MATHEMATICAL INSTITUTE

UNIVERSITY OF OXFORD

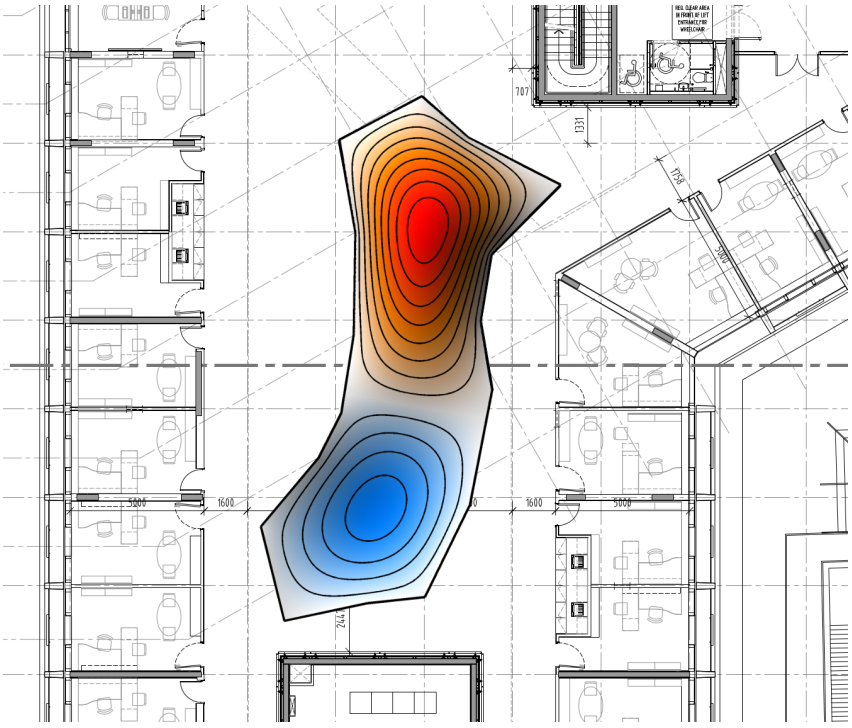
Computational Mathematics

Students' Guide

Hilary Term 2014

by

Dr Colin Macdonald



author: Colin Macdonald
compiled: January 22, 2014
git: HT2014-v1.2

Contents

Chapter 1

Introduction

The use of computers is widespread in all areas of life, and at universities they are used in both teaching and research. The influence and power of computing is fundamentally affecting many areas of both applied and pure mathematics. MATLAB is one of several systems used at Oxford for doing mathematics by computer; others include Mathematica, Maple, Sage and SciPy/NumPy. These tools are sufficiently versatile to support many different branches of mathematical activity, and they may be used to construct complicated programs.

1.1 Objectives

The objective of this practical course is to discover more about mathematics using MATLAB. Last term you were introduced to some basic techniques, by working through the Michaelmas Term Students' Guide which you are due to complete near the beginning of this term. After this, for the rest of Hilary Term, you will work alone on two projects. You will need to submit two projects from those in this manual.

While MATLAB complements the traditional part of the degree course, we hope the projects help you revise or understand topics which are related in some way to past or future lectures. It is hoped that at the end of this MATLAB course you will feel sufficiently confident to be able to use MATLAB (and/or other computer tools) throughout the rest of your undergraduate career.

1.2 Schedule

Deadlines

- 12 noon, Monday, week 6 (February 24) Submission of first project.
- 12 noon, Monday, week 9 (March 17) Submission of second project.

Computer and demonstrator access

This term, the practical sessions with demonstrators in Weeks 1 and 2 will be the same as those for weeks 7 and 8, respectively, of Michaelmas Term. From Week 3 onwards, there are no fixed hours for each college, and you may use the timetabled classrooms at the Mathematical Institute whenever suits you within the following times:

- Weeks 3, 4, and 7: Mon–Fri 3pm–4pm.
- Weeks 5 and 8: Mon, Tues, Weds 3pm–4pm; Thurs, Fri 3pm–5pm.

- Week 6: Mon 10am–noon; Tues-Fri 3pm–4pm.

It's probably a good idea to check the course website (https://www.maths.ox.ac.uk/courses/course/23071) for any possible updates to these times.

Note you will need to bring your laptop to the drop-in sessions. If you need to borrow a laptop, you will need to inform Nia Roderick (roderick@maths.ox.ac.uk) of your chosen drop-in session times in advance.

A demonstrator will be present during the above times. Demonstrators will help resolve general problems that you encounter in trying to carry out the instructions of this booklet, but will not assist in the actual project exercises.

