Mathematical Tripos: IA/IB Computational Projects

Contents

0 Introduction **1**

0.1 Overview . . . 1

0.2 Aims . . . 1

0.2.1 Understanding algorithms . . . 2

0.2.2 Being able to programme . . . 2

0.2.3 Writing up your results . . . 3

0.3 Why use computers in mathematics? . . . 4

1 Algorithms **6**

1.1 Matrix Multiplication . . . 6

1.1.1 Complexity (or the number of operations) . . . 7

1.1.2 MATLAB implementation . . . 7

1.1.3 Are there better algorithms? . . . 8

1.1.4 Triangular Matrices . . . 8

1.2 Matrix Multiplication in MATLAB (*unlectured*) . . . 10

1.2.1 My first MATLAB . . . 10

1.2.2 A function to multiply two matrices . . . 13

2 Numbers and Errors **15**

2.0 Introduction . . . 15

2.1 Big Numbers, Small Numbers and Precision . . . 15

2.2 Types of 'Rounding Error' . . . 19

2.3 Truncation Error . . . 22

3 Solution of Transcendental Equations **23**

3.0 Introduction . . . 23

3.1 The Basics . . . 23

3.2 Termination Criteria . . . 23

3.3 Bisection, Interval Halving or Binary Search . . . 23

3.3.1 Recursion . . . 26

3.4 Order of Convergence . . . 27

3.5 Other Methods . . . 28

4 Solution of Ordinary Differential Equations **29**

4.0 Introduction . . . 29

4.1 The Euler Method . . . 29

4.1.1 Example MATLAB code . . . 30

4.2 Accuracy . . . 32

4.2.1 Local truncation error of the Euler method . . . 32

4.2.2 Global truncation error of the Euler method (*Unlectured*) . . . 32

4.2.3 Testing accuracy without the exact solution, and a free lunch! . . . 33

4.2.4 Numerical confirmation of accuracy . . . 34

4.3 Stability of the Euler Method . . . 36

5 Gaussian Elimination **39**

5.0 Introduction . . . 39

5.1 LU factorization . . . 39

5.1.1 Definition . . . 39

5.1.2 Application: solution of linear systems . . . 39

5.2 The calculation of an LU factorization . . . 40

5.2.1 MATLAB code . . . 41

5.2.2 Check the code . . . 43

5.3 LU factorization with column pivoting . . . 44

5.3.1 More MATLAB . . . 46

5.4 Solution of linear equations . . . 47

6 The Nitty Gritty **51**

6.1 The Nature of CATAM Projects . . . 51

6.2 Examination Credit . . . 51

6.3 Timetable . . . 52

6.3.1 Planning your work . . . 52

6.4 Programming language[s] . . . 53

6.4.1 Your copy of MATLAB . . . 53

6.4.2 Programming manual[s] . . . 53

6.4.3 To MATLAB, or not to MATLAB . . . 54

7 Project Reports **55**

7.1 Project write-ups: examination credit . . . 55

7.2 Project write-ups: advice . . . 55

7.2.1 Project write-ups: advice on length . . . 57

7.3 Project write-ups: technicalities . . . 57

8 Information Sources **58**

9 Unfair Means, Plagiarism and Guidelines for Collaboration **60**

9.1 Oral examinations 62

10 Submission and Assessment **63**

10.1 Submission form 63

10.2 Submission of written work 63

10.3 Electronic submission 63

10.4 (Non)-return of written work 64

0 Introduction

0.1 Overview

CATAM stands for *Computer-Aided Teaching of All Mathematics*.[1] The course is different to anything else in the Tripos. In outline:

- it is examined in both Part IB and Part II (but the introductory lectures are given in the Easter Term of the Part IA year);
- there is no end-of-year examination;
- instead, in Part IB and Part II there are a number of projects that you do **throughout** the year, with the course being examined entirely through the submission of project write-ups, i.e. this is the part of the Tripos with an element of continuous assessment;
- in Part IB, there are two *compulsory/core* project write-ups that should be submitted (together with associated computer programs) early in the Lent Term of 2015, and two further *additional* projects that should be submitted early in the Easter Term of 2015;
- in Part IB the maximum credit obtainable is 160 marks and there are no α or β quality marks;
- credit obtained is added directly to the credit gained in the written examination, and hence the maximum contribution to the final merit mark is 160 (which is roughly the same, averaging over the α weightings, as for a 16-lecture course);
- in past years the CATAM credit has often contributed 10%-15% of a person's final mark in the Part IB examination (i.e. more than a a typical 16-lecture course);

Remarks.

(i) Note the **throughout** the year. For some reason too many students leave matters too late, e.g. the vacation immediately before submission, and then it so happens that they are ill and/or suffer some other personal misfortune[2], and then . . . . If you are concerned about examination marks then you have been warned.

(ii) Of course you may have a higher calling: to quote a student email to the helpline:

> *Perhaps I should think of the projects as an opportunity to do some interesting maths, rather than as a means of picking up marks.*

(iii) In the eighth and final lecture on 20th May I will go into more detail about the nuts-and-bolts of the course, including warnings about plagiarism, etc. Please attend that lecture. Before that some of you may think that Part IA revision is more important; my experience is that if you are thinking that then you may well be the type of student who needs to attend the lectures, e.g. because you are type of student who hardly ever gets caught up, or needs an incentive to get up in the morning, or will need the CATAM marks, or . . . .

0.2 Aims

The blurb on the CATAM website (`http://www.maths.cam.ac.uk/undergrad/catam/`) states that

> the *Computational Projects* course provides an education in solving **mathematical** problems using a computing environment. The emphasis is on developing **mathematical** skills rather than programming abilities. The aim is for students to learn to use basic computational techniques and software packages to solve interesting problems, many of which are analytically intractable (or at least algebraically messy).

[1] Forty or so years ago it stood for *Computer-Aided Teaching of Applied Mathematics*, but since then pure mathematicians have discovered that computers are useful for them as well.

[2] E.g. their laptop is stolen, or there is a volcanic eruption.

The most important part of the course is therefore the *mathematics*. However, in order to do the mathematics, and get your marks, there are three other skills, all of which are important 'transferable' skills, that you need to employ (if not necessarily master):

(i) firstly, you need to be able to 'convert' a piece of mathematics that you want to do into one or more algorithms, i.e. a set of rules, that you programme on your computer;

(ii) secondly you need to be able to programme your computer using a suitable programming language;

(iii) thirdly, you need to be able to write-up your methods, calculations, results and conclusions in a way that is understandable to someone else (in your case, the Assessor). My second warning: it is the write-up, not the programming, that takes the time; leave enough time (see the first warning above).

As such CATAM projects are intended to be exercises in independent investigation somewhat like those a mathematician might be asked to undertake in the 'real world'. The questions posed in the projects are more *open-ended* than standard Tripos questions: there is frequently not a single 'correct' response, nor often is the method of investigation fully specified. This is *deliberate*, and is intended to allow you both to demonstrate your ability to use your own judgement in such matters, and also to produce mathematically intelligent, relevant responses to imprecise questions.

0.2.1 Understanding algorithms

The lectures will mainly be about how to go about finding, and then codifying, algorithms. This will be achieved by using as examples either mathematics that you already understand, or mathematics which you need a quick introduction to (e.g. in order to do one of the *Core Projects*). Part of the course may feel like an extension to *Vectors & Matrices* (and as such may help your revision).

0.2.2 Being able to programme

Once you understand how to construct an algorithm, then the next thing you need to know is how to convert the algorithm into a language that the computer will understand.

We all have our own preferred language in which to communicate, the choice often being an accident of birth. Some of us understand many languages, and there is almost always no 'right' language (although it helps if, say, aircraft controllers use the same language).

Computers understand many languages, and there is often no 'right' language. However, users may have their own preferred language in which to communicate with the computer, the choice often being an accident of birth.

This year, as for the last few years, the programming language the Faculty is 'supporting' is MATLAB (standing for MATrix LABoratory). MATLAB is designed to work with matrices (which has certain advantages, and certain disadvantages). One of it's advantages over languages such as C (the previously 'supported' language) is that it's easier to learn and it has a built-in user friendly interface; a disadvantage is that it's not as flexible. The feedback we have had is that, on the whole, students have found it easier to programme in MATLAB than C.

Your copy of MATLAB. All undergraduate students at the University are entitled to download and install MATLAB on their own computer, running Windows, Linux, or MacOS, for *non-commercial* University use. Installation instructions and the files for download are available at

`http://www.maths.cam.ac.uk/undergrad/catam/software/matlabinstall/matlab-personal.htm`

You will see that several versions of MATLAB are available; if you are downloading MATLAB for the first time it is recommended that you choose the latest version, currently R2013b (version R2014a should be available soon[3]).

[3] See also `http://www.maths.cam.ac.uk/computing/software/matlab/install_matlab.html`.

MATLAB is also available on the University Managed Cluster Service (MCS); in particular it is available

- on the Mathematics/CATAM MCS in room GL.04 of the CMS,
- on the Computing Service public MCS computers, and
- on the MCS computers at a number of Colleges.

Learning MATLAB. To some extent the Faculty works on the same principle as that by which you learnt your very first language; we let you pick it up as you go along by exposure (e.g. through these lectures).[4] To a greater or lesser extent, this works. However, some other help is at hand.

- You should have already worked through the Faculty's introductory programming manual that provides a step-by-step introduction to MATLAB suitable for beginners. This is available on-line at

  http://www.maths.cam.ac.uk/undergrad/catam/MATLAB/manual/booklet.pdf.

  A number of sessions were arranged so that you could work through this introductory programming manual in the Mathematics/CATAM MCS room (i.e. room GL.04 in the CMS) with a adviser present who could help you with any difficulties. If you have not taken advantage of these sessions and subsequently need help, do not worry too much (see later).
- There are also a number of excellent books, and Google returns about 24,300,000 hits for the search 'MATLAB introduction' (up from 13,400,000 hits last year), and about 3,210,000 hits for the search 'MATLAB introduction tutorial' (slightly down from 3,340,000 hits).

To MATLAB, or not to MATLAB. While the use of MATLAB is recommended, especially if you have not programmed before, its use is not compulsory, and *you are free to write your programs in any computing language whatsoever*. Python, C, C++, C#, Java, Visual Basic, Mathematica, Maple, php, Haskell (if you must), etc. have been used by several students in the past, and Excel has often been used for plotting graphs of computed results. The choice is your own.[5]

However, you should be aware that

- we do *not* promise to help you with programming problems if you use a language other than MATLAB;
- not all languages have the *breadth of mathematical routines* that come with the MATLAB package, and hence a language that is ideal for writing apps may not be ideal for doing maths;
- if the language you are using does not have the needed routines then it will be up to you to either write your own routines (i.e. re-invent the wheel) or find reliable replacements.[6]

0.2.3 Writing up your results

Almost all future employment will require you to be able to write-up your work in a coherent fashion. This is the part of the Mathematical Tripos where you can hone this skill. Further, there is an incentive in that, while 40% of marks are awarded for computing, 50% of marks are awarded for the mathematics in your write-ups, and 10% of marks are awarded for the 'quality' of your write-ups. Hence, if a write-up is incoherent you stand to lose over half your marks.

[4] One of the issues is that you already have enough lectures, etc., so there is a reluctance to give you even more.

[5] However, if you choose a high-level package that can perform some advanced mathematical operations automatically (for instance, complete prime factorisation of an integer with a single command), then you should check whether use of such commands are permitted in a particular project.

[6] The converse of this is that there are so many official and unofficial MATLAB routines that there is almost certainly a routine to do whatever you want ... but beware of breaking the plagiarism rules. Also, a further word of warning, if you find a routine that does nearly what you want, and start editing it, you may find that the routine uses more sophisticated MATLAB than you understand or need, and this can turn into a sink of time (as some students have found in the past).

As part of your training in this aspect, over the summer you will have the opportunity to do an *Introductory Project*.[7] The *Introductory Project* carries no marks, and a model answer will be made available at the start of the Michaelmas Term.[8] Your College is at liberty to arrange a supervision on the *Introductory Project*. As an innovation this year, we hope to run an Example Class on the *Introductory Project* towards the start of the Michaelmas Term.

0.3 Why use computers in mathematics?

Forty or so years ago some might have argued (well did argue) that computers had little role in mathematics (or more accurately pure mathematics). However, we have come a *long* way from just using computers to work out the correct trajectory of a space rocket that reaches the moon - and comes back (well we have come *some* way). Computers are now used in all branches of mathematics.

Pure

Possibly the most famous example in Pure Mathematics was the proof of the 4-colour theorem in 1976; the problem was first reduced by analysis to 1936 special cases, and then a computer was programmed to colour each case. In the last couple of years Harald Helfgott has claimed a computer-aided proof of the weak Goldbach conjecture.

Nowadays computers are not primarily used for simply enumerating cases; often the idea is to use computers to investigate different examples of some construct or entity (e.g. groups, graphs, number systems) and for the mathematician to observe and thence to spot patterns or connections: a proof can then be attempted in the usual way.

A computer is a tool that can be used to obtain insight.

Pure mathematical concepts can also lead to practical applications, e.g. large primes and modular arithmetic (computer security), or graph colouring (mobile phone networks).

Specialist pure mathematics software has also been developed, e.g. MAGMA (for algebra, number theory, geometry and combinatorics).

Applicable

From the outset, computers have been used to carry out statistical analyses of large amounts of data.

However, computers can also be used, say, to carry out simulations of random processes to see if there are global features of interest which can then be modelled. For instance, microscopic Brownian motion, a random process, can be simulated and the macroscopic qualities assessed (but in this simple case you can also do much analytically).

Similarly, the measurements of stocks, shares, derivatives and options (which on short time scales might be random[ish]) can be modelled and large scale movements investigated.

Applied

Use here varies from simple integration of ODEs and modelling dynamical systems (e.g. to calculate required trajectories), to simulations of weather, climate change, volcanic eruptions (and where the ash goes), semi-conductors and other materials, flow around aircraft, cars and underwater moving objects, and the start and evolution of the Universe.

[7] This year's project is not yet available formally available, but is highly likely to be remarkably similar to last year's project which is on-line at

http://www.maths.cam.ac.uk/undergrad/catam/IB/0pt1.pdf

[8] It's relatively simple to find the answer before that, but it's educationally better to do the project blind.

Some of these real-life situations can be modelled extremely well (e.g. much aircraft design is done computationally), and sometimes not (e.g. dispersion of ash, but that has improved over the last couple of years). The applied mathematician is on the lookout for common features and interesting behaviour (which *might* then be examined on paper), or improving the computational methods so that bigger and better simulations are possible.

Previous Questionnaire Responses

Comment. I get the impression that most mathmos have already done some coding, and would benefit more from MATLAB specific guidance than from a general introduction to coding.

Reply. Last year 60% had no experience, or little previous experience, of coding.

Comment. The lectures ought to have covered less technical things and done actually mildly interesting topics, asked interesting questions so that students would actually have a reason to learn/practice coding.

Reply. The course is about about how to start coding something you understand (and about some of the pitfalls), not about teaching new material. Familiarity might breed contempt, but it may also help see the wood for the trees. This is especially so in the case of the *Introductory Project* where the aim is to learn how to *write-up* your work (rather then concentrate on new mathematics).

1 Algorithms

Definition. An algorithm is a process or set of rules used in calculations or other problem solving operations.

You have already encountered them.

