PURE Math 2012 Seminar
Week 2 Computer Laboratory Exercises

Background and Goals

This week, we will begin to work with some of the commands related to monomial orders and computing Gröbner bases in Sage. Note: this time the lab problems you are to submit are "sprinkled through" the discussion, marked *Exercise 1*, *Exercise 2*, and so forth.

Days 1,2: Lists, Defining Monomial Orders, Polynomial Division

Before we start in with the division algorithm, we need to take a bit of time and look at some of the details of *lists* in Sage.

Lists and Vectors

To represent points in affine space, we use coordinate vectors. The same idea is also useful to represent any *ordered collection* of information. In Sage, these ordered collections are called *lists*, and the elements of a list can be *anything*. A list is indicated by a pair of *square brackets* ([,]) enclosing the items in the list, separated by commas. For instance, the input line

```
list1 = ['a','b','c','d','e']
```

creates a list with five items (the letters a, b, c, d, e, treated as character strings, not as symbolic variables), and assigns the list to the name `list1`. We have said that lists are *ordered*. What this means, for instance, is that `alist` above is *different from* the list defined by `list2 = ['a','d','c','e','b']`. You can test this by entering the definition of `list2`, then the command

```
list1 == list2
```

(The == is the comparison operator – not assignment; it returns a True or False value indicating whether the two lists are identical.)

A list can have any number of items, including *no items* at all – the empty list is written as `[]`. The same item can also occur at several places in a list (unlike the case for a *set*, which is an unordered collection of information with no repetitions).

The items in a list *do not* all have to be items of the same *type*. For instance, another perfectly OK Sage list is the following:

```
list3 = [['a','b','c'],'d','e']
```

where there are three items in all. (The first list item in `list3` is the list `['a','b','c']`, and the second and third items are the character strings `'d','e'`.)

Some useful list operations

- The built-in `len` function returns the number of items in a list. (If one of the items is a list itself, it is counted as a single item.)
- To pick out the ith item in a named list (like `list1`), you can use the notation `list1[i]`. Caution: Lists are always numbered starting from 0. So `list1[1]` is actually the *second* element(!)
- You can also pick out a "slice" or consecutive sublist – the items in the slots numbered `first` to `last - 1` – from a list by using the format `list1[first:last]`. This gives a list as output.
- The `append` operator inserts a new entry at the end of a list. Try `list1.append('k')` to see the result.
- Sage has an interesting feature called "tab-completion" that can be used as a quick reference tool. Try entering `list1.` and then press the TAB key. You should see a listing of different ways the partial command could be completed.

Tuples and sets

Sage also has data types called *tuples* and *sets*. A tuple is similar to a list, but once it is created it cannot be changed (it is *immutable*). A set is an unordered list. You can find information about these in the online tutorials and manual.

Exercise 1

Explain, and give examples showing:

a) how to insert a new item at the *start* of a list
b) how to create a list containing the entries of a given list, followed by a second copy of that same list
c) how to insert a new item as the third entry from the start in a list (assuming it has at least two entries to begin with) Be careful to test your idea both on lists with two and with more than two entries!

Note: There are other builtin list commands not discussed above. Some of these tasks may be covered by one of them, but they can also be done using the commands above.

Matrices in Sage

If we think of a list as a row vector, then it is natural to think of a list of lists (all of the same size) as a way to represent a *matrix*. We can do this in Sage as well, as a basic way to deal with square or more general rectangular matrices. For example,

```
mat = [[1,2],[3,4]]
```

creates a list of two row vectors. This can also be thought of as a 2×2 matrix, but Sage will not see it that way unless we explicitly say

```
Mat = Matrix(mat); Mat
```

Now you will see something like the stacked matrix format as the output.

To select the entry in row i and column j from a matrix defined this way, we could use, for instance `Mat[i,j]`. Rows and columns are also numbered starting from 0. In the list-of-lists form we would use `mat[i][j]` – here `mat[i]` is the ith row, so `mat[i][j]` is the jth entry in that row.

Working in Sage/Singular

Note: From now on, we will essentially be using Sage as an interface to Singular (a package for computational commutative algebra, algebraic geometry, and singularity theory). There are several ways to do this, giving progressively more direct access to the underlying functionalities of the Singular system. But by far the easiest way to get started is to use Sage-level commands for defining polynomial rings, ideals, and then computing Gröbner bases, etc. This has the advantage that the output of the commands will be Sage objects of types we can predict (rather than Singular objects).

