MS6021 Scientific Computing

using MatLab and **Python**

(the absolute beginners are ok…)

- Natalia Kopteva
- Email: natalia.kopteva@ul.ie
- Web: https://staff.ul.ie/natalia/
- Office Hours: Q&A sessions during scheduled on-campus classes

  or via Skype for Business (see announements via SULIS)
- MS6021 webpage: https://staff.ul.ie/natalia/node/1202/
- see also SULIS for announcements + recordings etc…

Assessment

- **100% continuous assessment**
  (i.e. NO final exam)
- **5 Assignments**,
  (15% + 20% + 20% +20% + 25%).
  - Marked on **code (emailed to me)** +
    LATEX/Word/hand-written
    descriptions/justifications/conclusions/answers to ALL questions…
    (NO zip files please)
  - **Assignment 1** is given in this file;

due end of Week 3 (or get in touch before the deadline!)

Plagiarism & related

- Discussing ideas and algorithms is fine.
- Discussing/**copying (parts of) code** without citation is NOT.

(had such situations in the past; similarities in codes are easily recognizable…)

- **I'm going to run your codes**.
  If I can't reproduce your results from your code I may ask you to demonstrate it for me.

- If I don't virtually meet you for a long time, I may request that we meet and you show me your work…

What this module is about?

- What is Scientific Computing??

Numerical Mathematics
(or ***Numerical Analysis***)

+

Computer Programming (languages, operating systems, management of large quantities of data) **for Scientific Problems**

Computing component:

- MatLab
  (Octave is similar, but free)

- Python
  (instead, we also considered teaching Fortran, C, or C++…
  Julia is becoming popular for Scientific Computing)
  All languages suitable for Scientific Computing have a lot of similarities.

Both have good libraries of subroutines useful for Scientific Computing programming!

This course will touch on: Datatypes and control structures. Functions, procedures and subroutines and modules. Inputs and output.

(Object-Oriented Programming in MatLab or Python won't be considered in this module. Neither linking MatLab to compiled Fortran/C. Nor many other things…)

Numerical Maths component

(official syllabus)

- **Linear algebra:** Norms and conditions numbers, linear equations, over and under-determined systems, inverse and pseudo-inverse, factorisations, singular value decomposition, eigenvalue problems, practical case studies. Non-linear equations: Root finding, optimisation, practical case studies.
- **Differential equations:** Ordinary differential equations, boundary value problems, singular perturbations, boundary layers, partial differential equations, first order PDE method of characteristics, parabolic, hyperbolic and elliptic problems, case studies.

Course Style

- Many ways to teach a course like this:
  (i) lots of details on algorithms (even a bit of theory),
  (ii) how to do x using black boxes.
  - Try to do a bit of both.

- What we'll do:
  Mostly worksheets + assignments
  rather than lectures.

Why not always use black boxes?

- Sometimes you need to know how an algorithm works to understand
  - what it is doing
  - whether it's applicable for your problem (more important!)
  - be able to tweak it if required…

- If a standard algorithm doesn't work or there is no black box, you may need to (call specialists to) address this.

General Programming Style Tips

- It is surprising how hard it is to understand code you've written a even few weeks ago!

For Clarity

- Put lots of comments in your code
- Split problem up into components to be handled by separate
- Create (reusable) functions/subroutines

"The computer is your side"
BUT

- **Computers do what you instruct them to do**
  not what you want them to do.
  So you need to use the language your computer can understand:
  e.g. do NOT write 2x but instead use 2*x ,
  do NOT write sin x but instead use sin(x) …

- BEWARE: Modern, expensive simulation packages will happily produce beautiful, compelling but totally WRONG pictures and movies.

  ➔**TEST your programs** on cases you can solve analytically. Never do a simulation unless you know (roughly) what the answer will be.

Matlab

- Interactive system for **Numerical Computation**
- Written by **Cleve Moler** late 1970s in FORTRAN (now in C)

Matlab Advantages

- High level language - can do a lot with a few statements
- Easy data structures - arrays created and extended automatically
- Interactive - easy to experiment with and debug code
- Reasonably good Graphics
- Programs/Subroutines can be saved as M files
- Toolboxes (commercial, free, or homemade) add area specific capability to MatLab.
- Lots of free m-files available over Internet

Matlab Disadvantages

- Interpreted and therefore slow
- Quite Expensive: detailed knowledge of matlab useless if you work somewhere that can't afford it or prefers mathematica
- Does NOT protect constants: pi=4 accepted (this gives one more flexibility, but dangerous if working on big projects as part of a big team…)
- Encourages trust in black boxes

MatLab part - Textbooks + Links

(see the module website…)

- *An Introduction to Matlab* by David F. Griffiths, University of Dundee.
- *Programming for Computations - MATLAB/Octave* by Svein Linge and Hans Petter Langtangen (Springer Open Access).

ALSO:

- *Matlab Guide*. Higham and Higham.
- *Yet Another Guide TO Matlab* by Matt Dunham and Kevin Murphy, University of British Columbia.

NOTE:
UL offers students and staff a **MatLab licence** so you should be able to install it on your personal laptop/pc.

So NOW we start with MatLab

(You may skip some exercises if already familiar with MatLab)

1. Open MatLab on your computer

2. Open the module website
   https://staff.ul.ie/natalia/node/1202/
   and start working through:

– William Lee's ***Worksheet #1*** (excluding the final section on the Mandelbrot Set, which is optional, i.e. may be done at home)

– William Lee's ***Worksheet #4***

– *An Introduction to Matlab* by David F. Griffiths:
Sections 3, 6, 7, 8, 10, 12-16 (excluding 16.1).
(Other sections are optional.)

