x) = r(X) + q(x)p(x);$$

onde $r(x) = 0$ ou $\partial(r(x)) < \partial(p(x))$.

Agora, olhando para classe de equivalência de $f(x)$ temos o seguinte:

$f(x) + J = r(x) + q(x)p(x) + J \Rightarrow f(x) + J = r(x) + J \Rightarrow f(x) + J = ax + b + J$. Logo,

$$L = \{ax + b + J | a, b \in K\}$$

Analisando as possíveis possibilidades para a e b temos:

a	b
0	0
1	0
0	1
1	1

Portanto, $L = \{J, x + J, 1 + J, x + 1 + J\}$ é um corpo contendo exatamente 4 elementos.

Corpo com 8 elementos

Seja $K = \mathbb{Z}_2 = \{\bar{0}, \bar{1}\}$ um corpo e seja ainda, $p(x) = x^3 + x + 1$, $p(x) \in K[x]$. Temos que $p(x)$ é irredutível, e, de maneira análoga o procedimento anterior, podemos construir um corpo.

Seguindo o processo demonstrado no item anterior, chegaremos no seguinte resultado:

$$L = K[x]/J = \{J, 1+J, x+J, x+1+J, x^2+J, x^2+x+J, x^2+1+J, x^2+x+1+J\},$$

onde $J = K[x] \cdot p(x)$ é um ideal maximal de $K[x]$.

O Teorema que acabamos de ver nos dá uma ótima ferramenta para construir um corpo finito, basta apenas obter um polinômio irredutível sobre um determinado corpo K. Agora vamos conhecer uma segunda maneira de construir um corpo, mas antes disso, precisamos de algumas definições.

Dizemos que $\alpha \in L$ é algébrico sobre K se existe $f(x) \in K[x] - \{0\}$ tal que $f(\alpha) = 0$. Caso contrário dizemos que α é transcendente sobre K.

Se todo $\alpha \in L \supset K$ é algébrico sobre K, então dizemos que $L \supset K$ é uma extensão algébrica.

Definição 15. *Seja $\alpha \in L$ algébrico sobre um corpo $K \subset L$. O único polinômio de menor grau entre os polinômios $f(x)$ em $K[x]$ satisfazendo:*

1. *$f(\alpha) = 0$.*
2. *$f(x)$ é mônico.*

é chamado o polinômio irredutível de α sobre K e denotado por $irr(\alpha, K)$.

O seguinte resultado é importante, pois usamos suas equivalências para construir corpos finitos. A demonstração segue diretamente de [2].

Teorema 16. *Se $\alpha \in L \supset K$ e se*

$$\begin{aligned} \phi : K[x] &\to L \\ f(x) &\mapsto f(\alpha) \end{aligned}$$

então, ϕ é um homomorfismo tal que:

1. *$Im(\phi) = K[\alpha]$, $K \subset K[\alpha] \subset L$.*

2. *α é transcendente sobre K se, e somente se, $Ker(\phi) = \{0\}$.*

3. *Se α é algébrico sobre K e $p(x) = irr(\alpha, K)$ então $Ker(\phi) = K[x] \cdot p(x)$ é um ideal maximal de $K[x]$.*

4. *$K[x]/Ker(\phi) \simeq K[\alpha]$.*

Corolário 17. *Seja $\alpha \in L \supset K$.*

1. *Se α é algébrico sobre K então $K[\alpha]$ é um subcorpo de L que contém K.*

2. *Se α é transcendente sobre K então $K[\alpha]$ é um subdomínio de L isomorfo ao domínio $K[x]$ dos polinômios na indeterminada x.*

Corolário 18. *Se $\alpha, \beta \in L \supset K$ são raízes de um mesmo polinômio irredutível sobre K, então, $K[\alpha]$ e $K[\beta]$ são corpos isomorfos.*

As demonstrações dos resultados acima podem ser encontradas em [1].

