

Notas Minicurso de GAP

Ayrton Anjos Teixeira

teixeiraayrton99@gmail.com

Abril de 2022



Sumário

1	Primeiros passos	5
1.1	O que é o GAP?	5
1.2	Instalação e materias de consulta	5
1.3	Iniciando o GAP	6
2	Começando a programar no GAP	8
2.1	Operadores	8
2.2	Variáveis e funções	9
2.3	Listas	10
2.4	Condicionais e loops	11
2.5	Funções (de novo)	13
2.6	Alguns exemplos	14
2.7	Exercícios	15
3	Grupos	18
3.1	Definindo grupos	18
3.2	Algumas funções	19
3.3	Homomorfismos	20
3.4	Comutadores	21
3.5	Exercícios	23
4	Apêndice	24
4.1	Links importantes	24
4.2	Aulas no Youtube	24
5	Soluções de alguns exercícios	25

Preâmbulo e agradecimentos

Essas notas foram escritas para um minicurso (online) que realizei pelo PETMAT-UnB em 2021. Após esse momento gravei alguns vídeos baseado nesse material, caso o leitor esteja interessado, no Apêndice pode-se encontrar mais detalhes e os links das aulas.

Esperamos que o leitor, após a leitura do material, consiga:

1. Entender a linguagem do GAP.
2. Conseguir desenvolver algoritmos.
3. Testar conjecturas e calcular exemplos.

Para tanto, adotaremos uma perspectiva mais teórica onde as funções e objetos do GAP são apresentados, juntamente de exercícios para fortalecerem o entendimento dos capítulos. Alguns exercícios não são tão precisos, nesses, recomendamos apenas que o leitor se empenhe até onde julgar interessante.

O problema principal que trataremos aqui será o de calcular *grupos com elementos não-comutadores*, para entendermos essa questão, precisamos de algumas definições.

Sejam G um grupo e $x, y \in G$, definimos o comutador de x, y como elemento $[x, y] = x^{-1}y^{-1}xy$. O conjunto dos comutadores de G é denotado por $\Gamma(G)$ e o subgrupo $G' = \langle \Gamma(G) \rangle$ é chamado de subgrupo derivado. Desmond MacHale publicou em 1981 um artigo ([3]) com algumas conjecturas sugerindo que os matemáticos tentem encontrar contra-exemplos minimais para elas. A conjectura 7 desse artigo é a seguinte:

CONJECTURA. *Seja G um grupo, o conjunto dos comutadores $\Gamma(G)$ é um subgrupo de G .*

No capítulo 3 veremos como utilizar o GAP para calcular um grupo, de menor ordem, que contradiz essa afirmação.

Para uma ideia da complexidade de fazer esses cálculos sem auxílio computacional veja ([2]). Esse artigo é fruto do doutorado do matemático Robert Guralnick, um dos maiores nomes da Teoria de Grupos.

Apresentado o minicurso, falaremos um pouco do seu surgimento.

Por volta de maio de 2020 comecei a minha pesquisa individual no PETMAT UnB sob orientação do Prof. Dr. Raimundo Bastos (UnB) em Teoria de Grupos. Logo fui apresentado ao GAP e desde então venho o utilizando.

O minicurso surgiu de uma sugestão do meu orientador e, depois de algum tempo amadurecendo a ideia, decidi realizá-lo. Inclusive, deixo meus agradecimentos ao Prof. Raimundo Bastos pela sugestão e por todas as outras que vieram enquanto planejava o minicurso e até mesmo após sua realização.

Por fim, deixo também meus agradecimentos aos colegas e à tutora do PET, Profa. Dra. Luciana Ávila; e ao FNDE/MEC pelo auxílio financeiro.

1 Primeiros passos

1.1 O que é o GAP?

A sigla GAP significa “Groups, Algorithms, Programming”. De forma sucinta, o GAP é um software livre de Álgebra Computacional focado especialmente em Teoria de Grupos. De acordo com o site oficial do programa, o GAP “fornece uma linguagem de programação, uma biblioteca com milhares de funções que implementam algoritmos algébricos assim como uma grande biblioteca de objetos algébricos”. Nessas notas veremos como utilizar o GAP para trabalhar com grupos finitos.

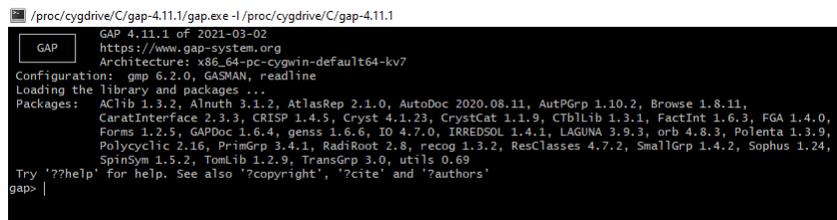
1.2 Instalação e materias de consulta

No site oficial do GAP

<https://www.gap-system.org/>

é possível encontrar instruções de como baixar e instalar o software. Uma alternativa, o GGAP, pode ser encontrada em <https://www.math.colostate.edu/~hulpke/CGT/education.html>, apesar da versão utilizada no GGAP ser antiga, ele pode ser uma boa ferramenta para começar por possuir uma interface bastante amigável, grande parte das imagens utilizadas nesse documento foram feitas utilizando o GGAP.

