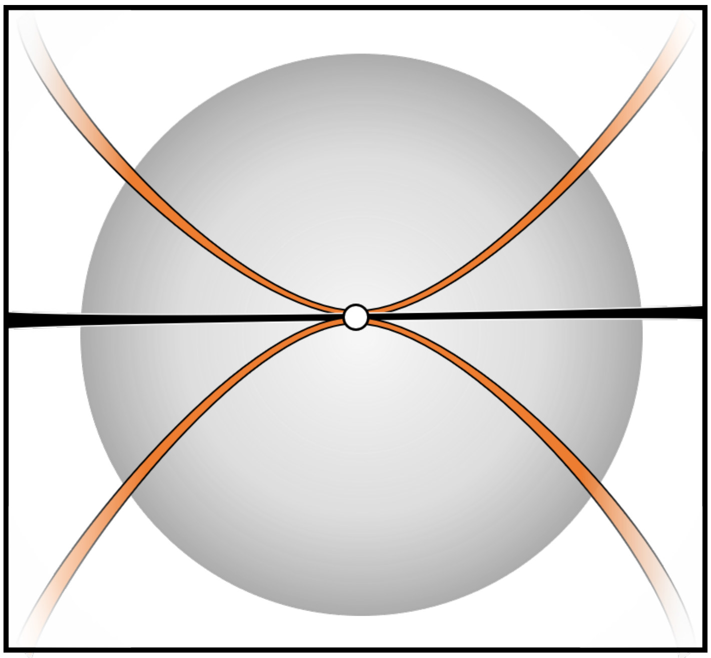
MATHEMATICAL INSTITUTE, UNIVERSITY OF OXFORD

# Computational Mathematics

**Students' Guide, Hilary Term 2020**

**by**

**Prof Vidit Nanda**



**Acknowledgments**: The course guide is based on previous versions by Dr Andrew Thompson, Dr Alberto Paganini and others. I am grateful to all previous authors.

# 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 any two of the three given projects.

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   Submission of first project.
- 12 noon, Monday, week 9   Submission of second project.

You are free to **choose any two of the three projects described here**. And you don't have to do them in order. For instance, if you choose Projects (A) and (C), you are allowed to submit (C) in Week 6 and (A) in Week 9.

### 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 3–8: Mon 3pm–4pm; Thurs 3pm–4pm.

There will be additional sessions ahead of the project deadlines:

- Week 5: Weds 3pm–5pm; Fri 3pm–5pm.
- Week 8: Fri 3pm–5pm.

It's probably a good idea to check the course website (`https://courses.maths.ox.ac.uk/node/43979` ) for any possible updates to these times. Note that you will need to bring your laptop to the drop-in sessions.If you need to borrow a laptop, you will need to inform the Academic Administrator (`acadadmin@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 project instructions, 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 all the exercises and try to fully understand 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 6 and 9, 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". This includes **thoroughly adding comments** so that your code is readable by other human beings. 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. Several of the exercises require you to make plots; please make sure your plots are legible, and all their components are suitably labelled.

### 1.3.1 Getting help

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 those found in the MATLAB help pages. You may also ask the the Course Director to clarify any obscurity in the projects.

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

### 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://courses.maths.ox.ac.uk/node/43979

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 https://www.maths.ox.ac.uk/members/it/it-notices-policies/rules.

# Chapter 2

# Preparing your project

To start, say, Project A, find the template 'projAtemplate.m' on the course website https://courses.maths.ox.ac.uk/node/43979 . Save this file as `projectA.m`, in a folder/directory also called `projectA`. 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 `projectA.m` should produce your complete answer. We will use the MATLAB 'publish' system.

```
publish('projectA.m','pdf')
```

This will create a PDF report in `projectA/html/projectA.pdf`. 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 `projectA.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 `projectA.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 include '`type other_function.m`' where appropriate in your `projectA.m`.
- Include appropriate MATLAB output: don't include pages and pages of output, but you must show that you have answered the question. This will require some thought and good judgment but it's 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 `projectA.zip` or `projectA.tar.gz` file of your projectA 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

Full instructions on the submission system will be emailed to you nearer the time. The projects are to be submitted electronically at `https://courses.maths.ox.ac.uk/node/43979/assignments` (from anywhere with internet access). Submission deadlines are given in Section 1.2. 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. Penalties for late submission are specified in the Examination Conventions

`https://www.maths.ox.ac.uk/members/students/undergraduate-courses/examinations-assessments/examination-conventions`.