Seja α algébrico sobre K e seja $p(x)$ um polinômio mônico de menor grau em $K[x]$, tal que $p(\alpha) = 0$. Pela minimalidade do grau de $p(x)$, temos claramente que $p(x)$ é o único polinômio mônico irredutível sobre K, tal que, $p(\alpha) = 0$, o qual denotaremos por $p(x) = irr(\alpha, K)$.

A seguinte proposição nos dá uma ideia de como construir corpo fazendo adjunção de raízes.

Proposição 3. *Seja $L \supset K$, $\alpha \in L$ algébrico sobre K. Se o grau do polinômio $irr(\alpha, K)$ é n, então:*

1. *$\forall f(x) \in K[x]$, $f(\alpha)$ pode ser expresso de modo único na forma $f(\alpha) = a_0 + a_1\alpha + ... + a_{n-1}\alpha^{n-1}$, onde $a_i \in K$.*

2. *$K[\alpha] = \{a_0 + a_1\alpha + ... + a_{n-1}\alpha^{n-1} : a_i \in K\}$ é um subcorpo de L que contém K.*

3. *se $K = \mathbb{Z}_p$ então $K[\alpha]$ é um corpo contendo exatamente p^n elementos.*

Demonstração: Considere $p(x) = irr(\alpha, K)$. Por hipótese, temos que o grau de $p(x)$ é n, ou seja, $\partial(p(x)) = n$. Vamos demonstrar os itens 1, 2 e 3. De fato,

1) Seja $f(x) \in K[x]$, pelo Algoritmo da Divisão existem $q(x), r(x) \in K[x]$ tais que

$$f(x) = q(x) \cdot p(x) + r(x),$$

onde, $r(x) = 0$ ou $\partial(r(x)) < \partial(p(x))$. Desse modo,

$$r(x) = a_0 + a_1x + ... + a_{n-1}x^{n-1} \;\; a_i \in K \; i \in \{0, ..., n-1\}$$

Observe que

$$f(\alpha) = q(\alpha) \cdot p(\alpha) + r(\alpha)$$

Como α é algébrico em K tem-se que $p(\alpha) = 0$ e daí

$$f(\alpha) = r(\alpha) = a_0 + a_1\alpha + ... + a_{n-1}\alpha^{n-1}.$$

Para mostrar a unicidade vamos supor que $f(x)$ não se escreve de maneira única, ou seja,

$$f(\alpha) = a_0 + a_1\alpha + ... + a_{n-1}\alpha^{n-1} \; f(\alpha) = b_0 + b_1\alpha + ... + b_{n-1}\alpha^{n-1}$$

com $b_i, a_i \in K$. Como $f(\alpha) = f(\alpha)$, temos o seguinte:

$$a_0 + a_1\alpha + ... + a_{n-1}\alpha^{n-1} = b_0 + b_1\alpha + ... + b_{n-1}\alpha^{n-1}$$

e daí,

$$(a_0 - b_0) + (a_1 - b_1)\alpha + (a_{n-1} - b_{n-1})\alpha^{n-1} = 0$$

Seja $q(x) \in K[x]$, tal que

$$q(x) = (a_0 - b_0) + (a_1 - b_1)x + (a_{n-1} - b_{n-1})x^{n-1}$$

Observe que $q(\alpha) = 0$ e $\partial(q(x)) = n-1 < n$. O que é um absurdo, pois α é algébrico em K, que por definição, $p(x)$ é o único polinômio mônico de menor grau satisfazendo $p(\alpha) = 0$. Portanto $q(x) = 0$ e, consequentemente, $a_i = b_i, i \in \{0, ..., n-1\}$.

2) Este item é consequência direta do item 1).

3) Considere $K = \mathbb{Z}_p$, pelo item 1), temos:

$$\mathbb{Z}_p[\alpha] = \{a_0 + a_1\alpha + ... + a_{n-1}\alpha^{n-1} | a_i \in \mathbb{Z}_p\}$$

Observe que$\forall \; a_i \in \mathbb{Z}_p = \{\bar{0}, \bar{1}, ..., \overline{p-1}\}$, temos p possibilidades para a_i. Portanto, para cada n-upla do conjunto $\{a_0, a_1, ..., a_n - 1\}$ é possível estabelecer uma correspondência bijetiva com $\mathbb{Z}_p$, assim ,$\mathbb{Z}_p[\alpha]$ tem p^n elementos.

