Computer Algebra in Introductory Group Theory:

Exploring Permutation Groups with Maple V

Suda Kunyosying
Department of Mathematics & Engineering
Shepherd College
Shepherdstown, WV 25443
skunyosy@shepherd.edu

Abstract:

This paper discusses the use of Maple V's ***group*** and ***combinat*** packages in the study of elementary abstract algebra or introductory group theory. The main focus of the paper is to show how to introduce discovery learning into abstract algebra class and how to use a laboratory approach to teach basic concepts of group theory with the help of elementary group theory commands found in MAPLE V's packages ***group*** and ***combinat***.

1. Introduction

An "Introductory Group Theory" course may vary greatly in contents. To a typical Asian university, what the author intends to do may appear a bit too elementary or too shallow for such a course. Regardless of the level of sophistication in the course contents, we ought to let students perform some important basic experiments that help encourage the discovery of abstract mathematical idea.

A computer algebra software such as Maple V, a very powerful tool for doing mathematics, can be effectively used in the undergraduate abstract algebra course to encourage the discovery of mathematical ideas through guided experiments. The main focus of the paper is to show how to introduce discovery learning into abstract algebra class and how to use a laboratory approach to teach basic concepts of group theory. We will present some elementary group theory commands in MAPLE V's packages ***group*** and *combinat* which may be useful in a first abstract algebra course or introductory group theory.

Focusing on the notion that "One cannot teach a computer how to do something without learning it better oneself," several of our exercises require the students to write functions or procedures to implement abstract mathematical ideas in a concrete way, such as "l**i**sting elements of a permutation group", or "constructing an abstract group's table." Those exercises really require the students to think about what the computer is doing when it is performing the instructions given it. As a result, they will begin to form a mental image of the ideas they are learning, and come to truly understand these ideas**.**

Another type of assignments requires students to identify a group by means of a set of generators and a set of relations between those generators. The students then verify their answer by constructing the group using *grelgroup* and *subgrel* commands of Maple V. Such construction is useful in distinguishing finite groups from infinite groups.

Another useful command of Maple V is the *permrep* command. The paper will present an example illustrating the use of Maple V's *permrep* command in the study of Cayley's Regular Representation Theorem.

It must be mentioned also that any Computer Algebra Software, such as ISETL or Mathematica, may be used to do the experiments suggested by the author. However, the author has found that students become familiar with Maple V's syntax much quicker than any other system.

2. L**isting elements of a permutation group using MAPLE's functions.**

2.1. *permgroup(degree, {generators})*

This routine constructs a permutation group of specified degree and generators.
Note: All permutations in MAPLE V must be written as disjoint cycles.
Example: The symmetric group S_3. >with(group):

```
> S_3 := permgroup(3,{[[1,2]],[[1,2,3]]});
    S_3 := permgroup(3, {[[1, 2]], [[1, 2, 3]]})
```

2.2. *grouporder (group)*

This routine gives the order (or number of elements) of a group.
Example : To verify S_3 above has 6 elements:

```
> grouporder(S_3);
    6
```

2.3. *cosets(PG,SG)*

This routine gives a complete list of right coset representatives for a subgroup SG of a permutation group PG. A set of permutations in disjoint cycle notation is returned.

2.3.1 Note:

1. PG and SG must be permutation groups of the same degree.
2. This routine can be used to list elements of a permutation group when SG is the trivial subgroup, { }, of the same degree.

2.3.2 Example

```
> ident := permgroup (3, {[]} ):
> cosets(S_3, ident);
    {[], [[1, 2]], [[1, 2, 3]], [[1, 3, 2]], [[1, 3]], [[2, 3]]}
```

3. Listing elements of permutation groups without the cosets command.

The following MAPLE V' s functions will be required.

(i) *permute(n)*
This function (found in MAPLE V's combinat package) constructs a list of all permutations of n objects.

(ii) convert(permlist, 'disjcyc')
This function converts permutations in a list into disjoint cycles.