1.1 Matrix Multiplication

Let $\mathsf{A} = \{A_{ij}\}$ and $\mathsf{B} = \{B_{ij}\}$ be $n \times n$ matrices. Suppose that $\mathsf{C} = \mathsf{AB}$. Then from *Vectors & Matrices*

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \quad (i = 1, \dots, n,\ j = 1, \dots, n)\,.$$

For instance, if

$$\mathsf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \qquad \mathsf{B} = \begin{pmatrix} p & q \\ r & s \end{pmatrix},$$

then

$$\mathsf{AB} = \begin{pmatrix} ap + br & aq + bs \\ cp + dr & cq + ds \end{pmatrix}.$$

How do we express this as an algorithm? Well both indices i and j need to run through 1 to n; we can express this in MATLAB using two `for` loops. We write `C(i,j)` for C_{ij}, and we need to remember to set `C(i,j)` to zero before starting the sum, which we express using another `for` loop. Combining the above we obtain

```
>> for i = 1:n
     for j = 1:n
       C(i,j) = 0;
       for k = 1:n
         C(i,j) = C(i,j) + A(i,k)*B(k,j);
       end
     end
   end
```

Remarks. Recall from the MATLAB booklet/sessions that

(i) `i=1:n` means run `i` from `1` to `n` in steps of `1`;

(ii) `end` indicates the end of a `for` loop;

(iii) a semi-colon at the end of a statement means do not print out the results of any assignment;

(iv) despite what a mathematician might think, the line

```
C(i,j) = C(i,j) + A(i,k)*B(k,j)
```

does not imply that `A(i,k)*B(k,j)` is equal to zero. It means set `C(i,j)` equal to its old value plus `A(i,k)*B(k,j)`. The line is better read as 'assign `C(i,j)` to be equal to `C(i,j)+A(i,k)*B(k,j)`', rather than '`C(i,j)` equals `C(i,j)+A(i,k)*B(k,j)`'. To avoid confusion some computer languages (e.g. Ada[9]) use := for the assignment operator and use = for the boolean operator to test equality of values, others (e.g. MATLAB, C) use = for the assignment operator and use == for the boolean operator to test equality of values, and others (e.g. BASIC) use = for both (so as to maximise confusion).

Suppose instead that $\mathsf{A} = \{A_{ij}\}$ was a $m \times n$ matrix and $\mathsf{B} = \{B_{ij}\}$ was a $n \times q$ matrix. Then $\mathsf{C} = \mathsf{AB}$ would be the $m \times q$ matrix given by

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \quad (i = 1, \dots, m,\ j = 1, \dots, q)\,.$$

[9] Named after Ada Lovelace.

The MATLAB to evaluate the matrix C would be

```
>> for i = 1:m
     for j = 1:q
       C(i,j) = 0;
       for k = 1:n
         C(i,j) = C(i,j) + A(i,k)*B(k,j);
       end
     end
   end
```

1.1.1 Complexity (or the number of operations)

In the case when $m = n = q$, we have three `for` loops running through n variables. Hence in calculating the matrix $\mathsf{C} = \mathsf{AB}$ the computer performs

(a) n^2 assignments of `C(i,j)` to zero;

(b) n^3 multiplications;

(c) n^3 additions;

(d) n^3 further assignments.

This is a total of $3n^3+n^2$ operations, including n^3 multiplications and n^3 additions. Matrix multiplication by this method is said to be an $O(n^3)$ operation.

1.1.2 MATLAB implementation

The unlectured section §1.2 below contains an explanation of how to write MATLAB code to multiply two matrices together. The result is a MATLAB function **`matmult`** with two *input* arguments, namely the matrices `A` and `B`, and one *output* argument, namely the matrix `C`.

```
function C=matmult(A,B)
%
% MATLAB code illustrating matrix multiply.
%
% This code is inefficient so should NOT be used in anger.
[m,n] = size(A);
[p,q] = size(B);
%
if n ~= p
  error('Columns of the left matrix do not match rows of the right matrix.')
else
  for I = 1:m
    for J = 1:q
      C(I,J) = 0;
      for K = 1:n
        C(I,J) = C(I,J) + A(I,K)*B(K,J);
      end
    end
  end
end
```

Remark. MATLAB distinguishes between upper and lower case. Compared with the earlier code `i`, `j` and `k` have been replaced by `I`, `J` and `K` respectively. This is because it is best to avoid the use of `i` and `j` in MATLAB, since they are both the built-in root of `-1`.[10]

[10] If you have set `i` to something other than the square root of `-1`, and you need the square root of `-1`, then you can use `j`, or failing that use `clear i` and/or `clear j`.

We can test how quickly this MATLAB code runs with the built-in timing commands `tic` and `toc`. We also need an easy way to generate a matrix; this can be done with the built-in *function* `rand(n)`, which generates a $n \times n$ matrix containing pseudo-random values drawn from the standard uniform distribution on the open interval(0,1). It is then possible to compare the times of multiplying 500×500 and 1000×1000 matrices using `matmult` as follows, where we also compare with the time taken using the MATLAB built-in function `*`.

```
>> n= 500; A=rand(n); tic; matmult(A,A); toc
>> n=1000; A=rand(n); tic; matmult(A,A); toc
>>                    tic; A*A         ; toc
```

Remark. The function `matmult` has been introduced for educational reasons. The MATLAB built-in function `*` is far preferable.

1.1.3 Are there better algorithms?

A natural question to ask is whether we could do better than $O(n^3)$ operations. To have an inkling why the answer is yes consider the multiplication of two 2×2 matrices. Let

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}),\\
M_2 &= (A_{21} + A_{22})B_{11},\\
M_3 &= A_{11}(B_{12} - B_{22}),\\
M_4 &= A_{22}(B_{21} - B_{11}),\\
M_5 &= (A_{11} + A_{12})B_{22},\\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{21}),\\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}).
\end{aligned}$$

Then $\mathsf{C} = \mathsf{AB}$ can be calculated as follows:

$$\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7,\\
C_{12} &= M_3 + M_5,\\
C_{21} &= M_2 + M_4,\\
C_{22} &= M_1 - M_2 + M_3 + M_6.
\end{aligned}$$

This is *Strassen's algorithm*. It requires 7 multiplications and 18 additions, compared with 8 multiplications and 8 additions by our previous method. This does not look like an improvement until one realises that multiplications tend to be more expensive for computers than additions.

If this idea is now applied recursively we find that for an $n \times n$ matrix the multiplication cost is $O(n^{\log_2 7}) \approx O(n^{2.807})$ operations. Hence for $n \gg 1$ *Strassen's algorithm* is much cheaper than the algorithm we first thought of (and is used in 'black box' multiplication routines when n is large enough).

There are other algorithms. For instance, the Coppersmith-Winograd algorithm requires $O(\kappa n^{2.376})$ multiplications, where κ is a *constant pre-multiplier*. Unfortunately κ is so large in this case that the method is not useful for practical values of n.

Remark. There have been recent improvements to the bound by Andrew Stothers (2010) to $O(n^{2.373})$, Virginia Williams (2011) to $O(n^{2.3728642})$, and François Le Gall (2014) to $O(n^{2.3728639})$. It is an
01/14 open question as to whether an $O(n^2)$ or $O(n^2(\log n)^m)$ algorithm is possible.

1.1.4 Triangular Matrices

If

$$\mathsf{A} = \begin{pmatrix} a & b \\ 0 & d \end{pmatrix}, \qquad \mathsf{B} = \begin{pmatrix} p & q \\ 0 & s \end{pmatrix},$$

then

$$\mathsf{AB} = \begin{pmatrix} ap & aq + bs \\ 0 & ds \end{pmatrix}.$$

If we used our original algorithm then we would perform 8 multiplications, but 4 of them would be multiplications by zero. It is computationally cheaper if we do not multiply by zero!

Suppose that both A and B are upper triangular. Then $A_{ik} = 0$ if $i > k$ and $B_{kj} = 0$ if $k > j$, and

$$C_{ij} = \sum_{k=i}^{j} A_{ik} B_{kj} \quad (i = 1, \ldots, n,\ j = 1, \ldots, n)\,.$$

where if $j < i$ it is assumed that there is no sum. In terms of MATLAB we can write this as

```
>> for I = 1:n
     for J = 1:I-1
       C(I,J) = 0;
     end
     for J = I:n
       C(I,J) = 0;
       for K = I:J
         C(I,J) = C(I,J) + A(J,K)*B(K,J);
       end
     end
   end
```

Remark. Note that for `I=1` it is assumed that the computer is smart enough not to execute the `for J = 1:I-1` loop.[11]

Complexity. If we stop multiplying by zero then

$$\begin{aligned}
\text{number of multiplications} &= \sum_{i=1}^{n} \sum_{j=i}^{n} (j - i + 1) \\
&= \sum_{i=1}^{n} \sum_{\ell=1}^{n-i+1} \ell && \text{where } \ell = j - i + 1 \\
&= \sum_{i=1}^{n} \frac{1}{2}(n - i + 1)(n - i + 2) \\
&\approx \frac{1}{2} \sum_{i=1}^{n} (n - i)^2 \\
&\approx \frac{1}{2} \sum_{k=1}^{n} k^2 && \text{where } k = n - i \\
&\approx \frac{1}{2} \left(\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \right) \\
&\approx \frac{1}{6} n^3
\end{aligned}$$

01/10 For $n \gg 1$ we can save $\frac{5}{6}$ of the operations by not multiplying by zero.

01/11
01/12
01/13

[11] Sometimes the computer is smart enough, sometimes it executes the loop once just for the hell of it (which would not be a problem with the current ordering of the `for J = ...` loops, but would be if the order of the `for J = ...` loops was reversed), and sometimes it executes the loop forever (if you are very unlucky).

1.2 Matrix Multiplication in Matlab (*unlectured*)

Throughout these lectures it will be assumed that you have worked through the Matlab introductory programming manual available on-line at

http://www.maths.cam.ac.uk/undergrad/catam/MATLAB/manual/booklet.pdf

If you have not, you need to do it now. The following sub-section covers a few of the very basics.

1.2.1 My first Matlab

In Matlab if we enter

```
>> I=1
```

then Matlab assigns the value `1` to the variable `I` and outputs the result. We can check the value of `I` by

```
>> I
```

If we put a semi-colon at the end of the line, then the assignment is done quietly, but it is still done:

```
>> I=2;
>> I
```

If we want to save screen real-estate then the command `format compact` helps:

```
>> format compact;
>> I=-1;
>> I
```

There is a good reason why we have used `I` rather than `i`

```
>> i
```

We can increase the value of `I` by one with the Matlab command

```
>> I=I+1
```

Vectors and matrices. In Matlab it is straightforward to create row vectors and column vectors, to check their sizes, and to print out the values of individual elements, e.g.

```
>> A=[1,2]
>> B=[-3,-4]'
>> size(A)
>> size(B)
>> A(1,1)
>> B(2,1)
```

It is also straightforward to create larger matrices (and find an alternative use for `I`):

```
>> I=[1,0;0,1]
>> F=[0,1;1,0]
>> G=[1,2,3;4,5,6]
>> size(I)
>> size(F)
>> size(G)
```

It is possible to store the size of matrices, e.g.

```
>> [m,n]=size(A);
>> [p,q]=size(B);
>> m, n, p, q
```

Since m=q using the code on page 6 in §1.1 we can calculate C=AB

```
>> for I = 1:m
     for J = 1:q
       C(I,J) = 0;
       for K = 1:n
         C(I,J) = C(I,J) + A(I,K)*B(K,J);
       end
     end
   end
>> C
```

Further, since n=p, we can also the above code to calculate D=BA

```
>> for I = 1:p
     for J = 1:n
       D(I,J) = 0;
       for K = 1:m
         D(I,J) = D(I,J) + B(I,K)*A(K,J);
       end
     end
   end
>> D
```

Typing less. If you are doing similar tasks more than once, you do not want to type the same thing in a number of times. What you would like to do is save your program somewhere so that you can run it after minor changes.

The precise commands now depend on what version of MATLAB you are using. Here we assume you are running version R2012b or later (enter the MATLAB command `version` to see what version you are running).

Left-click on `New` towards the top left of the MATLAB window, followed by `Script`.[12] A new *'Editor'* window should open containing a cursor on line 1. Type in the following code:

```
A=[1,2];
B=[-3;-4];
[m,n]=size(A);
[p,q]=size(B);
for I = 1:m
  for J = 1:q
    C(I,J) = 0;
    for K = 1:n
      C(I,J) = C(I,J) + A(I,K)*B(K,J);
    end
  end
end
C
```

To save the code in a file left-click on the arrow beneath `Save` towards the top left of the *'Editor'* window, followed by `Save As...`.[13] A new window should appear with a default `File Name` of something like

[12] Or, depending on the version of MATLAB you are running, left-click on `File` on the top line of the MATLAB window, followed by `New` and `Script` (or `M-File` or `Blank M-File`).

[13] Or, depending on the version of MATLAB, left-click on `File` on the top line of the *'Editor'* window, followed by `Save As...`.

`untitledn.m`. To change the file name left-click in the box next to `File Name` and change the entry to, say, `testmult`. Then left-click `Save` to accept this name. If you look in the *'Current Folder'* pane a file `testmult` should now have appeared. Having saved your code, return to the *'Command Window'* and enter

```
>> testmult
```

The result `C=-11` should be output.

If you have made a mistake you will need to return to the *'Editor'* window and modify your code. Once you have made your corrections save the code by left-clicking on `Save` towards the top left of the *'Editor'* window. [14]

Suppose now we wish to multiply two difference matrices. Rather than typing in the commands again we can edit `testmult.m` to read, say:

```
A=[0,1;1,0];
B=[1,2,3;4,5,6];
[m,n]=size(A);
[p,q]=size(B);
for I = 1:m
  for J = 1:q
    C(I,J) = 0;
    for K = 1:n
      C(I,J) = C(I,J) + A(I,K)*B(K,J);
    end
  end
end
C
```

`Save` your changes, then type `testmult` in *'Command Window'* to confirm that

$$\mathtt{C} = \begin{pmatrix} 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix}.$$

Sanity checks. Note that we are assuming that we are smart enough to type in matrices that can be multiplied together, i.e. we are assuming that if the sizes of `A` and `B` are $m \times n$ and $p \times q$ respectively, then $n = p$. It is best to check for this (otherwise it's rubbish in and rubbish out) by building in sanity checks into your code. We can do this using an `if` statement and the `error` function by modifying the code to obtain

```
A=[1,2,3;4,5,6];
B=[0,1;1,0];
[m,n]=size(A);
[p,q]=size(B);
if n ~= p
  error('Columns of the left matrix do not match rows of the right matrix.')
else
  for I = 1:m
    for J = 1:q
      C(I,J) = 0;
      for K = 1:n
        C(I,J) = C(I,J) + A(I,K)*B(K,J);
      end
    end
  end
end
C
```

[14] Or, depending on the version of MATLAB, left-click on `File` on the top line of the *'Editor'* window, followed by `Save`.

Here the expression ~= means not equal, and the `error` function terminates the code while outputting the *character string* between quotes. Typing `testmult` in the *'Command Window'* now produces an error (with a somewhat helpful message).

1.2.2 A function to multiply two matrices

A better way to implement the above code for multiplying two matrices together, is as a MATLAB *function*, say called `matmult` (see the first line below). To this end save the following code in a file called `matmult[.m]`.

```
function C=matmult(A,B)
%
% MATLAB code illustrating matrix multiply.
%
% This code is inefficient so should NOT be used in anger.
[m,n] = size(A);
[p,q] = size(B);
%
if n ~= p
  error('Columns of the left matrix do not match rows of the right matrix.')
else
  for I = 1:m
    for J = 1:q
      C(I,J) = 0;
      for K = 1:n
        C(I,J) = C(I,J) + A(I,K)*B(K,J);
      end
    end
  end
end
```

Remarks.

(i) The function `matmult` has two *input* arguments, namely the matrices `A` and `B`, and one *output* argument, namely the matrix `C` (see the first line).