1.3 Completing the projects

To carry out a project successfully, you need to master two ingredients—the actual mathematics of the topic under investigation, and the construction of the MATLAB commands needed to solve the relevant problems. Picking up the mathematics is probably a familiar activity that you practice when you attend a lecture or read your notes. Building up a repertoire of MATLAB commands and algorithmic ideas requires a perhaps different skill that in some ways is more akin to learning a language. There is a tendency to do things in an inefficient way to begin with, but eventually one achieves fluency in most practical situations.

Before you get started on a project, it is a good idea to glance through the exercises to try to appreciate what is being asked. To answer most of the exercises you will have to find the relevant commands that enable MATLAB to do what you want. There are clues and guidance given for this within each project, although it will often be necessary (or at least helpful) to consult the MATLAB help system.

Each project is divided into several exercises, and earns a total of 20 marks. The projects must be completed to your satisfaction and submitted electronically before the respective deadlines in weeks 5 and 8, according to the instructions given below. The marks will count towards Prelims and will not be released until after the Preliminary Examinations.

Your answers will ideally display both your proficiency in MATLAB and appreciation of some of the underlying mathematics.

Each project has some marks set aside for "MATLAB code which is elegant and concise". The lecturer will (try to) give examples of such during the MATLAB lectures this term. As always, the presentation of your work also counts towards your grade.

1.3.1 Getting help

The description of each project gives references to books covering the relevant mathematics; if you cannot understand some of this then you are free to consult your tutors or others about it, but not about the project itself. You may discuss with the demonstrators and others the techniques described in the Michaelmas Term Students' Guide, the commands listed in the Hilary Term Students' Guide, and found in the MATLAB help pages. You may ask the Head Demonstrator or the Course Director to clarify any obscurity in the projects, although at least the latter has a reputation for "terse" replies.[1]

The projects must be your own unaided work. You will be asked to make a declaration to that effect when you submit them.

[1] e.g, "Sometimes in life you have to figure something out for yourself; this is one of those times."

1.3.2 Debugging and correcting errors

"Debugging" means eliminating errors in the lines of code constituting a program. When you first devise a program for an exercise, do not be too disheartened if it does not work when you first try to run it. In that case, before attempting anything else, type `clear` at the command line and run it again. This has the effect of resetting all the variables, and may be successful at clearing the problem.

If the program still fails, locate the line where the problem originates. Remove semicolons from commands if necessary, so that intermediate calculations are printed out and you can spot the first line where things fail. You may also want to display additional output; the `disp()` command can be useful. If the program runs but gives the wrong answer, try running it for very simple cases, and find those for which it gives the wrong answer. Remove all code that is not used in that particular calculation, by inserting comments so that MATLAB ignores everything that follows on that line.

Website

A copy of this manual can be found at:

https://www.maths.ox.ac.uk/courses/course/23071

This site will also incorporate up-to-date information on the course, such as corrections of any errors, possible hints on the exercises, and instructions for the submission of projects.

Legal stuff

Both the University of Oxford and the Mathematical Institute have rules governing the use of computers, and these should be consulted at http://www.maths.ox.ac.uk/notices/it/rules.

Chapter 2

Preparing your project

To start, say, Project 1, find the template "proj1template.m" on the course website https://www.maths.ox.ac.uk/courses/course/23071 . *Important:* save this file as `project1.m`, in a folder/directory also called `project1`. Do not use other names.

You will be submitting this entire folder so please make sure it contains only files relevant to your project. You will almost certainly end up creating several `.m` files within this folder as part of your project.

2.1 Matlab publish

The file `project1.m` should produce your complete answer. We will use the MATLAB "publish" system.

```
>> publish project1
```

This will create an HTML report in `project1/html/proj1.html` which can be viewed with a webbrowser (Firefox/Chrome/Safari/etc). The lecturer will give examples of "publish" in your MATLAB lectures and post an example file on the course website. You should also read "`help publish`" and "`doc publish`".

The examiners will read this published report in assessing your project. It is important that the report be well-presented.

- Divide `project1.m` into headings for each exercise (perhaps more than one heading for each exercise).
- You can and should call other functions and scripts from within `project1.m`
- Make sure you answer all the questions asked using text in comment blocks—if it asks why explain why!
- Some questions ask you to create a function in an external file. A good way to make this code appear in your published results is to you "`type foo.m`" where appropriate in your `project1.m`.
- Include appropriate MATLAB output: don't include pages-and-pages of output, but you must show that you have answered the exercise. This will require some thought and good judgment but its worth the effort to avoid losing points if the examiners cannot determine your answer.

The examiners may also run your various codes and test your functions.

Make sure you run publish one last time before submitting your project. Then double-check the results.

2.2 Zip up your files

Make a `project1.zip` or `project1.tar.gz` file of your project1 folder or directory including all files and subfolders or subdirectories. No `.rar` files please. It is highly recommended you make sure you know how to do this well before the deadline.

