PURE Math 2012 Seminar
Week 1 Computer Laboratory Exercises

Background and Goals

A part of our work this summer will be carrying out calculations of Gröbner bases for solving systems of equations and other computations within the computer algebra system *Sage*, using its connection with **Singular**. This week, we will want to get familiar doing mathematics in this context.

General Information about Sage

Launching Sage

If you are running Sage on a Mac, you can just launch it by XXX. On a PC, you will need to launch the VirtualBox, "power on" Sage, open a web browser and launch `http://localhost:8000` to access the Sage notebook server.

Sage worksheets

Worksheets are integrated text/graphics/mathematics documents where any or all of the following can be done:

1) you can type in commands from the keyboard to ask Sage to perform many different kinds of calculations, read data from external files, store work in files, etc.
2) output generated by Sage from your input commands (numerical values, symbolic formulas, and graphics) will be displayed,
3) you can modify commands, generating new output, store your results for use later, etc.
4) you can enter text to annotate and explain the results of computations.

Some of your interactive "management" of a in progress will take place through the pull-down menus across the top the "scroll bar" on the right that you can use to move around within the worksheet, to see previous input and output lines, etc.

When you start a new worksheet, Sage will ask you immediately for a name. This can be almost anything, but it will be good to use descriptive names so that you can tell what is in a worksheet from its name.

Input and output in a worksheet

After you name a new worksheet, or open an existing one, you will see a collection of outlined boxes, with spaces between them for output, etc. All Sage input is entered in these boxes; there can be any number of input lines within each box. Clicking on a box highlights the frame, makes that box "active" for input, and generates a link marked `evaluate` below the box. Typically a session will consist of you entering new input commands, evaluating, checking the results, correcting errors if necessary, and repeating the process until you have what you are looking for. Worksheets are saved periodically automatically; there is also a Save button at the top that you can use at any time.

If you want to enter a new input box, note that moving the cursor over the region below the output from an existing box produces a heavy blue horizontal line above the next input box. This is an *insertion point.* If you left click on that line, you will insert a new input box at that point in the worksheet.

Caution: You can enter new commands in any box in a worksheet at any time. The current values of variables and the sequence of commands performed, though, is determined by the order of the evaluations you have done. To keep things straight, it is probably better (at least at first) not to "jump around" too much.

As we mentioned before, Sage worksheets can contain text as well as commands and output. After you have generated the output you want, you can bring up the insertion point as above. If you shift-left click now (not just left click), Sage will generate a text input box for you). Note: If you anticipate adding stuff to one of these boxes over several input/output cycles, don't save it right away. Once you save, you cannot go back and edit the text(!) But with that caveat, you can enter text almost anywhere in the worksheet to annotate your work.

Saving and reloading your Sage worksheets

When you open the `localhost:8000` web page in your browser, you will see a list of active worksheets (if you have any). Clicking on the name of one will lauch that worksheet.

Getting Out

The Home link at the top of the Sage worksheet will take you back to the list of active worksheets. From there you can stop (the most common operation), archive, or delete any of the active worksheets. The download and upload options there make it easy to share worksheets, store them remotely, etc.

Days 1, 2: First Sample Sage Sessions, Plotting Commands

Let's get right down to work and walk through a first sample session! First, you will need to launch a Sage worksheet as described in the General Information section above.

Sage commands

Sage commands typically create a new object, perhaps assign it a name, and/or perform one or more operations on it. The basis for Sage is the programming language Python, and Sage has builtin interfaces to a number of very powerful open-source packages, which makes Sage a very flexible system for performing calculations of various kinds. For instance we will be using the Sage interface to Singular (one of the most powerful packages packages for Gröbner basis calculations) extensively this summer.

The assignment operator =

A Sage command of the form

```
name = expression;
```

assigns the result of evaluating the right hand side to the name on the left. Usually, the right hand side will involve a known expression, or performing some operation on some information we already know. We want to take the result of the operation and give it an abbreviated "name" for later use. For instance,

```
var('x')
cube = (x^2 - 16)^3
```

Then later on, we could "reuse" the value of `cube` in other expressions, just by putting in the name `cube` at the appropriate place (e.g. `cube + x^4`).