A opção que recomendamos é utilizar uma versão online do GAP disponível em <https://github.com/gap-system/try-gap-in-jupyter>, nesse caso, nenhum tipo de instalação é necessária e mesmo assim todas as funcionalidades do GAP que serão utilizadas nessas notas estão presentes nessa versão.



```
/proc/cygdrive/C/gap-4.11.1/gap.exe -! /proc/cygdrive/C/gap-4.11.1
GAP
GAP 4.11.1 of 2021-03-02
https://www.gap-system.org
Architecture: x86_64-pc-cygwin-default64-kv7
Configuration: gmp 6.2.0, GASMAN, readline
Loading the library and packages ...
Packages:  AClib 1.3.2, Alnuth 3.1.2, AtlasRep 2.1.0, AutoDoc 2020.08.11, AutPGrp 1.10.2, Browse 1.8.11,
CaratInterFace 2.3.3, CRISP 1.4.5, Cryst 4.1.23, CrystCat 1.1.9, CTblLib 1.3.1, FactInt 1.6.3, FGA 1.4.0,
Forms 1.2.5, GAPDoc 1.6.4, genss 1.6.6, IO 4.7.0, IRREDSOL 1.4.1, LAGUNA 3.9.3, orb 4.8.3, Polenta 1.3.9,
Polycyclic 2.16, PrimGrp 3.4.1, RadlRoot 2.8, recog 1.9.2, ResClasses 4.7.2, SmallGrp 1.4.2, Sophus 1.24,
SpinSym 1.5.2, TomLib 1.2.9, TransGrp 3.0, utils 0.69
Try '?help' for help. See also '?copyright', '?cite' and '?authors'
gap>
```

Figura 1: GAP rodando no prompt do Windows

Para utilizar o GAP, é indispensável ter em mãos o Manual de Referência ([4]), além disso, uma boa complementação para o minicurso pode ser o Tutorial ([5]), lá são apresentados comandos para se trabalhar com outras estruturas algébricas (como corpos e espaços vetoriais).

Um outro texto interessante são as notas do minicurso do Prof. Dr. Csaba Schneider ([6]), nele você pode ver, entre outras coisas, como trabalhar com Grafos e com o Cubo Mágico no GAP.

1.3 Iniciando o GAP

É possível iniciar o GAP de formas distintas dependendo do processo de instalação utilizado. Em geral, será necessário abrir o prompt do sistema operacional e digitar *gap*. No Windows pode ser um pouco diferente, após o processo de instalação será criada uma pasta com o nome *gap-4.10.2* (ou uma numeração diferente dependendo da versão instalada), dentro estará a pasta *bin*, será necessário escrever todo o caminho até o arquivo *gap.bat* dentro dessa pasta, por exemplo `C:\gap-4.10.2\bin\gap.bat`. Para encerrar a sessão basta escrever *quit*; e apertar *enter*.

Agora veja a imagem abaixo:



```
>2+3;  
[5
```

Figura 2: Uma operação no GAP

Após escrever $2 + 3$; e apertar *enter* o GAP retornou o valor dessa expressão, se retirássemos o ponto e vírgula o GAP entenderia que ainda não acabamos de digitar a expressão (em geral, espaços não são considerados), apesar de algumas funções terem uma sintaxe mais complexa sempre precisamos encerrar com ponto e vírgula para que o GAP leia a expressão, avalie e retorne o resultado. Também podemos encerrar com dois ponto e vírgula (“;;”), dessa forma o GAP irá ler e avaliar a expressão mas não retornará o resultado.

Escrever códigos complexos no prompt do sistema pode ser bastante trabalhoso, uma alternativa é escrever utilizando um editor de texto e utilizar o comando *Read()* para que o GAP leia o arquivo, dentro dos parênteses é preciso colocar todo o caminho até o arquivo, por exemplo:

```
gap> Read("C:/gap-4.11.1/bin/testeread.txt");  
1  
4  
9  
16  
25  
gap> |
```

Figura 3: Exemplo Read

Também podemos escrever comentários no arquivo que serão ignorados pelo GAP, para isso basta utilizar o símbolo *hashtag* (#).

2 Começando a programar no GAP

2.1 Operadores

O GAP possui três tipos de operadores:

- *Aritméticos*;
- *Comparação*;
- *Lógicos*.

