MATHEMATICAL INSTITUTE

UNIVERSITY OF OXFORD

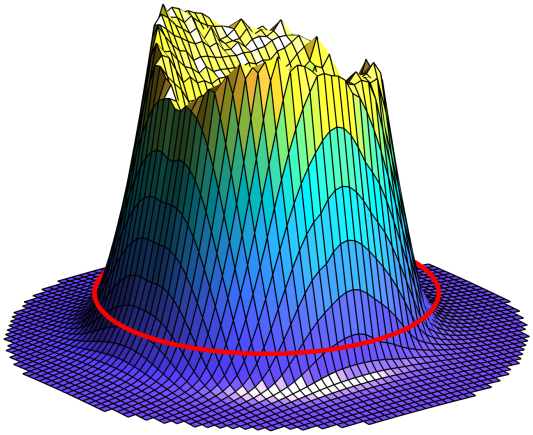
Computational Mathematics

Student Guide

Hilary Term 2022

by

Prof. Nick Trefethen



Acknowledgments: This course guide is based on previous editions by Andrew Thompson, Alberto Paganini, Vidit Nanda, Estelle Massart 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. Computing fundamentally influences 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.

By the way, the correct way to write it is in all caps, "MATLAB". But I've always found this ugly, so in this document we'll go with the incorrect "Matlab".

1.1 Objectives

The objective of this course is to help you learn about doing mathematics using Matlab. Last term you were introduced to some basic techniques, by working through the Michaelmas Term Student Guide, which you are due to complete near the beginning of this term. After this, for the rest of this term, you will work alone on any two of the three projects presented in this booklet.

While Matlab complements the traditional part of the degree course, we hope the projects may help you revise or understand topics which are related to your past or future studies. It is hoped that at the end of this 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

Lectures

There will be two lectures this term, at 11:00 on Wednesdays of weeks 1 and 3 in lecture theatre L1.

Deadlines

- 12 noon, Monday, week 6: Online submission of first project.

- 12 noon, Monday, week 9: Online 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 may submit (C) in Week 6 and (A) in Week 9.

Computer and demonstrator access

This term, the schedule for the practical sessions with demonstrators in Weeks 1 and 2 will be the same as for weeks 7 and 8, respectively, of Michaelmas Term. From Week 3 onwards, there are no fixed hours for each college, but drop-in sessions with demonstrators will be scheduled at times to be announced. (Currently these are tentatively set at Mondays 3–4 and Thursdays 3–4 of Weeks 3–8 and also Wednesday 3–5 of Week 5, Friday 3–5 of Week 5, and Friday 3–5 of Week 8.) See the course website (`https://courses.maths.ox.ac.uk/course/view.php?id=44`) for updates and further information.

Demonstrators will be happy to help resolve general problems, but will not assist in the details of the actual project exercises.

1.3 Completing the projects

To carry out a project successfully, you need to master two ingredients: the mathematics, and the Matlab programming. Picking up the mathematics is a familiar activity that you practice when you attend a lecture or read your notes or mathematics texts. Building up a repertoire of Matlab commands and algorithmic ideas is a different skill that in some ways is more akin to learning a language. It is perfectly normal to do things somewhat inefficiently at first, and to achieve greater fluency as time goes by.

Before you get started on a project, it is a good idea to look over all the exercises to understand what is being asked. To complete most of the exercises, you will have to find the relevant commands that make Matlab do what you want. There are clues and guidance given for this within each project, although it will often be necessary to consult the Matlab help system.

Each project is divided into several exercises, and earns a total of 20 marks, which are **split between mathematical content and clarity of presentation**. The projects must be completed and submitted electronically before the Monday 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 examinations.

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

Try to make your Matlab code elegant and concise — with comments as necessary so that it readable by human beings. Many of the exercises require you to make plots. Please be sure your plots are legible, with all their components well labelled and their fonts not too tiny.

1.3.1 Getting help

You may discuss with the demonstrators and others the techniques described in the Michaelmas Term Student Guide, as well as those found in the Matlab help pages and other sources. You may also ask the Course Director, i.e., me, to clarify any unclear points in the projects.