(iii) groupmember(element, PG)
This function tests if a given element is a member of the permutaion group PG.

3.1 Outline of Procedure (for listing elements of a permutation group)

(i) Create a list L0 of all permutations on n letters using *combinat[permute](n)*
(ii) Convert all elements in L0 to disjoint cycles with *convert(L0, 'disjcyc')*.
(iii) Create the permutation group G of degree n generated by generators in the set L using *permgroup(n, L)*.
(iv) Test each elements in L0 for membership in G using *groupmember(element, G)*
(v) Put those that are members of G in a set L1. Display L1.

3.2 Procedure *gpElements (n, L)*
(lists all elements of a permutation group of degree n generted by a list of generators L)

```
> gpElements := proc(n::integer,G)
>   local i,j,L0,L1;
>   L0 := combinat[permute](n);
>   for i from 1 to nops(L0) do
>     g.i := convert(L0[i],'disjcyc');
>   od;
>   L1 := []: for i from 1 to nops(L0) do
>     if groupmember(g.i,G) then
>       L1 := [op(L1),g.i];
>     fi;
>   od;
> RETURN(L1);
> end;
```

3.3 Example

```
> S_3 := permgroup(3,{[[1,2]],[[1,2,3]]});
      S_3 := permgroup(3, {[[1, 2]], [[1, 2, 3]]})
> gpElements(3, S_3);
     [[], [[2, 3]], [[1, 2]], [[1, 2, 3]], [[1, 3, 2]], [[1, 3]]]
> S_4 := permgroup(4,{[[1,2]],[[1,2,3,4]]});
      S_4 := permgroup(4, {[[1, 2]], [[1, 2, 3, 4]]})
> gpElements(4, S_4);
[[], [[3, 4]], [[2, 3]], [[2, 3, 4]], [[2, 4, 3]], [[2, 4]], [[1, 2]], [[1, 2], [3, 4]],
[[1, 2, 3]], [[1, 2, 3, 4]], [[1, 2, 4, 3]], [[1, 2, 4]], [[1, 3, 2]], [[1, 3, 4, 2]],
[[1, 3]], [[1, 3, 4]], [[1, 3],[2, 4]], [[1, 3, 2, 4]], [[1, 4, 3, 2]], [[1, 4, 2]],
[[1, 4, 3]], [[1, 4]], [[1, 4, 2, 3]],  [[1, 4], [2, 3]]]
```

4. Embedded Subgroups of a Symmetric Group and Cayley's Theorem

By comparing an embedded subgroup of a symmetric group with a symmetric group of lower degree having exactly the same elements (in disjoint cycle notations), the students will see the distinction between subgroup of a symmetric group and a subset which is also a group

but not a subgroup of that symmetric group. In particular, this experiment will illustrate the following points

(i) S_3 is not a subgroup of S_4. Although, written as disjoint cycle, every element of S_3 is in S_4 and S_3 is a group. However, a subgroup of a permutation group must have the same degree. I.e. they must act on the same set of letters.

(ii) S_3 is isomorphic to an embedded subgroups of S_4.

(iii) Any group is isomorphic to a subgroup of a symmetric group. (Cayley's Theorem)

4.1 *Procedure Symm (n, degree)*

(creates an embedded subgroup S_n of a symmetric group S_degree)

```
> Symm := proc (n::integer,degree::integer)
>    local i, S:
>    if degree < n then ERROR (`argument 2 must be > argument 1`); fi;
>    S := permgroup(degree,{[[1,2]],[[seq (i, i = 1..n)]]});
>    RETURN(S);
> end;
```

4.2 **E**xample.

```
> S3 := Symm(3,4):
> S_4 := permgroup(4,{[[1,2]],[[1,2,3,4]]}):
> issubgroup(S3,S_4);
        true
> S_3:=permgroup(3,{[[1,2]],[[1,2,3]]}):
> issubgroup(S_3,S_4);
        false
> gpElements(3,S3);
      [[], [[2, 3]], [[1, 2]], [[1, 2, 3]], [[1, 3, 2]], [[1, 3]]]
> gpElements(3,S_3);
      [[], [[2, 3]], [[1, 2]], [[1, 2, 3]], [[1, 3, 2]], [[1, 3]]]
```