Os operadores aritméticos nos possibilitam efetuar cálculos com números inteiros (ou com outros objetos), podemos somar ou subtrair (+ ou -), multiplicar ou dividir (* ou /) e ainda calcular potências (^) ou restos (mod). Vejamos alguns exemplos:

```
>2 + 3;2 - 3;
[
  5
  -1
]
>2*3;
[
  6
]
>7^400; #O GAP consegue trabalhar com números bem grandes
[
  109450060433611308542425445648666217529975487335970618633541940751543906316349\
  209002147856846968715280739995373528253861552495710170702637728891720852868384\
  710440066743972862761169960663579079291058878933088274875698178024977088223396\
  398265555596916473536792437134632739719389969690630523317113111727683195819839\
  003492006097994729312240001
]
>72/14;100/5;
[
  36/7
  20
]
>(2+3)*5;
[
  25
]
>119 mod 5;
[
  4
]
```

Figura 4: Operadores aritméticos

Já os operadores de comparação retornam *true*, *false* ou *fail* (valores booleanos) dependendo dos objetos avaliados. Por exemplo, dados os inteiros a e b , podemos escrever $a < b$ para avaliar se a é menor que b , em caso

positivo o operador retorna *true* e em caso negativo retorna *false*. Outros operadores dessa classe são:

- $a = b$ - retorna *true* se a e b são iguais e *false* caso contrário.
- $a <> b$ - retorna *true* se a e b são diferentes e *false* caso contrário.
- $a \leq b$ - retorna *true* se a é menor ou igual a b e *false* caso contrário.

Vejamos alguns exemplos:

```
>2 < 3;  
[ true  
>10<>3+7;  
[ false  
>2^3 >= 3^2;  
[ false
```

Figura 5: Operadores de comparação

Por fim, os operadores lógicos permitem manipular os valores booleanos. O operador *not* retorna *true* se o valor booleano avaliado for *false* e *false* caso contrário. Outros operadores dessa classe são *and* e *or*, ambos avaliam dois valores booleanos, digamos b_1 e b_2 , o primeiro retorna *true* apenas se b_1 e b_2 forem *true* e *false* caso contrário, já o segundo retorna *true* se ao menos um entre b_1 e b_2 for *true* e *false* caso contrário. Vejamos como podemos utilizá-los:

```
>2 < 3 or 3 < 4;  
[ true  
>2 < 3 and 3>4;  
[ false  
>not 2=3;  
[ true
```

Figura 6: Operadores lógicos

2.2 Variáveis e funções

Usamos as variáveis para “atribuir nome” a objetos no GAP. Para identificar uma variável precisamos utilizar pelo menos uma letra (1234a serve, mas

12345 não) e podemos atribuí-la quase qualquer objeto no GAP, como inteiros, matrizes ou grupos. O comando utilizado é *nomedavariavel := objeto*, note que utilizamos *:=* e não apenas *=*. Sempre que o GAP retorna um objeto, esse objeto é atribuído à variável *last*.

Há também as variáveis *last2* e *last3*, que recebem os penúltimos e antepenúltimos objetos retornados pelo GAP, respectivamente. Veja o exemplo:

```
>(5+2)*7;  
[ 49  
>last*3; Note que foi atribuido 49 à variável last.  
[ 147  
>last2=49; Note que agora o valor 49 foi atribuido à variável last2.  
[ true
```

Figura 7: Exemplo last e last2

Antes de continuarmos é preciso alguns comentários sobre um tipo particular de objeto do GAP, as funções. Em geral, esses objetos funcionam como uma função no sentido matemático da palavra, por exemplo, se escrevermos *Factorial(5)*; o GAP retornará 120. Há também funções que retornam valores booleanos, como a função *IsPrimeInt(n)* que retorna *false* se for verificado que *n* é composto e *true* caso contrário (para mais detalhes veja o Capítulo 14.4 do Manual de Referência do GAP).

2.3 Listas

As listas são coleções de objetos entre colchetes e separados por vírgula, por exemplo *[1, 2, 3]*. Existem muitas funções implementadas no GAP para se trabalhar com listas, vejamos alguns exemplos, outros podem ser encontrados nos exercícios.

```

[ Definindo Listas:
>lista1:=[]; lista2:=[1,2,3];
[ ]
[ 1, 2, 3 ]
[ Append(l1,l2) Concatena l1 e l2; Add(l1,n) Adiciona n no final de l1 .
>Append(lista1,lista2); Add(lista1,8); lista1;
[ 1, 2, 3, 8 ]
[ lista[n] Retorna o n-ésimo elemento da lista; lista[n]:=m Altera o n-ésimo elemento da lista por m.
>lista1[2]; lista1[1]:=4;
2
4
[ Length(lista) Calcula o tamanho da lista.
>Length(lista1);
4
[ x in lista1 Retorna true se x está na lista1 e false caso contrário.
>7 in lista1; 0 in lista1;
false
false

```

Figura 8: Exemplos com listas.

No mais, sempre que quiser realizar algo com listas, vale checar o manual e ver se já existe algo implementado.