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

1.3.2 Debugging and correcting errors

Debugging means eliminating errors in a program. When you write a program, do not be 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, a good strategy is to locate the line where the problem originates. Remove semicolons if necessary, so that intermediate calculations are printed out and you can spot the first place where things fail. You may also want to display additional output, for which the `disp` command can be useful. If the program runs but gives the wrong answer, try running it for simpler and simpler cases, until you reduce the problem to a minimal instance for which it gives the wrong answer. Remember that you can always remove code from execution that is not relevant to a particular calculation by inserting the comment character %, so that Matlab ignores everything that follows on that line. The command `return` can also be used to halt execution of a Matlab program at a particular point.

The more advanced way to debug is to use the Matlab debugger, which you may enjoy getting to know. Information can be found in a number of places including `https://www.mathworks.com/help/matlab/matlab_prog/debugging-process-and-features.html`.

Website

This manual can be found at `https://courses.maths.ox.ac.uk/course/view.php?id=44`. 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 note

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`.

Cover image

If you're curious about the image on the cover, take a look at the March 2021 essay at `https://people.maths.ox.ac.uk/trefethen/essays.html`.

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/course/view.php?id=44`. 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 probably end up creating several `.m` files within this folder as part of your project.

2.1 Matlab publish

Execution of 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 the lectures and post an example file on the course website. You should also read `help publish` and `doc publish`.

The assessors will read this published report in assessing your project. It is important that the report be well-presented. You will definitely lose points if your `projectA` just executes the maths without any discussion.

- Divide `projectA.m` into headings for each exercise.
- You can call other functions and scripts from within `projectA.m`. A good way to make this code appear in your published document is to write `type <name of function>` where appropriate in `projectA.m`.
- Make sure that your discussion answers all the questions.
- Include appropriate Matlab output: not pages and pages, but enough to make it clear you have understood and answered the question. This will require some judgment, but it's worth the effort. Clear presentation is a lifelong skill.

The examiners may also run your codes.

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 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

Projects will be submitted online via Inspera; see `https://www.ox.ac.uk/students/academic/exams/open-book/online-assessments`. Further information about the submission process will be emailed to you later on if necessary.

Submission deadlines were given in Section 1.2, and these deadlines are strict. It is vital that you meet them, for the submission system will not allow submissions after these 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 lost track of your details you will need to sort them out with OUCS well before the first deadline.

Chapter 3

(A) Random Fourier series

For some $m \geq 1$, let a_j $(0 \leq j \leq m)$ and b_j $(1 \leq j \leq m)$ be independent samples of the standard normal distribution with variance 1, i.e., numbers of the kind generated by the Matlab `randn` command. Let f_m be the function of $x \in [0, 2\pi]$ defined by the *finite random Fourier series*

$$f_m(x) = a_0 + \sqrt{2} \sum_{j=1}^{m} \Big[a_j \cos(jx) + b_j \sin(jx) \Big].$$

We call f_m a *smooth random function.*

3.1 Exercise (A1)

Write a Matlab function

```
fm = smooth(m)
```

which takes $m \geq 1$ as input and produces an anonymous function `fm` corresponding to the smooth random function f_m as output. You may assume that `fm` takes as its argument a row vector `x`. For reproducibility, ensure that `smooth` calls `randn` to compute its random numbers exactly in the order $a_0, a_1, b_1, a_2, b_2, \ldots, a_m, b_m$. Show your result by executing

```
npts = 5000;
xx = linspace(0,2*pi,npts);
seed = 1; rng(seed), fm = smooth(20);
plot(xx,fm(xx))
```

and then adding appropriate labels to the plot. The point of the "`seed`" part of the code is to make the random numbers reproducible. If you run this multiple times without the `rng` command, you'll get a different smooth random function each time.

3.2 Exercise (A2)

As m increases, the size of f_m can be expected to increase at a rate that scales with $\sqrt{m}$. The standard deviation of this distribution is in fact

$\sigma_m = \sqrt{2m+1}$. (The limit $m \to \infty$ corresponds to that elusive concept known as "white noise".) To illustrate this numerically, first plot `fm(xx)` for $m = 50$ and $m = 200$, including for comparison a pair of horizontal dashed lines at heights $\pm\sigma_m$. Then compute `cm = max(abs(fm(xx)))` with `fm = smooth(m)` for $m = 20, 40, \ldots, 1000$ and plot `cm` against m with `subplot(1,2,1)` and `cm` against $\sqrt{m}$ with `subplot(1,2,2)`. In both cases include for comparison a dashed curve showing $4\sigma_m$. (The number 4 is just a rough approximation; for a more precise analysis one would call upon the field known as extreme value statistics.)