(ii) Lines beginning with % are *comment* lines that are added for explanation. The first few lines are built-in `help`. If you have saved the above code in a file `matmult[.m]`, then in a *'Command Window'* try

```
>> help matmult
```

We might test the function `matmult` by entering into the *'Command Window'*

```
>> F=[0,1;1,0];
>> G=[1,2,3;4,5,6];
>> H=matmult(F,G);
>> H
>> H=matmult(G,F);
```

Remark. Note the use of `F`, `G` and `H` rather than `A`, `B` and `C`. The variables `A`, `B` and `C` within the function `matmult` are, so-called, *dummy variables*. The function `matmult` manipulates `F` and `G` as if they were `A` and `B` respectively, and outputs the answer to `H` as if it were `C`.

Alternatively there is an easier way of multiplying two matrices together in MATLAB (note that MATLAB has its own built-in sanity checks):

```
>> F*G
>> G*F
```

As noted earlier, the easier way is far preferable:

```
>> n=500; A=rand(n); tic; matmult(A,A); toc
>>                   tic; A*A         ; toc
```

Exercises. MATLAB has multiple multiplications. As an exercise try the following:

```
>> A=[1,2]
>> B=[-3,-4]'
>> E=[5,-6];
>> A*B
>> B*A
>> A.*B
>> B.*A
>> A*E
>> E*A
>> A.*E
>> E.*A
>> sum(E.*A)
```

If MATLAB generates an error, make sure you understand why.

2 Numbers and Errors

2.0 Introduction

Understanding the accuracy of a solution is a key skill.

- The first part of this section (on how numbers are stored in computers) is for your general education: it is something that you ought to see at some point in your university career. It is also something that you may end up using, but at a time that is at present unknown.
- The second part of the section on rounding and truncation errors will be directly useful for at least one of the projects next year.

2.1 Big Numbers, Small Numbers and Precision

(i) Computers store everything in *bits* (or *binary digits*). Bits have only two possible values, which are usually denoted by 0 and 1. Depending on the context, the two values can be interpreted as logical values (true/false, yes/no), algebraic signs (+/-), activation states (on/off), or any other two-valued attribute.[15]

(ii) Computers store numbers in binary as a sequence of bits, and can represent integers exactly, as long as there are enough bits. For instance, with 32 bits per integer it is possible to represent *positive* integers from 0 to $2^{32}-1$, or signed integers from -2^{31} to $2^{31}-1$.

(iii) However not all numbers are integers, and it is important to realise that computers cannot store most numbers accurately. The problem is not just with irrational numbers such as $\sqrt{2}$, π, and e, which would apparently need an infinite machine[16], but also with very large (in magnitude) numbers, very small (in magnitude) numbers, and numbers with large numbers of significant digits.

(iv) Computers represent *floating-point* real numbers, r, as

$$r = (-1)^s\, f\, b^e\,, \tag{2.1}$$

where s, f, b and e are integers; s determines the sign, f is the *significand* (or 'coefficient'), b is the base (usually 2), and e is the exponent. The possible finite values that can be represented in a given format are determined by the number of digits in the significand f (the precision, p), the base b, and the number of digits in the exponent e.

So that results produced on one computer can be [exactly] reproduced on another, there are standards for precisely how the numbers are represented. According to the IEEE 754 standard, the exponent e is stored in *biased format* where a constant called the *bias*, say e_{max}, is added to e so that the lowest representable exponent is stored as 1. If n is the number of bits in the exponent
02/11 field, according to the IEEE 754 standard, $e_{max} = 2^{n-1}-1$.

(v) There are a number of basic formats according to IEEE 754; we consider the two binary floating-point basic formats encoded using 32 and 64 bits (each has one sign bit, and 23 or 52 significand bits, and 8 or 11 exponent bits, respectively).

Bits	*Common Name*	*b*	*p*	e_{min}	e_{max}	*Decimal Digits*	*Decimal* e_{max}
32	Single precision	2	23+1	-2^7+2	2^7-1	7.22	38.23
64	Double precision	2	52+1	$-2^{10}+2$	$2^{10}-1$	15.95	307.95

[15] A *qubit* has some similarities to a classical bit. Like a bit, a qubit can have two possible values, say a 0 or a 1. The difference is that whereas a bit must be either 0 or 1, a qubit can be 0, 1, or a superposition of both.
[16] Although *algebraic manipulators* [sometimes] have a way round this.

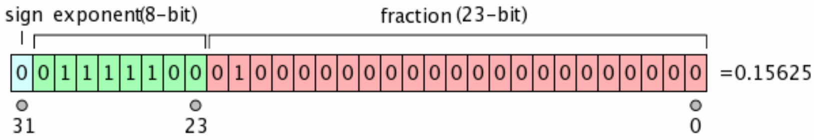


Figure 1: A 32-bit IEEE 754 (a.k.a. *single-precision*) floating-point number has 1 sign bit, 8 exponent bits, and 23 fraction (or significand) bits. Here $s = 0$, $e = (1111100)_2 = 124$, $f = (1.01)_2 = 1.25$, and so the number is $(-1)^0 \times 2^{124-127} \times 1.25 = 0.15625$.

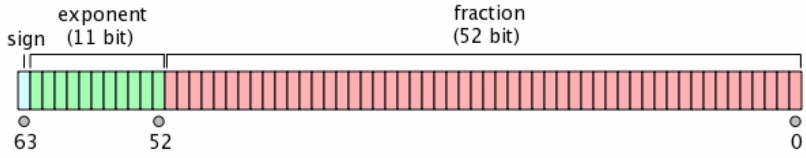


Figure 2: A 64-bit IEEE 754 (a.k.a. *double-precision*) floating-point number has 1 sign bit, 11 exponent bits, and 52 fraction (or significand) bits.

Remarks

(a) The precision of the binary formats is one greater than the width of its significand; this is because there is an implied (hidden) 1 bit at the start, so that

$$r = (-1)^s \left(1 + \sum_{i=1}^{p-1} f_i 2^{-i}\right) 2^{e-e_{max}} \,,$$

where f_i is the i^{th} bit of f.

(b) One of the two missing values in the exponent is used to represent zero (and *subnormal* numbers),[17] while the other value is used to represent infinity and undefined numbers.[18] For instance try the following in MATLAB

```
>>       1/0,        -1/0,        0/0
    ans = Inf, ans = -Inf, ans = NaN
```

(vi) Floating-point numbers lie at discrete points on the real number line. At higher values the absolute spacing between numbers increases, although the fractional change between adjacent numbers does not.

If a number cannot be represented exactly (e.g. 0.15625 can be represented exactly, but 0.001 cannot), the nearest floating-point number is used to represent it. The difference between the true value and this value is the round-off error (which is one type of *rounding error*).

When numbers get too small or large, *underflow* and *overflow* can occur. The IEEE 754 standard defines $-\infty$, ∞ and NaN, and decrees what should happen when things go wrong, e.g. 1/0 and 0/0.

[17] More precisely, for one of these two values, if the significand is zero then the binary represents zero, while if the significand is non-zero, then the binary represents some very small numbers with less than p digits of precision (so called *subnormal, or denormal, numbers*).

[18] More precisely, for the other reserved value of the exponent, if the significand is zero the binary represents infinity, while if the significand is non-zero the binary represents *Not a Number*, i.e. NaN.

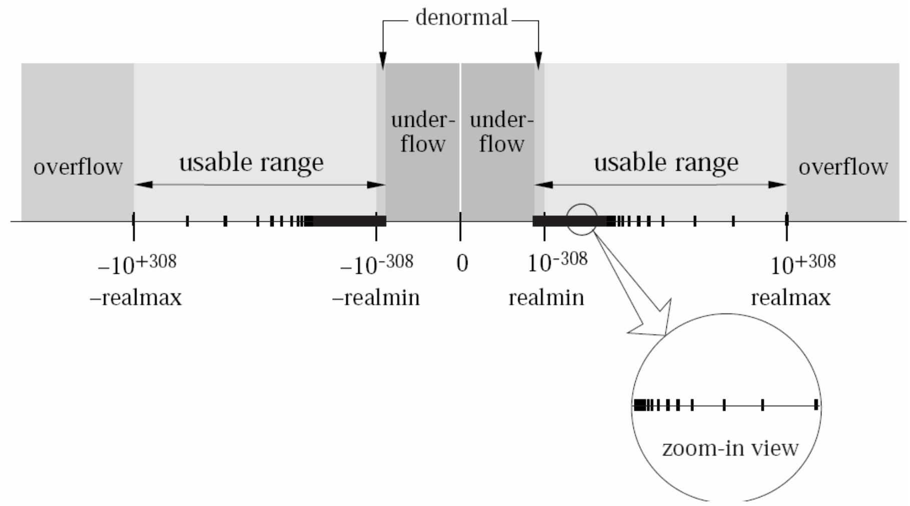


Figure 3: The floating-point number line

(vii) Floating-point numbers do not necessarily obey the usual rules of arithmetic such as associativity, e.g. it may be true that

$$\begin{aligned}(a+b)+c &\neq a+(b+c)\,,\\ (ab)c &\neq a(bc)\,.\end{aligned}$$

However, normally $a+b=b+a$ and $ab=ba$.

(viii) We will refer to the smallest number, ε, that when added to 1 (viewed as a real) that produces a real number different from 1, as the *machine accuracy* or *machine epsilon*. So

$$1+\varepsilon \neq 1\,, \tag{2.2a}$$

$$1+\tfrac{1}{2}\varepsilon = 1\,. \tag{2.2b}$$

ε is emphatically **not** the smallest number that can be represented (see point (ix) below). It is an **indication** of the number of significant figures.

With this definition ε is the spacing of floating-point numbers with zero exponent, i.e. $\varepsilon = b^{-(p-1)}$. In MATLAB it can be found as follows:

```
>> format long g; eps, 2^(-52)
   ans = 2.22044604925031e-16
   ans = 2.22044604925031e-16
```

`eps` with no arguments, or equivalently `eps('double')`, returns the result for double-precision numbers. By default MATLAB uses 'double-precision' numbers, giving about 16 significant figures. Also available are 'single-precision' numbers giving about 7 significant figures; `eps('single')` returns the machine accuracy for single precision numbers:

```
>> eps('single'), single(2)^(-23)
   ans = 1.192093e-07
   ans = 1.192093e-07
```

Roll it yourself. You can find an *approximation* to ε for double and single-precision numbers with the following MATLAB code:

```
>> a=1; while 1+a ~= 1, a=a/2; end; 2*a
   a = 2.22044604925031e-16
>> a=single(1); while 1+a ~= 1, a=a/2; end; 2*a
   a = 1.192093e-07
```

Note that

(a) a mathematical expression involving both single and double-precision numbers returns a single-precision number;

(b) in versions of MATLAB prior to R2012b, it was necessary execute the undocumented `'feature accel off'` before the single-precision statement because of an 'accelerator bug'.

Remark. The IEEE standard does not define the term *machine epsilon*. Above we use the MATLAB definition. Another [more] widely used definition, which we refer to as *unit round-off*, is the maximum relative error of the rounding procedure. Since machine epsilon is a bound for relative error, it suffices to consider numbers with exponent $e = 0$. It also suffices to consider positive numbers. For the usual round-to-nearest kind of rounding, the absolute rounding error is at most half the spacing, i.e. with this definition the *unit round-off* is $b^{-(p-1)}/2$, i.e. half MATLAB's definition.

(ix) As noted above, precision is different from size. To find the largest double-precision floating-point number representable with MATLAB try

```
>> realmax
   ans = 1.79769313486232e+308
```

Anything larger overflows:

```
>> (1+eps)*realmax
   ans = Inf
```

In single precision the largest floating-point number is smaller:

```
>> realmax('single')
   ans = 3.402823e+38
```

Similarly you can find the smallest positive *normalized* floating-point number (note that this is different from ε):

```
>> realmin
   ans = 2.2250738585072e-308
```

Remark. The *normalised* in the preceding sentence is important. You can get numbers smaller, but they have less precision (see also footnote 17). As an example suppose that we are working in decimal rather than binary, and we have five digits of mantissa and two of exponent. It is then possible to represent numbers like $1.2345 \times 10^{+12}$. The minimum positive value, with full precision achievable, with this format would be 1.0000×10^{-99}. However, you could represent smaller numbers with less precision, i.e. numbers as small as 0.0001×10^{-99}. Anything smaller would *underflow* and/or be set to zero. To see this in MATLAB try

```
>> realmin*eps/2, realmin*eps/(2-eps)
```

05/10

(x) Finally, having noted that [most] numbers cannot be stored accurately, let us return to integers (which can be stored exactly if they are small enough). To find the maximum and minimum integers that MATLAB can store exactly type

```
>> intmax
   ans = 2147483647
>> intmin
   ans =-2147483648
```

Now check:

```
>> 2^31-1, -2^31
   ans = 2147483647
   ans =-2147483648
```

Remarks.

(a) First note that 1, 2, ... are treated as reals, unless declared as integers. Try

```
>> m=10; n=int32(10); whos m n
   Name     Size     Bytes     Class     Attributes
   m        1x1          8     double
   n        1x1          4     int32
```

Remark. A byte [most commonly] consists of eight bits.

(b) Second, note that in MATLAB you need to be **careful** when manipulating integers near the maximum value. *Inter alia* this is because if the maximum integer `intmax` is exceed in a calculation it is replaced by `intmax` (i.e. `int32` integers saturate on overflow). Further, an expression containing both integers and reals (single or double), is converted to an integer at the end of the calculation. To see the consequences of these rules you might like to try:

```
>> 10*intmax-intmax
```

and compare with

```
>> 10*double(intmax)-double(intmax)
>> 10*double(intmax)-intmax
>> int32(10)*double(intmax)-double(intmax)
>> int16(10)*double(intmax)-double(intmax)
>> int8(10)*double(intmax)-double(intmax)
>> ...
```

For more explanation enter `'help int32'` within MATLAB, or search for `'Numeric Classes'` from within MATLAB help.

(c) There is also a `int64` class (but not all arithmetic operations are defined, see `'help int64'`). However, often it is sufficient to represent integers using double-precision floating-point numbers, which are exact between -2^{52} and 2^{52}. Try

```
>> f=@(n,m) (int64(2)^n+m)-double(int64(2)^n+m)
>> n=52, f(n,-1), f(n,0), f(n,1), f(n,2), f(n,3)
     52,      0 ,      0 ,     -1,       0 ,     -1
```

02/12

2.2 Types of 'Rounding Error'

Absolute and relative errors. If x is the actual number we are interested in, and x_c is the number actually stored in the computer, then

$$x - x_c \quad \text{is the } \textit{absolute error} \text{ of } x_c, \tag{2.3a}$$

$$\frac{x - x_c}{x} \quad \text{is the } \textit{relative error} \text{ of } x_c. \tag{2.3b}$$

Rounding error.

(a) $\frac{1}{3}$ might be stored in a computer as 0.333333.

(b) Precision can also be lost in calculation, e.g.

$$0.1234 \times 1.001 = 0.1235234 \tag{2.4a}$$

might be rounded to

$$0.1234 \times 1.001 = 0.123523 \tag{2.4b}$$

Cancellation error. Suppose that we know a number $x = 1.234567$ correct to 1×10^{-6}. Then in the following calculation an initial relative error of $O(10^{-6})$ ends up as a relative error of $O(10^{-1})$:

02/13 $$(1.234567 \pm 10^{-6}) - 1.23456 = 0.000007 \pm 10^{-6}\,.$$
02/14

Notation. All the above types of error are sometimes referred to as *rounding error* as a 'catch-all'.