Getting Help On-Line

The exact format Sage is expecting for each type of command is specified in the programming of the Sage system. Usually, there are several different ways to do any particular operation, but there is *little or no freedom* in the ordering of what has to go where and in what format. Much useful information on this for all the built-in commands, and LOTS of instructive examples are contained in the Sage on-line documentation. (Unfortunately, there is often much more in the documentation than you are probably looking for, so you will often need to sift through the pages for the exact information you want!) For example try looking at the on-line help page for `2D Plotting`. We'll see some other examples in a moment.

Sage Expressions

Formulas or expressions are entered in something like usual mathematical notation:

1) The symbols for addition, subtraction, multiplication and division are $+, -, *, /$ respectively.
2) The caret (ˆ) is the symbol for raising to a power.

Everything must be entered in one linear string of characters, so you will need to use *parentheses* to group terms to get the expressions you want. The rule to keep in mind is: *Sage always evaluates expressions by doing powers first, then products and quotients, then sums and differences, all left to right, unless parentheses are used to override these built-in rules*. For example, the expression `a + b c^2/d + e` is the same as the mathematical formula:

$$a + \frac{bc^2}{d} + e.$$

If you really want

$$\frac{a + bc^2}{d + e}$$

you will need to enter the expression `(a + b c^2)/(d + e)`. What if you really wanted:

$$\frac{(a + bc)^2}{d + e}?$$

3) Sage "knows" all the usual elementary functions from calculus. The names of the most common ones are `sin, cos, tan, exp, log, sqrt`. To use one of these functions in a Sage formula, you put the name, followed by the "argument" (that is, the constant or expression you are applying the function to) *in parentheses*.
4) If you have any question about whether a formula has been entered correctly (i.e. the way you wanted it to be), Sage can "pretty-print" it in mathematically-typeset form, which should make it easier to read. For instance, in an input box enter

```
var('x')
f = (x - 7)/(x^2 + 4*x + 12)
f
```

and then click the `evaluate` link under the active box. This should generate the formula you entered for f in the same form you entered it. If you want to see the "pretty-print" version, you can edit the content of the input box to make the last line

```
show(f)
```

You should see an output formula like this:

$$\frac{x - 7}{x^2 + 4x + 12}$$

Notes:

- The first line in the original input lines is necessary here unless you have done something else previously in your Sage session to tell it that x is the name of a symbolic variable. Without it you will get one of Sage's (unfortunately cryptic) error messages.
- Unless you put in `print` or `show` commands, only the output produced by the last line of an input box is normally displayed. (This is because it is frequently the case that you don't need to see intermediate results in multi-step computations.)
- The `show(f)` line can also be entered as `f.show()`. This illustrates the Python lineage of Sage. Many Sage operations can either be specified using something like mathematical function notation (the `show(f)` form) or using a *postfix operator* form (the `f.show()`) form. You will get to like the second form the more you use it, mainly because it allows a nice way to compose operators while editing your input lines only at their ends(!)

Some practice

How would you enter each of the following formulas as Sage expressions? First, declare x, y, a, b, c as symbolic variables with an input command

```
var('x,y,a,b,c')
```

Enter commands to assign these expressions to the given names, and "pretty-print" the results.

1)
$$f = 4xy^2 - 2x^3 + 3y - 7$$

2)
$$g = x^3 + 3x + 4 + \frac{1}{x^2} - \frac{x+2}{x^4 + 8x}$$

3) (Note: `sqrt(x)` is the Sage syntax for $\sqrt{x}$.)
$$q = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

2D plotting

The most basic Sage command for 2D plotting, used for graphs of the form $y = f(x)$ among other things, is called `plot`. The format is

```
plot(function,range,options)
```

where

1) `function` is the function to be plotted – the simplest way to specify one is via a formula for $f(x)$ (an *expression* in Sage)
2) `range` is the range of x-values you want to see plotted, entered in the form

```
(x, low, high)
```

3) `options` can be used to control the form of the plot if desired. No options need be specified however if you don't want to. More on this later.

