Grupos de Permutações com o Maple

Lenimar Nunes de Andrade
UFPB – Dep. Matemática
lenimar@mat.ufpb.br

1 de fevereiro de 2003

1 Introdução

O Maple é um programa de Computação Algébrica de uso geral que vem sendo desenvolvido na Waterloo University desde 1981. Ele pode ser usado para efetuar cálculos numéricos ou analíticos envolvendo uma grande variedade de temas: simplificação de expressões algébricas, cálculo de raízes de polinômios, aritmética modular, corpos finitos, limites, derivadas, integrais, equações diferenciais, gráficos planos e tridimensionais, entre muitos outros.

Neste artigo mostramos, através de vários exemplos, como o Maple pode ser usado em situações que envolvam grupos de permutações tais como cálculo do normalizador e do centralizador de um subgrupo, cálculo de subgrupos de Sylow, cálculo de séries subnormais etc.

Em geral, os comandos digitados para o programa podem ser digitados à direita de um aviso denotado por um sinal de maior. A linha de comando deve encerrar com um ponto e vírgula ou com dois pontos. Se encerrar com ponto e vírgula, o resultado da operação é mostrado logo a seguir; se a linha encerrar com dois pontos, o resultado ficará guardado internamente e não será mostrado imediatamente.

2 Permutações

Diversas operações com grupos de permutações podem ser feitas com os comandos do pacote `group`. Logo, eles exigem que tenha sido digitado antes um `with(group)` ou que sejam usados na forma `group[`*comando*`]`.

```
> with(group);   # listagem dos comandos do pacote group
```

[DerivedS, LCS, NormalClosure, RandElement, SnConjugates, Sylow, areconjugate,
center, centralizer, core, cosets, cosrep, derived, elements, groupmember, grouporder,
inter, invperm, isabelian, isnormal, issubgroup, mulperms, normalizer, orbit, parity,
permrep, pres, transgroup]

O Maple interpreta uma lista $[a_1, \cdots, a_n]$ formada com os inteiros de 1 a n como sendo uma permutação f de S_n na qual $f(i) = a_i$. Por exemplo, a lista $x = [4, 5, 1, 3, 2, 6]$ pode ser interpretada como sendo a permutação

$$x = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 5 & 1 & 3 & 2 & 6 \end{pmatrix}$$

Um ciclo de um grupo de permutações na forma $[a_1, a_2, \ldots, a_n]$ e o elemento identidade por [ ]. O produto de ciclos é indicado colocando-os entre colchetes e separados por vírgulas. Por exemplo, o produto de dois ciclos é representado na forma $[[a_1, a_2, \ldots, a_n], [b_1, b_2, \ldots, b_p]]$.

O comando `convert(`*permutação*`, disjcyc)` converte a permutação fornecida como primeiro parâmetro em um produto de ciclos disjuntos.

Exemplo 2.1 *Sejam $\beta \in S_6$ e $\sigma \in S_{13}$ as permutações*

$$\beta = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 1 & 2 & 6 & 5 \end{pmatrix}$$

e

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 11 & 3 & 12 & 10 & 13 & 1 & 9 & 2 & 6 & 4 & 7 & 8 & 5 \end{pmatrix}.$$

Vamos usar o `convert` *para escrevê-las como produtos de ciclos disjuntos.*

```
> beta := [3, 4, 1, 2, 6, 5]:
> convert(beta, disjcyc);
```

$$[[1,3],[2,4],[5,6]]$$

```
> sigma := [11, 3, 12, 10, 13, 1, 9, 2, 6, 4, 7, 8, 5]:
> sigma := convert(sigma, disjcyc);
```

$$\sigma := [[1,11,7,9,6],[2,3,12,8],[4,10],[5,13]]$$

Obtivemos $\beta = (1\ \ 3)\,(2\ \ 4)\,(5\ \ 6)$ *e* $\sigma = (1\ \ 11\ \ 7\ \ 9\ \ 6)\,(2\ \ 3\ \ 12\ \ 8)\,(4\ \ 10)\,(5\ \ 13)$.

O `convert` também pode ser usado para converter produtos de ciclos disjuntos em permutações. Para isso, basta usar o segundo parâmetro como sendo `permlist` e acrescentar um terceiro parâmetro com um inteiro que corresponde ao grau da permutação.

Exemplo 2.2 *Neste exemplo, convertemos* $(2\ \ 7)\,(1\ \ 4\ \ 3)$ *e* $(2\ \ 4\ \ 1)\,(7\ \ 3)$ *em listas que representam permutações em* S_{10} *e* S_8*, respectivamente.*

```
> convert( [[2, 7], [1, 4, 3]], permlist, 10);
```

$$[4,7,1,3,5,6,2,8,9,10]$$

```
> delta := convert([[2,4,1], [7,3]], permlist, 8);
```

$$\delta := [2,4,7,1,5,6,3,8]$$

3 Comandos do pacote `group`

Para o pacote `group`, as permutações devem ser fornecidas como produtos de ciclos disjuntos. Um grupo de permutações pode ser definido na forma

`permgroup(grau,` {*geradores*}`)`.

