

TP2 : classes, structure quotient et systèmes générateurs

forts

Eléments de corrigé

```
> restart: with(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
]
> multperm:=proc(g1,g2);
return(mulperms(g2,g1));
end:
> S:=proc(n);
if n=2 then return(permgroupe(2,[[[1,2]]])) else
return(permgroupe(n,[[[1,2]],[[1..n]]])); fi;
end:
> A:=proc(n);
if n=2 then return(permgroupe(2,{}))
elif n=3 then return(permgroupe(3,[[[1,2,3]]]))
elif type(n,odd) then
return(permgroupe(n,[[[1,2,3]],[[1..n]]]))
else return(permgroupe(n,[[[1,2,3]],[[1..n],[1,2]]]));
fi;
end:
> typ:=proc(n,g) local g1,N,res,i,N1;
if not type(g,disjcycc(n)) then g1:=convert(g,disjcycc) else
g1:=g; fi;
N:=nops(g1);
if N=0 then return([1$n])
else res:=[seq(nops(op(i,g1)),i=1..N)];
N1:=sum(res[i],i=1..nops(res));
return([1$(n-N1),op(sort(res))]);
fi;
end:
> typ:=proc(n,g) local g1,N,res,i,N1;
if not type(g,disjcycc(n)) then g1:=convert(g,disjcycc) else
g1:=g; fi;
N:=nops(g1);
if N=0 then return([1$n])
else res:=[seq(nops(op(i,g1)),i=1..N)];
N1:=sum(res[i],i=1..nops(res));
return([1$(n-N1),op(sort(res))]);
fi;
end:
```

Question 1

```
> SnConjugates(S(6),[1,2,3]);
```

120

Il y a 120 éléments dans $S(6)$ de type $[1,2,3]$ (dénombrement à la main), donc 120 conjugués à $[[1,2,3],[4,5]]$. C'est bien ce que donne Maple.

La procédure suivante teste si deux éléments a et b sont conjugués dans G ; pour cela, on teste d'abord s'il sont conjugués dans $S(n)$ (où par définition G est un sous-groupe de $S(n)$), et cela en comparant les types.

```
> areconj:=proc(G,a,b) local n,g,res;
res:=false; n:=op(1,G);
if typ(n,a)=typ(n,b) then
for g in elements(G) do
```

```
if multperm(invperm(g),multperm(a,g))=b then res:=true;
break fi;
od;
fi;
return(res);
end:
```

```
> G:=permgroupe(4,[[[1,2,3,4]],[[1,3]]]): grouporder(G);
```

8

```
> a:=[[1,2],[3,4]]: groupmember(a,G);
```

true

```
> b:=[[1,3],[2,4]]: groupmember(b,G);
```

true

```
> typ(4,a); typ(4,b); areconj(S(4),a,b);
```

[2,2]

[2,2]

true

Ces deux éléments sont de même type, donc conjugués dans $S(4)$.

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

false

Mais ils ne sont pas conjugués dans G .

```
> areconjugate(S(4),a,b); areconjugate(G,a,b);
```

true

false

Ces commandes sont déjà implémentées (sans doute moins naïvement, i.e. plus efficacement) sous Maple.

Question 2

```
> classeconj:=proc(a,G) local L,g;
L:={};
for g in elements(G) do
L:={op(L),multperm(invperm(g),multperm(a,g))}; od;
return(L);
end:
```

```
> classeconj(a,S(4)); classeconj(b,S(4));
```

{[[1,2],[3,4]], [[1,3],[2,4]], [[1,4],[2,3]]}

{[[1,2],[3,4]], [[1,3],[2,4]], [[1,4],[2,3]]}

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

{[[1,2],[3,4]], [[1,4],[2,3]]}

{[[1,3],[2,4]]}

```
> listeclasse:=proc(G) local L,T,C;
```

```
L:=elements(G) minus {}; T:=[1];
```

```
while nops(L)>0 do
```

```
C:=classeconj(op(1,L),G);
```

```
T:=[op(T),nops(C)];
```

```
L:=L minus C;
```

```
od;
```

```
return(sort(T));
```

```
end:
```

```

> listeclasses(S(4));
[1, 3, 6, 6, 8]
> eqclasses:=proc(G) local L,n,s;
n:=grouporder(G); s:=convert(listeclasses(G), '+');
return(n,s);
end:
> eqclasses(S(4));
24, 24
> eqclasses(G);
8, 8