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 Student Self-Service). If you have forgotten your details you must contact OUCS well before the first deadline. The system will only allow one submission per project.

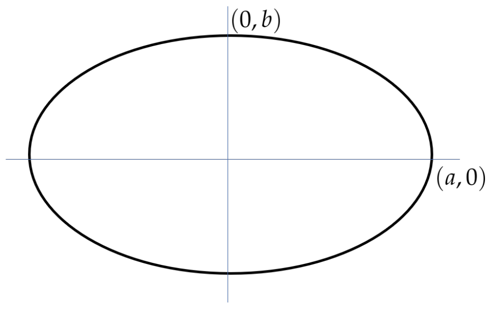
# Chapter 3

# (A) Elliptic Integrals

This project is designed to show you a simple but extremely important class of integrals which can not be explicitly solved, and hence require numerical approximation. If this topic is appealing to you, I suggest also searching the internet for *complete elliptic integrals of the second kind*. I know it's a mouthful, but there's a lot of interesting geometry and analysis related to these integrals.

## Introduction

An ellipse, you might remember from ye olde geometry class, is a closed curve which (once you align and center it around the origin in the plane) looks like this:



The longest radius (called $a$ here) is the *major axis* and the shortest radius (called $b$ in the picture) is the *minor axis*. Even if it has been several years since you last saw an ellipse, you may recall that its equation in the $(x, y)$ plane is:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1. \tag{3.1}$$

A little bit of calculus will tell you that the area is $\pi ab$; when $a = b$ the ellipse becomes a circle and this area formula recovers the much better-known $\pi a^2$. But what of the circumference $C$ as a function of $a$ and $b$? It is a matter of some surprise that *there is no simple formula* for $C$ in terms of $a$ and $b$. What goes wrong?

Well, here's what happens if you try to use calculus for computing $P(a, b)$. First you need to try and parametrize the ellipse. No problem, you can tell from (3.1) that a perfectly good parametrization is

$$x(t) = a\cos(t) \text{ and } y(t) = b\sin(t) \tag{3.2}$$

for $t$ in the interval $[0, 2\pi)$. So the length element becomes

$$dC(t) = \sqrt{(dx)^2 + (dy)^2} = \sqrt{a^2 \sin^2(t) + b^2 \cos^2(t)}\, dt,$$

and all we have to do is solve

$$C(a,b) = \int_0^{2\pi} \sqrt{a^2 \sin^2(t) + b^2 \cos^2(t)}\, dt.$$

When $a = b$, there is no problem because the stuff under the square root is just $a$. But in general, when $a \neq b$ there is no magic trick that will simplify our life and offer a clean solution. Thus, in order to satisfy our curiosity about elliptic integrals, we will turn to numerics.

## 3.1 Exercise A1

Write `function[x,y] = ellipse(a,b,n)` which generates $n$ points of the form $(x_i, y_i)$ on an ellipse with major axis $a$ and minor axis $b$ using the parametrization from (3.2). You should first chop up the interval $[0, 2\pi]$ into $n$ equally spaced subintervals $[t_i, t_{i+1}]$, and then store $x_i = x(t_i)$ and $y_i = y(t_i)$ for the endpoints of those pieces[1].

Now **plot the output points** for $(a, b, n) = (1, 0.75, 800)$ to confirm that they lie on an ellipse with the correct semi-major and semi-minor axis.

## 3.2 Exercise A2

Write `function [len] = arclength(x,y)` which takes as input the points which were output by `ellipse` and computes the **approximate circumference** as a sum

$$C_{\text{app}}(a,b) = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}.$$

**Check** that $C_{\text{app}}(a,a)$ is close to the expected circular circumference for values of $a$ in $\{1, 0.8, \ldots, 0.2\}$ by comparing the difference between your $C_{\text{app}}$ (using $n = 100$ points in each case) and the true circumference $2\pi a$. In general does $C_{\text{app}}$ tend to be an over-estimate, an under-estimate, or neither? Can you explain why?