■

Corpo de 4 elementos:

Seja $K = \mathbb{Z}_2$ um corpo e L uma extensão algébrica sobre K. Seja $p(x) = x^2+x+1 \in K[x]$, tal que, $p(x)$ é o $irr(\alpha, K)$ para algum $\alpha \in L$. Então, de acordo com a Proposição , temos o seguinte:

$K[\alpha] = \{a_0 + a_1\alpha : a_1, a_2 \in K\}$ é um corpo contendo exatamente 4 elementos.

Observe que de acordo com o Teorema 16, $K[\alpha]$ é isomorfo ao corpo $K[x]/J$ com 4 elementos construído anteriormente, onde J é um ideal maximal gerado por um polinômio irredutível.

Polinômios irredutíveis são de fundamental importância para construção de corpos, aliados ao Teorema 14 temos uma ótima ferramenta que permite construir um corpo finito. Nos exemplos dados é fácil identificar um polinômio irredutível sobre um corpo K dado, e assim, construir um novo corpo a partir dele, entretanto, nem sempre é fácil encontrar um polinômio irredutível sobre um determinado corpo K. Para isso, na seção seguinte vamos exibir um algoritmo que permite encontrar um polinômio irredutível sobre um corpo finito qualquer, facilitando assim a construção de novos corpos.

5.3 Caracterização de Corpos Finitos

Nesta seção, mostraremos a existência e unicidade dos corpos finitos, caracterizando assim, os corpos finitos.

Se uma extensão V sobre um corpo K possui uma base com n elementos, dizemos que n é a dimensão de V sobre K e denotamos por $[V : K] = n$. Se K é um corpo qualquer e se L é uma extensão de K, dizemos que a extensão $L \supset K$ é finita se $[L : K] = n < \infty$. Caso contrário, dizemos que $L \supset K$ é uma extensão infinita.

Lema 19. *Seja F um corpo finito contendo um subcorpo K com q elementos. Então F possui q^m elementos, onde $m = [F : K]$.*

Demonstração: Seja F uma extensão vetorial do corpo K. Sabemos que F é finito, assim, seja $[F : K] = m$, logo F possui uma base sobre K contendo m elementos, seja $\beta = \{b_1, b_2, ..., b_m\}$ esta base. Cada elemento $v \in F$ é expresso de maneira única da forma: $a_1b_1 + a_2b_2 + ...a_mb_m$, onde $a_1, a_2, ..., a_m \in K$. Para cada $a_i \in K$, a_i pode assumir q valores, pois K possui q elementos, sendo assim, F possui exatamente q^m elementos.

■

Teorema 20. *Seja F um corpo finito. Então F possui p^n elementos, onde p é a característica de F e n é o grau de F sobre um subcorpo primo.*

Demonstração: Seja F um corpo finito. De acordo com o Teorema 12 da seção anterior, temos que F possui característica p, onde p é um número primo.

Considere o subcorpo primo K de F, sabemos que K é isomorfo a F_p pelo Teorema 16 da seção anterior que contém exatamente p elementos. Sendo assim, usando o lema anterior, F possui p^n elementos.

■

Lema 21. *Se F é um corpo finito com q elementos, então, para todo $a \in F$ satisfaz $a^q = a$.*

Demonstração: Observe inicialmente que para $0 \in F$ a identidade $a^q = a$ é satisfeita. Considere $G = F - \{0\}$, G é um grupo multiplicativo com $q-1$ elementos, ou seja, $|G| = q-1$, logo, para todo $a \in G$, temos $a^{q-1} = 1$, e multiplicando esta última igualdade por a temos, $a^q = a$, o que demonstra o resultado.