2.4 Condicionais e loops

Nessa seção veremos o comando *if*, o *for* e o *while*. No GAP esses comandos funcionam como em outras linguagens de programação, mas é importante entendermos a estrutura delas. Para o *if* a estrutura padrão é *if condição then ação fi*;. Por exemplo, ao escrever

```
if 10 > 9 then Print("dez é maior que nove"); fi;
```

o GAP retornará *10 é maior que nove*. Podemos complementar com os comandos *elif* e *else*. O *elif* permite adicionarmos outra condição ao loop *if*, veja exemplo:

```
>if x>10 then Aqui x representa um número qualquer.
Print("maior que 10");
elif x<10 then
Print("menor que 10");
fi;
```

Figura 9: Exemplo if-elif

O *else* permite que realizemos alguma ação caso nenhuma das condições anteriores for satisfeita, por exemplo:

```
>if x>10 then  Aqui x representa um número qualquer.
Print("maior que 10");
elif x<10 then
Print("menor que 10");
else
Print("igual a 10");
fi;
```

Figura 10: Exemplo if-elif-else

Um característica importante do *if-elif-else* é que após uma das condições serem satisfeitas o GAP já interrompe o comando. Em geral, o comando *if* precisa vir acompanhado de algo mais para que tenha uma maior utilidade, normalmente utilizamos loops para que possamos checar a condição para vários objetos.

O loop *for* serve para que possamos percorrer completamente uma lista, escrevendo *for x in valores do ação od*; faremos a variável *x* percorrer a lista *valores* e, para cada valor que *x* assume, o GAP realizará a *ação* definida. Exemplo:

```
>for x in [1,2,3,12,500] do
Print(x^2,"\n");
od;
1
4
9
144
250000
```

Figura 11: Exemplo for

Por fim, o loop *while* permite que possamos realizar uma ação enquanto uma condição for satisfeita, a estrutura do comando é *while condição do ação*. Por exemplo:

```

>c:=0;;
>while c<4 do
  Print(c,"\n"); c:=c+1;
od;
0
1
2
3

```

Figura 12: Exemplo while

2.5 Funções (de novo)

Na seção 3.2 vimos alguns exemplos de funções do GAP, porém é comum nos depararmos com situações onde é interessante poder definir novas funções, para realizar tal tarefa o GAP nos proporciona duas opções:

- Para funções simples e com somente um argumento o operador *maps-to* ($- \rightarrow$) é bastante conveniente. Por exemplo, escrevendo $quadrado := x \rightarrow x^2$ definimos uma função chamada *quadrado* que calcula o quadrado do argumento, ao escrevermos $quadrado(3)$; o GAP retornará 9.
- Para funções mais complexas ou com mais de um argumento precisamos escrever *nome:=function (argumentos) ações end;*. Veja os exemplos:

Se precisarmos usar uma variável dentro da função precisamos declará-la usando o comando *local*, por exemplo:

```

>exemplo:=function(n)
  local i,j;
  i:=n^2; j:=n^3;
end;

```

Figura 13: Exemplo local

Para podermos definir as variáveis i e j precisamos escrever *local i,j;*, caso contrário o GAP retorna o erro *Syntax error: warning: unbound global variable.*

2.6 Alguns exemplos

Nessa seção veremos alguns exemplos de como utilizar os comandos que vimos até o momento. Para tanto, considere o seguinte problema:

Problema 1: Determinar todos os naturais menores que 1000 que possuem somente um divisor primo.

Veremos duas formas de resolver esse problema, a primeira é a seguinte:

```
>lista:=[];;  
>for x in [1..1000] do  
  if IsPrimePowerInt(x) then Add(lista,x); fi; od; Print(lista);
```

Figura 14: Exemplo 1 - Problema 1

O comando $IsPrimePower(n)$ retorna *true* se $n = p^k$ para algum primo p e *false* caso contrário. Nesse caso, utilizamos o comando *for* para percorrer uma lista contendo os naturais menores que 1000 e o comando *if* para que o GAP retornasse apenas os naturais que estamos interessados.

Agora veja uma outra forma de resolvermos esse problema:

```
>potenciadeprimo:= function(n)  
  if IsPrimePowerInt(n) then Print(n,"\n");fi;  
end;  
[ function( n ) ... end  
>for x in [1..1000] do  
  potenciadeprimo(x);od;
```

Figura 15: Exemplo 2 - Problema 1

Aqui definimos uma função $potenciadeprimo(n)$ que retorna n , se n é da forma p^k com p primo. Depois aplicamos essa função em todos os naturais menores que 1000.

Problema 2: Encontre 10 pares de primos p, q tais que $q = p + 2$.

Podemos resolver esse problema da seguinte forma:

```

>c:=0;;
>for x in [1..1000] do
  if IsPrime(x) and IsPrime(x+2) then
    if c<10 then
      Print(x, " ",x+2,"\n");c:=c+1;
    fi;fi;od;

```

Figura 16: Exemplo Problema 2

Note que a variável c conta quantos pares já encontramos. A cada par encontrado adicionamos 1 à variável c e o condicional *if* garante que o código irá parar quando $c = 10$.