– OPTIONAL (you may check at home): *Programming for Computations*
Sections 1.2-1.5,
Section 1.6: Ex. 1.4

More Homework (trivial part)

1. Download
   (see the links at the module webpage)
   - *An Introduction to Matlab* by David F. Griffiths, University of Dundee.
   - *Programming for Computations - MATLAB/Octave* by Svein Linge and Hans Petter Langtangen (Springer Open Access).
   - William Lee's Worksheets #9

2. Install Anaconda (NOT urgent! ; unless you already have Python on your computer)
   - Check whether you have a 64bit or 32bit machine
   - Install the relevant **Python 3.6 version** (Python 2.7 is slightly different, but can be used as well)
   - On completing the installation, try to open **Spyder**

ASSIGNMENT #1

due end of Week 3

(negotiable, but before the deadline)

NOTE: Lecture #2 + Assignment #2 have been considered the most difficult by many students over the years, and will be published by end of Week 2. So, you may consider starting on them early (e.g. from Week 3)…

Comments on Assignment #1

Assignment #1 is given on the next page

- WARNING:
  Assignment #1 is at the **CORE** to certain parts of all other Assignments.
  (So it will be very difficult for you to successfully work on forthcoming assignments unless you complete #1 with full understanding of what you are doing…)

- TIP: **Worksheet #9** and **Sections 15-16 in [Griffiths]** will be useful for this assignment.

- Don't be scared before you start:
- I will **further discuss** how one can approach this assignment **on the whiteboard**… Also, you may **consult me** on further specifics + discuss you codes and results with me...

- **SUBMISSION** (no zip files, please)= **MatLab code(s)** emailed to me **+** (LATEX/Word/hand-written) **method description + conclusions**
  Carefully read what you are requested to do.

  All Questions should be fully addressed to get full points.

i.e. sort of a ***report of all your work*** on this assignment.

(If in doubt, consult me before submission, rather than after getting a possibly disappointing grade :)

Assignment #1 (15%, due end Week 3)

Part I: Consider the boundary value problem:

$$-u'' = \exp(2x), \quad u(0)=u(1)=0.$$

1. Solve the problem analytically (at home; the solution t.b. submitted).
2. Discretise the equation and solve it constructing a sparse matrix using spdiags. Use the spy function to visualise the matrix
   (TIP: see **next page** + **Worksheet #9**, and **Sections 15-16 in [Griffiths]**).
3. Then solve the equation using a full matrix
   (NEVER use this INEFFICIENT approach in future!!!)
4. Find (roughly) the largest number of points N in this method for which MatLab will allow you to construct (i) a full matrix; (ii) a sparse matrix .
5. Compare your computed solutions (using numbers of points N= 4, 8, 16, …) with the analytical solution.
   Present a scientifically convincing argument that larger numbers of points yield more accurate computed solutions. For example, you may
   - plot computed solutions + the analytical solution on the same graph,
   - and/or evaluate the maximum errors,
   - and/or use other means.

NOTE: **to get full marks**, you are requested to use **NO LOOPs** (i.e. NO while or for operators) when working on this assignment.
(Typically, vectorization is more efficient than loops, although there may be exceptions.)

Assignment #1 (tips on Part I)

- Divide the interval (0,1) into N equal subintervals by the points $x_i = (i - 1)\, h, \quad i = 1, \ldots N+1, \quad$ where $h = 1/N$

  (but remember that NO loops are allowed)

- Discretize the differential equation using

$$u''(x_i) \approx \frac{u(x_i + h) - 2u(x_i) + u(x_i - h)}{h^2}$$

- Represent the resulting numerical method as a matrix equation:

  $A\boldsymbol{U} = F$ where A is an (N-1) by (N-1) matrix,

  F is a column vector of length (N-1),

  $\boldsymbol{U} = [U_2, U_3, \ldots, U_N]'$ is the unknown column vector of computed solution values U_i associated with x_i

- Plot the computed solution $\{U_i\}$ for $i = 1, \ldots, N + 1$

Assignment #1 (continued)

Part II: Modify the code of Part I for the problem:

$$-u'' + a(x)u = f(x), \quad u(0)=u(1)=0.$$

NOTE: as now we have a variable coefficient, the exact solution is unknown (unlike the problem in Part I)

The functions $a(x)$ and $f(x)$ are specified on next page (based on your ID).

- To estimate the error, compare computed solutions for the numbers of points N and 2N by evaluating

$$Error := \max |U^N - U^{2N}| \quad \text{for each } N=16, 32, 64,\ldots,$$

where U^N and U^{2N} are the computed solutions obtained with (0,1) respectively divided into N and 2N subintervals,

and the maximum is computed at the nodes of the mesh used for U^N

NOTE: this is NOT an actual error, but a good indicator of how small the error is, and whether the error becomes smaller for larger N and how fast…

- Complete the table

N	16	32	64	128	256	512
Error						

The coefficients a(x) and f(x) are specified below based on your ID:

- If your ID starts with “20”, then
  $a(x) = \exp(x^2 - x)$
- If your ID starts with “16”, then
  $a(x) = 3 + \cos(2x^3)$
- Otherwise,
  $a(x) = \ln(2 + x^2)$

- If sum of the digits in your ID is odd, use
  $f(x) = \cos(2x) + 2 - x^2$
- Otherwise,
  $f(x) = 7/(2 + x^2 + x)$