■

Lema 22. *Se F é um corpo finito com q elementos e K é um subcorpo de F, então o polinômio $x^q - x$ em $K[x]$ pode ser fatorado em $F[x]$ como:*

$$x^q - x = \prod_{a \in F} (x - a) \tag{5.1}$$

e F é um corpo de decomposição de $x^q - x$ sobre K.

Teorema 23 (Existência e Unicidade)**.** *Para todo primo p e todo inteiro positivo n existe um corpo finito com p^n elementos. Qualquer corpo finito com $q = p^n$ elementos é isomorfo ao corpo de decomposição de $x^q - x$ sobre F_p.*

Demonstração: (Existência) Seja $q = p^n$, considere o polinômio $p(x) \in F_q[x]$, tal que $p(x) = x^q - x$ e seja F o corpo de decomposição de $p(x)$ sobre F_q. Sabemos que $p(x)$ possui q raízes distintas em F, considere então $p'(x) = qx^{q-1} - 1$, onde $p'(x) \in F_q[x]$. Observe que $p'(x)$ e $p(x)$ não possui raízes em comum.

Seja $S = \{a \in F : a^q - a = 0\}$, S é um subcorpo de F, de fato:

1. $0, 1 \in S$;
2. Seja $a, b \in S$, $(a-b)^q = a^q - b^q = a - b$, e então $a - b \in S$.
3. $a, b \in S$, e $b \neq 0$, temos que $(ab^{-1})^q = a^q(b^{-1})^q = ab^{-1}$, e assim, $ab^{-1} \in S$.

Entretanto, $x^q - x$ se decompõem em S, pois S contêm todas as suas raízes. Logo $F = S$, e como S possui q elementos, F é um corpo finito com q elementos.

(Unicidade) Seja F um corpo finito com $q = p^n$ elementos. Logo, F possui Característica p, com p primo. De acordo com o Teorema 14, F contém algum subcorpo primo F_p. Usando o Lema 16, temos que F é um corpo de decomposição de $x^q - x$ sobre F_p, portanto esse resultado segue do Teorema da existência e unicidade de corpos de decomposição, que diz o seguinte, se K é um corpo e seja $p(x) \in F_p$ um polinômio não constante, então existe um corpo de decomposição de $p(x)$ sobre F_q.

■

As demonstrações dos próximos resultados podem ser encontradas em [8].

Teorema 24 (Critério para subcorpo)**.** *Seja F_q um corpo finito com $q = p^n$ elementos. Então todo subcorpo de F_q tem ordem p^m, onde m é um divisor positivo de n. Reciprocamente, se m é um divisor positivo de n, então existe exatamente um subcorpo de F_q com p^m elementos.*

Teorema 25. *Para todo corpo finito F_q o grupo mutiplicativo F_q^* é cíclico.*

Corolário 26. *Para todo corpo finito F_q e todo inteiro positivo n, existe um polinômio irredutível em $F_q[x]$ de grau n.*

5.4 Polinômios Irredutíveis sobre Corpos Finitos

Um dos principais problemas desta monografia é construir corpos finitos, e vimos que uma das melhores formas de construir um corpo finito é usando polinômios irredutíveis. Sabemos também que sempre existe um polinômio irredutível sobre um corpo finito, nos resta agora determinar esses polinômios irredutíveis.

Para tal, vamos considerar dois teoremas importantes que justificarão o algoritmo na qual usaremos para encontrar polinômios irredutíveis. As demonstrações podem ser encontradas em [5], [6] e [7].

Teorema 27. *Considere $\mathbb{F}_q$ um corpo finito com q elementos. Seja $f(x) \in \mathbb{F}_q[x]$ um polinômio irredutível sobre $\mathbb{F}_q$ de grau n, então $f(x)$ divide $(x^{q^k} - x)$ se, e somente se, n divide k.*

Teorema 28. *Para cada corpo finito $\mathbb{F}_q$ e cada $k \in \mathbb{N}$, o produto de todos os polinômios mônicos irredutíveis sobre $\mathbb{F}_q$ cujo grau divide k é igual a $x^{q^k} - x$.*

Uma consequência dos teoremas citados é o Algoritmo de Rabin, que permite encontrar um polinômio $p(x)$ irredutível sobre um corpo finito $\mathbb{F}_q$, onde $q = p^n$.