```

Question 3

L'équation aux classes d'un groupe abélien est toujours $\text{card}=1+1+\dots+1$; il suffit donc de trouver deux groupes abéliens de meme cardinal non isomorphes.

```

> H:=permgroupe(4, {[[1,2,3,4]]});
K:=permgroupe(4, {[[1,2]], [[3,4]]});
H:=permgroupe(4, {[[1,2,3,4]]});
K:=permgroupe(4, {[[1,2]], [[3,4]]});
> listeclasses(H); listeclasses(K);
[1, 1, 1, 1]
[1, 1, 1, 1]

```

L'équation aux classes est la meme, mais ces deux groupes ne sont pas isomorphes : le premier possède un élément d'ordre 4 mais pas le second. Noter que H est le groupe cyclique d'ordre 4 isomorphe à $\mathbb{Z}/4\mathbb{Z}$ et K est isomorphe à $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ en tant que groupe abstrait (il est engendré par deux éléments d'ordre deux qui commutent).

Question 4

Cette procedure calcule la classe de x à droite modulo H :

```

> classeg:=proc(x,H) local res,h;
res:={};
for h in elements(H) do res:={op(res),multperm(x,h)}; od;
return(res);
end:

```

Et celle-ci la classe de x à gauche :

```

> classed:=proc(x,H) local res,h;
res:={};
for h in elements(H) do res:={op(res),multperm(h,x)}; od;
return(res);
end:

```

Question 5

```

> V4:=permgroupe(4, {[[1,2],[3,4]], [[1,4],[2,3]]});

```

Les deux générateurs correspondent aux symétries par rapport aux médiatrices des cotés du rectangle (lorsque les sommets sont numérotés circulairement). Il est isomorphe à $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$ en tant que groupe abstrait.

```

> quot1:=cosets(S(4),V4);
quot1 := {[[2,4,3]], [[2,3]], [[2,3,4]], [[3,4]], [[2,4]], []}

```

C'est une liste de representants des classes à gauche de S(4) modulo H

Voici les classes à gauche modulo H :

```

> cg1:=seq(classeg(q,V4),q=quot1);
cg1 := {[[1,4,2]], [[1,2,3]], [[2,4,3]], [[1,3,4]]},
{[[1,4]], [[2,3]], [[1,2,4,3]], [[1,3,4,2]]}, {[[1,4,3]], [[1,2,4]], [[2,3,4]], [[1,3,2]]},
{[[1,4,2,3]], [[1,2]], [[1,3,2,4]], [[3,4]]}, {[[1,2,3,4]], [[1,3]], [[1,4,3,2]], [[2,4]]},
{[[1,2],[3,4]], [[1,3],[2,4]], [[1,4],[2,3]], []}

```

Et les classes à gauche :

```

> cd1:=seq(classed(invperm(q),V4),q=quot1);
cd1 := {[[1,4,3]], [[1,2,4]], [[2,3,4]], [[1,3,2]]},
{[[1,4]], [[2,3]], [[1,2,4,3]], [[1,3,4,2]]}, {[[1,4,2]], [[1,2,3]], [[2,4,3]], [[1,3,4]]},
{[[1,4,2,3]], [[1,2]], [[1,3,2,4]], [[3,4]]}, {[[1,2,3,4]], [[1,3]], [[1,4,3,2]], [[2,4]]},
{[[1,2],[3,4]], [[1,3],[2,4]], [[1,4],[2,3]], []}

```

Les classes a droite coincident avec celles a gauche.

```

> evalb({op(cg1)}={op(cd1)});
true

```

```

> isnormal(S(4),V4);
true

```

Ce n'est pas étonnant : V est constitué de l'identité et des permutations de S(4) de type (2,2) ; comme la conjugaison conserve le type, gV_4g^{-1} est inclus dans V_4 pour toute permutation g de S(4) et donc $gV_4 = V_4g$.

Le quotient $S(4)/V$ est isomorphe à S(3) : l'application qui à un élément de quot associe sa classe définit un isomorphisme entre S(3) (identifié aux permutations de {2,3,4}) et $S(4)/V$. En effet, l'application de passage au quotient $S(4) \rightarrow S(4)/V$ est un morphisme surjectif dont l'application précédente est la restriction à S(3) ; on montre la bijectivité en regardant les cardinaux, la surjectivité résultant de la description des classes donnée par Maple.

