Group Theory and SAGE: A Primer

Robert A. Beezer
University of Puget Sound

Revision: December 9, 2008

Introduction

This compilation collects SAGE commands that are useful for a student in an introductory course in group theory. It is not intended to teach SAGE or to teach group theory. (Pointers to other resources here.) Rather, by presenting commands roughly in the order a student would learn the corresponding mathematics they might be encouraged to experiment and learn more about mathematics and learn more about SAGE. The "E" in SAGE once stood for "Experimentation."

Basic Properties of the Integers

Integer Division

`a % b` will return the remainder upon division of a by b. In other words, the value is the unique integer r such that (1) $0 \leq r < b$, and (2) $a = bq + r$ for some integer q (the quotient). Then $(a - r)/b$ will equal q. For example,

```
r = 14 % 3
q = (14 - r)/3
```

will set `r` to `2`, and set `q` to `4`.
(Add caution about division on integers.)

`gcd(a,b)`

Returns the greatest common divisor of a and b, where in our first uses, a and b are integers. Later, a and b can be other objects with a notion of divisibility and "greatness," such as polynomials. For example, `gcd(2776,2452)` will return `4`.

xgcd(a,b)

"Extended gcd." Returns a triple where the first element is the greatest common divisor of a and b (as with the `gcd` command above), but the next two elements are the values of r and s such that $ra + sb = \gcd(a, b)$. For example, `xgcd(633,331)` returns `(1, 194, -371)`. Portions of the triple can be extracted using `[ ]` to access the entries of the triple, starting with the first as number `0`. For example, the following should return the result `True` (even if you change the values of `a` and `b`).

```
a = 633
b = 331
extended = xgcd(a, b)
g = extended[0]
r = extended[1]
s = extended[2]
g == r*a + s*b
```

Divisibility

A remainder of zero indicates divisibility. So `(a % b) == 0` will return `True` if b divides a, and will otherwise return `False`. For example, `(9 % 3) == 0` is `True`, but `(9 % 4) == 0` is `False`.

factor(a)

As promised by the Fundamental Theorem of Arithmetic, `factor(a)` will return an expression for a as a product of powers of primes. It will print in a nicely-readable form, but can also be manipulated with Python as a list of pairs (p_i, e_i) containing primes as bases, and their associated exponents. For example, `factor(2600)` returns $2\hat{3} * 5\hat{2} * 13$. We can strip off pieces of the prime decomposition using two levels of `[ ]`. This is a good example to study in order tolearn about how to drill down into Python lists.

```
n = 2600
decomposition = factor(n)
print n, " decomposes as ", decomposition
secondterm = decomposition[1]
print "Base and exponent (pair) for second prime: ", secondterm
base = secondterm[0]
exponent = secondterm[1]
print "Base is ", base
print "Exponent is ", exponent
thirdbase = decomposition[2][0]
thirdexponent = decomposition[2][1]
print "Base of third term is ", thirdbase, " with exponent ", thirdexponent
```

Multiplicative Inverse, Modular Arithmetic

`inverse_mod(a, n)` yields the multiplicative inverse of a mod n (or an error if it doesn't exist). For example, `inverse_mod(352, 917)` yields `508`. (Check that `352*508 = m*917+1` for some integer m.)

Powers, Modular Arithmetic

`power_mod(a, m, n)` yields a^m mod n. For example, `power_mod(15, 831, 23)` yields `10`. If $m = -1$, then this duplicates the function of `inverse_mod()`.

Euler ϕ-function

`euler_phi(n)` will return the number of positive integers less than n, and relatively prime to n (i.e. having greatest common divisor with n equal to 1). For example, `euler_phi(345)` should return 176. Try the following experiment:

```
m = random_prime(10000)
n = random_prime(10000)
print(m, n, euler_phi(m*n) == euler_phi(m)*euler_phi(n))
```

Feel a conjecture coming on? Can you generalize this result?

Primes

`is_prime(a)` returns `True` or `False` depending on if a is prime or not. For example, `is_prime(117371)` returns `True`, while `is_prime(14547073)` returns `False` ($14547073 = 1597 * 9109$).

`random_prime(a,True)` will return a random prime between 2 and a.
Experiment with `random_prime(10^21,True)`. (Replacing `True` by `False` will speed up the search, but there will be a very small probability the result will not be prime.)

`prime_range(a,b)` returns the list of all the primes from a to $b-1$, inclusive. For example, `prime_range(500,550)` returns `[503, 509, 521, 523, 541, 547]`.