• O algoritmo de Rabin consiste basicamente em três passos:

1. Gerar um polinômio mônico $g(x)$ aleatório de grau d sobre o corpo finito $\mathbb{F}_q$;

2. Verificar se $g(x)$ divide $(x^q - x)$;

3. Testar se o máximo divisor comum entre $g(x)$ e $(x^{p^{n_i}} - x)$ é igual a 1 para todo $n_i = \frac{n}{k_i}$ onde k_i são todos os divisores primos de n.

Se esses três passos obtiverem sucesso significa que encontramos um polinômio irredutível sobre $\mathbb{F}_q$.

Implementamos esse Algoritmo computacionalmente com o GAP, cujo código se encontra no Apêndice B. Para exemplificar o Algoritmo, vamos determinar um polinômio irredutível sobre um corpo finito.

Primeiramente, vamos considerar um corpo finito no GAP, tomando $q = 2^4$, vamos trabalhar com o corpo $\mathbb{F}_q$.

```
gap> q:=16;
16
gap> Fq:=GF(q);
GF(2^4)
```

A função GF("parâmetro") retorna um corpo finito com p^n elementos, onde p é um número primo, logo o "parâmetro" é uma potência de um número primo.

Vamos definir agora a variável indeterminada "x" na qual usaremos para trabalhar com polinômios.

```
gap> x:=X(Fq,"x");
x
```

Agora, é suficiente executar o algoritmo para encontrar o polinômio irredutível sobre o corpo F_q, para tal, precisamos efetuar a leitura da função.

```
gap> Read("/home/tulioaltoe/RabinOtimizado.g");
gap> repeat
> t:=Rabin(Fq,2);
> until t<>0;
x^2+Z(2^2)*x+Z(2^2)
```

Portanto $t(x)$ é um polinômio irredutível sobre F_q.

```
gap> t;
x^2+Z(2^2)*x+Z(2^2)
```

Podemos verificar agora, se o polinômio encontrado satisfaz o Teorema 21, observe que $t(x)$ tem grau dois e tomando $k = 6$, temos:

```
gap> k:=6;
```

```
6
gap> pol:= x^(q^k) - x;
x^16777216+x
gap> pol mod t;
0*Z(2)
gap> Int(0*Z(2));
0
```

Portanto $t(x)$ é o nosso polinômio irredutível, e a partir dele podemos construir um novo corpo $F_q[x]/J$, onde $J = F_q[x] \cdot t(x)$ é um ideal maximal.

6 Conclusão

O trabalho no qual encerramos, constituiu no esforço de estudar alguns resultados clássicos da Álgebra, como a classificação de grupos e construção e caracterização de corpos finitos.

Em virtude da teoria de grupos ser muito ampla, nos focamos na classificação de grupos a menos de isomorfismo. Classificamos algebricamente os grupos de ordem 4, 6 e 8 e desenvolvemos uma aplicação que lista e permite operar e trabalhar com determinados grupos e utilizamos o método de Kronecker para fatorar um polinômio sobre um domínio de integridade.

Construímos corpos a partir de polinômios irredutíveis, e ainda, por adjunção de raízes. Vimos na sua caracterização que dado um corpo finito, o mesmo sempre terá p^n elementos, onde p é um número primo e n um natural qualquer, e que, dado um primo p e um número natural n, sempre existirá um corpo finito com p^n elementos. Garantimos também, que sempre existe um polinômio irredutível sobre um corpo finito, possibilitando assim a construção de novos corpos.

Em geral, não é fácil achar um polinômio irredutível sobre um corpo, como garantimos sua existência, utilizamos o algoritmo de Rabin para determinar polinômios irredutíveis sobre um corpo finito.