Double-check that you have all files for your project and only those files for your project.

2.3 Submitting the projects

Submission deadlines are given above. The projects are to be submitted electronically (from anywhere with internet access). These deadlines are *strict*. It is *vital* that you meet them because the submission system will not allow submissions after the above times. You should therefore give yourself plenty of time to submit your projects, preferably at least a day or two in advance of the deadline.

You will need your University Single Sign On username and password in order to submit each project, and also your examination candidate number (available from OSS). If you have forgotten your details you must contact OUCS well before the first deadline. The system will only allow one submission per project.

Chapter 3

Planar Graph Game (Project 1)

3.1 Introduction

Let's make a game in MATLAB.

Exercise 1 We need random integers in this project. Let's do some tests on the `randi()` function. Generate 10000 random integers between 1 and 10. Plot a histogram of them. Do these dice seem fair?[1]) □

3.2 Graphs

That was not the game, be patient. Let's make a graph. A *graph* is a list of vertices (or nodes) and edges connecting them. In our case, the edges will be undirected. Graphs have lots of interesting mathematical properties. They are also very useful in applications of mathematics (think social networks with people as nodes, or the graph of all webpages with links as edges).

We need some random nodes/vertices for our graph. We will have n of them (or maybe N, depending on how consistent I can be).

```
>> n = 10;
>> V = rand(n,2);
```

We have also given them some coordinates in the plane. Here the first column of V should be taken as the x-coordinates and the second column as the y-coordinates. The coordinates are not very important for the mathematics of a graph, but they are pretty important for drawing it.

What is important are the edges between. We define an adjacency matrix E as an $n \times n$ matrix. $E(i, j) = 1$ if node i is connected to node j. $E(i, j) = 0$ if node i is not connected to node j.

Exercise 2 Write a function that builds a random $n \times n$ adjacency matrix with m edges. Your matrix should *carefully* follow this specification:

- must be called `make_rand_edges.m`.
- E must be symmetric.
- E must have *exactly* $2m$ nonzero entries (note the 2, why?).
- E must have a zero diagonal (no local loops).

[1] Insert NSA/GCHQ joke here—just a joke as it would take much more careful testing than this to find problems in a pseudorandom number generator.

- each node should have at least one edge (we don't want disconnected floating nodes, although disconnected components are ok).
- should raise an error if m is less than n.
- E must be random (we don't want node 1 always connected to node 2 for example, even if that would make some of the previous specs easier).

That is a lot of specs; each spec is an opportunity for bugs; test your code carefully! □

If this question is too hard, just hardcode to one of these for now and come back to it:

```
>> e = zeros(n,1); e(2) = 1; E = toeplitz(e)
>> E = ones(n,n) - diag(ones(n,1))
```

Exercise 3 Write a function `plot_graph.m` which draws a graph. The nodes should be red and the edges should be blue. Make the 'markersize' at least 12 and the 'linewidth' at least 2. You could search the internet a bit for help with this last part. □

3.3 Intersections and planar graphs

We want to be able to check if two line segments intersect. This is a common problem in computational geometry. Like many problems in computational geometry, its easy to write implementations which seem to work but fail in some cases.[2]

Exercise 4 Download the following MATLAB code
http://www.mathworks.com/matlabcentral/fileexchange/27205-fast-line-segment-intersection
Write a code which uses `lineSegmentIntersect(AB, CD)` to find the intersection of line segments AB and CD. Demonstrate that works on 2 or 3 test cases. Specifically consider the case where the segments share one end point. □

Exercise 5 Write a function `is_planar_embedding(V, E)` which returns true if the graph is drawn without intersecting edges. If at least one edge intersects another, return false. We call this a planar embedding. More correctly, it is a "planar straight-line graph". □

Exercise 6 In the above code, you have to be careful: two edges that meet at a common vertex should not count as intersecting! Describe in words how you deal with this. □

Exercise 7 Modify the above code as follows. If a third argument of `true` is passed, as in `is_planar_embedding(V, E, true)`, it should also draw purple triangles at any points of intersection. It should continue to work if only two arguments are passed (Hint: look up "`nargin`"). □

If there exists a planar embedding then we say a graph is planar. This is a topological property of the graph: it does not depend on any particular realization which may have crossing edges. Any planar graph has a planar straight-line graph (Fáry's theorem, 1948, https://en.wikipedia.org/wiki/F%C3%A1ry%27s_theorem)

[2] For example, as of writing, Tom März has still not fixed this bug https://github.com/cbm755/cp_matrices/issues/2 in our surface computing software package! Perhaps StCatz students can give him a hard time about that...