2.7 Exercícios

Os exercícios servirão tanto para fixar o que foi visto quanto para apresentar novos comandos no GAP, é sugerido consultar o Manual para ler sobre os comandos apresentados, alguns possuem sutilezas que podem causar algum problema.

1. Calcule expressões numéricas no GAP. (Sugestão: tente utilizar potências para atingir um número grande o suficiente para que o GAP não o retorne).
2. Escreva algumas expressões envolvendo os operadores de comparação e os operadores lógicos.
3. Vimos que a função *IsPrimeInt()* retorna *false* se for provado que o argumento é composto e *true* caso contrário, talvez isso faça parecer que essa função não é “boa” para verificar se um número é primo. Procure essa função no capítulo 14 do Manual de Referência (<https://www.gap-system.org/Manuals/doc/ref/chap14.html>) e leia mais sobre ela.
4. Há um tipo especial de lista chamado *range*, que consiste basicamente de uma progressão aritmética de inteiros. Para definir um *range* escrevemos [*primeiro,segundo..último*], por exemplo, [1,6..16] define a lista [1,6,11,16]. Utilize esse comando para criar uma lista dos naturais ímpares menores que 1000.
5. Defina uma lista chamada *primos* com todos os primos até 17.

- (a) Troque o primeiro elemento da lista por 10.
 - (b) Adicione o número 19 à lista
 - (c) Utilize as funções *Product()* e *Sum()* na lista *primos*, você consegue adivinhar o que essas funções fazem?
6. Defina uma função chamada *dobro* que multiplica o argumento por 2 e uma função chamada *vezes* que recebe dois argumentos e os multiplica.
7. Caso desejemos aplicar uma função em cada elemento de uma lista podemos usar o comando *List(lista,função)*, exemplos: *List([1, 2, 3], IsPrime)* ou *List([1, 2, 3], x -> x²)*.
- (a) Use esse comando para calcular o fatorial de cada elemento da lista *primos* definida no exercício 4.
 - (b) Também podemos usar funções definidas pelo usuário, repita a parte (a) utilizando a função *dobro* definida no exercício 5.
8. Também podemos filtrar uma lista deixando apenas os elementos que satisfaçam alguma condição, para tanto podemos usar o comando *Filtered(lista, função)*, onde a função precisa retornar um valor booleano.
- (a) Utilize o comando para criar uma lista com todos os primos menores que 1000.
 - (b) Podemos criar “filtros” mais complexos utilizando os operadores lógicos *or* e *and*. Crie uma lista com todos os primos menores que 1000 que deixam resto 1 na divisão por 4.

Os próximos exercícios devem ser encarados como desafios, não por serem necessariamente difíceis, mas porque podem exigir alguma ideia ou função ainda não apresentada.

9. *Primos de Wieferich*. Seja p um primo ímpar, pelo Pequeno Teorema de Fermat $p \mid 2^{p-1} - 1$. Se $p^2 \mid 2^{p-1} - 1$ o primo p é dito primo de Wieferich.
- (a) Crie uma função *IsWieferich(n)* que retorna *true* se n for primo de Wieferich e *false* caso contrário.
 - (b) Atualmente só dois primos de Wieferich são conhecidos, encontremos!

10. *Recursividade.* Defina a sequência (L_n) da seguinte forma, $L_0 = 1$, $L_1 = 3$ e $L_n = L_{n-1} + L_{n-2}$ se $n \geq 2$. Os termos dessa sequência são conhecidos como *números de Lucas*, escreva uma função $nLucas(n)$ que calcula o n -ésimo termo da sequência acima.

3 Grupos

Nesse capítulo começaremos a trabalhar com grupos finitos no GAP. Veremos como podemos definir grupos e homomorfismos, calcular subgrupos, ordem e ainda outras coisas.

3.1 Definindo grupos

No GAP podemos definir grupos de algumas formas, a primeira delas é usando sua biblioteca. Tal biblioteca já possui implementado todos os grupos de ordem até 2000, com exceção dos grupos de ordem 1024 (veja o exercício 4 desse capítulo). Para cada ordem os grupos são armazenados em uma lista e para acessá-los usamos a função $SmallGroup(n,i)$, que calcula o i -ésimo grupo de ordem n . Veja o exemplo:

```
>NumberSmallGroups(48); Determina o número de grupos de ordem n
52
>SmallGroup(48,5);
[ <pc group of size 48 with 5 generators>
>AllSmallGroups(8); Retorna uma lista dos grupos de ordem n
[ <pc group of size 8 with 3 generators>,
  <pc group of size 8 with 3 generators> ]
```