Ao concluirmos este trabalho, podemos afirmar que nossos objetivos foram alcançados e esperamos que o mesmo possa ser de utilidade para pesquisadores e alunos que queiram seguir na Álgebra computacional, podendo tomar os resultados aqui apresentados como base.

Com a realização deste trabalho, idealizamos também novas propostas para trabalhos futuros, como por exemplo, o estudo de métodos que possibilitam fatorar polinômios sobre corpos finitos, cálculo de raízes de polinômios sobre corpos finitos, e é claro que, seguindo a mesma analogia deste trabalho, vamos implementar com o GAP e aplicar seus resultados.

Referências

1 GONÇALVES, A. **Introdução à Álgebra.** Rio de Janeiro: IMPA, 1999. Citado 6 vezes nas páginas 26, 27, 46, 48, 53 e 59.

2 GARCIA, A.; LEQUAIN, Y. **Elementos de Álgebra.** Rio de Janeiro: IMPA, 2007. Citado 4 vezes nas páginas 26, 46, 53 e 58.

3 The GAP Groups, GAP-Groups, Algorithms, and Programming, Version 4.4.12; 2008. (http://www.gap-system.org/). Citado na página 13.

4 SCHNEIDER, Csaba. Notas - Minicurso sobre o sistema Computacional GAP, Escola de Álgebra, UEM, Maringá/PR, 2014. (http://www.mat.ufmg.br/∼csaba/GAPMaringa/chap1_mj.html). Citado na página 20.

5 MASUDA, A. e PANARIO, D., *Tópicos de Corpos Finitos com Aplicações em Criptografia e Teoria de Códigos*. IMPA, Rio de Janeiro, 2007. Citado na página 63.

6 ZANOELLO, S. F. Raízes Polinomiais em Corpos Finitos, Dissertação de Mestrado, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2004. Citado na página 63.

7 ZATESKO, L. M. Irredutibilidade Polinomial e Algoritmos em Corpos Finitos, Monografia de Bacharelado em Ciência da Computação, Universidade Federal do Paraná, Curitiba, 2008. Citado na página 63.

8 RUDOLF, L. E HARALD, N. Introduction to finite fields and their applications. Cambridge University Press, New York, 1994. Citado 4 vezes nas páginas 11, 53, 54 e 63.

9 GERHARD, O. An analytic approach to the Collatz 3n + 1 Problem, Universitaet Hamburg, May 2011. Citado 2 vezes nas páginas 22 e 23.

10 DAMAZIO, M. F. Classificação de Alguns Grupos de Ordem Finita. Universidade do Estado de Santa Catarina, Joinville, 2012. Citado 2 vezes nas páginas 31 e 35.

11 JOSEPH, H. S. Wieferich's Criterion and abc-Conjecture. January, 1988. Citado na página 20.

12 WILHIAM, T. H. **Graduate Texts in Mathematics**. New YorK, Springer, 12ª ed., 2003. Citado na página 31.

13 BIAZZI, R. N. Polinômios Irredutíveis: Critérios e Aplicações. Universidade Estadual Paulista, 2014. Citado na página 48.

APÊNDICE A – Método de Kronecker

Segue abaixo o algoritmo do Método de Kronecker utilizado no Capítulo 3.