Writing an `is_planar()` function is more challenging.[3] Instead we'll make a human being work on it—the Dear Player, probably you—and we'll call it a "game" which automatically makes it fun![4]

It is a theorem that a planar graph with $n \geq 3$ vertices and number of edges e satisfies $e \leq 3n - 6$. So if we violate this bound, we can't possibly have a planar graph. Its worth keeping this in mind as we write the game, to avoid torn hair or sleepless nights on the part of Dear Player.

Exercise 8 Write a function called `closest_node(V, x, y)`. Here `x` and `y` are scalars specifying a coordinate (x, y) and `V` is the $N \times 2$ list of node/vertex coordinates. It should return the index of the node closest to (x, y). □

3.4 A graphical user interface

Here's some boilerplate code for a very basic MATLAB GUI. It queries for the player to click on points (x, y) repeatedly until the player presses 'q'.

```
>> while(1)
>>   [x,y,button] = ginput(1);
>>   if button == 'q'
>>     title('quiting');
>>     break
>>   end
>>   % do something with x and y coordinates...
>> end
```

(there are much fancier ways to make MATLAB GUIs: this will be enough for our purposes.)

Exercise 9 Write a script called `planar_game.m`. Your game should alternate between two modes.

Mode 1: the game starts in mode 1. Wait for the user to select a node (use your `closest_node` function). Selecting a node should draw a red star on it to show that it is highlighted. Now change to mode 2.

Mode 2: wait for the user to click to specify a new position for the currently selected node. Redraw the graph. Check if solved (label intersections in purple). If so, user wins, otherwise, back to mode 1.

Use the `title()` of the figure window to keep the user informed of what mode they are in, if they have won yet, etc. □

Warning: this game can be really hard, particularly if $e >> n$. Many graphs are *not* planar. This can lead to frustration.

If you want (i.e., optional, not part of the assessed project), download the "MatlabBGL"[5] software which, among other things, has a `boyer_myrvold_planarity_test()` function. It

[3] Its an easy calculation for computer to perform: algorithms exist which have linear time in the number of vertices: the challenge is in understanding those algorithms. Another year, another project.

[4] This project is inspired by the game "gPlanarity" by Christopher "Monty" Montgomery (creator of Ogg Vorbis, a popular open audio codec and patent-free alternative to MP3). Monty's version is itself based on a flash game by John Tantalo.

[5] http://www.mathworks.com/matlabcentral/fileexchange/10922-matlabbgl

also has `chrobak_payne_straight_line_drawing()` which should also take all the fun out of game.

Also optional: add drag-n-drop touch screen editing. Recode in Java. Put on Android's Play Store and Apple's App Store. Add annoying banner ads. Profit! Give your intructor a geneous cut.

Chapter 4

The South Crystal (Project 2)

Do you like the South Crystal Skylight in the new Andrew Wiles Building? If so, that was me. If not, it was all the work of graduate students Ingrid von Glehn and Yujia Chen, who are both excellent at everything else they do.

The department wants me to prepare documentation explaining the design. I need a project for 200 bright and highly literate undergraduates. Two birds with one stone.[1]

4.1 Triangles

The skylight is made of triangles. We'll need to deal with triangles in MATLAB.

Download `annies_pig.ply` and `stanford_bunny.ply` from the course website. We want to import data from these files input MATLAB. Let's take a look at one of these files:

```
ply
format ascii 1.0
comment Annie Hui's pig from AIM@SHAPE.net
element vertex 606
property float x
property float y
property float z
element face 1208
property list uchar int vertex_indices
end_header
0.1813917544 0.1701947 0.2795133644
0.1636665322 0.176398 0.2634560022
0.2127072944 0.3249817022 0.2477207666
  ...
-0.02928901 0.6892266622 -0.8351676518
-0.03595689 0.6897660022 -0.8323835538
3 389 388 394
3 132 127 128
3 132 129 130
  ...
```

The three columns of floating point numbers are the x, y and z coordinates of the vertices of the triangles. The lines that look like "`3 <integer> <integer> <integer>`" describe triangles: the integers are indices into the list of triangles. We need to load this data into MATLAB. There are many ways to do this, one simple way is to download `simple_ply_loader.m`

[1] No birds and definitely no stones near the skylight please!

from the course website.[2]

Exercise 1 After loading the data from the ply files above, use the `trisurf` command to plot the Stanford Bunny and the `trimesh` command to plot Annie Hui's Pig. □

4.1.1 A hole in the floor

The file `crystal_bdy.txt` on the website contains the x and y coordinates of the boundary of the hole. This shape was specified by the architects.

Our task is to define the triangular skylight above it.[3]

Exercise 2 Write a code which draws the polygonal boundary of the hole. Try to use a thick black line (the "linewidth" property). Now create a `meshgrid()` large enough to enclose the polygon. Define a function over the meshgrid which is 1 inside and 0 outside. Hint: look at the `inpolygon` function. Use `pcolor` to plot the meshgrid and display it on the same figure as the black outline. □

Now we need to design the triangulation above the hole.