## 3.3 Exercise A3

The mathematician S Ramanujan developed an approximate formula for the circumference of an ellipse:

$$C_{\text{ram}}(a,b) = \pi \left[ 3(a+b) - \sqrt{(3a+b)(a+3b)} \right].$$

Compare your $C_{\text{app}}$ to his $C_{\text{ram}}$ for a family of ellipses that starts from the unit circle ($a = b = 1$) and shrinks to the interval ($a = 1, b = 0$) by decreasing the minor axis in 100 evenly-spaced steps $1 = b_1, b_2, \ldots, b_{100} = 0$ while keeping the major axis $a$ fixed at 1.

**Plot**, on the same graph, $C_{\text{app}}(1, b_i)$ and $C_{\text{ram}}(1, b_i)$ as a function of $b_i$ for $i$ between 1 and 100. What do you notice about the general shape and slope of the two graphs? How well do they resemble each other? When are they the most different?

[1]Hint: `help linspace`

# Chapter 4

# (B) Image Compression

This project explores some magical properties of the *singular value decomposition*, affectionately called SVD. Secretly, this project is also about ellipses and I hope that you will spend a few seconds on Wikipedia to independently discover why.

## Introduction

Start with any $n \times k$ matrix $A$ (with real entries) whose transpose I will denote by $A^t$. It is not difficult to see that the $n \times n$ matrix $B_1 = AA^t$ and the $k \times k$ matrix $B_2 = A^tA$ are both symmetric, i.e., they equal their own transposes. What is somewhat more surprising is that all the **eigenvalues** of $B_1$ and $B_2$ are non-negative real numbers — this is a consequence of a result called *the spectral theorem* in linear algebra and beyond.

Even more surprising than that is the fact that the **non-zero eigenvalues of** $B_1$ **and** $B_2$ **are exactly the same**! Index the *square roots* of these eigenvalues in descending order:

$$\sigma_1 > \sigma_2 > \cdots > \sigma_r$$

(where $r \leq \min(n,k)$ is the rank of $A$). This collection of decreasing positive numbers is called the set of **singular values** of $A$. The **singular value decomposition** of $A$ is a factorization

$$A = U \cdot D \cdot V^t,$$

where $D$ is an $n \times k$ diagonal matrix described as follows. The first $r$ entries of its diagonal contain the singular values $\sigma_1, \ldots, \sigma_r$ of $A$, and all other entries are zero. The matrix $U$ is $n \times n$ and its columns contain while $V$ is $k \times k$. Their columns are given by eigenvectors corresponding to the common eigenvalues $\sigma_i^2$ of $B_1$ and $B_2$ respectively.

The point of all this is the following. For each matrix $M$, let's write $M_{[a,b]\times[c,d]}$ to indicate the sub-matrix of shape $(b-a+1)\times(d-c+1)$ which is obtained by restricting the rows to $a, a+1, \ldots, b-1, b$ and columns $c, c+1, \ldots, d-1, d$. Then, for any $s \leq r = \text{rank}(A)$, **the best rank-$s$ approximation to** $A$ can be extracted from the SVD by

$$A_{s-\text{app}} = U_{[1,n]\times[1,s]} \cdot D_{[1,s]\times[1,s]} \cdot V^t_{[1,n]\times[1,s]} \tag{4.1}$$

We will use this fact to compress digital images!

## 4.1 Exercise B1

From the course webpage `https://courses.maths.ox.ac.uk/node/43979` , download the image file `arches.bmp`. Use the MATLAB's in-built function `imread` to read in this file (the output will be a $1440 \times 2560 \times 3$ matrix, which I will call $M$). Here the last $\times 3$ indicates that we have three stacked matrices of pixel intensities, containing red, blue and green for each pixel of this image in the $1440 \times 2560$ grid. In case you have never seen these before, each pixel is represented by a triple $(R, G, B)$ of three integers between 0 and 255.