Algoritmo 1: Método de Kronecker

```
1  KroneckerGAP:= function(p,beta,phi,gama)
2     #Declarações de variáveis
3     local A,b,z,X,aux,d1,d2,d3,divD1,divD2,matrizsolucao,
4     divD3,n,n1,n2,n3,L,U,tamanhotabela,i,j,k,s,t,w,m,pren,
5     teste,valor,soma,cont,tamanhofatores,fatores;
6     #Inicializando os vetores
7     b:=[];
8     z:=[];
9     X:=[];
10    fatores:=[];
11    #Inicializando a Matriz
12    A:= [[gama^2, gama , 1],[phi^2,phi,1],[beta^2, beta, 1]];
13    #imprimindo a matriz
14    Print(``\n A matriz A:\n'');
15    PrintArray(A);
16    Print(``\n'');
17    #Avaliando os valores no polinômio
18    d1:= Value(p,beta);
19    d2:= Value(p,phi);
20    d3:= Value(p,gama);
21    Print(``\n Divisores: \n'');
22    Print(``\n d1='',d1);
23    Print(``\n d2='',d2);
24    Print(``\n d3='',d3);
25    Print(``\n'');
26    if d1 <> 0 then
27       divD1:= DivisorsInt(d1);
28    fi;
29    if d2 <> 0 then
30       divD2:= DivisorsInt(d2);
31    fi;
32    if d3 <> 0 then
33       divD3:= DivisorsInt(d3);
```

```
34      fi;
35      Print(``\n Divisores do divisor: \n'');
36      Print(``\n divD1='',divD1);
37      Print(``\n divD2='',divD2);
38      Print(``\n divD3='',divD3);
39      Print(``\n'');
40      divD1:=Concatenation(divD1,-divD1);
41      divD1:=Set(divD1);
42      divD2:=Concatenation(divD2,-divD2);
43      divD2:=Set(divD2);
44      divD3:=Concatenation(divD3,-divD3);
45      divD3:=Set(divD3);
46      Print(``\n Divisores com a parte negativa: \n'');
47      Print(``\n divD1='',divD1);
48      Print(``\n divD2='',divD2);
49      Print(``\n divD3='',divD3);
50      Print(``\n'');
51      tamanhotabela:= Length(divD1)*Length(divD2)*Length(divD3);
52      Print(``\n tamanho tabela:'', tamanhotabela);
53      Print(``\n'');
54      n1:=Length(divD1);
55      n2:=Length(divD2);
56      n3:=Length(divD3);
57      n:= Length(A);
58      matrizsolucao:=NullMat(tamanhotabela,n);
59      cont:=1;
60          for i in [1..n1] do
61              for j in [1..n2] do
62                  for k in [1..n3] do
63                      b:=[divD3[k],divD2[j],divD1[i]];
64                      aux:=[divD3[k],divD2[j],divD1[i]];
65                      L:=IdentityMat(n);
66                      for s in [1..n-1] do
67                          for t in [s+1..n] do
68                              m:= (A[t][s]/A[s][s]);
69                              L[t][s]:= m;
70                              b[t]:= b[t] - m*b[s];
71                                  for w in [s..n] do
72                                      A[t][w] := A[t][w] - m*A[s][w];
```

```
73                          od;
74                      od;
75                  od;
76                  U:=IdentityMat(n);
77                  U:=A;
78                  A:= L * U;
79                  z[1]:=b[1];
80                  for s in [2..n] do
81                      soma:=0;
82                      for t in [1..s-1] do
83                          soma := soma + L[s][t] * z[t];
84                      od;
85                      z[s]:= aux[s] - soma;
86                  od;
87                  X[n]:= (z[n] / U[n][n]);
88                  for s in Reversed([1..n-1]) do
89                      soma:=0;
90                      for t in [s+1..n] do
91                          soma := soma + U[s][t] * X[t];
92                      od;
93                          X[s] := ((z[s] - soma) / U[s][s]);
94                  od;
95                  for s in [1..n] do
96                      matrizsolucao[cont][s]:= X[s];
97                  od;
98                  cont:=cont + 1;
99              od;
100         od;
101     od;
102 Print(``\n\tD3\tD2\tD1\ta\tb\tc\n'');
103 cont:=1;
104     for i in [1..n1] do
105         for j in [1..n2] do
106             for k in [1..n3] do
107                 Print("----------------------------------
108                                                 ----------------------'');
109                 Print(``\n'');
110                 Print(cont);
111                 Print(``\t'');
```

```
112                 Print(divD3[k]);
113                 Print(``\t'');
114                 Print(divD2[j]);
115                 Print(``\t'');
116                 Print(divD1[i]);
117                 Print(``\t'');
118                 Print(matrizsolucao[cont][1]);
119                 Print(``\t'');
120                 Print(matrizsolucao[cont][2]);
121                 Print(``\t'');
122                 Print(matrizsolucao[cont][3]);
123                 Print(``\n'');
124                 cont:=cont+1;
125             od;
126         od;
127     od;
128     for s in [1..tamanhotabela] do
129         if matrizsolucao[s][1] = 1 then
130             Add(fatores, matrizsolucao[s][1]*x^2 +
131                     matrizsolucao[s][2]*x + matrizsolucao[s][3]);
132         fi;
133     od;
134     Print(``\n'');
135     Print(`` Possíveis divisores de p(x):\n'');
136     PrintArray(fatores);
137     tamanhofatores:= Length(fatores);
138     for s in [1..tamanhofatores] do
139         for t in [1..tamanhofatores] do
140             if fatores[s]*fatores[t] = p then
141                 Print(``\n A fatoração é dada por: \n'');
142                 Print(p);
143                 Print(``='');
144                 Print(``('',fatores[s],``)'');
145                 Print(``('',fatores[t],``)'');
146                 Print(``\n'');
147                 return;
148             fi;
149             teste:= p mod fatores[s];
150             valor:=0*x;
```

```
151             if teste = valor then
152                 Print(``\n A fatoração é dada por: \n'');
153                 Print(p);
154                 Print(``='');
155                 Print(``('',fatores[s],``)'');
156                 Print(``('',(p/fatores[s]),``)'');
157                 Print(``\n'');
158                 return;
159             fi;
160         od;
161     od;
162     Print(``\n'');
163 end;
```