4.2 Eigenvalue problems

You have probably encountered eigenvalues λ and eigenvectors x in linear algebra as solutions to the problem

$$Ax = \lambda x, \tag{1}$$

where A is a matrix. Here we investigate this concept for differential operators and in particular the *Laplacian* operator Δ. At least in terms of symbols, it appears straightforward to pose a version of the problem (1): we want to find nonzero eigen*functions* u and associated eigenvalues λ such that

$$-\Delta u = \lambda u. \tag{2}$$

In one dimension, the Laplacian operator is simply the second derivative operator, so (2) is $-u_{xx} = \lambda u$ or "find a nonzero function whose (negative) second derivative is a multiple of itself." And we know this: $u(x) = \cos 3x$ for example with $\lambda = 9$.[4]

Just as in all PDE problems, in addition to the equation (2), we need to know the domain and the boundary conditions. We assume the problem is posed on a bounded domain Ω with Dirichlet boundary conditions $u = 0$ on $\partial\Omega$. This is known as the *Laplace–Dirichlet* eigenvalue problem.

4.2.1 Applications

The eigenfunctions are also known as *eigenmodes* and are of fundamental importance to understanding many mathematical operators. They are also one of the principle mathematical tools of physics and have many practical engineering applications. The set of eigenvalues, the *spectrum* is equally important. Applications include mechanical vibration, electromagnetic cavities, photonic crystals, and population dynamics.

[2] Ply files can be more complicated than those given here. If you want to play with other triangulations—for example from the AIM@Shape website—you should probably get a better tool. I use readPly by Pascal Getreuer. Also useful are "ParaView" and "MeshLab" which are standard alone visualization and meshing tools.

[3] Tom März wanted a giant Stanford Bunny: we suspect he's a saboteur from Palo Alto.

[4] Opinions vary about where to put the inevitable minus sign! In this project, we will consider $-\Delta u$ as in (2) which makes a positive definite operator—all nonnegative eigenvalues.

4.3 Squares and rectangles

Exercise 3 In your calculus courses you will (have) encounter(ed) *separation of variables* as a technique for solving the heat equation $u_t = \Delta u$. By writing $u(x, y) = F(x)G(y)$, use this technique to solve the Laplace–Dirichlet eigenvalue problem on a square with side lengths π. The bottom-left corner of the square is at the origin. You can do this by hand or with MATLAB as you wish. Write a function called `eiglap_square` (in an m-file called `eiglap_square.m`) which takes 4 input arguments: two integers n and m and two symbolic variables x and y. For example, after

```
>> syms x y
>> [ef,lambda] = eiglap_square(4,5,x,y)
```

`ef` should contain a symbolic expression for the eigenfunction and `lambda` should contain a number, the eigenvalue (I suggest in `double` format not symbolic). The two integers n and m parameterize the eigenvalues and eigenfunctions (this should be clearer after you've worked through the separation of variables). Carefully determine what the admissible values are for n and m and make sure your function calls the MATLAB `error` function if appropriate. □

4.4 General domains

For more complicated domains, separation of variables is tricky or impossible. We turn to numerical methods to look for an approximate numerical solution. Download `ellipse_ev.m` from the website and study the code. The ellipse code uses "finite differences" and the treatment of the curved boundary leads to a low-order of accuracy in the solution; this could be improved by various techniques.[5]

Exercise 4 Copy `ellipse_ev.m` to `crystal_ev.m`. Modify `crystal_ev` to use the boundary of skylight instead. You may need to make other adjustments to the code. Make some plots of Laplace–Dirichlet eigenmodes of the skylight domain. Make a 3D `surf()` plot of at least one of the eigenmodes.

Your function should work as follows

```
>> [ews, evs] = crystal_ev(XV, YV, true)
```

where `XV` and `YV` specify the vertices of the polygon. □

Exercise 5 The crystal is not supposed to hang down into the mezzanine. Plot the absolute value of several eigenmodes. □

The architects choose the second eigenmode (well, its absolute value). We'll do the same.

[5] So many potential projects, so little time. The Numerical Analysis Part A option touches on many parts of this. There are also Part B options which deal specifically with the numerical solution of differential equations. For our crystal design, we used a more accurate second-order scheme rather than the first-order accurate one here.

4.4.1 Bring on the triangles

Download `crystal_flat.ply`. This is a flat 2D triangulation generated from the skylight boundary using the software Triangle (`www.cs.cmu.edu/~quake/triangle.html`).

Exercise 6 We need to assign the z values of each triangle vertex. Use the `interp2` function to "sample" the eigenfunction at the appropriate xy points.