The first task is to generate an equivalent greyscale image by extracting a single matrix $B$ of size $1440 \times 2560$ whose $(i, j)$-th entry is computed as follows:

$$B_{i,j} = \left\lfloor \frac{M_{i,j,1} + M_{i,j,2} + M_{i,j,3}}{3} \right\rfloor$$

Here the $\lfloor \quad \rfloor$ indicates that you should round down to the nearest integer. Using the function `imwrite`, create the black-and-white file `arches-bw.bmp`. Use `imshow` to **plot** this greyscale image.

## 4.2 Exercise B2

Next, write `function C = BestApprox(B,s)` which takes in your grayscale matrix $B$ (produced by the previous exercise) along with an integer $s$. The output $C$ is a new matrix that consists of the best rank $s$-approximation of $B$ as produced by (4.1). Please make sure to check within your function that $s$ is not too small (i.e., negative) or too large (exceeding $\min(n, k)$ where $n$ and $k$ count the number of rows and columns of $B$.

Using `imshow` again, **plot** the image you obtain using the best rank-50 approximation to $B$. You may use MATLAB's in-built `svd` function.

## 4.3 Exercise B3

For each $s$ in $\{10, 20, \ldots, 490, 500\}$, compute the worst-case difference of absolute values between the grayscale matrix $B$ and the best rank-$s$ approximation $C^s$ computed using (4.1):

$$d(s) = \max_{i,j} \left\{ |B_{i,j} - C^s_{i,j}| \right\},$$

for $1 \leq i \leq 1440$ and $1 \leq j \leq 2560$. (Hint: don't call `BestApprox(B,s)` individually for each $s$, because computing the SVD so many times will be expensive and lead to slow code. Instead, try to compute the SVD once and for all, and then extract all the different rank $s$-approximations from that single pre-computed SVD).

**Plot** a graph showing $d(s)$ against $s$ for all $s$ in $\{1, 10, 20, \ldots, 990, 1000\}$. How quickly does the approximation improve as the rank is increased?

# Chapter 5

# (C) Spectra of Random Matrices

The *spectrum of a matrix* is just a scary term that means its *set of eigenvalues*. This third project is about examining a surprising bit of structure in the eigenvalues of symmetric matrices with random entries. You may be pleased (or exasperated) to hear that this project is also about ellipses — I'm only half-joking. There is no introduction for this project because I don't want to completely ruin the surprise.

## 5.1 Exercise C1

Write `function e = RandSpec(n)` which does the following. It takes in a positive integer $n$, and generates an $n \times n$ matrix $A$ with independent standard Gaussian entries (the MATLAB function `randn` will be helpful). Then it returns the vector $e$ containing all $n$ eigenvalues of the *symmetric random matrix* $S = A + A^t$. As usual the second term means the transpose of $A$. You are allowed to use the `eig` function.

**Plot** a histogram (with 100 bins) of the output generated by calling `RandSpec(800)` once. If you call the function many times, you will see many different plots (after all, the process is random). But do you see any patterns in the histogram that are common across all these plots? If so, what are they?

## 5.2 Exercise C2

Now let's see how the *maximum* eigenvalue of $S$ changes with the size $n$. Write `function e = MaxEval(n)` which takes in a number $n$, calls `RandSpec(i)` for every $i$ in $\{1, \ldots, n\}$, and returns a vector $e$ so that the largest eigenvalue of the random $i \times i$ matrix is stored in $e(i)$. Now **plot** the entries of the vector $e$ as a function of $i$ by calling `MaxEval(800)`. If things go according to plan, $e(i)$ should roughly be equal to $2.8\sqrt{i}$. Confirm this by plotting $2.8\sqrt{i}$ on the *same* plot.

## 5.3 Exercise C3

Generate a vector $e$ by calling `RandSpec(1000)` and **plot** the histogram with 100 bins as in Exercise C1. Let $b$ be the height of the tallest bin you see, and let $a = 2.8\sqrt{1000}$ be the predicted maximum eigenvalue. *On the same histogram*, plot points from the ellipse $x^2/a^2 + y^2/b^2 = 1$. What have you discovered about the eigenvalues of random matrices? If this last question is crushing your soul, see *Wigner's semicircle law* on the internet.