Example 1

To plot $y = x^4 - 2x^3 + x - 5\sin(x^2)$ for $-2 \leq x \leq 1$, you could use the Sage `plot` command with no options:

```
plot(x^4 - 2 x^3 + x - 5 sin(x^2),(x,-2,1))
```

From the formula $y = x^4 - 2x^3 + x - 5\sin(x^2)$, you might guess that there is at least one other x-intercept for this graph for $x > 2$ (why?). To see that part of the graph as well, edit your previous command line to change the right hand endpoint of the interval of x values (do not retype the whole command). Click the `evaluate` link to have Sage execute the command again. Experiment until you are sure that your plot shows all the x-intercepts of this graph. (You can repeat this process of editing a command and re-running it as often as you want; the previous output is replaced by the new output each time.)

Example 2

Next, let's move to another 2D plotting command, to see the parametric curve $\alpha(t) = (t^2 - 1, t^3 - t)$. The basic format for parametric curves $\alpha(t) = (x(t), y(t))$ is

```
parametric_plot([x-comp,y-comp],t-range,options)
```

where

1) `x-comp` and `y-comp` are the x- and y-component functions of the curve to be plotted (each of these can be an *expression* involving the variable (parameter) t,
2) `t-range` is the range of t-values you want to see plotted, and
3) `options` can be used to control the form of the plot if desired. No options need be specified however if you don't want to.

Use this `parametric_plot` command to display a plot of the curve $\alpha(t) = (t^2 - 1, t^3 - t)$ for $t \in [-3, 3]$:

```
parametric_plot([t^2 - 1, t^3 - t],(t,-3,3))
```

(Did you get an error message? Look at it carefully and figure out what needs to be done to fix it.)
Example 3

Frequently, it's the relationship between two or more different graphs that you want to understand by looking at a plot. Sage has a very straightforward method for combining plots. Basically, you can use the `plot` or `parametric_plot` commands as above to generate the component plots, but you assign the output to names instead of displaying them. (The input lines given below also illustrate some plotting options we have not discussed before.) Then you can enter a command that combines the plots and displays them together like this:

```
sp1 = plot(sin(x),(x,0,4*pi),color='red')
sp2 = plot(sin(x+2),(x,0,4*pi),color='blue',linestyle='dashed')
sp1 + sp2
```

Implicit 2D plotting

The 2D plots above are all either graphs of functions $y = f(x)$, or parametric curves. In addition to these, we will also want to be able to plot the affine variety

$$\mathbf{V}(g(x, y)) = \{(x, y) \in \mathbf{R}^2 : g(x, y) = 0\}$$

for a general polynomial $g(x, y)$ in two variables. The technical name for this in Sage is *implicit curve plotting*. To plot these curves, we need a new command called `implicit_plot`. The format for the `implicit_plot` command is

```
implicit_plot(expression,x-range,y-range)
```

The expression should be the equation $g(x, y)$ (Sage assumes you mean to plot $g(x, y) = 0$.) This command generates a plot of the part of the variety $\mathbf{V}(g(x, y))$ in the rectangular box in the plane defined by the x- and y-ranges.

Example 4

For instance, try the following command:

```
implicit_plot(x^3 - 3 x y^2 - 3,(x,-3,3),(y,-3,3))
```

to plot the variety $\mathbf{V}(x^3 - 3xy^2 - 3)$. (Both x and y must be declared as symbolic variables for this to work(!)) Some questions to think about as you look at this: If you set $x = c$ (a constant), how many points (c, y) are there on the curve? How many points (x, d) are there for $y = d$? Do your answers depend on c, d? How?