Do this inside a function called `compute_vertices.m` which should take as inputs the eigenfunction, the meshgrid coordinates and the triangle vertex xy coordinates. It should return the vertex xyz coordinates (in the same order given in the ply file). □

Exercise 7 Then use trisurf/trimesh to display your triangulation. Finally, draw the absolute value of the eigenmode and the triangulation (using thick lines) on the same plot. You might need to play with transparency (e.g., `alpha()`). □

In case you want to make your own design, there were various restrictions on the triangles (imposed by the architects). They had to be not too big, not too small, and fairly close the equilateral (precise specs were given).

Chapter 5

Groups, Rings and Fields with Sage (Project 3)

5.1 Introduction

This project is different from the others in that it does not use MATLAB. Instead, you'll gain some experience with Sage.

Sage is a Free and Open Source Software (FOSS) package for doing mathematics with a strong focus on pure mathematics such as number theory. It bundles together the wide range of domain-specific mathematical tools and glues them together using the Python programming language. You can find out more about Sage at http://www.sagemath.org.

For this project we will use Sage through the exciting new SageMath Cloud:

http://cloud.sagemath.org

You should be able to use this from any computer with a recent web browser. If you prefer and you use a GNU/Linux or Mac system, you can easily download Sage onto your own machine.

Warning: Sage is *not* MATLAB: you'll need to learn some aspects of the Python programming language to complete this project. The course demonstrators may or may not be able to help. To compensate, the exercises here should be a bit easier.

5.1.1 Getting started

1. Go to http://cloud.sagemath.org
2. Create an account.
3. Create a new *private* project.
4. Add a new "Sage Worksheet" to the project. Name this "project3_12345678", replacing 12345678 with your candidate number. (You can rename it later before submission if you prefer).
5. Put your answers into the worksheet.
6. See Section 5.4 for notes on submitting your Sage project.

5.2 Modular arithmetic

With the administration out of the way, let's get started.

Type "`1 + 1`" into the worksheet in Sage. Press shift-enter. Verify you get 2. Now type `x = 1 + 1` and press shift-enter.

```
>> x = 1 + 1; x
```

Exercise 1 Clearly label this exercise, for example by adding

```
>> # Exercise 1
```

to a line in your worksheet. Now assign $2^{(3^4)}$ to a variable `y`. Compute $\frac{y}{4^{(3^2)}}$. □

Exercise 2 You can use mod() to compute modular arithmetic. What is $3^{(4^5)} \mod 137$? □

Exercise 3 In MATLAB, what would be the result of: "`x = mod(12,7), y = x + 8`"? Include your answer as a comment in your Sage worksheet, for example:

```
>> # Matlab gives 42
```

Now repeat this calculation in Sage. Can you explain carefully what happened? Hints: entering `x?` will access help about `x`. What do you think "`ZZ/nZZ`" means? □

This gives some suggestion about a really powerful aspect of Sage: it keeps track of what sort of mathematical objects we're dealing with.

5.2.1 Python programming

We need some flow control and programming basics in Python. These are somewhat similar to MATLAB and at any rate "Google is Your Friend" if you get stuck: Sage (and Python) have lots of online documentation.

Probably the most important distinction is that whitespace (intending) is important in Python whereas it is mostly cosmetic in MATLAB. Here is an example of defining a function (procedure) in Python

```
>> def square(x):
>>     y = x**2
>>     return y
>>
>> square(7)
```

And here is an example of a "for loop" and "if-then-else" flow control in Python. Careful with the whitespace.

```
>> print " num  sqrn"
>> print "-----------"
>> for i in range(0, 10):
>>     j = square(i)
>>     if j < 25:
>>         msg = ''
>>     else:
>>         msg = 'so big!'
>>     print "  %d    %d   %s" % (i,j,msg)
```

```
 num sqr
----------
  0   0
  1   1
  2   4
  3   9
  4   16
  5   25   so big!
  6   36   so big!
  7   49   so big!
  8   64   so big!
  9   81   so big!
```

Note that 10 is not included in the range.

5.2.2 Rings

Briefly, a mathematical ring is algebraic structure consisting of a set (for example, the integers, or a subset of them) and two operations, typically called addition and multiplication (typically, but not always, for good reason).[1]

Let's be concrete: we'll be (mostly) discussing the *Ring of Integers Modulo N*. For example, if $N = 7$, this is the set $\{0, 1, 2, 3, 4, 5, 6\}$ *with the usual addition and multiplication* but taken modulo N. Consider taking an element x of that set. Does it have a *multiplicative inverse*? That is, does there exist some y (also in the set $\{0, 1, 2, 3, 4, 5, 6\}$) such that $xy = 1 \mod 7$?