5. Cayley's Group Table

A useful tool for studying properties of a finite group is its abstract group table. So one of our projects is to have the students write a procedure to construct a table for a permutation group G. The procedure calls the already defined function *gpElements* to create a list of all elements of G then it constructs a two dimensional array of size o(G) x o(G). The product of two elements of G is computed by the MAPLE V's function: *group[mulperms](L0[i],L0[j]),* where L0 = ordered list of elements of G and L0[i] = ith entry in the list. All entries in the list are in disjoint cycle notations

5.1 Procedure for constructing Abstract Group Table

```
> abs_gp_table:=proc(n::integer,G)
> local i, j, L0, S_table, g, k ;
```

```
> L0:=gpElements(n,G);
> S_table:=array(1..(nops(L0)+1),1..(nops(L0)+1));
> for i from 1 to nops(L0) do
>     for j from 1 to nops(L0) do
>       for k from 1 to nops(L0) do
>           if L0[k]=group[mulperms](L0[i],L0[j]) then
>              S_table[i+1,j+1]:=a.k; fi;
> od; od; od;
> S_table[1,1] := `*` ;
> for j from 1 to nops(L0) do
>      S_table[1,j+1]:=a.j;
>      S_table[j+1,1]:=a.j;
> od;
> print(convert(S_table, matrix));
> print(seq(a.i=L0[i],i=1..nops(L0)));
> end;
```

6. Regular Permutation Representations.

This section will introduce MAPLE V's function: *group[permrep]* for finding a permutation representation of a group. The calling Sequence is *permrep(sbgrl)* , where sbgrl is a subgroup of a group described by generators and relations (i.e. a subgrel)

6.1 Description of *group[permrep]*

This function finds all the right cosets of the given subgroup in a given group then assigns integers consecutively to these cosets and constructs a permutation on these coset numbers for each group generator. It returns the permutation group generated by these permutations. Thus the permutation group will be a homomorphic image of (but not necessarily isomorphic to) the original group. A permgroup is returned whose generators are named the same as the original group generators.

Trick: To find a regular representation of elements of finite group G, we let the *subg*r to be the trivial subgroup { }.

6.2 **Example.**

Suppose G is generated by the relations: y^2 = 1 and yxy = x^2. We let SG to be the trivial subgroup of G generated by {[]}.

```
> G := grelgroup({x,y}, {[y,x,y,1/x,1/x],[y,y]}):
> SG := subgrel({y=[]},G):
> PG:= permrep(SG);
    PG := permgroup(6,{x = [[1, 5, 4], [2, 3, 6]], y = [[1, 2], [3, 4],[5, 6]]})
> grouporder(PG);
        6
> gpElements(6,PG);
      [[], [[1, 2], [3, 4], [5, 6]], [[1, 3], [2, 5], [4, 6]], [[1, 4, 5],
     [2, 6, 3]], [[1, 5, 4],  [2, 3, 6]], [[1, 6], [2, 4], [3, 5]]]
```

Since the abstract group Table for G, will be the same as the abstract group table for PG we can use our procedure *abs_gp_table(n,PG*) to construct an abstract group table for G.