Example: solution of a quadratic equation. We know that the exact solution to

$$ax^2 + bx + c = 0$$

is

$$2ax = -b \pm \sqrt{b^2 - 4ac}\,.$$

However, even with an exact formula, large rounding errors can creep in, e.g. if $b^2 \gg 4ac$. For instance the computational solution to

$$x^2 + 2000x + 1 = 0$$

is, with ε representing machine precision,

$$\begin{aligned} x &= 1000(-1 \pm \sqrt{1 - 10^{-6} + \varepsilon}) \\ &\approx 1000(-1 \pm (1 - \tfrac{1}{2}10^{-6} + \tfrac{1}{2}\varepsilon)) \\ &\approx \begin{cases} -2000(1 + \frac{1}{4}10^{-6} - \frac{1}{4}\varepsilon) \\ -500(10^{-6} - \varepsilon) \end{cases} . \end{aligned}$$

The second solution has a much larger relative error as a result of cancellation.

Example: Patriots and scuds. A [crude] guided missile system might need to calculate the velocity of the target from knowing the positions of the target, say x_1 and x_2, at times t_1 and t_2:

$$v = \frac{dx}{dt} \approx \frac{x_2 - x_1}{t_2 - t_1}\,.$$

There is the potential for an error here if, say, t_1 and t_2 are large, but the difference is small. On the [old] patriot missiles time was calculated in tenths of a second, with a maximum precision of 24 bits, i.e. with a relative error of

$$2^{-24} \approx (0.6)10^{-7}\,.$$

Hence after 100 hours, or $(3.6)10^6$ tenths of a second, the error was about $0.6(3.6)10^{6-7} = 0.2$ seconds[19]. Since a scud travels about $\frac{1}{3}$ km in this time, the patriots were not very accurate after 100 hours of up-time. The solution was to 'reboot' the patriots every 3 hours or so.

Example: Ariane. The maiden flight of the Ariane 5 launcher ended in a failure because of a software exception caused during a data conversion from 64-bit floating point to 16-bit signed integer. In particular the floating-point number had a value greater than that which could be represented by a 16-bit signed integer: e.g. see http://www.di.unito.it/~damiani/ariane5rep.html.

Example: solution of a difference relation. Rounding errors can arise in other, possibly surprising, ways. Consider the difference equation

$$a_{n+1} = a_{n-1} - a_n\,. \tag{2.5a}$$

The exact general solution to (2.5a) is

$$a_n = \alpha\left(\frac{\sqrt{5}-1}{2}\right)^n + \beta\left(\frac{-\sqrt{5}-1}{2}\right)^n . \tag{2.5b}$$

where α and β are constants. Suppose that

$$a_0 = 1 \quad \text{and} \quad a_1 = \tfrac{1}{2}(\sqrt{5} - 1)\,. \tag{2.5c}$$

[19] A slightly more refined argument shows that the error was 0.34 seconds: e.g. see

http://www.ima.umn.edu/~arnold/disasters/patriot.html.

then the exact solution has $\alpha = 1$ and $\beta = 0$, with the result that

$$a_n = \left(\frac{\sqrt{5}-1}{2}\right)^n < 1\,.$$

Suppose we calculate the solution to (2.5a) with initial condition (2.5c). To this end try the following MATLAB code, where we have replaced a_n by `a(n+1)` (because we cannot use a zero index in MATLAB); the solution calculated recursively using (2.5a) is stored in `a(n+1)`, and that calculated directly from (2.5b) is stored in `b(n+1)`.

```
>> clear a, b;
>> a(1)=1;
>> a(2)=(sqrt(5)-1)/2;
>> b(1:2)=a(1:2);
>> for n=2:80
     a(n+1)=a(n-1)-a(n);
     b(n+1)=b(n)*(sqrt(5)-1)/2;
   end;
>> a(1:n+1)-b(1:n+1)
```

If there was no rounding error, each element of `a(1:n+1)-b(1:n+1)` would be zero. However, because of rounding error, β in (2.5b) is not exactly zero, with the result that the second term of (2.5b) eventually magnifies sufficiently to dominate the solution.

Exercise: In the above code replace '`a(1)=1`' with '`a(1)=single(1)`'.

Example: numerical instability. Suppose we wish to compute the definite integral

$$\gamma_{n+1} = \int_0^1 x^n e^{x-1} dx\,, \tag{2.6a}$$

say for $n = 15$. Using integration by parts, we find that

$$\gamma_{n+1} = 1 - n\gamma_n\,, \quad n = 1, \dots\,, \tag{2.6b}$$

$$\gamma_{n+1} = \sum_{r=0}^{n-1} \frac{(-)^r n!}{(n-r)!} + (-)^n n!\gamma_1\,, \tag{2.6c}$$

$$\gamma_1 = 1 - e^{-1} = 0.632121\dots\,. \tag{2.6d}$$

Next, suppose we use the recursion relation (2.6b) to calculate γ_{16} starting with $\gamma_1 = 0.632121$ (i.e. with γ_1 truncated to six decimal places). We find that $\gamma_{16} =$ -576909, which is a significant error given that $0 < \gamma_{n+1} < 1$ for $n \geqslant 0$, and that the correct value is $\gamma_{16} = 0.0590175\dots$ (which it is straightforward to compute with, say, the trapezoidal rule or Simpson's rule).

The problem here is that the algorithm (2.6b), or equivalently (2.6c), is *unstable*. If we think of (2.6c) as an operator taking the data γ_1 to the solution γ_{n+1}, then it is an *unstable* scheme: a perturbation ε in the data leads to an error of $O(n!\varepsilon)$ in the solution. So for our specific example a perturbation of the data of less than 10^{-6} leads to a change in the solution of nearly 6×10^5.

Remarks

(a) The rounding error destroys the calculation, but the underlying problem is not the rounding error, but the *unstable* algorithm. In a numerical calculation making the right choice of algorithm can be key.

(b) The above difficulties can be illustrated with the following MATLAB code, where `integral` is a MATLAB function to evaluate integrals using adaptive quadrature (see Part IB *Numerical Analysis* and/or `help integral`); e.g. try `diffquad(15,6)`.

```
function [ ] = diffquad(n,p)
%diffquad(n,p)
%
% Round initial condition to p decimal places
%
format compact; format long g
%
gamma(1)=round((1-exp(-1))*10^p)/10^p;
for m=1:n
    gamma(m+1)=1-m*gamma(m);
end
%
fquad = @(x,n) x.^n.*exp(x-1);
tol=10*eps;
recur=gamma(n+1)
integ=integral(@(x)fquad(x,n),0,1,'AbsTol',tol,'RelTol',tol)
error=gamma(n+1)-integ
return
```

2.3 Truncation Error

Truncation error occurs when an infinite process is truncated. For example:

- truncation of the series for the exponential to 5 terms,

$$\exp(x) \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!}\,,$$

  or more generally to N terms

$$\exp(x) \approx \sum_{r=0}^{N-1} \frac{x^r}{r!}\,;$$

- a *finite difference* approximation of an infinitesimal

$$\frac{df}{dx}(x) \approx \frac{f(x+h) - f(x)}{h}\,.$$

3 Solution of Transcendental Equations

3.0 Introduction

Many of you will have already seen the underlying mathematics herein. The aim of this section is to review the mathematics, emphasise accuracy and convergence, and indicate how to formulate appropriate MATLAB code for use in the *Introductory Project*. Then the aim of the *Introductory Project* is to hone your programing and writing up skills (not to teach you new mathematics).

3.1 The Basics

Given a *continuous* function $f(x)$, suppose that we wish to solve the equation

$$f(x) = 0\,. \tag{3.1}$$

Particularly where f and x are multi-dimensional, this is a common requirement (although we will work in $\mathbb{R}^1$). A typical approach is *iterative*. For instance, given x_0 use a rule

$$x_{n+1} = g_n(x_0, x_1, \ldots, x_n) \tag{3.2a}$$

such that as $n \to \infty$

$$x_n \to x^* \quad \text{where} \quad f(x^*) = 0\,. \tag{3.2b}$$

It is often preferable that only a small number of previous estimates are used, e.g.

$$x_{n+1} = g_n(x_{n-1}, x_n)\,. \tag{3.2c}$$

03/11 For such a method to work we need a *rule* and a *termination criterion*.

3.2 Termination Criteria

We want to stop iterating when the error in our estimate of the root is small enough, i.e. when $|x_n - x^*| < \eta$, for some small η. The problem is that we do not know x^* (since it is what we are trying to find), so we cannot use this criterion.

We need to find a good termination criterion since, at best, a badly posed condition can require needless iterations but, at worst, it may lose the root. Possible termination criterion include,

$$|x_n - x_{n-1}| < \delta \tag{3.3a}$$

and

$$|f(x_n)| < \delta\,. \tag{3.3b}$$

for some small δ. While neither of these termination criterion require knowledge of the answer x^*, neither of these termination criterion imply that $|x_n - x^*| < \delta$.

3.3 Bisection, Interval Halving or Binary Search

While x is allowed to be a vector in (3.1), for simplicity we will assume that x is a scalar, i.e. we will restrict to one dimension.

Suppose that we wish to solve equation (3.1). Suppose also that we know points a_0 and b_0 such that $f(a_0)$ and $f(b_0)$ have opposite signs, i.e. suppose that

$$f(a_0)\,f(b_0) < 0\,.$$

This then implies that $f(x)$ has at least one root in the interval $[a_0, b_0]$ (where we assume wlog that $a_0 < b_0$).

To use the method of bisection to find a root, let

$$c = \tfrac{1}{2}(a_0 + b_0)\,.$$

If $f(c)$ has the same sign as $f(b_0)$, then we know that there is a root in $[a_0, c]$; otherwise there is a root in $[c, b_0]$. We have therefore halved the size of the interval containing the root. The *rule* we adopt is thus (initially with $n = 0$)

$$\begin{array}{rlll} \text{if} & f(b_n)f(c) > 0 & \text{then let} & a_{n+1} = a_n, \quad b_{n+1} = c\,, \\ \text{else if} & f(b_n)f(c) \leqslant 0 & \text{then let} & a_{n+1} = c\ , \quad b_{n+1} = b_n\,. \end{array}$$

We now repeat this process to deduce that

$$a_0 \leqslant a_1 \leqslant \ldots \leqslant a_n \leqslant x^* \leqslant b_n \leqslant \ldots \leqslant b_1 \leqslant b_0\,,$$

and that

$$|b_n - a_n| = \frac{|b_0 - a_0|}{2^n}\,.$$

Suppose that we wish to find the root x^* with a maximum error of $\eta > 0$. The root is no more than $\frac{1}{2}|b_n - a_n|$ from the mid-point of the interval $[a_n, b_n]$. Hence we can guarantee that

$$|\tfrac{1}{2}(a_n + b_n) - x^*| < \eta\,, \tag{3.4a}$$

if

$$\tfrac{1}{2}|b_n - a_n| = \frac{|b_0 - a_0|}{2^{n+1}} < \eta\,, \tag{3.4b}$$

i.e. if

$$n > \frac{\ln(|b_0 - a_0|/\eta)}{\ln 2} - 1\,. \tag{3.4c}$$

Convergence is guaranteed because we always bracket the root.

Remarks.

(i) For an error of $\eta > 0$ the complexity is proportional to $\ln(1/\eta)$, i.e. the complexity is logarithmic.

(ii) The *truncation error* is the size of the interval when we stop the iteration. Note that if we keep iterating, eventually

$$|b_n - a_n| = \varepsilon = \text{machine precision}\,,$$

at which point there can be no further improvement. In many cases a reasonable choice of η is thus

$$\eta = \varepsilon \max(|a_n|, |b_n|, 1)\,.$$

(iii) Note that the termination criterion (3.4b) does **not** guarantee that

$$\left|f(\tfrac{1}{2}(a_n + b_n))\right| < \eta\,.$$

Although we have found the root to within a specified accuracy, we cannot be sure that the value of f at the estimated root is equally small. In fact

$$\begin{aligned} f(\tfrac{1}{2}(a_n + b_n)) &= f(x^*) + \left(\tfrac{1}{2}(a_n + b_n) - x^*\right) f'(x^*) + O(\eta^2) \\ &= \left(\tfrac{1}{2}(a_n + b_n) - x^*\right) f'(x^*) + O(\eta^2) \end{aligned}$$

and hence

$$\begin{aligned} \left|f(\tfrac{1}{2}(a_n + b_n))\right| &\lessapprox \left|\tfrac{1}{2}(a_n + b_n) - x^*\right| |f'(x^*)| \\ &\lessapprox \eta|f'(x^*)|\,. \end{aligned}$$

It follows that f at the computed 'root' might be appreciable if $|f'(x^*)|$ is large.

Initial guesses. We can often get a good first guess by plotting the function, which is relatively easy in MATLAB. For instance suppose that we are interested in the function

$$f(x) = x^3 - 2x^2 - x + 2\,,$$

then $f(x)$ can be plotted with either of the following MATLAB commands:

```
>> fplot('x^3-2*x^2-x+2',[-2,3])
```

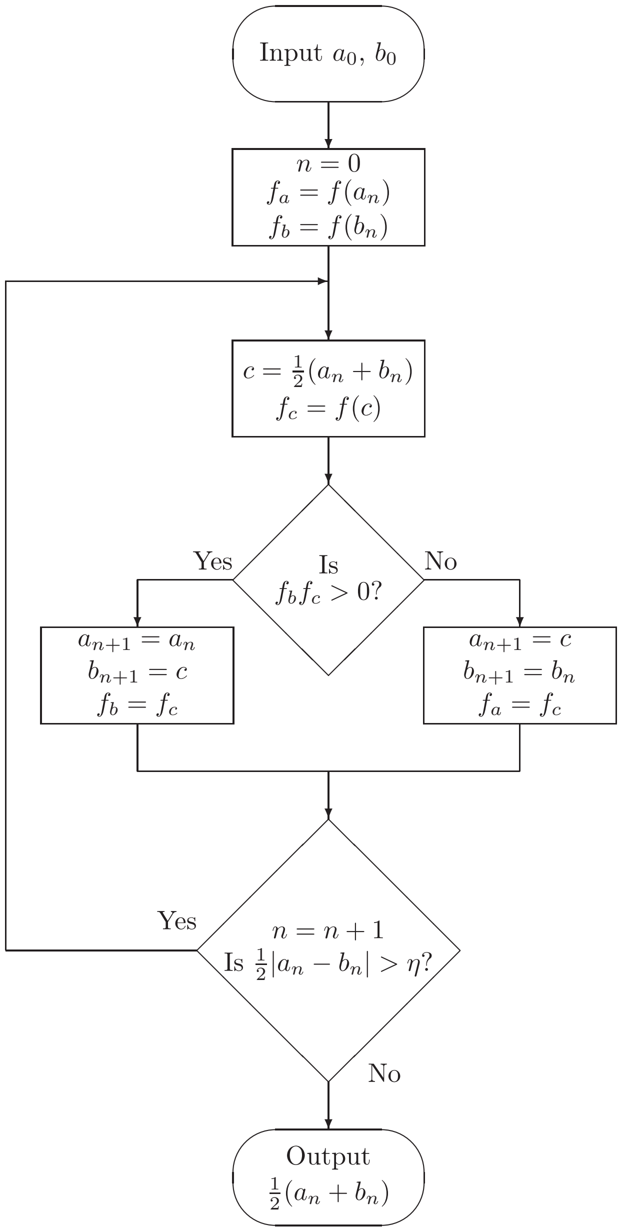
or

```
>> x=-1.5:.01:2.5; y=x.^3-2*x.^2-x+2; plot(x,y)
```

However, there are many other ways of plotting the same thing (or nearly the same thing):

```
>> fplot(@(x) x^3-2*x^2-x+2, [-2,3])
>> fplot(@(y) [y^3-2*y^2-y+2,0], [-2,3])
>> x=-2:.01:3; y=x.^3-2*x.^2-x+2; z=zeros(size(x));
plot(x,y,'r',x,z,'g')
>> x=-2:.1:3; y=x.^3-2*x.^2-x+2; plot(x,y,'r--',x,0,'b:')
>> f=inline('x^3-2*x^2-x+2'); fplot(f, [-2,3]); hold;
fplot('0', [-2,3])
```

06/10
03/14
Flowchart. In order to program an algorithm it sometimes helps to construct a flowchart.