Figura 17: Exemplo SmallGroup

O GAP também possui comandos para definir alguns grupos particulares, como grupos cíclicos, diedrais, simétricos ou alguns grupos de matrizes. Veja os exemplos:

```
>c8:=CyclicGroup(8); Define o grupo cíclico de ordem n
[ <pc group of size 8 with 3 generators>
>d8:=DihedralGroup(8); Define o grupo diedral de ordem 2n
[ <pc group of size 8 with 3 generators>
>d:=DihedralGroup(5); Note que o argumento precisa ser par
[ Error, no method found! For debugging hints type ?Recovery from NoMethodFound
  Error, no 2nd choice method found for `DihedralGroupCons' on 2 arguments
```

Figura 18: Exemplo Grupos particulares

Por fim, veremos como definir grupos utilizando geradores, para tanto usaremos o comando $Group(gens)$, onde $gens$ representa uma lista de geradores do grupo, também podemos escrever $Group(gens, id)$ para especificar a identidade do grupo.

```
>S8:=SymmetricGroup(8);
[ Sym( [ 1 .. 8 ] )
>s8:=Group((1,2),(1,2,3,4,5,6,7,8));
[ Group([ (1,2), (1,2,3,4,5,6,7,8) ])
>S8=s8; Note que os grupos definidos são iguais
[ true
```

Figura 19: Exemplo Group

3.2 Algumas funções

O GAP já possui implementado uma grande variedade de funções para se trabalhar com grupos, podemos calcular por exemplo:

- subgrupos específicos, como o centro, subgrupos de Sylow, centralizadores ...
- verificar se um grupo satisfaz alguma condição, como ser simples, abeliano, solúvel...

Veja a imagem a seguir:

```
>IsAbelian(CyclicGroup(8)); IsSimple(SymmetricGroup(5));
[ true
[ false
>Order(SymmetricGroup(5));
[ 120
>Center(DihedralGroup(8)); DerivedSubgroup(SymmetricGroup(5));
[ Group([ f3 ])
[ Group([ (1,3,2), (2,4,3), (2,3)(4,5) ])
>NormalSubgroups(DihedralGroup(8));
[ [ Group([ ]), Group([ f3 ]), Group([ f1*f2, f3 ]), Group([ f1, f3 ]),
[ Group([ f2, f3 ]), <pc group of size 8 with 3 generators> ]
```

Figura 20: Exemplo Contas com Grupos

Na imagem vemos que podemos verificar se um grupo G é abeliano ou simples com os comandos $IsAbelian(G)$ e $IsSimple(G)$, respectivamente. O comando $Order(G)$ calcula a ordem do grupo. Já os comandos $Center(G)$ e $DerivedGroup(G)$ calculam o centro $Z(G)$ e o subgrupo derivado G' , respectivamente. Por fim, $NormalSubgroups(G)$ calcula uma lista com os subgrupos normais de G .

3.3 Homomorfismos

No GAP é possível definir homomorfismos de grupos de quatro formas diferentes, aqui veremos três delas, a quarta forma consiste em induzir o homomorfismo a partir de uma ação de grupo, para mais detalhes veja o *Manual*. Também veremos como fazer alguns cálculos com homomorfismos (como a Imagem e o Núcleo).

Lembre-se que um homomorfismo fica totalmente determinado se soubermos a imagem dos geradores, o primeiro comando que veremos para definir um homomorfismo utiliza esse fato, podemos escrever $GroupHomomorphismByImages(G,H,gens1,gens2)$ para definir um homomorfismo de G em H , $gens1$ é uma lista de geradores de G que serão levados nos elementos de $gens2$ que ocupam a mesma posição nas respectivas listas.

Esse comando também verifica se a função definida é um homomorfismo e retorna *false* se não for o caso. Tal verificação pode ser bastante trabalhosa, se escrevermos $GroupHomomorphismByImagesNC$ o GAP pula essa etapa, nesse caso os resultados futuros podem ser inesperados pois podemos estar trabalhando com uma função que não é um homomorfismo.

Por fim, veja um exemplo:

```
>s5:=SymmetricGroup(5);; s4:=SymmetricGroup(4);;
>f:=GroupHomomorphismByImages(s4,s5,[(1,2),(1,2,3,4)],[(2,3),(1,4,3,2)]);
[[ (1,2), (1,2,3,4) ] -> [ (2,3), (1,4,3,2) ]
>IsInjective(f); IsSurjective(f);
[ true
[ false
```

Figura 21: Exemplo 1 - Homomorfismo

Também podemos utilizar o comando $GroupHomomorphismByFunction(G,H,fun)$ para definir um homomorfismo de G em H onde cada elemento x em G é

levado em $fun(x)$, f precisa ser uma função de G em H . Veja o exemplo:

```
>h:=function(x) Note que definimos uma função de S4 em S3
  if SignPerm(x)=-1 then return (1,2);
  else return (); fi;end;;
>f:=GroupHomomorphismByFunction(SymmetricGroup(4),SymmetricGroup(3),h);
[ MappingByFunction( Sym( [ 1 .. 4 ] ), Sym(
[ 1 .. 3 ] ), function( x ) ... end )
>Kernel(f);Image(f);
[ Group([ (), (1,2,4), (1,3,4), (1,4,2), (2,3,4), (1,2,3), (1,3)(2,4), (1,4,3),
(2,4,3), (1,2)(3,4), (1,3,2), (1,4)(2,3) ])
Group([ (1,2), (1,2) ])
```

Figura 22: Exemplo 2 - Homomorfismo

Uma observação importante é que com esse comando o GAP não verifica se de fato estamos definindo um homomorfismo.