Uma breve descrição de alguns comandos do pacote `group` é mostrada a seguir.

DerivedS(G) Série derivada de **G**, útil para determinar se **G** é solúvel;

LCS(G) Série central inferior de **G**, útil para determinar se **G** é nilpotente;

NormalClosure(H, G) Fecho normal de **H** em **G** (o menor subgrupo normal de **G** que contém **H**);

RandElement(G) Um elemento de **G** escolhido aleatoriamente;

SnConjugates(G , x) Número de elementos de **G** que contém um ciclo **x** ;

Sylow(G, p) **p**-Subgrupo de Sylow de **G**;

areconjugate(G, a, b) Determina se duas permutações **a** e **b** são conjugadas em **G**;

center(G) Calcula o centro de **G**;

centralizer(G, H) Calcula o centralizador em **G** de um conjunto de permutações **H**;

core(H, G) Calcula o maior subgrupo normal de **G** contido em **H**;

derived(G) Calcula [**G**, **G**], o subgrupo derivado de **G**;

elements(G) Listagem de todos os elementos de **G**;

groupmember(x, G) Verifica se um elemento x pertence ao grupo **G**;

grouporder(G) Calcula a ordem de **G**;

inter(G, H) Calcula a interseção de dois grupos **G** e **H**;

invperm(x) Calcula a permutação inversa de **x**;

isabelian(G) Verifica se o grupo **G** é abeliano;

isnormal(G, H) Verifica se um subgrupo **H** é normal em **G**;

issubgroup(H, G) Verifica se **H** é subgrupo de um grupo **G**;

mulperms(a, b) Calcula o produto das permutações **a** e **b**;

normalizer(G, H) Calcula o normalizador de **H** em **G**;

orbit(G, i) Calcula a órbita de um ponto **i**;

parity(x) Verifica se uma permutação **x** é par ou ímpar;

permgroup(n, A) Define um grupo de permutações de grau **n** com um conjunto de geradores **A**.

A seguir, exemplificamos o uso desses comandos.

4 Exemplos

Exemplo 4.1 *O subgrupo G do S_5 gerado pelos elementos* (1 4)(2 3) *e* (1 2)(3 5) *é definido com os seguintes comandos:*

```
> with(group):
> G := permgroup(5, {[[1,4],[2,3]], [[1,2],[3,5]]});
```

Para listar os elementos do grupo, é necessário usar o comando `elements` *:*

```
> elements(G);
```

que neste caso fornece a seguinte listagem:

$$\{\,[\,],[[1,3],[4,5]],[[2,5],[3,4]],[[1,2,3,4,5]],[[1,2],[3,5]],[[1,5],[2,4]],[[1,4],[2,3]],$$
$$[[1,4,2,5,3]],[[1,5,4,3,2]],[[1,3,5,2,4]]\,\}$$

O Maple usa o algoritmo de Dimino para listar os elementos de um grupo finito definido por um conjunto finito de geradores (conforme [1]).

Para saber quantos elementos formam esse grupo basta usar um `grouporder`*:*

```
> ‘Ordem de G‘ = grouporder(G);
```

$$Ordem\ de\ G = 10$$

Exemplo 4.2 *Neste exemplo definimos um subgrupo G de S_5 gerado pelos elementos* (3 2 5 4 1) *e* (1 5)(3 4 2)*:*

```
> with(group):
> G := permgroup(5, {[[3,2,5,4,1]], [[1,5],[3,4,2]]}):
> isabelian(G);   # G é abeliano?
```

$$false$$

Depois, pegamos dois elementos a e b de G de forma aleatória e calculamos seus inversos $x = a^{-1}$ e $y = b^{-1}$:

```
> a := RandElement(G);  b := RandElement(G);
```

$$a := [[1,4],[2,5,3]]$$
$$b := [[1,2,4,3,5]]$$

```
> x := invperm(a); y := invperm(b);
```

$$x := [[1,4],[2,3,5]]$$

$$y := [[1,5,3,4,2]]$$

Agora calculamos os sinais dos elementos a e b. Pela resposta dada pelo programa, temos que a é ímpar e que b é par.

```
> parity(a);
```

$$-1$$

```
> parity(b);
```

$$1$$

Perguntamos agora se a e b são conjugados em G:

```
> areconjugate(G, a, b);
```

$$false$$

Finalmente, calculamos os produtos $c = ab$ e $d = ba$ e perguntamos se c pertence ao grupo G.

```
> c: = mulperms(a, b); d := mulperms(b, a);
```

$$c := [[1,3,4,2]]$$

$$d := [[1,5,4,2]]$$

```
> groupmember(c, G);
```

$$true$$

Exemplo 4.3 *Neste exemplo definimos inicialmente um grupo G subgrupo de S_6 gerado por* (1 2)(3 4), (1 4) *e* (5 6 1). *Calculamos a ordem de G e escrevemo-la em forma fatorada:* $2^4 3^2 5$. *Depois, definimos H como sendo um 2-Subgrupo de Sylow de G. A partir da expressão fatorada da ordem de G, é claro que a ordem de H é* $2^4 = 16$. *Confirmamos isso listando todos os elementos de H e calculando sua ordem. Finalmente, verificamos que H não é normal em G.*

```
> with(group):
> G := permgroup(6, {[[1, 2],[3, 4]], [[1, 4]], [[5, 6, 1]]});
```