3D curve and surface plotting

Now we move up a dimension and consider plotting curves and surfaces in $\mathbf{R}^3$. The `parametric_plot3d` command can be used to draw parametric curves and parametric surfaces in $\mathbf{R}^3$. To use it to draw a parametric curve, you would enter a command of the form

```
parametric_plot3d([x-comp,y-comp,z-comp],(t,low,high))
```

where `x-comp` $= x(t)$, `y-comp` $= y(t)$, `z-comp` $= z(t)$ are the parametric equations of the curve and the range of parameter values to be plotted is $low \leq t \leq high$.

Example 5

For instance, try entering

```
parametric_plot3d([t,t^2,t^3],(t,-2,2))
```

to plot a portion of the *twisted cubic* curve from class. The first plot you see here might be rather uninformative. Fortunately, Sage also lets you look at a 3D plot from different viewpoints, by dragging and dropping the viewing box in the worksheet output. Hold down the left mouse button and move the cursor to rotate the graph. Experiment with this until you feel comfortable.

Parametric surface plotting

The `parametric_plot3d` command is also used for plotting parametric surfaces in $\mathbf{R}^3$. The differences between this use of the command and that above is that the component functions will depend on *two parameters*, say u, v, and you will need to (define and) specify plotting ranges for each one.

Plotting a graph $z = f(x, y)$.

The command for plotting graphs of functions of two variables is called `plot3d` (naturally enough!) Its format is similar to, but not exactly the same as, the format of `plot`. To draw a graph with `plot3d`, you use a command of the format:

```
plot3d(function,xrange,yrange,options)
```

The `function` is the function $f(x, y)$, entered in the usual Sage syntax for expressions. The `xrange` and `yrange` specify a rectangular box in the plane that the plot will be constructed over; the `options` can be used to specify how the plot is drawn. Look at the online help for `plot3d` if you want to see what things are possible. You can change viewpoint (rotate the graph); The method is the same as that for the `parametric_plot3d` command described above.

Implicit surface plots

To plot an surface defined as a variety $\mathbf{V}(g(x, y, z))$, you use the `implicit_plot3d` command. Look up the on-line help listing for this command.

Assignment on Plotting

Prepare and submit a Sage notebook showing the plots asked for in the following questions. Answer any questions posed here with text annotations.

A) Generate a plot of the variety $\mathbf{V}(x^3 - 3x + 2xy^2 - y^4 - 1) \subset \mathbf{R}^2$. Add a text region answering the following questions: How many intersections of this variety are there with vertical lines ($x = a$) and horizontal lines ($y = b$) in $\mathbf{R}^2$? Does the answer depend on the values of a, b? Is it possible to “see” these numbers from the form of the equation $g(x, y)$? Explain.

B) What happens if you look at the twisted cubic curve from Example 5 above, from viewpoints that are located along the three coordinate axes (”far out” from $(0, 0, 0)$)? Show each of these plots and explain their shapes. (Hint: If we look at the curve from a point far out on the x-axis, what are we seeing?)

C) Generate a plot of the following variety in $\mathbf{R}^3$:

$$S = \mathbf{V}(z^2 - (16 - x^2 - y^2)((x + 2)^2 + y^2 - 1)((x - 2)^2 + y^2 - 1)/50)$$

and explain the shape you see. You will want to “walk around” this one a lot by rotating and looking at it from different viewpoints (For example, for which $(x, y) \in \mathbf{R}^2$ are there points (x, y, z) on S and why (that is, what is the projection of S into the (x, y) plane? You will also need to think about how you choose the x-, y-, and z-ranges. Be sure you take the x- and y- ranges big enough to see all the points on the variety.)

Choose either one of the following two problems, or both if you are really up for a challenge!

D) The line segment from (a, b, c) to (d, e, f) can be parametrized as follows

$$\alpha(t) = (a + t(d - a), b + t(e - b), c + t(f - c))$$

for $0 \leq t \leq 1$. The points $P = (0, 0, 0)$, $Q = (2, 0, 0)$, $R = (0, \sqrt{3}, 1)$, $S = (0, -1, \sqrt{3})$ are four corners of a cube in $\mathbf{R}^3$ with edges PQ, PR, and PS, because the vectors $\vec{u} =$

$Q - P, \vec{v} = R - P, \vec{w} = S - P$ all have magnitude 2, and $\vec{u} \cdot \vec{v} = \vec{u} \cdot \vec{w} = \vec{v} \cdot \vec{w} = 0$. Your assignment, should you decide to accept it (just kidding!), is to create a picture of this cube by drawing the 12 edges together on the same set of axes in $\mathbf{R}^3$ (begin by drawing the line segments from P to Q, P to R, and P to S). To plot several parametric curves together on the same set of axes, I suggest generating each line segment separately (one at a time), and then combining them as discussed above. Also explain how you found the other 4 corners of the cube. This can be done in a text region.