Por último, dados um grupo G e N um subgrupo normal de G podemos definir o homomorfismo canônico de G em G/N , para tanto escrevemos *NaturalHomomorphismByNormalSubgroups*(G,N). Veja o exemplo:

```
>NaturalHomomorphismByNormalSubgroup(SymmetricGroup(6),AlternatingGroup(6));
[[ (1,2,3,4,5,6), (1,2) ] -> [ f1, f1 ]
>Image(f);Kernel(f);
[ Group([ (1,2), (1,2) ])
Group([ (), (1,2,4), (1,3,4), (1,4,2), (2,3,4), (1,2,3), (1,3)(2,4), (1,4,3),
(2,4,3), (1,2)(3,4), (1,3,2), (1,4)(2,3) ])
```

Figura 23: Exemplo 3 - Homomorfismo

3.4 Comutadores

Começaremos relembando algumas definições. Sejam G um grupo $x, y \in G$, definimos o comutador de x, y como elemento $[x, y] = x^{-1}y^{-1}xy$. O conjunto dos comutadores de G é denotado por $\Gamma(G)$ e o subgrupo $G' = \langle \Gamma(G) \rangle$ é chamado de subgrupo derivado.

No começo das notas vimos a seguinte conjectura:

CONJECTURA. *Seja G um grupo, o conjunto dos comutadores $\Gamma(G)$ é um subgrupo de G .*

Note que é equivalente verificar que $\Gamma(G) = G'$, dessa forma, veremos como encontrar um grupo G (de menor ordem) com $\Gamma(G) \neq G'$, para tanto, considere o comando $Comm(g, h)$ que calcula o comutador do par g, h e o comando $DerivedSubgroup(G)$ que calcula o subgrupo derivado de G .

Como $\Gamma(G) \subseteq G'$ basta verificar se $|\Gamma(G)| < |G'|$. Utilizaremos essa ideia e os comandos anteriores para definir uma função que tem como argumento um grupo G e que, caso $|\Gamma(G)| < |G'|$, devolve o grupo G , $|\Gamma(G)|$ e $|G'|$.

```
>derivado:=function(g)
  local i,j,gama;
  gama=[];
  for i in g do
  for j in g do
  Add(gama,Comm(i,j));
  od;od;
  gama:=Set(gama);
  if Size(gama)<Size(DerivedSubgroup(g)) then
  Print(StructureDescription(g)," ",Size(gama)," ",Size(DerivedSubgroup(g))," ");fi;end;
```

Figura 24: Função Derivado

Por fim, podemos aplicar essa função nos grupos já implementados no GAP, adiantando que o primeiro exemplo possui ordem 96, procuraremos apenas até essa ordem.

```
>for i in [1..96] do
  for j in [1..NumberSmallGroups(i)] do
  derivado(SmallGroup(i,j));
  od;od;
  [((C4 x C2) : C4) : C3 29 32 (C2 x C2 x Q8) : C3 29 32
```

Figura 25: Exemplo $\Gamma(G) \neq G'$

Portanto conseguimos construir o contra-exemplo que procurávamos. Há portanto dois grupos de ordem 96 que possuem elementos não-comutadores, a saber:

$$((C_4 \times C_2) \rtimes C_4) \rtimes C_3 \text{ e } (C_2 \times C_2 \times Q_8) \rtimes C_3$$

Além disso, temos $|\Gamma(G)| = 29$ e $|G'| = 32$.

3.5 Exercícios

Os três primeiros exercícios a seguir foram baseados em conjecturas apresentadas por Desmond MacHale em [3]. Para cada uma delas, encontre um contra-exemplo de ordem mínima.

1. *Recíproca do Teorema de Lagrange.* Seja G um grupo, então para todo divisor d de $|G|$ existe $H \leq G$ com $|H| = d$.
2. Todo grupo de ordem ímpar é abeliano.
3. Se todo subgrupo de G é normal em G , então G é abeliano.
4. Suponha que queremos sortear aleatoriamente um grupo G entre todos os grupos de ordem menor ou igual a 2000. Qual a probabilidade de $|G| = 1024$?
5. Entre todos os grupos abelianos de ordem n , qual possui a menor quantidade de subgrupos? Teste para $2 \leq n \leq 50$, você consegue conjecturar algo?
6. O comando *DihedralGroup*($2n$) calcula o grupo diedral de ordem $2n$. Para $n = 3, 4, 5, 6$ calcule a ordem do centro $Z(D_{2n})$ desse grupo. Você consegue conjecturar qual seria a ordem do centro $Z(D_{2n})$ para n qualquer?
7. *Grupos de Matrizes.* Escrevendo *GL*(n, p) o GAP retorna o grupo linear geral com entradas no corpo \mathbb{Z}_p . Analogamente *SL*(n, p) retorna o grupo especial linear com entradas em \mathbb{Z}_p . Calcule a ordem dos grupos *GL*($2, p$) e *SL*($2, p$) para $p = 2, 3, 5$. Você vê alguma relação entre essas ordens e $p - 1$? Você consegue conjecturar a ordem do grupo *GL*($2, p$) para p qualquer?
8. Calcule o grupo de automorfismos $\text{Aut}(G)$ para $G = S_3, S_4, S_5$. Compare a ordem de $\text{Aut}(G)$ com a ordem de G nesses casos. Será que o mesmo vale para S_6 ou para S_7 ?