APÊNDICE B – Algoritmo de Rabin

Segue abaixo a implementação do algoritmo de Rabin utilizado no Capítulo 5.

Algoritmo 2: Algoritmo de Rabin

```
1  Rabin := function(gf, n)
2     local listaCoefiAlea, f, i,listaelementos, q, pol, teste,
3     divisoresinteiros,valor,ki,s,ni,cont,gcd, cond1, cond2,p;
4     # Primeiro passo: Gerar um polinômio aleatório de grau n;
5     cont:=0;
6     listaCoefiAlea:= [];
7     cond1:= false; # Condições que validam o Algoritmo de Rabin.
8     cond2:= false; # Condições que validam o Algoritmo de Rabin.
9     ki:=[]; # Divisores primos de n.
10    ni:= []; # n/ki.
11    f:= x^n; # f mônico
12    for i in [1..n] do
13       Add(listaCoefiAlea, Random(gf));
14    od;
15    listaelementos:= List(gf);
16    for i in [1..n] do
17       f:= f + listaCoefiAlea[i]*x^(i-1);
18    od;
19    # Segundo passo: Testar se f(x) divide x^q - x;
20    q:= Length(listaelementos);
21    pol:= x^q - x;
22    teste:= EuclideanRemainder(pol,f);
23    valor:=0*x; # Polinômio nulo.
24    if teste = valor then
25       cond1 := true; # Primeira condição obteve sucesso.
26    else
27       cond1:= false;
28    fi;
29    # Terceiro Passo: Verificar se o M.D.C(f,x^(p^ni)-x) = 1
30    p:=PrimeDivisors(q);
31    divisoresinteiros:= DivisorsInt(q);
32    for i in [1..Length(divisoresinteiros)] do
33       if IsPrime(divisoresinteiros[i]) = true then
```

```
34            Add(ki, divisoresinteiros[i]);
35         fi;
36      od;
37      for i in [1..Length(ki)] do
38            Add(ni, n/ki[i]);
39      od;
40      for i in [1..Length(ni)] do
41         if (Gcd( f , x^(p[1]^ni[i]) - x ) = 1*x^0) then
42            cont:= cont + 1;
43         fi;
44      od;
45      if cont = Length(ni) then
46         cond2:= true; # Segunda condição obteve sucesso.
47      else
48         cond2:= false;
49      fi;
50      if (cond1 and cond2 = true) and (IsIrreducible(f) = true) then
51         Print("\n",f,"\n");
52         Print("\n");
53         return f;
54      else
55         Print("\n");
56         return 0;
57      fi;
58  end;
```