Program. We can implement the bisection method on the basis of the flowchart, e.g. the following MATLAB code should find the root of $x^3 - 2x^2 - x + 2 = 0$:

```
>> f=inline('x^3-2*x^2-x+2');
>> a=0; fa=f(a);
>> b=1.5; fb=f(b);
>> while abs(a - b)/2 > eps*max([abs(a) abs(b) 1])
     c=(a+b)/2; fc=f(c);
     if (fb > 0) == (fc > 0)
       b=c; fb=fc;
     else
       a=c; fa=fc;
     end
   end
>> root=(a+b)/2
```

Remark. There is one function evaluation each iteration.

Check. MATLAB has a built-in function to find zeros (the code for which can be listed with '`type fzero`'). Try

```
>> f=inline('x^3-2*x^2-x+2'); fzero(f,0)
```

or

```
>> fzero(f,1/3,optimset('TolX',1e-3))
>> fzero(@(x) x^3-2*x^2-x+2,[0,1.5])
>> fzero('x^3-2*x^2-x+2',[1.5 2.5])
```

Exercise. Bisection is not perfect: try

03/12

```
>> fzero('x^2',0.001)
```

03/13

3.3.1 Recursion

Recursion is a method where the solution to a problem depends on solutions to smaller instances of the same problem. This approach can be applied to many types of problems, and is a central idea of computer science.

Functions can be defined recursively. For instance for $n \in \mathbb{N}_0$ we might define 2^n as follows

$$2^n = \begin{cases} 1 & \text{if } n = 0 \\ 2 \times 2^{n-1} & \text{if } n \geqslant 1 \end{cases}.$$

Similarly a MATLAB function can be defined recursively. For instance the following MATLAB function calculates 2^n for $n \in \mathbb{N}_0$:

```
function [ y ] =twoexp(n)
% y=twoexp(n). This is a recursive program for computing y=2^n.
%
% First, some sanity checks.
%
if n < 0
  error('n should be positive.')
elseif n ~= round(n)
  error('n should be a positive integer.')
end
%
if n==0
  y=1;
else
  y=2*twoexp(n-1);
end
```

A binary search is a recursive operation. The following MATLAB code uses the bisection method to find a zero of a function `fun` to an accuracy `tol`; the root is to be in an interval [`xlow`, `xhigh`].

```
function [ xroot ] = binarysearch(fun, xlow, xhigh, tol)
% fun is a function
% The root lies between xlow and xhigh
% Iterate until have calculated the root to a tolerance tol
xmid = (xhigh + xlow)/2;
if ( xhigh-xlow < 2*tol )
  xroot=xmid;
  return
elseif (feval(fun,xmid)*feval(fun,xhigh) > 0)
  xroot=binarysearch(fun, xlow, xmid, tol);
else
  xroot=binarysearch(fun, xmid, xhigh, tol);
end
```

To pass a function (as opposed to a variable) to a function we need (as before) a `function handle`:

```
f=@(x) x^3-2*x^2-x+2
```

or

```
f=inline('x^3-2*x^2-x+2')
```

We can now find a root to $f(x) = x^3 - 2x^2 - x + 2$ by, say,

```
binarysearch(f, 0, 1.5, eps)
binarysearch(f,-3, 0  , 10^-3)
```

Remark. The recursive code is in some respects more elegant than the code on page 26; however the recursive code is twice as expensive since it requires two function evaluations per iteration.

3.4 Order of Convergence

We may well be interested in how fast we can obtain an answer; the *order of convergence* is a measure of this. Let

$$\delta_n = x_n - x^* \tag{3.5a}$$

then it is 'often' possible to show, for sufficiently small δ_n, that

$$|\delta_{n+1}| \leqslant c|\delta_n|^p\,, \tag{3.5b}$$

for some positive constants c and p. The smaller c and, more importantly, the larger p, the faster the convergence.

Notation.

The *order of the method* is said to be p.

The *asymptotic error constant* is said to be c.

Speed of convergence. It follows that

$$|\delta_n| \leqslant \begin{cases} c^n|\delta_0| & \text{if } p = 1 \text{ (require } c < 1 \text{ for convergence)}\,, \\ c^{\frac{1-p^n}{1-p}}|\delta_0|^{p^n} & \text{if } p > 1\,. \end{cases} \tag{3.6a}$$

Hence, in order to achieve an accuracy $|\delta_n| < \varepsilon < 1$, the number of iterations scales as

$$n \geqslant \begin{cases} \frac{\log(\varepsilon/|\delta_0|)}{\log c} & \text{if } p = 1,\ 0 < c < 1\,, \\ \\ \frac{1}{\log p}\log\left(\frac{\log\epsilon}{\log\left(c^{\frac{1}{p-1}}|\delta_0|\right)}\right) & \text{if } p > 1,\ n \gg 1\,. \end{cases} \tag{3.6b}$$

Bisection method. For the bisection method

$$|\delta_{n+1}| \leqslant \tfrac{1}{2}|\delta_n|\,,$$

and hence $p = 1$ and $c = \frac{1}{2}$. Since $p = 1$, the bisection method is said to be a *first-order method*, and the method has *linear convergence*. The maximum number of iterations required to improve the accuracy of the root by $O(10^{-q})$ is

04/11

$$n = O\left(\frac{\ln 10}{\ln 2}\,q\right).$$

3.5 Other Methods

There are many other methods of finding roots, including

(a) fixed-point (or Picard) iteration in which the equation $f(x) = 0$ is 're-written' in one of the *non-unique* forms, $x = g(x)$, followed by the iteration scheme

$$x_{n+1} = g\left(x_n\right)\,; \tag{3.7a}$$

(b) Newton-Raphson iteration (which is a special case of fixed-point iteration), which uses the scheme

$$x_{n+1} = x_n - \frac{f\left(x_n\right)}{f'\left(x_n\right)}\,; \tag{3.7b}$$

(c) the secant method, which uses the scheme

$$x_{n+1} = x_n - \left(\frac{x_n - x_{n-1}}{f\left(x_n\right) - f\left(x_{n-1}\right)}\right) f(x_n)\,; \tag{3.7c}$$

For (3.7a) and (3.7b) a suitable initial guess x_0 is required; for (3.7c) two suitable initial values x_0 and x_1 are required.

Order of Convergence. It can be shown that, when convergent,

(a) fixed-point iteration is *in general* first-order convergent for a simple root, i.e. when $f'(x_*) \neq 0$;

(b) Newton-Raphson iteration is second-order convergent for a simple root, but only first-order convergent for a multiple root;

(c) the secant method *in general* has an rate of convergence given by $p = \frac{1}{2}\left(1 + \sqrt{5}\right)$, but that if $f'(x_*) = 0$ then the rate of convergence degrades to linear.

Remarks.

(i) The theoretical background to these methods is covered in most textbooks on *Numerical Analysis.*

(ii) The *Introductory Project* is on *Root Finding in One Dimension*, and explores [some of] the above methods in more detail.

Exercise. In the bisection method, instead of using

$$c = \tfrac{1}{2}(a_n + b_n) \tag{3.8a}$$

as the point of bisection, assume that the function is approximately a straight line over the interval $[a_n, b_n]$. If the assumption is correct, the root is likely to be near (cf. (3.7c))

$$c = b_n - \left(\frac{b_n - a_n}{f\left(b_n\right) - f\left(a_n\right)}\right) f(b_n)\,. \tag{3.8b}$$

Verify that $c \in [a_n, b_n]$. Investigate whether this results in a reduced number of iterations if this value of c is used.

4 Solution of Ordinary Differential Equations

4.0 Introduction

Differential equations can be used to model many phenomena ranging from climate change (and other physical processes), through epidemiology (and other biological processes) to derivatives (and other financial products). As elsewhere in these notes we shall start by 'toddling', and leave walking and running to other courses. What follows, and in particular the error analysis, is directly relevant to the first Part IB Computational Project.

We will consider the simplest method for the numerical solution of an ordinary differential equation (ODE). In particular, for some known function f and initial conditions $y(a)$, we aim to solve the first-order ODE

$$y' \equiv \frac{dy}{dx} = f(x, y)\,, \tag{4.1}$$

for the value of $y(b)$, where $b > a$ wlog.

4.1 The Euler Method

The interval $[a, b]$ contains an uncountable number of points, hence we first divide it into N sub-intervals

$$x_0 = a, x_1, \ldots, x_{N-1}, x_N = b\,, \tag{4.2a}$$

where $h_n = x_n - x_{n-1}$ is the n^{th} *step size*. The sub-intervals need not be equi-spaced, but initially we will assume that each is of width

$$h = \frac{b-a}{N}\,, \quad \text{whence} \quad x_n = a + nh\,, \quad n = 0, 1, \ldots, N. \tag{4.2b}$$

Let Y_n be an approximation to the exact solution $y_n = y(x_n)$, so that

Position	**Exact Solution**	**Approximate Solution**
$x_0 = a$	$y_0 = y(a)$	$Y_0 = y_0$
$x_1 = a + h$	y_1	Y_1
$\vdots$	$\vdots$	$\vdots$
$x_n = a + nh$	y_n	Y_n
$\vdots$	$\vdots$	$\vdots$
$x_N = a + Nh = b$	y_N	Y_N

To solve the equation (4.1) using the Euler method we assume that $\frac{dy}{dx}$ is constant over each step, in particular we assume that in $[x_0, x_1)$ that

$$\frac{dy}{dx} \approx f(x_0, y_0)\,.$$

Thence, from geometry, we let

$$Y_1 = Y_0 + hf(x_0, Y_0) = y_0 + hf(x_0, y_0)\,.$$

Next we assume that in $[x_1, x_2)$ that

$$\frac{dy}{dx} \approx f(x_1, Y_1)\,,$$

and thence, from geometry, we let

$$Y_2 = Y_1 + hf(x_1, Y_1)\,.$$

Finally we recurse:

$$Y_{n+1} = Y_n + hf(x_n, Y_n)\,. \tag{4.3}$$

Remarks.

(i) We approximate $\frac{dy}{dx}$ by $f(x_n, Y_n)$ rather than $f(x_n, y_n)$, because we do not know y_n.

(ii) y could be a [column] vector.

(iii) The step size h need not be constant.

4.1.1 Example MATLAB code

Suppose as a test we consider

$$f(x, y) = xy^2\,. \tag{4.4a}$$

The exact solution to (4.1) is then

$$y = \frac{2}{c - x^2}\,, \tag{4.4b}$$

where c is a constant. Note that $y(0) = 2/c$, and that the solution has singularities at $x = \pm\sqrt{c}$.

The following MATLAB code implements n steps of the core of the above algorithm. It integrates from $x = 0$ to $x = 1$ with a constant step size $h = 0.05$ (so 20 steps are required), for an initial condition $y(0) = 1$ (in which case the exact solution has $c = 2$).

```
>> x=0;
>> y=1;
>> h=0.05;
>> n=round(1/h);
>> for i=1:n
     y=y+h*x*y^2;
     x=x+h;
   end
>> x, y
   x=1.0000, y=1.8342
```

Remarks.

(i) Note that in the `for` loop it is important to update `y` before updating `x`.

(ii) The exact solution at $x = 1$ is $y(1) = 2$; hence the error is -0.1658.

A natural question to ask is how the error depends on the step size h. To investigate this we bundle the above code into a function, `eulergraph`, that takes the initial conditions, range and step size as input, and outputs the solution at the calculated grid points in arrays `x` and `y`:

```
function [x, y] = eulergraph(xstart,ystart,xend,h)
% [x, y] = eulergraph(xstart,ystart,xend,h)
%
x(1)=xstart;
y(1)=ystart;
n=round((xend-xstart+eps)/h);
for i=1:n
  y(i+1)=y(i)+h*x(i)*y(i)^2;
  x(i+1)=x(i)+h;
end
return
```

We can then run this function for successively halved step sizes and plot the results.

```
% Set initial conditions and range
%
xstart=0;  ystart=1; if ~ exist('xend','var') xend=1.0; end
%
% Initialise parameters and plot
%
jmax=5; clf; hold on;
for j=1:jmax
    %
    % Choose step size, call eulergraph, plot solution & inverse
    %
    h=0.2/2^(j-1);
    [x,y]=eulergraph(xstart,ystart,xend,h);z=1./y;
    k=(j-1)/jmax; plot(x,y,x,z,'Color',[k/2 1-k k],'LineWidth',2);
end
%
% Plot asymptote, tidy up axes and print graph
%
xasymp=sqrt(2/ystart); ylower=ystart; yupper=(2*ystart)/abs(2-ystart*xend^2);
if xasymp <= xend
    ylower=0; plot([xasymp xasymp],[ylower yupper]);
end
axis([xstart xend ylower yupper]); xlabel('x'); ylabel('y');
hold off; print -depsc eulergraph
```

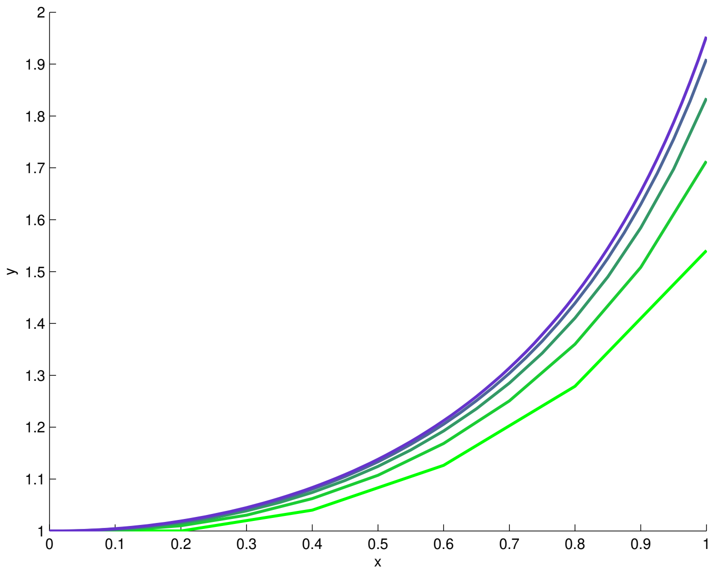


Figure 4: Solutions to (4.1) and (4.4a) using Euler's method, with $x_0 = 0$, $y_0 = 1$ and $h = 0.2$, 0.1, 0.05, 0.025 and 0.0125.

Remarks

(i) As might be expected, decreasing the step size h increases the accuracy. However, ideally this improvement should be *quantified*.[20]

(ii) By setting `xend=1.5` we see that it is possible to integrate [inadvertently] through a singularity; however refinement of the step size suggests that the solution near $x = 1.5$ is problematic. Matters can be made clearer by plotting $1/y$ (which is bounded) rather than y.

[20] As will be required in the Computational Projects.

4.2 Accuracy

It would be good to have a more formal measure of the errors associated with the Euler method and other methods.

Global error. Define the *global error* after n steps to be

$$E_n = Y_n - y_n\,, \tag{4.5a}$$

where we recall that y_n is the exact solution starting from $x = x_0$ and $y = y_0$.

Local error. First define $z_n = z(x_n)$ to be the *exact* solution to (4.1) at $x = x_n$ starting from $x = x_{n-1}$ and $y = Y_{n-1}$. In general $z_n \neq Y_n$ and $z_n \neq y_n$. Define the *local error* after step n to be

$$e_n = Y_n - z_n\,. \tag{4.5b}$$

Local truncation error. In the case of infinite precision arithmetic, so that the local error is solely due to truncation error, then a numerical method is said to be of *order p*, if as $h \to 0$

04/12

$$e_n \approx c\,h^{p+1}\,. \tag{4.5c}$$

Arm waving. If we take n steps, then from (4.5c) the cumulative local error is [likely to be] $O(nh^{p+1})$. So after we have taken $N = (b-a)/h$ steps, say, in order to calculate an approximation to $y(b)$, we might *hope* that, on the assumption of no rounding errors,

$$\text{global truncation error at } b = E_N = O(Nh^{p+1}) = O(h^p)\,. \tag{4.6}$$

This is an *arm-waving* estimate of the global truncation error. However, the argument can [some-
04/13 times] be tightened up with suitable assumptions: see (4.9) below.