`next_prime(a)` and `previous_prime(a)` are other ways to get a single prime number of a desired size.

Permutation Groups

A good portion of SAGE's support for group theory is based on routines from GAP (Groups, Algorithms, and Programming). The most concrete way to represent groups is via permutations (of the integers 1 through n), using function composition as the operation.

Writing Permutations SAGE uses "disjoint cycle notation" for permutations, see Section 4.1 of Judson for more on this. Composition occurs *left to right*, which is not what you might expect and is exactly the reverse of what Judson uses. There are two ways to write the permutation $\sigma = (1\,3)(2\,5\,4)$,

1. As a text string: `"(1,3)(2,5,4)"`
2. As a Python list of "tuples": `[(1,3), (2,5,4)]`

Groups SAGE knows many popular groups as sets of permutations. More are listed below, but for starters, the full symmetric group can be built with the command `SymmetricGroup(n)` where n is the number of symbols.

Permutation Elements Elements of a group can be created, and manipulated, as follows

```
G = SymmetricGroup(5)
sigma = G("(1,3)(2,5,4)")
rho = G([(1,4), (1,5)])
rho^-1*sigma*rho
```

Available functions include `sigma.order()`, which will return 6, the smallest power of k such that σ^k equals the identity element (), and `sigma.sign()` which will be 1 for an even permutation and -1 for an odd permutation (here σ is an odd permutation).

Groups An annotated list of some small permutation groups known to SAGE.
(More in `/sage/devel/sage/sage/groups/perm_gps/permgroup_named.py`)
`SymmetricGroup(n)`: All permutations on n symbols.

`DihedralGroup(n)`: Symmetries of an n-gon.

`CyclicPermutationGroup(n)`: Rotations of an n-gon (no flips).

`AlternatingGroup(n)`: Alternating group on n symbols, $n = 4$ is symmetries of tetrahedron.

`KleinFourGroup()`: The non-cyclic group of order 4.

Group Functions

Define a group, say `H = DihedralGroup(6)`, and then a variety of functions become available:

is_abelian()

The command `H.is_abelian()` will return `False`.

order()

`H.order()` will return the order of the group. So `H.order()` returns `12`.

list(), cayley_table()

The command `H.list()` will return the elements of H in a fixed order, and `H.cayley_table ()` will construct the Cayley table of H using the elements of the group in the same order as the `list()` command, and denoting them as `xN` where N is an integer.

center()

The command `H.center()` will return a subgroup that is the center of the group H (see Exercise 2.46 in Judson). Try `H.center().list()` to see which elements of H commute with *every* element of H.

Cayley Graph

For fun, try `show(H.cayley_graph())`.

Subgroups

Cyclic Subgroups

If `G` is a group, and `a` is an element of the group (try `a=G.random_element()`), then
`H=G.subgroup([a])`
will create `H` as the cyclic subgroup of `G` with generator `a`.

Consider the following example (note that the indentation of the third line is critical):

```
C20 = CyclicPermutationGroup(20)
for g in C20:
  print (g, g.order())
```

which will list the elements of a cyclic group of order 20, along with the order of each element. We can cut/paste an element of order 5 and build a subgroup,

```
H = C20.subgroup(["(1,17,13,9,5)(2,18,14,10,6)(3,19,15,11,7)(4,20,16,12,8)"])}
H.list()
```

For a cyclic group, the following command will list all of the subgroups.

```
C20.conjugacy_classes_subgroups()
```

Be careful, this command uses some more advanced ideas, and will not usually list *all* of the subgroups of a group — here we are relying on special properties of cyclic groups (but see the next section).

All Subgroups

If H is a subgroup of G and $g \in G$, then $gHg^{-1} = \{ghg^{-1} \mid h \in G\}$ will also be a subgroup of G. If `G` is a group, then the command `G.conjugacy_classes_subgroups()` will return a list of subgroups of `G`, but not all of the subgroups. However, every subgroup can be constructed from one on the list by the gHg^{-1} construction with a suitable g.

Symmetry Groups

You can give SAGE a short list of elements of a permutation group and SAGE will find the smallest subgroup that contains those elements. We say the list "generates" the subgroup. We list a few interesting subgroups you can create this way.