3.3 Exercise (A3)

Consider now the indefinite integral of f_m,

$$g_m(x) = \int_0^x f_m(s)ds.$$

We call this a "smooth random walk", and it can be proved that in a certain precise sense, g_m approaches a Brownian path with probability 1 as $m \to \infty$. (You do not need to know anything about Brownian paths for this exercise.) To verify this numerically, define the anonymous function

```
gmxx = @(fm) (2*pi/npts)*cumsum(fm(xx));
```

to approximate g_m by a cumulative sum over the grid `xx`. Now plot `gmxx` for $m = 10^{4-k}$, $k = 1, 2, 3$, making sure to reinitialize the random number seed with `rng(1)` before each plot. (The reason for taking the powers in decreasing order is just to make the plot a little nicer.) Comment on what you see. Among the remarkable properties of Brownian paths are that, with probability 1, they are continuous but nowhere differentiable.

3.4 Exercise (A4)

Illustrate Brownian paths by producing a plot with five different trajectories `gmxx` on it for $m = 1024$, beginning with `rng(1)` but now without reinitializing the seed between plots. Also illustrate 2D Brownian paths by plotting a `gmxx` produced with `rng(1)` as the x coordinate against another `gmxx` produced with `rng(2)` as the y coordinate. Be sure to set `axis equal` so that your x and y components are equally scaled. Produce two plots: one with `subplot(1,2,1)` for $m = 100$ and another with `subplot(1,2,2)` for $m = 1000$.

Chapter 4

(B) Julia sets

Julia sets are named after Gaston Julia; they have nothing to do with the computer language Julia. These are sets in the complex plane that give beautiful examples of complicated geometry, in particular, fractals.

To define a Julia set we fix a real or complex number c and then we consider the iteration

$$w := w^2 + c. \tag{1}$$

For some complex initial values w, the iteration blows up to infinity as we take more and more steps. For others, it remains bounded. The Julia set is the set in the complex plane defined as the boundary between the sets with these two types of behavior. In this project we plot these domains (approximately) on the computer.

4.1 Exercise (B1)

Write a Matlab function `Z = makegrid(npts)` that uses the commands

```
s = linspace(-1.7,1.7,npts);
[X,Y] = meshgrid(s,s);
Z = X + 1i*Y;
```

to set up a `npts` $\times$ `npts` grid of points in the square $-1.7 \le x, y \le 1.7$ of the complex plane. Test this function by executing

```
plot(makegrid(20),'.k','markersize',10), axis equal
```

Now write a function `plotW(W)` that applies commands that have this effect:

```
pcolor(double(abs(W)<2)), shading flat
axis square off
```

In addition, `plotW` should put a title on the plot along the lines of "34952 yellow pixels" indicating how many of the pixels have satisfied the condition $|w| < 2$. Explain what this function does and test it by executing

```
plotW(makegrid(1000))
```

(If 1000 causes trouble on your computer, use a slightly lower value.) Finally write a function `W = tensteps(W,c)` that applies ten steps of the iteration defined by (1) to a matrix of function values `W`. One could use a nested `for` loop for this, but it is better Matlab style, and more efficient, to operate on the whole matrix at once using the Matlab operation `W.^2`. Test this function by executing

```
plotW(tensteps(makegrid(1000),0))
```

and comment on why the result looks the way it does for this special case of $c = 0$.

4.2 Exercise (B2)

Now write a final function `threeplots(c)` that places three plots on a row using `subplot(1,3,j)` for $j = 1, 2, 3$. The first plot should be the `plotW` result from one call to `tensteps` with the initial values `W` equal to the matrix `Z` of (B1). The second plot should be the result after a second call to `tensteps`, and the third after a third call to `tensteps`. Together, then, the plots show results corresponding to 10, 20 and 30 iterations of (1). Execute `threeplots` with $c = -1$ and comment on the results. What differences does you see between the three plots, and how do you explain these?

4.3 Exercise (B3)

Now execute `threeplots` with $c = 0.3$ and comment on the results. The three plots differ more profoundly in this case than in the last exercise. How do you interpret this?

4.4 Exercise (B4)

Further choices of c give a wide variety of different behaviors. In particular, produce plots for $c = 0.75i$ and $c = 0.4 + 0.6i$ and comment on the results. There has been a great deal of study of the geometry of sets like these, with generalizations in many directions. It is a fine example of how mathematics these days, like physics and chemistry for centuries, depends on laboratory experiments to guide its advance.