Question 6

```

> W:=permgroupe(4, {[[1,2]], [[3,4]]});
W := permgroupe(4, {[[1,2]], [[3,4]]})

```

```

> isnormal(S(4),W);
false

```

```

> quot2:=cosets(S(4),W);
quot2 := {[[1,3],[2,4]], [[1,3]], [[2,3]], [[1,2,4,3]], [[2,4]], []}

```

```

> cg2:=seq(classeg(q,W),q=quot2);
cg2 := {[[1,4,2,3]], [[1,3],[2,4]], [[1,4],[2,3]], [[1,3,2,4]]},
{[[1,2,3,4]], [[1,3]], [[1,2,3]], [[1,3,4]]}, {[[2,3]], [[2,3,4]], [[1,3,2]], [[1,3,4,2]]},
{[[1,4,3]], [[1,4]], [[1,2,4]], [[1,2,4,3]]}, {[[1,4,2]], [[2,4,3]], [[1,4,3,2]], [[2,4]]},
{[[1,2],[3,4]], [[1,2]], [[3,4]], []}

```

```

> seq(classed(q,W),q=quot2);
{[[1,4,2,3]], [[1,3],[2,4]], [[1,4],[2,3]], [[1,3,2,4]]},
{[[1,4,3]], [[1,3]], [[1,3,2]], [[1,4,3,2]]}, {[[1,2,3]], [[2,4,3]], [[2,3]], [[1,2,4,3]]},
{[[1,2,3]], [[2,4,3]], [[2,3]], [[1,2,4,3]]}, {[[1,2,3,4]], [[1,2,4]], [[2,3,4]], [[2,4]]},
{[[1,2],[3,4]], [[1,2]], [[3,4]], []}

```

Les classes à droite modulo H des éléments de quot ne sont pas distinctes ; quot constitue un système de representants des classes à gauche (et non à droite comme le mentionne l'aide de Maple !)

```

> cd2:=seq(classed(invperm(q),W),q=quot2);
cd2:=[[1,4,2,3],[1,3],[2,4],[1,4],[2,3],[1,3,2,4]],
      [[1,4,3],[1,3],[1,3,2],[1,4,3,2]], [[1,2,3],[2,4,3],[2,3],[1,2,4,3]],
      [[1,4,2],[1,4],[1,3,4],[1,3,4,2]], [[1,2,3,4],[1,2,4],[2,3,4],[2,4]],
      [[1,2],[3,4],[1,2],[3,4],[ ]]]
[ Noter que classes à gauche et à droite ne coïncident pas.
> evalb({op(cg)}={op(cd)});
false

```

Question 7

```

> classesg:=proc(G,H) local L,X,S;
L:={}; X:=elements(G);
while nops(X)>0 do
S:=classg(op(1,X),H);
L:=L union {S};
X:=X minus S;
od;
return(L);
end;
> classesg(S(4),W);
[[[1,2,3,4],[1,3],[1,2,3],[1,3,4]],
 [[2,3],[2,3,4],[1,3,2],[1,3,4,2]], [[1,4,3],[1,4],[1,2,4],[1,2,4,3]],
 [[1,4,2,3],[1,3],[2,4],[1,4],[2,3],[1,3,2,4]],
 [[1,4,2],[2,4,3],[1,4,3,2],[2,4]], [[1,2],[3,4],[1,2],[3,4],[ ]]]
> evalb(#{op(cg2)});
true
> classesd:=proc(G,H) local L,X,S;
L:={}; X:=elements(G);
while nops(X)>0 do
S:=classed(op(1,X),H);
L:=L union {S};
X:=X minus S;
od;
return(L);
end;
> classesd(S(4),W);
[[[1,2,3,4],[1,2,4],[2,3,4],[2,4]],
 [[1,4,2],[1,4],[1,3,4],[1,3,4,2]],
 [[1,4,2,3],[1,3],[2,4],[1,4],[2,3],[1,3,2,4]],
 [[1,4,3],[1,3],[1,3,2],[1,4,3,2]], [[1,2,3],[2,4,3],[2,3],[1,2,4,3]],
 [[1,2],[3,4],[1,2],[3,4],[ ]]]
> evalb(#{op(cd2)});
true