Exercise 4 Write a function `find_mult_inv(x,N)` which uses `mod()` and a "for" loop to test if a number x has a multiplicative inverse mod N. It should print a message like "3 has multiplicative inverse 5 (mod 7)". Or "3 has no multiplicative inverse (mod 6)". Your function should return the multiplicative inverse or `[]` if one does not exist. (There are fancier ways to do this in Sage: we'll get there, for now follow the instructions). □

Exercise 5 Write a similar function called `has_mult_inv(x,N)` which should return a boolean True/False. It should not print anything. □

So far, we've mostly been using fairly standard Python (albeit from within Sage). Let's get a little more into the spirit of Sage. Consider:

```
>> Z7 = IntegerModRing(7)
>> a = Z7(3)
>> a
>> a^-1
```

Exercise 6 Write a short block of code that starts with

```
>> for a in Z7:
>>     ...
```

and displays elements and their multiplicative inverses in pairs. You should only need to add one or two lines of code. □

Other interesting things are easy to do in Sage, we give one example:

[1] Much more detail and precision will follow in a fascinating course in your near future!

```
>> S.<x> = PolynomialRing(IntegerModRing(6), 'x');
>> S
>> x
>> f = x^2 - x
>> f.roots(multiplicities = False)
```

This solves for the roots of a polynomial defined *over the ring*. This is another glimpse of something exciting in Mathematics and Sage: just because we're doing algebra doesn't mean we have to do it over the usual real numbers!

5.2.3 Fields and finite fields

If every non-zero element of the Ring has an multiplicative inverse, we call it a *Field*. The degree of a field is the number of elements in it. A finite field has a finite number of elements in it.

Suppose we conjecture that there can only be finite fields of degree p where p is prime. Let's test this conjecture.

Exercise 7 Write code to test whether *every* non-zero element of a Ring of Integers Mod N has a multiplicative inverse. Run this test for rings mod N for $N \in [2, \ldots, 500]$. Display a message like:

```
Every element of the N=5 ring has mult invs: its a field.
```

(but display nothing in case its not a field). You might make use of your previous functions. □

Finite fields could be build with more general mult/add operations and on sets other than integers. Nevertheless, our conjecture is pretty close: there only exist $GF(p^n)$ where p is prime and n is positive integer. GF is short for Galois Field. In Sage you can also use `FiniteField()` if you prefer.

So we expect this will work:

```
>> F = GF(7);
>> F
```

and this will fail:

```
>> FailField = GF(6);
```

Exercise 8 Try this code:

```
>> F = GF(5^2, 'c');
>> F
>> type(F)
>> for f in F:
>>     print f
```

Play with the 5 and 2: what does they seem to control? Note you're not asked to understand this algebraic structure: time enough for that in another course! □

5.3 Permutations and Permutation Groups

Understanding permutations of elements of a set—for example, $(1, 2, 3) \to (3, 2, 1)$—is useful in many areas of mathematics and some applications. This particularly true in the study of *symmetry*, be it geometrical rotations of a cube or understanding chemistry and mathematical physics.

```
>> p1 = Permutation([(4,5)]); p1
>> p2 = Permutation([(1,2,3)]); p2
```

Alternatively,

```
>> p1 = Permutation('(4,5)'); p1
>> p2 = Permutation('(1,2,3)'); p2
```

Its somewhat a matter of aesthetics but you might prefer p_2 also work on the full set $\{1, 2, 3, 4, 5\}$:

```
>> p2 = Permutation('(1,2,3)(5)'); p2
```

For completeness, you can also do `p2 = Permutation([(1,2,3),(5,)])` (note the slightly strange comma).

It is common to denote permutations only by the changes (that is, `p1` does not need to specify what happens to 1, 2, and 3).

Exercise 9 Create a permutation that swaps 7 with 8, 9 with 10, and "rotates" (1,2,3,4). □

We can combine permutations by multiplying using "`*`": this corresponds to composition of the permutations.

```
>> p2*p1
>> p1*p2
```

Consider

```
>> p1*p1
```

which does nothing—it swaps whatever is in the 3rd position with the 4th position, then swaps them again. As everything is back to the original position, this is the *identity* operation. You can also raise the permutations to powers, which applies the permutation multiple times.

Exercise 10 Try to find all possible permutations that you can *generate* using `p1` and `p2`. □

So if we take all these possible permutations, we have closure under the operation of composition. This set of permutations is an example of a *group*, which is an algebraic structure consisting of set and a group operation. Here the set is the permutations themselves and the group operator is composition. The permutations p_1 and p_2 are *generators* of the group: we can build every element of the group by taking compositions of them.

Sage can automate quite a lot of this.

```
>> G = PermutationGroup([p1,p2]);  G
```

or you can give the generators directly as tuples

```
>> PermutationGroup([(1,2,3), (4,5)]);
```

`G.order()` tells you how many things are in the group G and `G.random_element()` will return a random element from the group.