$$G := permgroup(6, \{[[1,2],[3,4]],[[1,4]],[[5,6,1]]\})$$

```
> grouporder(G); # ordem de G
```

$$720$$

```
> ifactor(%);
```

$$(2)^4(3)^2(5)$$

```
> H := Sylow(G, 2);
```

$$H := permgroup(6, \{[[2,3],[5,6]],[[3,6]],[[1,4],[2,6,5,3]]\})$$

```
> grouporder(H);   # ordem de H
```

$$16$$

```
> elements(H);
```

$$\{[\,], [[1,4],[2,5],[3,6]], [[1,4],[2,3],[5,6]], [[2,3],[5,6]], [[1,4],[2,3,5,6]], [[2,3,5,6]],$$
$$[[2,6,5,3]], [[2,5]], [[2,5],[3,6]], [[1,4],[2,6],[3,5]], [[2,6],[3,5]], [[1,4],[2,5]], [[1,4]],$$
$$[[1,4],[3,6]], [[3,6]], [[1,4],[2,6,5,3]]\}$$

```
> isnormal(G, H);  # H é normal em G?
```

$$false$$

Cuidado: `isnormal(H, G)` *não faz sentido, mas o programa responde que é true.*

Exemplo 4.4 *Neste exemplo, definimos G como sendo um grupo de permutações de grau 8 gerado pelos elementos* (1 2 8), (1 2 3 4 5) *e* (5 6 7 8)*, pegamos aleatoriamente dois de seus elementos a e b e com eles geramos um subgrupo H.*

```
> with(group):
> G := permgroup(8, {[[1, 2, 8]], [[1, 2, 3, 4, 5]], [[5, 6, 7, 8]]}):
> a := RandElement(G); b := RandElement(G); H := permgroup(8, {a, b});
```

$$a := [[1, 2], [3, 8, 7], [4, 5]]$$

$$b := [[1, 4], [3, 5, 8, 7]]$$

$$H := permgroup(8, \{[[1, 2], [3, 8, 7], [4, 5]], [[1, 4], [3, 5, 8, 7]]\})$$

```
> isnormal(G, H); # H é normal em G?
```

$$false$$

Calculamos N o normalizador de H em G e calculamos sua ordem e a ordem de G. O fato da ordem de N ser menor do que a ordem de G mostra mais uma vez que H não é normal em G.

```
> N := normalizer(G, H);
```

$$N := permgroup(8, \{[[7, 8]], [[1, 2], [3, 8, 7], [4, 5]], [[1, 4], [3, 5, 8, 7]]\})$$

```
> ‘ |N| ‘ = grouporder(N); ‘|G|‘ = grouporder(G);
```

$$|N| = 5040$$

$$|G| = 40320$$

Exemplo 4.5 *Neste exemplo definimos um grupo G, subgrupo de S_6 gerado por três ciclos e calculamos sua ordem.*

```
> with(group):
> G := permgroup(6, {[[1, 2]], [[1, 3, 4]], [[5, 6]]}):
> grouporder(G);
```

48

Agora, construímos uma série subnormal de G.

```
> DerivedS(G);
```

$$[\text{permgroup}(6, \{[[5, 6]], [[1, 3, 4]], [[1, 2]]\}),\ \text{permgroup}(6, \{[], [[1, 2, 3]], [[2, 4, 3]]\}),$$
$$\text{permgroup}(6, \{[], [[1, 2], [3, 4]], [[1, 3], [2, 4]]\}),\ \text{permgroup}(6, \{[\ ]\})]$$

Obtivemos assim grupos $G_2 = \langle(1\ 2\ 3)(2\ 4\ 3)\rangle$ e $G_1 = \langle(1\ 2)(3\ 4), (1\ 3)(2\ 4))\rangle$ tais que $G \rhd G_2 \rhd G_1 \rhd \{(1)\}$.

Exemplo 4.6 *Podemos agrupar os comandos pré-definidos em procedimentos e assim criarmos novos comandos. Neste exemplo definimos um procedimento* `ordem` *que calcula a ordem de uma permutação fornecida na forma de lista.*

```
> ordem := proc(sigma)
>   local G, alpha;
>   alpha := convert(sigma, 'disjcyc');
>   G := permgroup(nops(sigma), {alpha});
>   grouporder(G);
> end proc:
>
> with(group):
> x :=  [5, 10, 1, 2, 3, 4, 6, 7, 9, 8]:
> o :=  ordem(x);
```

$$o := 6$$

Referências

[1] Waterloo Maple Inc. (2001) *Maple 7 Programming Guide*.

[2] Waterloo Maple Inc. (2001) *Maple 7 Learning Guide*.