Symmetries of a Tetrahedron

Label the 4 vertices of a regular tetahedron as 1, 2, 3 and 4. Fix the vertex labeled 4 and roate the opposite face through 120 degrees. This will create the permutation/symmetry (1 2 3). Similarly, fixing vertex 1, and rotating the opposite face will create the permutation (2 3 4). These two permutations are enough to generate the full group of the twelve symmetries of the tetrahedron. Another symmetry can be visualized by running an axis through the midpoint of an edge of the tetrahedron through to the midpoint of the opposite edge, and then rotating by 180 degrees about this axis. For example, the 1–2 edge is opposite the 3–4 edge, and the symmetry is described by the permutation (1 2)(3 4). This permutationj, along with either of the above permutations will also generate the group. So here are two ways to create this group,

```
tetra_one = PermutationGroup(["(1,2,3)", "(2,3,4)"])
tetra_two = PermutationGroup(["(1,2,3)", "(1,2)(3,4)"])
```

This group has a variety of interesting properties, so it is worth experimenting with. You may also know it as the "alternating group on 4 sysmbols," which SAGE will create with the command `AlternatingGroup(4)`.

Symmetries of a Cube

Label vertices of one face of a cube with 1, 2, 3 and 4, and on the opposite face label the vertices 5, 6, 7 and 8 (5 opposite 1, 6 opposite 2, etc.). Consider three axes that run from the center of a face to the center of the opposite face, and consider a quarter-turn rotation about each axis. These three rotations will construct the entire symmetry group. Use

```
cube = PermutationGroup(["(3,2,6,7)(4,1,5,8)",
                         "(1,2,6,5)(4,3,7,8)", "(1,2,3,4)(5,6,7,8)"])}
```

A cube has four distinct diagonals (joining opposite vertices through the center of the cube). Each symmetry of the cube will cause the diagonals to arrange differently. In this way, we can view an element of the symmetry group as a permutation of four "symbols" — the diagonals. It happens that *each* of the 24 permutations of the diagonals is created by exactly one symmetry of the 8 vertices of the cube. So this subgroup of S_8 is "the same as" S_4. In SAGE, `cube.is_isomorphic(SymmetricGroup(4))` will test to see if the group of symmetries of the cube are "the same as" S_4.

Normal Subgroups

is_normal()

Begin with the alternating group, A_4: `A4=AlternatingGroup(4)`.

Construct the three symmetries of the tetrahedron that are 180 degree rotations about axes through midpoints of opposite edges:
`r1=A4("(1,2)(3,4)")`, `r2=A4("(1,3)(2,4)")`, `r3=A4("(1,4)(2,3)")`

And construct a subgroup of size 4: `H=A4.subgroup([r1,r2,r3])`

Now check if the subgroup is normal in A_4: `H.is_normal(A4)` returns `True`.

quotient_group()

Extending the previous example, we can create the quotient (factor) group of A_4 by H.
`A4.quotient_group(H)` will return a permutation group generated by `(1,2,3)`. As expected this is a group of order 3. Notice that we do not get back a group of the actual cosets, but instead we get a group isomorphic to the factor group.

is_simple()

It is easy to check to see if a group is void of any normal subgroups.
`AlternatingGroup(5).is_simple()` is `True` and `AlternatingGroup(4).is_simple()` is `False`.

composition_series()

For any group, it is easy to obtain a composition series. There is an element of randomness in the algorithm used for this, so you may not always get the same results. (But the list of factor groups is unique, according to the Jordan-Holder theorem.) Also, the subgroups generated sometimes have more generators than necessary, so you might want to "study" each subgroup carefully by checking properties like its order.

An interesting example is: `DihedralGroup(105).composition_series()`
The output will be a list of 5 subgroups, each a normal subgroup of its predecessor.

Several other series are possible, such as the derived series. Use tab completion to see the possibilities.

Conjugacy

Given a group G, we can define a relation $\sim$ on G by: for a, $b \in G$, $a \sim b$ if and only if there exists an element $g \in G$ such that $gag^{-1} = b$.
Since this is an equivalence relation, there is an associated partition of the elements of G into equivalence classes. For this very important relation, the classes are known as "conjugacy classes.". A representative of each of these equivalence classes can be found as follows. Suppose `G` is a permutation group, then `G.conjugacy_classes_representatives()` will return a list of elements of G, one per conjugacy class. Given an element $g \in G$, the "centralizer" of g is the set $C(g) = \{h \in G \mid hgh^{-1} = g\}$, which is a subgroup of G. A theorem tells us that the size of each conjiugagcy class is the order of the group divided by the order of the centralizer of an element of the class. With the following code we can determine the size of the conjugacy classes of the sfull symmetric group on 5 symbols,

```
G = SymmetricGroup(5)
group_order = G.order()
reps = G.conjugacy_classes_representatives()
class_sizes = []
for g in reps:
  class_sizes.append( group_order/G.centralizer(g).order() )
print class_sizes
```

This should produce the list `[1, 10, 15, 20, 20, 30, 24]` which you can check sums to 120, the order of the group. You might be able to produce this list just by counting elements of the group with identical cycle structure.