E) One very interesting and useful class of parametric curves are the *Bezier* curves discussed in Chapter 1, §3 of our book. A single Bezier cubic is a plane parametric curve

$$\beta(t) = t^3 P_0 + 3t^2(1-t)P_1 + 3t(1-t)^2 P_2 + (1-t)^3 P_3$$

where the P_i are called the *control points* – they control the location and the shape of the curve. In Sage, a single Bezier cubic, and its control points, can be plotted via commands like this:

```
points = [(0,0),(1,-1),(1,3),(3,0)]
BP = bezier_path([points])
SP = scatter_plot(points)
BP + SP
```

Try varying the locations of the control points and redrawing the curve to see how this works, that is how the endpoints of the curve and the tangent directions at the end points are determined by the points P_0, P_1, P_2, P_3.

Sage also lets you define *piecewise* Bezier curves linked together in a very particular way. You can list any number of points in lists like the list `points` above. The curves that are generated use the control points in a similar fashion to the basic Bezier cubics, but as you will see, the curve passes through every other point, and the intermediate points control the tangent directions. Curves like this are a major tool in *computer-aided geometric design.* (They were originally invented by an engineer at the Renault automobile works in France to help specify shapes involved in manufacture of auto parts.) They can be used to specify curved shapes appearing in many objects. In this problem your goal will be to construct a composite Bezier curve (that is determine a suitable collection of control points) that has the shape of the outline of the letter "c" in the typeface used for this document. Try match the overall shape, proportions, and details as closely as you can. In particular, note

- the small "knob" at the upper end,
- the gradual increase in thickness (distance between left and right boundaries) in the vertical portion, and
- the different curvature on the bottom end.

Designing typefaces is one "real-world" application of this mathematical idea.

Days 3, 4: Symbolic computation in Sage

Much of our work starting next week will consist of *symbolic* computations in Sage – mainly various operations on polynomials in several variables. Today, we will look at some first examples of this sort of computation, with polynomials in one variable.

Polynomials versus expressions

Even though some of the formulas we have run into in examples above look to us like polynomials, Sage would not recognize them that way. Sage makes a distinction between general symbolic expressions and polynomials. This means, for instance, that there are operations permitted on one type of object that are not defined on the other. This is somewhat annoying at times, but fortunately, there are ways to get Sage to display what type of thing a given object is and to change that if necessary.

Defining the ring of polynomials with coefficients in a specific ring

Say we want to perform computations with polynomials in the ring $R = \mathbf{Q}[x]$. Sage lets us define this structure in several different ways. Here is the most direct:

```
R.<x> = PolynomialRing(QQ)
```

This makes the name of the polynomial ring R, the coefficient ring $\mathbf{Q}$ (the `QQ` on the right), and it defines the name of the variable as `x`). There is also a short-hand form that does the same thing:

```
R.<x> = QQ[]
```

(Other possibilities for the coefficient ring would be `ZZ` – the ring of integers, `Integers(n)` for the integers mod n (fill in the specfic integer n, of course!), `GF(p)` for the finite field of order p, and many, many others.)

If you need to determine whether something (say `f`) is really a polynomial (for example, if it is causing an error message), you can enter a command like this:

```
f.parent()
```

The output will indicate the "parent structure" in which `f` is defined. It should be something like `Univariate Polynomial ring over x over Rational Field` for some of the following commands to work. If the out is something like `Symbolic Ring`, then the object *is not a polynomial*; it is just a general symbolic expression.