Monomial orders in Sage/Singular

To set up Sage to do computations in a polynomial ring with a specified collection of variables, using a particular monomial order, we can use the `Polynomial Ring` command seen earlier, but include an option that specifies a particular monomial order to be used for all operations involving that information in that ring. For example, to set up a ring in 4 variables x, y, z, w with coefficients in $\mathbf{Q}$ and the lex order with $x > y > z > w$ we could say

```
R.<x,y,z,w> = PolynomialRing(QQ,order='lex')
```

Other builtin options for defining orders include:

- `'deglex'` for the graded lex order
- `'degrevlex'` for the graded reverse lex order
- `'wdegrevlex'` for the order $>_{w,grevlex}$ defined by comparing monomials with a positive integer component weight vector w first, breaking ties with graded reverse lex. Here is an example showing the syntax for the monomial order specification using the weight vector $w = (4, 3, 2, 1)$ of length 4 (the number of variables):

```
S.<x,y,z,w> = PolynomialRing(QQ,order=TermOrder('wdegrevlex',(4,3,2,1)))
```

- Sage also has a completely general way to specify monomial orders using matrices that we will see shortly.

Once the ring has been defined with a specified order, there are built-in operators that select leading monomials and leading coefficients:

- `f.lt()` gives the leading term (including the coefficient)
- `f.lm()` gives the leading monomial (without the coefficient)
- `f.lc()` gives the leading coefficient.

There is also an operator that performs the multivariable polynomial division algorithm using a given list of divisors, and returns the remainder:

- `f.reduce(L)`

The list `L` here can be any list of polynomials in the base ring containing the polynomial f. *The division is performed using an algorithm that does not always give the same results as the one we discussed in class, though.* If you want to duplicate that entirely, the following Sage function can be used, after setting up the appropriate ring definition:

```
def PolyDiv(f,PList):
   s = len(PList)
   a = [0 for i in range(0,s)]
   r = 0
   p = f
   while p <> 0:
     i = 0
     divocc = False
     while (i < s) and (divocc == False):
       q = p.lt()/PList[i].lt()
       if q in R:
         a[i] = a[i] + q
         p = R(p - q*PList[i])
         divocc = True
       else:
         i = i + 1
     if divocc == False:
       r = r + p.lt()
       p = p - p.lt()
   return(a,r)
```

Exercise 2

We did problem 1 from Chapter 2, §3 of "IVA" in discussion. Check your work using Sage.

Exercise 3

Do problems 5-8 in Chapter 2, §3 of "IVA" using Sage.

Defining a Monomial Order by a Matrix

In discussion today, we saw that we can generalize the weight orders $>_w$, to define monomial orders on $k[x_1, \ldots, x_n]$ starting from any $m \times n$ matrix M with

- $m \geq n$,

- $\text{rank}(M) = n$,
- all entries non-negative integers.

Namely, suppose the rows of M are the vectors $w_1, \ldots, w_n$. Then we can compare monomials x^α and x^β by first comparing their w_1-weights, then breaking ties sucessively with the w_2-weights, w_3-weights, and so on until the w_m-weights. In symbols:

$$\begin{aligned} x^\alpha >_M x^\beta \Leftrightarrow\ & w_1 \cdot \alpha > w_1 \cdot \beta \\ & \text{or } [(w_1 \cdot \alpha = w_1 \cdot \beta) \text{ and } (w_2 \cdot \alpha > w_2 \cdot \beta)] \\ & \text{or } [(w_1 \cdot \alpha = w_1 \cdot \beta) \text{ and } (w_2 \cdot \alpha = w_2 \cdot \beta) \text{ and } (w_3 \cdot \alpha > w_3 \cdot \beta)] \\ & \text{or } \cdots \\ & \text{or } [(w_1 \cdot \alpha = w_1 \cdot \beta) \text{ and } \cdots \text{ and } (w_{m-1} \cdot \alpha = w_{m-1} \cdot \beta) \text{ and } (w_m \cdot \alpha > w_m \cdot \beta)] \end{aligned}$$

All monomial orders can be specified as $>_M$ orders for appropriate matrices M. In Sage, M must be an invertible square matrix with integer entries.