Sylow Subgroups

Sylow's Theorems assert the existence of certain subgroups. For example, if p is a prime, and p^r divides the order of a group G, then G must have a subgroup of order p^r. Such a subgroup could be found among the output of the `conjugacy_classes_subgroups()` command by checking the orders of the subgroups produced. The `map()` command is a quick way to do this. The symmetric group on 8 symbols, S_8, has order $8! = 40,320$ and is divisible by $2^7 = 128$. Let's find one example of a subgroup of permutations on 8 symbols with order 128. The next command takes a few minutes to run, so go get a cup of coffee after you set it in motion.

```
G = SymmetricGroup(8)
subgroups = G.conjugacy_classes_subgroups()
map( order, subgroups )
```

The `map(order, subgroups)` command will apply the `order()` method to each of the subgroups in the list `subgroups`. The output is thus a large list of the orders of many (296) subgroups. if you count carefully, you will see that 259^{th} subgroup has order 128. You can retrieve this group for further study by referencing it as `subgroups[258]` (remember that counting starts at zero).
If p^r is the highest power of p to divide the order of G, then a subgroup of order p^r is known as a "Sylow p-subgroup." Sylow's Theorems also say any two Sylow p-subgroups are conjugate, so the output of `conjugacy_classes_subgroups()` should only contain each Sylow p-subgroup once. But there is an easier way, `sylow_subgroup(p)` will return one. Notice that the argument of the command is just the prime p, not the full power p^r. Failure to use a prime will generate an informative error message.

Groups of Small Order as Permutation Groups

We list here constructions of all of the groups of order less than 16, as permutation groups.

Size	Construction	Notes
1	`SymmetricGroup(1)`	Trivial
2	`SymmetricGroup(2)`	Also `CyclicPermutationGroup(2)`
3	`CyclicPermutationGroup(3)`	Prime order
4	`CyclicPermutationGroup(4)`	Cyclic
4	`KleinFourGroup()`	Abelian, non-cyclic
5	`CyclicPermutationGroup(5)`	Prime order
6	`CyclicPermutationGroup(6)`	Cyclic
6	`SymmetricGroup(3)`	Non-abelian, also `DihedralGroup(3)`
7	`CyclicPermutationGroup(7)`	Prime order
8	`CyclicPermutationGroup(8)`	Cyclic

8	`D1=CyclicPermutationGroup(4)`
	`D2=CyclicPermutationGroup(2)`
	`G=direct_product_permgroups([D1,D2])`
		Abelian, non-cyclic

8	`D1=CyclicPermutationGroup(2)`
	`D2=CyclicPermutationGroup(2)`
	`D3=CyclicPermutationGroup(2)`
	`G=direct_product_permgroups([D1,D2,D3])`
		Abelian, non-cyclic

8	`DihedralGroup(4)`	Non-abelian
8	`PermutationGroup(["(1,2,5,6)(3,4,7,8)",` `"(1,3,5,7)(2,8,6,4)" ])`	Quaternions The two generators are I and J
9	`CyclicPermutationGroup(9)`	Cyclic

9	`D1=CyclicPermutationGroup(3)`
	`D2=CyclicPermutationGroup(3)`
	`G=direct_product_permgroups([D1,D2])`
		Abelian, non-cyclic

10	`CyclicPermutationGroup(10)`	Cyclic
10	`DihedralGroup(5)`	Non-abelian
11	`CyclicPermutationGroup(11)`	Prime order
12	`CyclicPermutationGroup(12)`	Cyclic

12	`D1=CyclicPermutationGroup(6)`
	`D2=CyclicPermutationGroup(2)`
	`G=direct_product_permgroups([D1,D2])`
		Abelian, non-cyclic

12	`DihedralGroup(6)`	Non-abelian
12	`AlternatingGroup(4)`	Non-abelian, symmetries of tetrahedron
12	`PermutationGroup(["(1,2,3)(4,6)(5,7)",` `"(1,2)(4,5,6,7)"])`	Non-abelian Semi-direct product $Z_3 \rtimes Z_4$
13	`CyclicPermutationGroup(13)`	Prime order
14	`CyclicPermutationGroup(14)`	Cyclic
14	`DihedralGroup(7)`	Non-abelian
15	`CyclicPermutationGroup(15)`	Cyclic