All is not lost, though. If you have a symbolic polynomial expression that contains only the variable x, you can "force" Sage to treat it it as an element of the ring R defined above with a command like this:

```
fR = R(f)
```

(you could also use the same name if you wanted to).

Operations on polynomials

Sage has built-in commands:

- `factor` to factor a polynomial. This gives the factorization over the specified coefficient ring. For instance, if we defined $\mathbf{Q}[x]$ as above, then the factor command does not know about radicals, complex numbers, etc. Formats: `factor(poly)` or `poly.factor()`. This works on polynomials in any number of variables.
- `quo_rem()` to compute the quotient and remainder on division of one polynomial $f(x)$ by another $g(x)$. This has a slightly strange format: `f.quo_rem(g)` Note: the polynomial `g` in the parentheses is used as the *divisor*; the polynomial being acted on (the `f`) is the dividend. If f or g contain other variables besides x, this will fail. The *output* from this is a Sage *list* – the quotient is the first element and the remainder is the second element. If you want to assign names to the outputs, you can do something like this:

  ```
  (q,r) = f.quo_rem(g)
  ```

  This divides g into f, assigns the quotient to the name q, and the remainder to r. (You can also extract the two components of the output by a sort of subscript notation. But caution – Sage lists are always numbered starting from 0 (another "Python thing!" This means that if you want to extract the quotient from a division and use it separately you would say

  ```
  QR = f.quo_rem(g)
  q = QR[0]
  ```
- `gcd` to compute the greatest common divisor of two polynomials $f(x)$ and $g(x)$. Format: `f.gcd(g)` The greatest common divisor of a set of several polynomials can be computed by nested gcd's (see p. 43 of "IVA"). For example $gcd(f, g, h)$ can be computed by `f.gcd(g).gcd(h)`.
- `diff(f,x)` computes the formal derivative of f with respect to x.

Here is a sample sequence of Sage input lines illustrating some of these commands. Try entering and executing them in sequence. Look carefully at the output and make sure you understand what happened in each case:

```
R.<x> = PolynomialRing(QQ)
cube = (x^2 - 16)^3
print cube
s = (x^2-x+1)*(x - 4)
print s
(q,r) = cube.quo_rem(s)
print "q = ", q," , r = ", r
q*s + r == s
```

(Note the double equals sign `==` in the last input line – this is different from the assignment operator `=`. What does it do?) Also try

```
print cube.gcd(s)
print s.gcd(cube)
```

Assignment on polynomial computations

Using Sage and the commands above, do problems 8, 9, 15 b from Chapter 1, §5 of "IVA". For 15 b, use the formula in 15 a.

A First Taste of Sage Programming

In addition to a having large collection of built-in commands for performing various mathematical computations, Sage is also a *programming language* that you can use to code and implement your own new commands or procedures. Here is a first example of a Sage procedure which takes as input two polynomials in one variable, $f(x)$ and $g(x)$, and computes their gcd $h(x)$, together with polynomials $A(x)$ and $B(x)$ satisfying

$$h(x) = A(x)f(x) + B(x)g(x)$$

(recall this is question 10 from Chapter 1, §5 of "IVA" from today's discussion). Here is Sage code for one way to do this:

```
def ExtEuc(f,g):
    A = 1; B = 0; C = 0; D = 1; s = g; h = f;
    while s <> 0:
        (q,r) = h.quo_rem(s)
        h = s
        s = r
        Csave = C
        Dsave = D
        C = A - q*C
        D = B - q*D
        A = Csave
        B = Dsave
    return (A,B,h)
```

Note that the syntax here is very similar to our pseudocode for algorithms. The indentation, though, is *not optional.* It determines how far the body of the while loop extends (so that the `return` at the end is not in the body). To use the procedure to compute the gcd and the polynomials A, B, you would enter a command like this:

```
ExtEuc(fpoly,gpoly)
```

where `fpoly` and `gpoly` are the two polynomials. The output from the procedure consists of the polynomials A and B and the gcd (last value of h). Use this to check your work from this morning.