To specify that we want to use the matrix order defined by a suitable square matrix `wmat`, in the ring definition, we enter

```
order = TermOrder(wmat)
```

The matrix `wmat` itself can be specified as above using `Matrix` applied to the list of lists as above, either first in a command assigning the matrix to a name, or else directly in the `TermOrder` option. The list of variables in the ring definition indicates which components in the weight vectors (the rows of `wmat`) apply to which variables. For example, if for some reason we did not want to use the builtin lex order, we could define something equivalent on $\mathbf{Q}[x, y, z]$ using

```
order = TermOrder(Matrix([[1,0,0],[0,1,0],[0,0,1]])
```

Exercise 4

What matrix would define the weight order on $\mathbf{Q}[x, y, z]$ where we compare exponents first with the weight vector $(1, 3, 7)$, then break ties using the lex order? Create an appropriate matrix term order in Sage, and repeat the division from Exercise 2a in Chapter 2, §3 from "IVA".

Additional Examples of Monomial Orders

Exercises 10 and 12 from Chapter 2, §5 of "IVA" contain two additional important examples of monomial orders – the *product* orders, and Bayer and Stillman's *elimination* orders.

Exercise 5

Let $R = \mathbf{Q}[x_1, x_2, x_3, y_1, y_2, y_3, y_4]$.

a) Using the matrix order setup in Sage, determine how to define a mixed order $>_{mixed}$ on R where

$$x^\alpha y^\beta >_{mixed} x^\gamma y^\delta \Leftrightarrow x^\alpha >_{lex} x^\gamma \text{ or } ((x^\alpha = x^\gamma) \text{ and } (y^\beta >_{grevlex} y^\delta)).$$

(Hint: Think of making the matrix from "blocks" like the ones we have seen for the orders separately.)

b) Using Sage's setup, determine how to define the elimination order $>_3$ from Exercise 12.

Days 3,4: Gröbner Bases

Now, we are ready to use Sage to compute Gröbner bases of some ideals. We will use these computations to decide ideal membership. In addition, with lexicographic Gröbner bases we will see additional examples related to the Elimination and Extension Theorems from Chapter 3, §1.

Exercise 6

Do parts a,b,d from Problem 5 in Chapter 2, §6 of "IVA" using lex order with $x > y > z$, lex with $z > y > z$, and then graded lex with $x > y > z$. (Note: Sage doesn't have a directly accessible S-polynomial operator, so you will need to do this "by hand.") What is the answer to the following problem 6 based on the results?

Buchberger's Criterion

Recall that Buchberger's Criterion says $G = \{g_1, \ldots, g_t\}$ is a Gröbner basis for the ideal it generates if and only if for all pairs $i \neq j$,

$$\overline{S(g_i, g_j)}^G = 0,$$

where $\overline{f}^G$ is the remainder on division by G. Although we will be skipping the proof of this fact, it is very important – it is the foundation for the algorithm for computing Gröbner bases that we will introduce tomorrow.

Exercise 7

Using Buchberger's Criterion, do parts a and c of problem 9 in Chapter 2, §6 of "IVA." The remainders can be computed via the builtin command `reduce` we discussed before.

Buchberger's Algorithm

Starting from any set of generators for an ideal I, Buchberger's Algorithm produces a Gröbner basis for I by building up a set of polynomials in the ideal for which Buchberger's Criterion is satisfied. This is done by adjoining any nonzero S-polynomials that

are discovered in applying the criterion to the current set of polynomials. The pseudocode description is given on p. 87 of "IVA" and was discussed in class.

Exercise 8

Using Sage to "automate" the calculation of the S-polynomials, but controling the current list of polynomials "by hand," apply Buchberger's Algorithm to find a Gröbner basis of the ideal

$$\langle x^2 - xy + 1, xy - y + 2 \rangle$$

using lex order with $x > y$, then lex with $y > x$, and finally the matrix order for

$$M = \begin{pmatrix} 2 & 5 \\ 3 & 1 \end{pmatrix}.$$

If you are feeling really confident of your Sage skills, it would also be possible to code a Sage function carrying out the basic Buchberger algorithm from the text.

Exercise 9

There is a builtin `groebner_basis()` operator in Sage as well that can be applied to an ideal (with a particular set of generators) in a polynomial ring. To apply it, you would first define the ring, second the ideal, and then compute the Gröbner basis like this:

```
RR.<x,y> = PolynomialRing(QQ,order='lex')
I = Ideal([x^2 + 7*x*y^3 + 1, x^2*y^2 - 4*y - 1])
B = I.groebner_basis()
B
```

What happens if you apply it to the ideal from Exercise 8 with the monomial orders given there? Show that the bases that Sage finds generate the same ideal as your answers. (Recall the ideal membership test based on remainders.)

Exercise 10

a) Do Problems 1 and 3 in Chapter 2, §8 using Sage.
b) Do Problem 4 in Chapter 3, §1 using Sage.