Chapter 5

(C) Gauss quadrature

Suppose we have an integral over an interval $[a, b]$, which we might as well take to be $[-1, 1]$:

$$I = \int_{-1}^{1} f(x)dx.$$

For most integrands f, I cannot be computed analytically and must be approximated numerically. A standard way to do this is by applying an n-point *quadrature formula*

$$I_n = \sum_{k=1}^{n} w_k f(x_k),$$

where $x_1, \ldots, x_n$ are a set of $n \geq 1$ distinct *nodes* in $[-1, 1]$ and $w_1, \ldots, w_n$ are a corresponding set of *weights*.

One measure of the quality of a quadrature formula is that it gives exactly the right answer, $I_n = I$, if f is any polynomial of a certain low degree d. It is natural to ask, given n, what's the highest such d that can be achieved by some n-point quadrature formula? The answer turns out to be $d = 2n - 1$, and the unique choice of $\{x_k\}$ and $\{w_k\}$ that achieves this corresponds to *Gauss quadrature,* introduced by Carl Gauss in 1814. (To be precise, it's unique up to a permutation of the indices.)

5.1 Exercise (C1)

For $n = 3$, the Gauss nodes are $-\sqrt{3/5}$, 0, $\sqrt{3/5}$ and the Gauss weights are 5/9, 8/9, 5/9. Write a Matlab code that verifies numerically that $I_3 = I$ for $f(x) = x^k$ with $0 \leq k \leq 5$ but $I_3 \neq I$ for $k = 6$. Specifically, print the errors $E_n = I_n - I$ for each of these cases. What happens to E_n for $k = 7, 8, 9, 10$? How do you explain this?

5.2 Exercise (C2)

A remarkable method for computing Gauss nodes and weights was introduced by Golub and Welsch in 1969. Let A be the $n \times n$ tridiagonal sym-

metric *Jacobi matrix* whose entries are all zero apart from the numbers

$$\frac{1}{\sqrt{1\cdot 3}},\ \frac{2}{\sqrt{3\cdot 5}},\ \frac{3}{\sqrt{5\cdot 7}},\ \dots,\ \frac{n-1}{\sqrt{(2n-3)\cdot(2n-1)}}$$

in the upper-diagonal positions $(1,2),(2,3),\dots,(n-1,n)$ of the matrix and also in the lower-diagonal positions $(2,1),(3,2),\dots,(n,n-1)$. Let $\{\lambda_j\}$ and $\{v_j\}$ be the eigenvalues and eigenvectors of A as computed e.g. by the Matlab `eig` command. (In particular, each eigenvector should be normalized so that the sum of the squares of its entries equal to 1.) Golub and Welsch showed that the node x_j is the eigenvalue λ_j, and the weight w_j is twice the square of the first component of the corresponding eigenvector v_j. Use these properties to write a Matlab function

```
[x,w] = gaussq(n)
```

that returns vectors of the n Gauss quadrature nodes and weights for any n. Verify that for $n = 3$, you get the results given in (C1).

5.3 Exercise (C3)

Consider the function $f(x) = e^x \tanh(4\cos(20x))$ for $x \in [-1,1]$, where tanh is as usual the hyperbolic tangent. Make a plot of f and estimate its integral I by Gauss quadrature with $n = 1000$. Let's call this estimate I even though of course it is not quite exact. Now make a semilogy plot of $|E_n|$ against n for $n = 50, 100, 150, \dots, 950$, making sure that the plot shows clearly where the data points lie. Comment on the shape of this curve, both its early stages and its later stages. (As it happens, the rate of convergence is determined by the locations of the singularities of $f(x)$ in the complex plane.)

5.4 Exercise (C4)

Determine analytically the exact integrals over $[-1,1]$ of the functions

$$\text{(a) } f(x) = \frac{1}{1+25x^2}, \quad \text{(b) } g(x) = \tanh(20(x+\tfrac{1}{2})), \quad \text{(c) } h(x) = |x|.$$

(You do not have to show your working.) Then make two plots with clearly-labeled curves indicating convergence of n-point Gauss quadrature for these three integrands as a function of n. One plot should be on semilogy axes, and the other on loglog axes. Approximately how large must n be for 6-digit accuracy in each of the three cases? What do the shapes of the curves on the two axes indicate about convergence rates?