Remark. Since we want the error to be small as $h \to 0$, a numerical method is as useful as a chocolate teapot unless $p > 0$.

4.2.1 Local truncation error of the Euler method

From Taylor's theorem

$$z(x_n) = z(x_{n-1}) + hz'(x_{n-1}) + \tfrac{1}{2}h^2 z''(\xi_{n-1})\,, \tag{4.7a}$$

for some $\xi_{n-1} \in [x_{n-1}, x_n]$. From our definitions of the Euler method (4.3) and the definition of local error (4.5b), then on the assumption of no rounding errors,

$$\begin{aligned}
e_n &= Y_n - z_n \\
&= Y_{n-1} + hf(x_{n-1}, Y_{n-1}) - (z(x_{n-1}) + hz'(x_{n-1}) + \tfrac{1}{2}h^2 z''(\xi_{n-1})) \\
&= Y_{n-1} + hf(x_{n-1}, Y_{n-1}) - Y_{n-1} - hf(x_{n-1}, Y_{n-1}) - \tfrac{1}{2}h^2 z''(\xi_{n-1}) \\
&= -\tfrac{1}{2}h^2 z''(\xi_{n-1}))\,.
\end{aligned} \tag{4.7b}$$

Hence from (4.5c) the Euler method has a local truncation error of order one.

4.2.2 Global truncation error of the Euler method (*Unlectured*)

Executive summary. Given suitable restrictions/conditions, the Euler method has a global truncation error of order one.

In order to calculate the global truncation error we need to know something about the function f. We suppose that f is bounded and continuous (as already implicitly assumed) and that it satisfies a *Lipschitz condition*:

$$|f(x,y) - f(x,z)| < L|y-z|\,, \tag{4.8}$$

for all x, y and z and for some positive constant L.

Then from (4.5a) and (4.3)

$$\begin{aligned} E_n &= Y_n - y_n \\ &= Y_{n-1} + hf(x_{n-1}, Y_{n-1}) - (y(x_{n-1}) + hy'(x_{n-1}) + \tfrac{1}{2}h^2 y''(\xi_{n-1})) \\ &= Y_{n-1} - y_{n-1} + h(f(x_{n-1}, Y_{n-1}) - f(x_{n-1}, y_{n-1})) - \tfrac{1}{2}h^2 y''(\xi_{n-1}) \\ &= E_{n-1} + h(f(x_{n-1}, Y_{n-1}) - f(x_{n-1}, y_{n-1})) - \tfrac{1}{2}h^2 y''(\xi_{n-1})\,. \end{aligned}$$

Further, from the Lipschitz condition (4.8)

$$|f(x_{n-1}, Y_{n-1}) - f(x_{n-1}, y_{n-1})| < L|Y_{n-1} - y_{n-1}|\,.$$

We will assume that the second derivative is bounded, i.e. for some positive constant σ and for all x

$$|y''(x)| < \sigma\,.$$

Then it follows from the triangle inequality, and summing a geometric progression, that

$$\begin{aligned} |E_n| &\leqslant |E_{n-1}| + hL|Y_{n-1} - y_{n-1}| + \tfrac{1}{2}h^2\sigma \\ &\leqslant (1+hL)|E_{n-1}| + \tfrac{1}{2}h^2\sigma \\ &\leqslant (1+hL)^{r+1}|E_{n-1-r}| + \tfrac{1}{2}h^2\sigma \sum_{j=0}^{r}(1+hL)^j \\ &\leqslant (1+hL)^n|E_0| + \tfrac{1}{2}h^2\sigma \sum_{j=0}^{n-1}(1+hL)^j \\ &\leqslant \frac{h\sigma}{2L}((1+hL)^n - 1)\,. \end{aligned}$$

For $h > 0$ it is true that $0 < 1 + hL < e^{hL}$, and hence that $(1+hL)^n < e^{nhL}$; further $nh = (x_n - x_0)$. Hence

$$\begin{aligned} |E_n| &\leqslant \frac{h\sigma}{2L}\left(e^{nhL} - 1\right) \\ &\leqslant \frac{h\sigma}{2L}\left(\mathrm{e}^{(x_n - x_0)L} - 1\right)\,. \end{aligned} \tag{4.9}$$

05/11 We conclude that the Euler method has a global truncation error of order one.

4.2.3 Testing accuracy without the exact solution, and a free lunch!

For our example equation (4.4a) we know the exact solution (4.4b); it is then straightforward to test the accuracy of the solution. However, suppose that we do not know the exact solution, but wish to test accuracy; what do we do then?

One way forward is to postulate that the global truncation error is $O(h^p)$, cf. (4.6), or more specifically that

$$y_h = y^* + \lambda h^p + O(h^{p+1})\,, \tag{4.10}$$

where y_h is the numerical solution obtained with a step size of h, y^* is the exact solution, and λ is a constant. Then,

$$y_h - y_{\frac{1}{2}h} = \left(1 - \tfrac{1}{2^p}\right)\lambda h^p + O(h^{p+1})\,, \tag{4.11a}$$

and

$$\log|y_h - y_{\frac{1}{2}h}| = \log\left(1 - \tfrac{1}{2^p}\right)\lambda + p\log h + \ldots\,. \tag{4.11b}$$

This suggests that the slope of a plot of $\log|y_h - y_{\frac{1}{2}h}|$ against $\log h$ should yield the order p.

Remark. If writing a numerical ODE solver or, probably more pertinently, a numerical PDE solver, it is best to theoretically calculate the order of the method, and then to confirm that the code is consistent with the theoretical order (e.g. by means of the aforementioned plot, or similar).

Further, if we know the order p then there is a free lunch! First, we note from that (4.11a)

$$\lambda h^p = \frac{2^p}{2^p-1}\left(y_h - y_{\frac{1}{2}h}\right) + O(h^{p+1})\,. \tag{4.12a}$$

Hence from (4.10) we conclude that

$$y^* = y_R + O(h^{p+1})\,, \tag{4.12b}$$

where

$$y_R = \frac{1}{2^p-1}\left(2^p y_{\frac{1}{2}h} - y_h\right)\,, \tag{4.12c}$$

is a better $O(h^{p+1})$ approximation to the solution. For the Euler method, $p=1$ from (4.9), hence a better $O(h^2)$ approximation to the solution should be

$$y_R = 2y_{\frac{1}{2}h} - y_h\,. \tag{4.12d}$$

Remark. Even higher order solutions can be obtained by similar extrapolation if three or more solutions with different step sizes are known, e.g. y_h, $y_{\frac{1}{2}h}$ and $y_{\frac{1}{4}h}$.

4.2.4 Numerical confirmation of accuracy

In order to examine the error of our model problem more systematically, we modify the `eulergraph` function to output only the solution at the end of the range:

```
function [x, y] = eulerverify(xstart,ystart,xend,h)
% [x, y] = eulerverify(xstart,ystart,xend,h)
%
% Integrate y'=x*y^2 with step h from xstart to xend with y(xstart)=ystart
x=xstart; y=ystart; n=round((xend-xstart+eps)/h);
for i=1:n
  y=y+h*x*y^2;
  x=x+h;
end
return
```

The code in the Annex on page 38 runs the function `eulerverify` for a range of successively halved step sizes, where the step sizes and values of x and y output by `eulerverify` are stored in arrays `h`, `x` and `y` respectively. The error at the end of the integration range, as calculated using the exact solution, is then
07/10 plotted on a `loglog` plot.

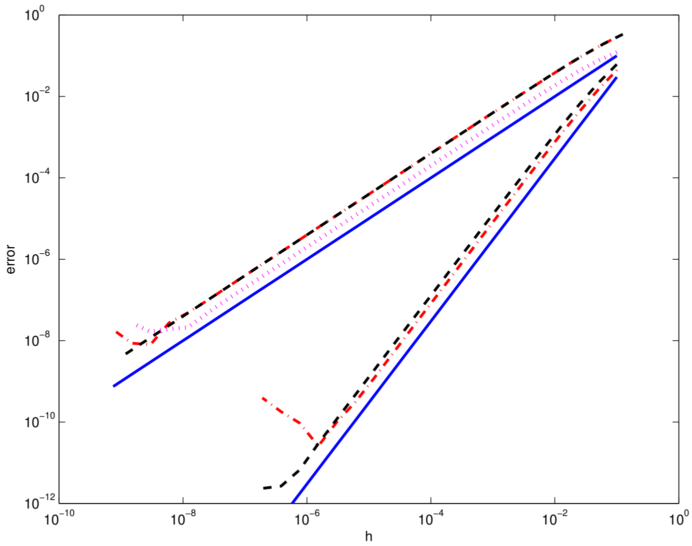


Figure 5: Error in y at the end of range plotted against step size. Upper '$- \cdot$' line: $h = 2^{-m}/10$; upper '$- \; -$' line: $h = 2^{-m}$. Dotted line is estimate of the error. Lower lines are the extrapolated second-order solutions using (4.12d). Solid lines are lines of slope 1 and 2.

Verifying the order. For Euler's method $E_N \propto h^p$ with $p = 1$ (see (4.9)), and hence we expect

$$\log |E| \propto \log |h| \,.$$

The reference line of slope one in the `loglog` plot is consistent with $p = 1$.

Estimating the order. Earlier we noted that it should be possible to use the slope of a plot of $\log |y_h - y_{\frac{1}{2}h}|$ against $\log\, h$ to estimate the order p without knowing the exact solution. The dotted line in figure 5 is a plot of $\log |y_h - y_{\frac{1}{2}h}|$ and is consistent with $p = 1$.

Rounding error. Note that if h is too small rounding error can be expected to become important. Suppose that our method has an $O(h^p)$ truncation error, and suppose that in the **worst** case the systematic rounding error accumulated over $1/h$ steps is $O(\texttt{eps}/h)$. Then

$$\text{total error} = O(h^p) + O(\texttt{eps}/h) \,.$$

This is minimised for

$$h = O((\texttt{eps})^{\frac{1}{1+p}}) \,,$$

so yielding

$$\text{minimum total error} = O((\texttt{eps})^{\frac{p}{1+p}}) \,. \tag{4.13a}$$

Remarks.

(a) If $p = 1$, as for the Euler method, the total error is minimised for $h = O((\texttt{eps})^{\frac{1}{2}})$, in which case

$$\text{minimum total error} = O((\texttt{eps})^{\frac{1}{2}}) \,. \tag{4.13b}$$

This is consistent with figure 5.

(b) The minimum total error decreases with increasing p. This makes *higher-order* methods, i.e. methods with larger p, more attractive than the Euler method. For instance, if $p = 4$ then

$$\text{minimum total error} = O((\texttt{eps})^{\frac{4}{5}}) \,. \tag{4.13c}$$

(c) The systematic rounding error accumulated over $1/h$ steps may not be as bad as the worst case estimate of $O(\texttt{eps}/h)$. One might hope that the rounding error accumulated over a number of steps cancels to some extent because of randomness, so that the accumulated error is $O(\texttt{eps}/h^{\frac{1}{2}})$ (cf. diffusion), which would result in

$$\text{minimum total error} = O((\texttt{eps})^{\frac{2p}{1+2p}})\,. \tag{4.13d}$$

Figure 6 illustrates an Euler solution for an integration where the step size h is adjusted at each step by $r\,h/10$ where r is a pseudo-random number in $(0,1)$. The step size at which the truncation error balances the rounding error is reduced, although the minimum total error is slightly larger than (4.13d) with $p = 1$. A similar reduction in the rounding error can also be achieved by choosing a step-size, 2^{-m}, that can be represented exactly in binary (see also figure 6).

Richardson extrapolation. That it is possible to extrapolate to a second-order solution using (4.12d) is illustrated both in figure 5 and in figure 6, where $\log|y_R - y^*|$ is plotted against $\log h$ for a number of different step sizes.

Remark. Note that with $p = 2$ in (4.13a), the minimum total error is $O(\texttt{eps})^{\frac{2}{3}}$, which is slightly smaller than the minimum of the extrapolation in figure 5. Similarly with $p = 2$ in (4.13d), the minimum total error is $O(\texttt{eps})^{\frac{4}{5}}$, which again is slightly smaller than the minimum of the extrapolation in figure 6

Warning. Do not use Euler's method if highly accurate results are needed; there are far better methods.

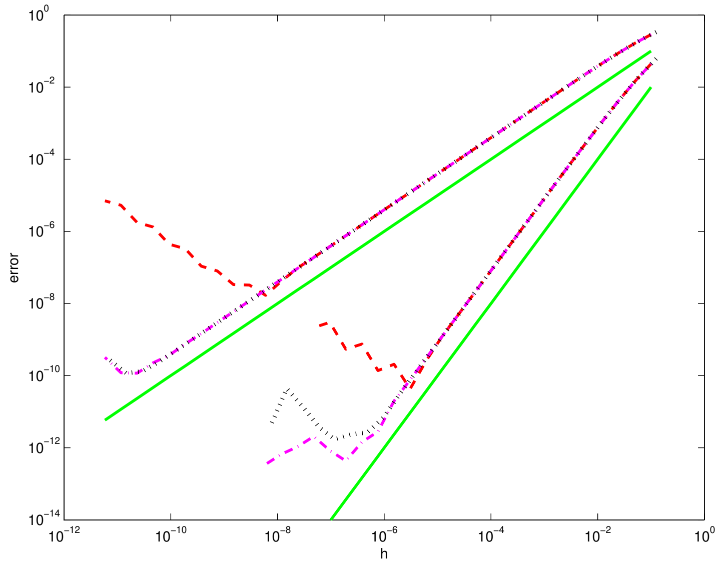


Figure 6: Error in y at the end of range plotted against step size. Upper '– –' line: $h = 2^{-m}/10$; upper '– ·' line: $h = 2^{-m}(1 - r/10)/10$; upper dotted line: $h = 2^{-m}$. Lower lines are the extrapolated second-order solutions using (4.12d). Solid lines are lines of slope 1 and 2.

4.3 Stability of the Euler Method

In computation the *stability* of a method refers to the sensitivity of the solution of a given problem to small changes in the data or the given parameters of the problem. Stability can be characterised a number of ways. One of the Part IB projects will investigate the stability of various methods for numerically integrating ODEs. Here we will examine the linear stability [domain] of the [forward] Euler method.

Specifically, suppose that we let $f(x, y) = -\lambda y$ for some real constant λ. The exact solution is

$$y = y_0 \exp(-\lambda(x - x_0))\,,$$

which decays as x increases if $\lambda > 0$; it is desirable that a numerical solution should similarly decay. So let us apply Euler's method to

$$y' = -\lambda y\,. \tag{4.14a}$$

Then from the definition of Euler's method, i.e. (4.3),

$$Y_{n+1} = Y_n + h(-\lambda Y_n) = Y_n(1-\lambda h)\,. \tag{4.14b}$$

By induction it follows that

$$Y_n = Y_0(1-\lambda h)^n\,, \tag{4.14c}$$

and hence the numerical solution only decays if

$$|1-\lambda h| < 1\,. \tag{4.14d}$$

If instead $|1-\lambda h| > 1$, i.e. if $h > 2/\lambda$, then the numerical solution increases with increasing n, i.e. increasing x. This is a simple example of *linear instability*. If (4.14d) is satisfied, i.e. if h is small enough, we say that the Euler method is *stable*.

Remarks.