Exercise 11 Use the `.list()` method of the group to get all elements of the group. How many are there? Is this the same number as you found "by hand"? □

Note the elements of the group are written in *cycle notation* whereas `p2` is the effect of the permutation on the set. This might be a good time look at the Wikipedia article on Permutation.

We can also iterate over the elements in a group

```
>> for g in G:
>>     print g
```

Exercise 12 Use `SymmetricGroup()` to create a group of *all* permutations of a set of 5 objects. Assign it to a variable `S6`. Can you ask Sage whether G above is a "subgroup" of S_6? Hint: type "`G.`" and press the Tab key for a list of methods of `G`: these are things that the object `G` knows how to do. □

You can see a bit more of the algebraic structure of the group by viewings its Cayley Table:

```
>> T = G.cayley_table()
>> T
>> T.column_keys()
```

where `T.column_keys()` describes the meaning of the shorthand of a, b, c, d, e, f for the elements of the group. Note the composition of any elements in the group will result in another element of the group (this is "closure" again). Any element of the group can be made by applying, possibly repeatedly, the generators.

5.3.1 Word problems

A train leaves Oxford at 8am travelling at 80 mph while another leaves Birmingham at 10am... Just kidding! The "*word problem*" here involves searching for a sequence of permutations to reach a particular state.

Exercise 13 Construct a permutation group called `G` in Sage of degree 10 with generators (1 2), (1 3 5 7), (2 4 6 8). Now define

```
>> e1 = G('(1,2)')
>> e2 = G('(5,6)')
```

Call the `word_problem` method of each of these elements to express them in terms of the generators of G. □

5.3.2 Rubik's cube

```
>> rubik = CubeGroup();
>> rubik.display2d("")
```

Note the Rubics cube consists of 48 coloured squares (in this mathematical model, the center squares are not considered as they are fixed in place).

```
>> e = rubik.random_element()
>> rubik.display2d(e)
>> P = rubik.plot3d_cube(e)
```

and in a new cell:

```
>> P.show()
```

Exercise 14 Assuming we play it *my way* (that is, ahem, by dismembering the cube and putting it back together), how many configurations are there? (see below for how to do this) Using legal moves how many configurations are there? What percentage of configurations are legal? □

How to do the above? The "dismembered" part is a bit tricky. Here's one approach based on a group structure obtained from Wikipedia:[2]

$$\mathbb{Z}_4^6 \times \mathbb{Z}_3 \wr \mathrm{S}_8 \times \mathbb{Z}_2 \wr \mathrm{S}_{12}.$$

Enter the following into a single Sage cell:

```
>> %gap
>> s8 := SymmetricGroup(8);
>> s12 := SymmetricGroup(12);
>> z4 := CyclicGroup(4);
>> z3 := CyclicGroup(3);
>> z2 := CyclicGroup(2);
>> D1 := DirectProduct(z4,z4,z4,z4,z4,z4);
>> W1 := WreathProduct(z3,s8);
>> W2 := WreathProduct(z2,s12);
>> w :=DirectProduct(D1,W1, W2);
```

The "`%gap`" tells Sage that this cell should work directly with GAP, the underlying software which Sage uses for manipulating groups.[3] We then need to get the size of that group out of GAP and back into a Sage variable. In a new cell, do:

```
>> dismember_size = Integer(gap.get('Size(w)'))
>> dismember_size
```

Rubik's Word Problem You could use the word problem to get a sequence of moves to solve a Rubik's Cube from a random configuration. Talk about taking the fun out of everything. Note you do not have to do this as part of the project.

[2] http://en.wikipedia.org/w/index.php?title=Rubik%27s_Cube_group&oldid=577669579, and special thanks to Dr Jennifer Balakrishnan for helping me do this in Sage!

[3] This is a neat feature of Sage: it glues together various packages (in this case the group theory system GAP http://www.gap-system.org) which are often state-of-the-art research codes in their area. One can access advanced features (in this case the "wreath product $\wr$ ") by talking directly to the lower-level software. And if a feature turns out to be useful to the Sage community, it often ends up enabled with Sage itself.

5.4 Sage submission notes

To submit this project, save (download) your work from cloud.sagemath.org as a file

`project3_12345678.sagews`

(a "Download file" icon appears if you hover over the file entry under the "Files" tab.) Replace 12345678 with your candidate number.

Also print your worksheet to a PDF file: preferably by using the print icon next to the green "Save" button.

Place both files inside a .zip file as described in the MATLAB-specific instructions and submit normally.

NOTE: simply sharing your cloud.sagemath.org project with your instructor won't work (Oxford grading must be anonymous).

Warning: cloud.sagemath.org is still in beta: do *not* leave this until the last minute.