```

Question 8

```

[ > SGF_I4:=[[],{[]},{[]},{[]}];

```

```

[ > SGF_C4:=[[],[[1,2,3,4]],[[1,3],[2,4]],[[1,4,3,2]]],{[]},{[]];
[ Pour G=A4, G1 est engendré par (2 3 4); on regarde G/G1 :
> classesg(A(4),permgroupe(4,{[[2,3,4]]}));
[[[1,2],[3,4],[1,2,3],[1,2,4]], [[1,3],[2,4],[1,3,4],[1,3,2]],
 [[2,4,3],[2,3,4],[ ]], [[1,4,2],[1,4,3],[1,4],[2,3]]]]
[ Comme G2=G3={id}, on peut prendre :
> SGF_A4:=[[],[[1,2,3]],[[1,3,2]],[[1,4,2]]],{[]],[[2,3,4]],[[2,4,3]],{[]];
[ Pour G=S4, G1 est engendré par (2 3), (2 4) et (3 4).
> classesg(S(4),permgroupe(4,{[[2,3]],[[2,4]],[[3,4]]}));
[[[2,4,3],[2,3],[2,3,4],[3,4],[2,4],[ ]],
 [[1,4,2],[1,4,3],[1,4,2,3],[1,4],[1,4],[2,3],[1,4,3,2]],
 [[1,3],[2,4],[1,3],[1,3,4],[1,3,2],[1,3,2,4],[1,3,4,2]],
 [[1,2],[3,4],[1,2,3,4],[1,2],[1,2,3],[1,2,4],[1,2,4,3]]]]
[ De plus, G2={id,(3 4)}; on regarde G1/G2 :
> classesg(permgroupe(4,{[[2,3]],[[2,4]],[[3,4]]}),permgroupe(4,
[[3,4]]));
[[[2,3],[2,3,4]], [[2,4,3],[2,4]], [[3,4],[ ]]]
[ et G3={id}, d'où :
> SGF_S4:=[[],[[1,2]],[[1,3]],[[1,4]]],{[]],[[2,3]],[[2,4]],{[
],[[3,4]]}];
> mul(nops(i),i=SGF_I4);
1
> mul(nops(i),i=SGF_C4);
4
> mul(nops(i),i=SGF_A4);
12
> mul(nops(i),i=SGF_S4);
24
[ Le produit est chaque fois égal au cardinal du groupe

```

Question 9

```

[ > card1:=SGF->mul(nops(c),c=SGF):
[ > card1(SGF_A4); card1(SGF_S4);
12
24
[ > elements1:=proc(SGF) local i,L;
L:={[]}; for i from 1 to nops(SGF) do
L:=seq(seq(multperm(g,h),g=L),h=op(i,SGF)); od;
return(L);
end;
[ > elements1(SGF_I4); elements1(SGF_C4);
[[ ]]]
[[[1,3],[2,4],[1,2,3,4],[1,4,3,2],[ ]]]

```

[Il faudrait traiter à part le cas du SGF trivial pour éviter l'erreur (mais l'intérêt est faible).

```

[ > elements1(SGF_A4);

```

```

[[[1, 4, 2]], [[1, 4, 3]], [[1, 2], [3, 4]], [[1, 3], [2, 4]], [[1, 4], [2, 3]], [[1, 2, 3]], [[2, 4, 3]],
[[1, 2, 4]], [[1, 3, 4]], [[2, 3, 4]], [[1, 3, 2]], [ ] ]
> evalb(=%elements(A(4)));
true
> elements1(SGF_S4);
[[[1, 4, 2]], [[1, 4, 3]], [[1, 2], [3, 4]], [[1, 4, 2, 3]], [[1, 4]], [[1, 3], [2, 4]], [[1, 4], [2, 3]],
[[1, 2, 3, 4]], [[1, 3]], [[1, 2]], [[1, 2, 3]], [[2, 4, 3]], [[2, 3]], [[1, 2, 4]], [[1, 3, 4]],
[[2, 3, 4]], [[1, 3, 2]], [[1, 2, 4, 3]], [[1, 4, 3, 2]], [[1, 3, 2, 4]], [[1, 3, 4, 2]], [[3, 4]], [[2, 4]],
[ ] ]
> evalb(=%elements(S(4)));
true