(i) We note that as $N \to \infty$

$$Y_N = Y_0(1-\lambda h)^N = y_0\left(1 - \frac{\lambda(b-a)}{N}\right)^N \to y_0 \exp(-\lambda(b-a))\,.$$

Hence as the step size tends to zero, the discrete solution at $x = b$ tends to the exact solution.

(ii) Different methods have different stability properties. A higher-order method that might look attractive from an accuracy viewpoint, would not be of practical use if it was unstable for all h. 05/14

Exercise. Examine the stability of the numerical solution to (4.14a) using the backward Euler method

$$Y_{n+1} = Y_n + hf(x_{n+1}, Y_{n+1})\,. \tag{4.15}$$

Annex

Gift-wrap 'housekeeping' code around the function `eulerverifyrun` used to produce figure 5.

```
% Choose output format and clear variables
%
format compact; format long g; clear h H y Y x X; clf;
%
% Check loop limit m <= 28 so "for" loop limit not exceeded in eulerverify
%
if ~ exist('m','var'), m=28; end
if m > 28, error('Reduce the size of m to less than or equal to 28'), end
%
for i=1:m
  h(i)=0.1/2^(i-1);
  [x(i), y(i)]=eulerverify(0,1,1,h(i));
end
fprintf('h=%3g, xend-1=%3g, yend-2=%3g\n',h(m),x(m)-1,y(m)-2)
loglog(h, abs(y-2), 'r-.','LineWidth',2);
xlabel('h'); ylabel('error'); axis([10^(-10) 1 10^(-12) 1]);
hold on; CurrentFig=gcf; WinOnTop(CurrentFig);
%
input('\nHit return to plot a reference line of slope 1');
%
loglog(h, h, 'b-','LineWidth',2);
%
input('\nHit return to use a step size of h=1/2^m');
%
% Use step sizes that can he stored exactly
%
for i=1:m
  H(i)=1/2^(i+2);
  [X(i), Y(i)]=eulerverify(0,1,1,H(i));
end
fprintf('h=%3g, xend-1=%3g, yend-2=%3g\n',H(m),X(m)-1,Y(m)-2)
loglog(H, abs(Y-2), 'k--','LineWidth',2);
%
input('\nHit return to obtain an estimate of the error');
%
loglog(h(1:m-1),abs(y(2:m)-y(1:m-1)),'m:','LineWidth',3);
%
input('\nHit return to obtain extrapolated solutions');
%
top=round(3*m/4);
loglog(h(1:top-1),abs(2*y(2:top)-y(1:top-1)-2),'r-.','LineWidth',2);
loglog(h(1:top-1),abs(2*Y(2:top)-Y(1:top-1)-2),'k--','LineWidth',2);
%
input('\nHit return to plot a reference line of slope 2');
%
loglog(h(1:m-1), 3*h(1:m-1).^2, 'b-','LineWidth',2);
hold off; print -depsc eulerverify
```

5 Gaussian Elimination

5.0 Introduction

One of the aims of this section is to write a slightly longer piece of MATLAB code, showing how it can be constructed piece by piece. It is generally easier to write a basic code, and then refine it to make it more sophisticated.

Remark. This section also illustrates how to write a standard method, in this case [a very close relative of] Gaussian elimination, in a neater algorithmic form using matrices. Needless to say, matrices are important; as the abstract of a research seminar given this time last year put it:

> *'Functions of matrices are widely used in science, engineering and the social sciences, due to the succinct and insightful way they allow problems to be formulated and solutions to be expressed. New applications involving matrix functions are regularly being found, ranging from small but difficult problems in medicine to huge, sparse systems arising in the solution of partial differential equations. . . . '*

5.1 LU factorization

5.1.1 Definition

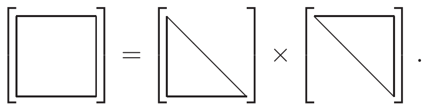
Let A be a real $n \times n$ matrix. We say that the $n \times n$ matrices L and U are an LU *factorization* of A if

(i) L is lower-triangular, i.e. $L_{i\,j} = 0$ for $i < j$,

(ii) U is upper-triangular, i.e. $U_{i\,j} = 0$ for $i > j$, and

(iii) $\mathsf{A} = \mathsf{LU}$.

Therefore the factorization takes the form



5.1.2 Application: solution of linear systems

Let $\mathsf{A} = \mathsf{LU}$ and suppose we wish to solve $\mathsf{A}\mathbf{x} = \mathbf{b}$. This is the same as $\mathsf{L}(\mathsf{U}\mathbf{x}) = \mathbf{b}$, which we decompose into

$$\mathsf{L}\mathbf{y} = \mathbf{b}, \quad \mathsf{U}\mathbf{x} = \mathbf{y}.$$

Both latter systems are triangular and can be solved using forward/backward substitution. Specifically, first we solve for $\mathbf{y}$; thus

$$\begin{aligned} L_{1\,1}y_1 &= b_1 && \text{gives } y_1\text{, next} \\ L_{2\,1}y_1 + L_{2\,2}y_2 &= b_2 && \text{yields } y_2 \text{ etc.} \end{aligned}$$

Then we solve for $\mathbf{x}$ by reversing the order; thus

$$\begin{aligned} U_{n\,n}x_n &= y_n && \text{gives } x_n\text{, next} \\ U_{n-1\,n-1}x_{n-1} + U_{n-1\,n}x_n &= y_{n-1} && \text{yields } x_{n-1}\text{, and so on.} \end{aligned}$$

05/12 *Remark.* This requires $\mathcal{O}(n^2)$ computational operations.[21]
05/13

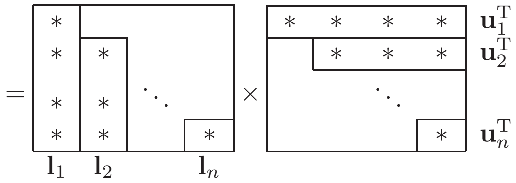
[21] Where, as usual, we only bother to count multiplications/divisions.

5.2 The calculation of an LU factorization

We present a method of the LU factorization that is based on a direct approach. So we assume that the factorization exists.

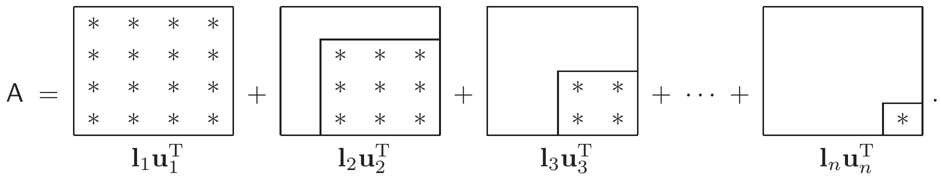
We denote the *columns* of L by $\mathbf{l}_1, \mathbf{l}_2, \ldots, \mathbf{l}_n$ and the *rows* of U by $\mathbf{u}_1^{\mathrm{T}}, \mathbf{u}_2^{\mathrm{T}}, \ldots, \mathbf{u}_n^{\mathrm{T}}$. Then

$$\mathsf{A} = \mathsf{LU} = \begin{bmatrix} \mathbf{l}_1 & \mathbf{l}_2 & \cdots & \mathbf{l}_n \end{bmatrix} \begin{bmatrix} \mathbf{u}_1^{\mathrm{T}} \\ \mathbf{u}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{u}_n^{\mathrm{T}} \end{bmatrix} \tag{5.1a}$$



$$= \sum_{k=1}^{n} \mathbf{l}_k \mathbf{u}_k^{\mathrm{T}}\,. \tag{5.1b}$$

Since the first $(k-1)$ components of $\mathbf{l}_k$ and $\mathbf{u}_k$ are all zero for $k \geqslant 2$, each rank-one matrix $\mathbf{l}_k \mathbf{u}_k^{\mathrm{T}}$ has zeros in its first $(k-1)$ rows and columns. Therefore



(5.1c)

We begin our calculation by extracting $\mathbf{l}_1$ and $\mathbf{u}_1^{\mathrm{T}}$ from A, and then proceed similarly to extract $\mathbf{l}_2$ and $\mathbf{u}_2^{\mathrm{T}}$, etc. First we note from (5.1c) the first row of the matrix $\mathbf{l}_1\mathbf{u}_1^{\mathrm{T}}$ is $L_{1\,1}\mathbf{u}_1^{\mathrm{T}} = \mathbf{u}_1^{\mathrm{T}}$, since the diagonal elements of L equal one. Hence $\mathbf{u}_1^{\mathrm{T}}$ coincides with the first row of A, while the first column of $\mathbf{l}_1\mathbf{u}_1^{\mathrm{T}}$, which is $U_{1\,1}\mathbf{l}_1 = A_{1\,1}\mathbf{l}_1$, coincides with the first column of A. Hence, with $\mathsf{A}^{(0)} = \mathsf{A}$,

$$\mathbf{u}_1^{\mathrm{T}} = [\text{the first row of } \mathsf{A}^{(0)}], \qquad \mathbf{l}_1 = [\text{the first column of } \mathsf{A}^{(0)}]/A_{1\,1}^{(0)}.$$

Next, having found $\mathbf{l}_1$ and $\mathbf{u}_1$, we form the matrix

$$\mathsf{A}^{(1)} = \mathsf{A}^{(0)} - \mathbf{l}_1\mathbf{u}_1^{\mathrm{T}} = \sum_{k=2}^{n} \mathbf{l}_k\mathbf{u}_k^{\mathrm{T}}. \tag{5.2}$$

By construction, the first row and column of $\mathsf{A}^{(1)}$ are zero. It follows that $\mathbf{u}_2^{\mathrm{T}}$ is the second row of $\mathsf{A}^{(1)}$, while $\mathbf{l}_2$ is its second column, scaled so that $L_{2\,2} = 1$; hence we obtain

$$\mathbf{u}_2^{\mathrm{T}} = [\text{the second row of } \mathsf{A}^{(1)}], \qquad \mathbf{l}_2 = [\text{the second column of } \mathsf{A}^{(1)}]/A_{2\,2}^{(1)}.$$

Continuing this way we can formulate an algorithm.

The LU *algorithm.* Set $\mathsf{A}^{(0)} = \mathsf{A}$. For all $k = 1, 2, \ldots, n$ set $\mathbf{u}_k^{\mathrm{T}}$ to be the k^{th} row of $\mathsf{A}^{(k-1)}$ and $\mathbf{l}_k$ to the k^{th} column of $\mathsf{A}^{(k-1)}$, scaled so that $L_{k\,k} = 1$. Next calculate

$$\mathsf{A}^{(k)} = \mathsf{A}^{(k-1)} - \mathbf{l}_k\mathbf{u}_k^{\mathrm{T}}\,, \tag{5.3}$$

and then increment k.

Hence we perform the calculations by the formulae for $k = 1, 2, \ldots, n$:

$$U_{k\,j} = A_{k\,j}^{(k-1)}, \qquad j = k, \ldots, n, \tag{5.4a}$$

$$L_{i\,k} = \frac{A_{i\,k}^{(k-1)}}{A_{k\,k}^{(k-1)}}, \qquad i = k, \ldots, n, \tag{5.4b}$$

$$A_{i\,j}^{(k)} = A_{i\,j}^{(k-1)} - L_{i\,k}U_{k\,j}, \qquad i, j > k. \tag{5.4c}$$

Remarks.

(i) This construction shows that the condition

$$A_{k\,k}^{(k-1)} \neq 0\,, \quad k = 1, \ldots, n-1\,, \tag{5.5}$$

is a *sufficient* condition for an LU factorization to exist and be unique. $A_{1\,1}^{(0)}$ is just $A_{1\,1}$, but the other values are only derived during construction.

(ii) We note that $\mathbf{l}_k \mathbf{u}_k^{\mathrm{T}}$ stays the same if we replace

$$\mathbf{l}_k \to \alpha \mathbf{l}_k, \quad \mathbf{u}_k \to \alpha^{-1} \mathbf{u}_k, \quad \text{where} \quad \alpha \neq 0\,.$$

It is for this reason, subject to (5.5), that we may assume w.l.o.g. that all diagonal elements of L equal one.

Complexity. In calculating complexity, or cost, we assume that it is only multiplications and divisions that matter. By the k-th step of the LU algorithm we see from (5.4b) and (5.4c) that we perform $(n-k)$ divisions to determine the components of $\mathbf{l}_k$ and $(n-k)^2$ multiplications in finding $\mathbf{l}_k \mathbf{u}_k^{\mathrm{T}}$. Hence

$$N_{LU} = \sum_{k=1}^{n-1} \left[(n-k)^2 + (n-k) \right] = \tfrac{1}{3}\, n^3 + \mathcal{O}(n^2). \tag{5.6a}$$

Solving $\mathsf{L}\mathbf{y} = \mathbf{b}$ by forward substitution we use k multiplications/divisions to determine y_k, and similarly for back substitution, thus

$$N_F = N_B = \sum_{k=1}^{n} k \sim \tfrac{1}{2}\, n^2. \tag{5.6b}$$

Relation to Gaussian elimination. The calculation of $\mathsf{A}^{(k)}$ from $\mathsf{A}^{(k-1)}$ is analogous to the row operations in Gaussian elimination. However, a difference between LU algorithm and Gaussian elimination is that in LU we do not consider the right hand side $\mathbf{b}$ until the factorization is complete.

5.2.1 Matlab code

In the following code $\mathsf{A}^{(k)}$ overwrites $\mathsf{A}^{(k-1)}$, and only the non-zero elements of L = `L` and U = `U` are calculated.

```
[m n]=size(A);
if m ~= n, error('Input must be a square matrix.'), end
L=zeros(n); U=zeros(n);
%
for k = 1:n
   for j = k:n
     U(k,j) = A(k,j);
   end
   for i = k:n
     L(i,k) = A(i,k)/A(k,k);
   end
   for i = k:n                              % Choice of i or j as the outer loop
     for j = k:n
       A(i,j) = A(i,j) - L(i,k)*U(k,j);
     end
   end
end
```

Remarks

(i) The function `zeros(m,n)` constructs a m×n matrix of zeros; `zeros(n)` is shorthand for `zeros(n,n)`.

(ii) There is a free choice of outer loop between `i` and `j`; however, the choice of `i` results in more elegant code in the sequel.

At the end of the calculation A is the zero matrix, while L = `L` and U = `U` are each half full. This suggests that with some thought it should be possible to store L and U in the elements of A that are being set to zero. We start by storing U in the upper triangular part of A:

```
[m n]=size(A);
if m ~= n, error('Input must be a square matrix.'), end
L=zeros(n);
%
for k = 1:n-1                             % Loop to n-1
   for i = k:n                            % U(k,j) = A(k,j), so do nothing
     L(i,k) = A(i,k)/A(k,k);
   end
   for i = k+1:n                          % Do not overwrite U in kth row of A
     for j = k:n
       A(i,j) = A(i,j) - L(i,k)*A(k,j);
     end
   end
end
L(n,n)=1;
%
U=triu(A, 0);
```

Remark. The function `triu(A,k)` extracts the elements on and above the $\mathtt{k}^{\text{th}}$ diagonal of the matrix `A`, where `k=0` is the main diagonal, `k>0` is above the main diagonal and `k<0` is below the main diagonal.

By careful examination of where we are overwriting `A`, we can store `L` in the lower triangular part of `A`. It is also possible to combine two `for` loops. Finally we convert the code into a function.

```
function [ L, U ] = LUfactor( A )
%LUfactor: calculate the LU factorisation of A
%
[m n]=size(A);
if m ~= n
  error('Input must be a square matrix.')
else
  for k = 1:n-1
    for i = k+1:n                         % Calculate only non-unit elements of L
      A(i,k) = A(i,k)/A(k,k);             % L(i,k) = A(i,k)/A(k,k)
      for j = k+1:n                       % Do not overwrite L in kth column of A
        A(i,j) = A(i,j) - A(i,k)*A(k,j); % A(i,j) = A(i,j) - L(i,k)*U(k,j)
      end
    end
  end
  %
  L=tril(A,-1)+eye(n);
  U=triu(A,0);
end
```

Remarks

(i) The function `eye(n)`: constructs a `n` × `n` identity matrix.