6.3 **Cayley's Table: an application of the procedure *abs_gp_table***

```
>  abs_gp_table(6,PG);
   [* a1 a2 a3 a4 a5 a6]
   [a1 a1 a2 a3 a4 a5 a6]
   [a2 a2 a1 a5 a6 a3 a4]
   [a3 a3 a4 a1 a2 a6 a5]
   [a4 a4 a3 a6 a5 a1 a2]
   [a5 a5 a6 a2 a1 a4 a3]
   [a6 a6 a5 a4 a3 a2 a1]
a1 = [], a2 = [[1, 2], [3, 4], [5, 6]],
a3 = [[1, 3], [2, 5], [4, 6]], a4 = [[1, 4, 5], [2, 6, 3]],
a5 =  [[1, 5, 4], [2, 3, 6]], a6 = [[1, 6], [2, 4], [3, 5]]
```

6.4. **A Conway's Problem: An application of *permrep.***

Problem. Suppose G is generated by the relations: ab = c, bc =a, ca = b.
Find the order of G and construct its Cayley's table.

Solution,

```
> C := grelgroup({a,b,c},{[a,b,1/c],[b,c,1/a],[c,a,1/b]});
  C := grelgroup({a, b, c}, {[b, c, 1/a], [c, a, 1/b], [a, b, 1/c]})
> grouporder(C);
       8
```

```
> sc := subgrel({y=[]},C):
> PC:=permrep(sc);
      PC := permgroup(8, {a = [[1, 3, 5, 8], [2, 6, 7, 4]],
c = [[1, 4, 5, 6], [2, 3, 7, 8]], b =   [[1, 2, 5, 7], [3, 4, 8, 6]]})
> grouporder(PC);
       8
> gpElements(8,PC);
     [ [], [[1, 2, 5, 7], [3, 4, 8, 6]], [[1, 3, 5, 8], [2, 6, 7, 4]], [[1, 4, 5, 6],
     [2, 3, 7, 8]], [[1, 5], [2, 7], [3, 8], [4, 6]], [[1, 6, 5, 4], [2, 8, 7, 3]], [[1, 7, 5, 2],
     [3, 6, 8, 4]],  [[1, 8, 5, 3], [2, 4, 7, 6]] ]

>  abs_gp_table(8,PC);
[* a1 a2 a3 a4 a5 a6 a7 a8]
[a1 a1 a2 a3 a4 a5 a6 a7 a8]
[a2 a2 a5 a6 a3 a7 a8 a1 a4]
[a3 a3 a4 a5 a7 a8 a2 a6 a1]
[a4 a4 a8 a2 a5 a6 a1 a3 a7]
[a5 a5 a7 a8 a6 a1 a4 a2 a3]
[a6 a6 a3 a7 a1 a4 a5 a8 a2]
[a7 a7 a1 a4 a8 a2 a3 a5 a6]
[a8 a8 a6 a1 a2 a3 a7 a4 a5]
```

The abstract elements of PC correspond to the following permutations

a1 = [], a2 = [[1, 2, 5, 7], [3, 4, 8, 6]], a3 = [[1, 3, 5, 8], [2, 6, 7, 4]],
a4 = [[1, 4, 5, 6], [2, 3, 7, 8]], a5 = [[1, 5], [2, 7], [3, 8], [4, 6]],
a6 = [[1, 6, 5, 4], [2, 8, 7, 3]], a7 = [[1, 7, 5, 2], [3, 6, 8, 4]],
a8 = [[1, 8, 5, 3], [2, 4, 7, 6]]

7. Conclusion

A computer algebra system, such as MAPLE V, can and should be used to help students learn abstract mathematical ideas by getting them involved in constructing the ideas. We strongly believe that students' active involvement with constructing mathematics for themselves is essential to understanding concepts. When students write a procedure, such as *gpElements*, that makes use of several other related concepts, they will learn those concepts through the action of having to describe them precisely to the computer. The mathemaical ideas they are able to put together for themselves with the help of MAPLE V will tend to be rich and meaningful to them.

8. References

1. **Baxter, Nancy, Dubinsky,Ed, & Levin, Gary**, *Learning Discrete Mathematics with ISETL*, Springer-Verlag, New York, 1988.
2. **Redfem, D.**, *The Maple Handbook*, Springer-Verlag, New York, 1993.
3. **Sawyer, W.W.,** *A Concrete Approach to Abstract Algebra* , Freeman & Co.,London. 1959.