4 Apêndice

4.1 Links importantes

Abaixo seguem os links que são importantes para minicurso:

- Site oficial do GAP
<https://www.gap-system.org/>
- Instalação (padrão e alternativa)
<https://www.gap-system.org/Download/index.html>
<https://www.gap-system.org/Download/alternatives.html>
- Manuais (tutorial e referência)
<https://www.gap-system.org/Manuals/doc/tut/chap0.html>
<https://www.gap-system.org/Manuals/doc/ref/chap0.html>
- GAP Online (interessante para começar)
<https://github.com/gap-system/try-gap-in-jupyter>
- GGAP (software indicado para usuários de Windows)
<https://www.math.colostate.edu/~hulpke/CGT/education.html>

4.2 Aulas no Youtube

Como foi dito no Preâmbulo do texto, foram gravadas algumas aulas baseadas nessas notas de aula, nem todo o conteúdo é coberto nos vídeos, mas acredito que pode ser uma boa introdução.

Essas aulas foram gravadas para uma atividade que chamei de Oficina de GAP e podem ser encontradas na playlist a seguir:

<https://youtube.com/playlist?list=PLcwKG1lKz0ke4MNXwaXTKtxYLmfZDXB6k>

5 Soluções de alguns exercícios

Aqui daremos uma solução de certos exercícios, para alguns daremos apenas o resultado esperado, para outros, deixaremos também um algoritmo que soluciona o exercício.

Capítulo 3

Exercício 9.

a)

```
IsWieferich:=function(n)
if IsPrime(n) and (2^(n-1)-1) mod n^2 = 0 then
Print(n,"\n"); fi;end;
```

b) 1093 e 3511

Exercício 10.

```
nLucas:=function(n)
local L0, L1;
L0:=1;L1:=3;
if n=0 then return 1;
elif n=1 then return 3;
else
return nLucas(n-1)+nLucas(n-2);
fi;end;
```

Capítulo 4

Exercício 1.

```
rtl:=function(g)
local cc,os;
cc:=List(ConjugacyClassesSubgroups(g),Representative);
os:=List(cc,Order);
if Size(os)<Size(DivisorsInt(Order(g))) then
Print(StructureDescription(g)," ", Order(g),"\n");
fi;end;
```

```

for i in [1..16] do
for j in [1..NumberSmallGroups(i)] do
rtl(SmallGroup(i,j)); od;od;

```

Exercício 2.

```

for i in [1,3..21] do
grupos:=AllSmallGroups(i);
impar:=Filtered(grupos, x-> not(IsAbelian(x)));
if Size(impar)>0 then Print(List(impar,StructureDescription)," ",i,"\n");
fi;od;

```

Exercício 4.

```

c:=0;;
for i in [1..2000] do
c:=c+NumberSmallGroups(i);
od;
Float(NumberSmallGroups(1024)/c);

```

Exercício 7.

```

for p in [2,3,5] do
Print([Order(GeneralLinearGroup(2,p)),Order(SpecialLinearGroup(2,p))]); od;

```

Temos $GL(2, p) = (p - 1)SL(2, p)$.

Referências

- [1] ROBINSON, D.J.S., A course in the theory of groups, 2nd edition, Springer-Verlag, New York, 1996.
- [2] GURALNICK, R. M., Commutators and Commutator Subgroups. Advances in Mathematics 45 (1982), 319-330.
- [3] Desmond MacHale (1981). Minimum Counterexamples in Group Theory. Mathematics Magazine, 54(1), 23–28.
- [4] Manual do GAP, disponível no site oficial <https://www.gap-system.org/Manuals/doc/ref/chap0.html>.
- [5] Tutorial do GAP, disponível no site oficial <https://www.gap-system.org/Manuals/doc/tut/chap0.html>.
- [6] SCHNEIDER, C., Notas do minicurso: Uma introdução à álgebra computacional com GAP. XXII Escola de Álgebra, Maringá, 2014.
- [7] RAINBOLT, J. G., GALLIAN, J. A., Abstract Algebra with GAP. Disponível em <http://math.slu.edu/~rainbolt/manual8th.htm>