```

Question 10

```

> image1:=proc(g,n,i) local gg;
gg:=convert(g,'permlist',n);
op(i,gg);
end;
[ Sans convertir g tout entier en permlist :
> image:=proc(g,n,i) local c,l,j;
for c in g do
l:=nops(c);
for j from 1 to l do
if op(j,c)=i then if j=1 then return(op(1,c)) else
return(op(j+1,c)); fi; fi;
od;
od;
return(i);
end;
> g:=[[1,3,5],[2,9],[7]]: image(g,10,8); image(g,10,3);
image(g,10,7); image(g,10,9);
8
5
7
2
> appart:=proc(g,SGF) local gg,n,x,i,test,h;
gg:=g; n:=nops(SGF)+1;
for i from 1 to n-1 do
x:=image(gg,n,i);
test:=false;
for h in op(i,SGF) do if image(h,n,i)=x then
gg:=multperm(invperm(h),gg); test:=true; break; fi; od;
if test=false then return(false); fi;
if gg=[] then return(true); fi;
od;
end;
> appart([[3,4]],SGF_S4);
true
> appart([[1,2,3]],SGF_A4);

```

```

true
> appart([[1,2]],SGF_A4);
false
> appart([[1,2],[3,4]],SGF_A4);
true

```

Question 11

```

> appart_i:=proc(g,SGF) local gg,n,x,i,test,h;
gg:=g; n:=nops(SGF)+1;
for i from 1 to n-1 do
x:=image(gg,n,i);
test:=false;
for h in op(i,SGF) do if image(h,n,i)=x then
gg:=multperm(invperm(h),gg); test:=true; break; fi; od;
if test=false then return([i,gg]); fi;
if gg=[] then return([n,gg]); fi;
od;
end;
> sgf_plus:=proc(g,SGF) local SGFg,n,ig,i,gg,j,h;
SGFg:=SGF; n:=nops(SGF)+1; ig:=appart_i(g,SGF);
i:=op(1,ig); gg:=op(2,ig);
if i<n then SGFg[i]:=op(i,SGFg) union {gg};
for j from 1 to i do for h in op(j,SGFg) do
SGFg:=sgf_plus(multperm(gg,h),SGFg); od; od;
fi;
return(SGFg);
end;
> sgf:=proc(sg,n) local g,SGF,i;
SGF:=[seq([[]],i=1..n-1)]; for g in sg do
SGF:=sgf_plus(g,SGF); od;
return(SGF);
end;
> SGF:=sgf({[[1,2]],[[1,2,3,4]]},4);
SGF:=
[[[[1, 3], [2, 4]], [[1, 2, 3, 4]], [[1, 4, 3, 2]], [ ]], [[2, 4, 3]], [[2, 3]], [ ]], [[3, 4]], [ ] ]
> card1(SGF); evalb(elements1(SGF)=elements(S(4)));
24
true
> SGF:=sgf({[[1,2,3]],[[1,2],[3,4]]},4);
SGF:=[[[[1, 4, 2]], [[1, 2], [3, 4]], [[1, 3, 2]], [ ]], [[2, 4, 3]], [[2, 3, 4]], [ ]], [ [ ] ]
> evalb(elements1(SGF)=elements(A(4)));
true
> time(grouporder(S(8))); time(elements(S(8)));
0.016
6.540
> t:=time(): SGF:=sgf({[[1,2]],[[1..8]]},8): time()-t;
0.384
> time(card1(SGF));
0.

```

```
[ > time(elements1(SGF));  
[  
[ 8.068  
[ > time(grouporder(S(15)));  
[ 0.336  
[ > time(sgf([[1,2]],[[1..15]],15));  
[ 25.121
```

Quelques subtilités se cachent encore derrière les procédures natives de Maple, mais c'est bien une variante de l'algorithme de Schreier-Sims qui est la base de ces procédures...

```
[ > elements2:=proc(G) local S,L,g,h,N;  
[ S:=op(2,G); for g in S do S:=S union {invperm(g)}; od;  
[ L:=S union {}; N:=S;  
[ while (nops(N)<>0) do  
[ N:={seq(seq(multperm(g,h),g=N),h=S)} minus L;  
[ L:=L union N;  
[ od;  
[ return(L);  
[ end;  
[ > time(elements2(S(8)));  
[ 20.465
```

[Le gain par rapport à l'algorithme naïf est par contre évident.