(ii) The function `tril(A,k)` extracts the elements on and below the $\mathtt{k}^{\text{th}}$ diagonal of the matrix `A`, where `k=0` is the main diagonal, `k>0` is above the main diagonal and `k<0` is below the main diagonal.

5.2.2 Check the code

In order to test our MATLAB code we need one or more suitable matrices A. MATLAB has a number of built-in matrices (some of which we have already encountered), e.g.

- `zeros(m,n)`: constructs a $\mathtt{m} \times \mathtt{n}$ matrix of zeros;
- `ones(m,n)`: constructs a $\mathtt{m} \times \mathtt{n}$ matrix on ones;
- `eye(n)`: constructs a $\mathtt{n} \times \mathtt{n}$ identity matrix;
- `magic(n)`: constructs a $\mathtt{n} \times \mathtt{n}$ matrix from the integers 1 through $\mathtt{n}^2$ with equal row, column, and diagonal sums (valid for all $\mathtt{n} > 0$ except $\mathtt{n} = 2$);
- `hilb(n)`: constructs a $\mathtt{n} \times \mathtt{n}$ Hilbert matrix with $(i,j)^{\text{th}}$ element $1/(i+j-1)$;
- `rand(m,n)`: constructs a $\mathtt{m} \times \mathtt{n}$ matrix containing pseudo-random values drawn from the standard uniform distribution on the open interval $(0,1)$.

As a test of the function `LUfactor` try

```
>> format compact
>> A=magic(5)
>> [L,U]=LUfactor(A)
```

But how do we know if the answer is correct? Well if `L` and `U` are such that $\mathsf{A} - \mathsf{LU} = 0$, then check:

```
>> A-L*U
```

If `n` was much greater than `5` the output would not be very user friendly. Alternatively we can sum the elements of the array:

- by using `sum(A-L*U)`, which is a row vector consisting of the sum over the elements of each column;
- or by using `sum(sum(A-L*U))`, which is better because it is the sum of all elements of `A-L*U`;
- or by using `sum(sum(abs(A-L*U)))`, which is even better because it is a sum of all the absolute values of the elements of `A-L*U` (and so avoids cancellations).

```
>> sum(A-L*U)
>> sum(sum(A-L*U))
>> sum(sum(abs(A-L*U)))
```

We can try different different matrices:

```
>> A=rand(5), [L,U]=LUfactor(A); sum(sum(abs(A-L*U)))
>> A=hilb(5), [L,U]=LUfactor(A); sum(sum(abs(A-L*U)))
>> n=11; A=rand(n) ; [L,U]=LUfactor(A); sum(sum(abs(A-L*U)))
>> n=11; A=hilb(n) ; [L,U]=LUfactor(A); sum(sum(abs(A-L*U)))
>> n= 7; A=magic(n); [L,U]=LUfactor(A); sum(sum(abs(A-L*U)))
>> n= 9; A=magic(n); [L,U]=LUfactor(A); sum(sum(abs(A-L*U)))
>> n=11; A=magic(n); [L,U]=LUfactor(A); sum(sum(abs(A-L*U)))
```

`magic` is clearly not so magic! To obtain an inkling as to what is going wrong try

```
>> for n=3:2:11, det(magic(n)), end
```

Exercise: evaluate the determinants when `n` is even.

As an even simpler example of the problem try

```
>> A=[0 1; 1 0]; [L,U]=LUfactor(A)
```

We will return to this issue in the next section, but first we note that MATLAB has its own LU factorisation function (which seems to have some advantages):

```
>> n=11; A=magic(11); [l,u]=lu(A); sum(sum(abs(A-l*u)))
>> A=[0 1; 1 0]; [l,u]=lu(A), sum(sum(abs(A-l*u)))
```

Note that the MATLAB function `lu` returns a matrix `l` and an upper triangular matrix `u` such that `l*u=A`. However, the matrix `l` returned is *not necessarily* lower triangular; although it might be viewed as 'psychologically lower triangular matrix', in that it is a product of a lower triangular matrix and a permutation matrix.

A modified call to the function `lu` can ensure that a lower triangular matrix is returned, (but at some cost to what `L*U` evaluates to):

```
>> [L,U,P]=lu(A)
>> sum(sum(abs(L*U-A)))
>> sum(sum(abs(L*U-P*A)))
>> sum(sum(abs(P'*P-eye(2))))
>> [l, u]=lu(A); sum(sum(abs(P'*L-l)))
```

06/11 Here `P` is a permutation matrix, and is the modification needed to fix the 'divide-by-zero' problem.

5.3 LU factorization with column pivoting

Column pivoting: description. Naive LU factorization fails when, for example, $A^{(k-1)}_{k\,k} = 0$. A remedy is to exchange rows of A by picking a suitable *pivotal equation*. This technique is called *column pivoting* (or *partial pivoting*), and is equivalent to picking a suitable equation for eliminating the unknown in Gaussian elimination. Specifically, column pivoting means that, having obtained $\mathsf{A}^{(k-1)}$, we exchange two rows of $\mathsf{A}^{(k-1)}$ so that the element of largest magnitude in the k^{th} column is in the 'pivotal position' (k,k). In other words,

$$\left|A^{(k-1)}_{k\,k}\right| = \max_{i=k\ldots n} \left|A^{(k-1)}_{i\,k}\right| .$$

The exchange of rows, say k and p, can be regarded as the pre-multiplication of the relevant matrix $\mathsf{A}^{(k-1)}$ by a permutation matrix P_k,

$$\mathsf{P}_k = \begin{array}{cl} \begin{pmatrix} 1 & & & & & & & & & 0 \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & 0 & & & 1 & & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1 & & & \\ & & & 1 & & & 0 & & & \\ & & & & & & & 1 & & \\ & & & & & & & & \ddots & \\ 0 & & & & & & & & & 1 \end{pmatrix} & \begin{array}{l} \\ \\ \\ \leftarrow \text{ row } k \\ \\ \\ \\ \leftarrow \text{ row } p \\ \\ \\ \\ \end{array} \end{array} . \tag{5.7}$$

Remarks

(i) $\mathsf{P}_k^{\mathrm{T}} = \mathsf{P}_k$.

(ii) $\mathsf{P}_k\mathsf{P}_k = \mathsf{I}$, hence $\mathsf{P}_k^{-1} = \mathsf{P}_k = \mathsf{P}_k^{\mathrm{T}}$.

(iii) $\det \mathsf{P}_k = -1$.

(iv) P_k is a reflection matrix.

06/13
06/14
Good

Algorithm Suppose $\mathsf{A}_{k\,k}^{(k-1)} = \delta$, where $|\delta| \ll 1$, i.e.

$$\mathsf{A}^{(k-1)} = \begin{pmatrix} 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & 0 & \delta & A_{k\,k+1}^{(k-1)} & \cdots & A_{k\,n}^{(k-1)} \\ \vdots & \ddots & 0 & A_{k+1\,k}^{(k-1)} & A_{k+1\,k+1}^{(k-1)} & \cdots & A_{k+1\,n}^{(k-1)} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & A_{n\,k}^{(k-1)} & A_{n\,k+1}^{(k-1)} & \cdots & A_{n\,n}^{(k-1)} \end{pmatrix}. \tag{5.8a}$$

First identify p such that

$$\left|A_{p\,k}^{(k-1)}\right| = \max_i \left|A_{i\,k}^{(k-1)}\right|, \tag{5.8b}$$

and then swap rows p and k of $\mathsf{A}^{(k-1)}$ using P_k as defined in (5.7). Next construct $\mathbf{l}_k$ and $\mathbf{u}_k$ from $\mathsf{P}_k\mathsf{A}^{(k-1)}$ using the same algorithm as before, and then form

$$\mathsf{A}^{(k)} = \mathsf{P}_k\mathsf{A}^{(k-1)} - \mathbf{l}_k\mathbf{u}_k^{\mathrm{T}}. \tag{5.9a}$$

By recursion

$$\begin{aligned} \mathsf{A}^{(k)} &= \mathsf{P}_k(\mathsf{P}_{k-1}\mathsf{A}^{(k-2)} - \mathbf{l}_{k-1}\mathbf{u}_{k-1}^{\mathrm{T}}) - \mathbf{l}_k\mathbf{u}_k^{\mathrm{T}} \\ &= \mathsf{P}_k\mathsf{P}_{k-1}\mathsf{A}^{(k-2)} - \mathsf{P}_k\mathbf{l}_{k-1}\mathbf{u}_{k-1}^{\mathrm{T}} - \mathbf{l}_k\mathbf{u}_k^{\mathrm{T}} \\ &= \dots. \end{aligned}$$

Hence, on defining for convenience $\mathsf{P}_n = \mathsf{I}$ and $\mathsf{P} = \mathsf{P}_n\mathsf{P}_{n-1}\dots\mathsf{P}_1$, we find on undoing the recursion that

$$\mathsf{PA} \equiv \mathsf{P}_n\mathsf{P}_{n-1}\dots\mathsf{P}_1\mathsf{A} = \sum_{k=1}^{n-1} \mathsf{P}_n\dots\mathsf{P}_{k+1}\mathbf{l}_k\mathbf{u}_k^{\mathrm{T}} \;+\; \mathbf{l}_n\mathbf{u}_n^{\mathrm{T}} = \mathsf{LU}, \tag{5.9b}$$

where

$$\mathsf{L} = \begin{bmatrix} \mathsf{P}_n\dots\mathsf{P}_2\mathbf{l}_1 & \dots & \mathsf{P}_n\dots\mathsf{P}_{k+1}\mathbf{l}_k & \dots & \mathbf{l}_n \end{bmatrix}, \quad \text{and} \quad \mathsf{U} = \begin{bmatrix} \mathbf{u}_1^{\mathrm{T}} \\ \mathbf{u}_2^{\mathrm{T}} \\ \vdots \\ \mathbf{u}_n^{\mathrm{T}} \end{bmatrix}. \tag{5.9c}$$

Remarks

(i) At each stage the permutation of rows is applied to the portion of L that has been formed already (i.e. the first $k-1$ columns of L), and then only to the bottom $n-k+1$ components of these vectors.

(ii) We need to record the permutation of rows to solve for the right hand side and/or to compute the determinant.

(iii) Note that $\mathsf{P}^{\mathrm{T}}\mathsf{P} = \mathsf{I}$, i.e. P is orthogonal.

The zero-column case. Column pivoting copes with zeros at the pivot position, except when the whole k^{th} column of $\mathsf{A}^{(k-1)}$ is zero. This can only happen if A is singular, and in that case we let $\mathbf{l}_k$ be the k^{th} unit vector, while taking $\mathbf{u}_k^{\mathrm{T}}$ as the k-th row of $\mathsf{A}^{(k-1)}$ as before. This choice preserves the condition that the matrix $\mathbf{l}_k\mathbf{u}_k^{\mathrm{T}}$ has the same k-th row and column as $\mathsf{A}^{(k-1)}$. Thus

$$\mathsf{A}^{(k)} = \mathsf{A}^{(k-1)} - \mathbf{l}_k\mathbf{u}_k^{\mathrm{T}}$$

still has zeros in its k-th row and column as required.

Round-off error. An important advantage of *column pivoting* is that every element of L has magnitude at most one, i.e. $|L_{ij}| \leqslant 1$ for all $i, j = 1, \dots, n$. This avoids not just division by zero but also tends to reduce the chance of very large numbers occurring during the factorization, a phenomenon that might lead to accumulation of *round-off error* and to *ill conditioning.*

Complexity. For a given A, the factorisation matrices L, U and P can again be calculated in $O(n^3)$ operations. Further, $\mathsf{A}\mathbf{x} = \mathbf{b}$, or equivalently $\mathsf{LU}\mathbf{x} = \mathsf{PA}\mathbf{x} = \mathsf{P}\mathbf{b}$, can again be solved post-factorisation in $O(n^2)$ operations; thus

$$\mathsf{L}\mathbf{y} = \mathsf{P}\mathbf{b} \quad \text{and then} \quad \mathsf{U}\mathbf{x} = \mathbf{y}\,. \tag{5.10}$$

5.3.1 More Matlab

Based on (5.9b), (5.9c) and the `LUfactor` code, a function to perform LU factorisation with partial pivoting is:

```
function [ L, U, Q ] = LUppivot( A )
%LUppivot: calculate the LU factorisation of A with pivoting
%
[m n]=size(A);
if m ~= n
  error('Input must be a square matrix.')
else
  for k = 1:n-1
    % Find row r+k-1 in column k with absolute largest element
    [~,r]=max(abs(A(k:end,k)));
    Q(k,:)=[k,r+k-1];
    % Swop rows k and r+k-1 in A
    A([k,r+k-1],:)=A([r+k-1,k],:);
    %
    for i = k+1:n
      A(i,k) = A(i,k)/A(k,k);
      A(i,k+1:n) = A(i,k+1:n) - A(i,k)*A(k,k+1:n);
    end
  end
  %
  L=tril(A,-1)+eye(n); U=triu(A,0);
end
```

Remarks.

(i) The function `max` returns the maximum element of a vector and the index of that element. Note also the use of `~` to indicate that a function argument should be ignored.

(ii) The permutation information is stored in `Q` as *transpositions* (although, as we shall see, this choice is not the slickest alternative).

(iii) The command `A([k,p],:)=A([p,k],:)` swaps rows `k` and `p` of `A`.

(iv) By storing `L` in the lower triangular part of `A` there is no need to permute `L` explicitly.

(v) Note the use of implicit `for` loops (e.g. '`:`'), including loops that do not start from `1` and which end with `end`.

In addition to `LUfactor` we need a function to calculate PA from using the permutations stored in `Q`:

06/12

We can test `LUppivot` as follows:

```
>> n=11; A=magic(n); [L,U,Q]=LUppivot(A);
>> B=permat(Q,A);
>> ssa=@(A) sum(sum(abs(A))); ssa(B-L*U)
```

However, a far slicker method of tracking the permutations is to track the permutations by swapping elements of a vector P=1:n; thus

```
function [ L, U, P ] = LUpivot( A )
%LUpivot:  calculate the LU factorisation of A with pivoting
%
[m n]=size(A);
if m ~= n
  error('Input must be a square matrix.')
else
  P=1:n;
  for k = 1:n-1
    % Find row r+k-1 in column k with absolute largest element
    [~,r]=max(abs(A(k:end,k)));
    % Swop rows k and r+k-1 in both A and P
    A([k,r+k-1],:) = A([r+k-1,k],:);
    P([k,r+k-1]  ) = P([r+k-1,k]  );
    %
    for i = k+1:n
      A(i,k) = A(i,k)/A(k,k);
      A(i,k+1:n) = A(i,k+1:n) - A(i,k)*A(k,k+1:n);
    end
  end
  %
  L=tril(A,-1)+eye(n); U=triu(A,0);
end
```

There is then no need for permat since L*U=A(P,:):

```
>> n=11; A=magic(n); [L,U,P]=LUpivot(A);
>> B(:,:)=A(P,:); ssa(B-L*U)
```

With partial pivoting it is possible to factorise matrices that defeated LUfactor (even when the determinant is very large):

```
>> n=101; A=magic(n); [L,U,P]=LUpivot(A);
>> clear B; B(:,:)=A(P,:); ssa(B-L*U), det(A)
```

We can now factorise large[ish] matrices, as long as we are willing to wait:

```
>> n=501; A=rand(n); tic; [L,U,P]=LUpivot(A); toc
>> tic; [l u p]=lu(A); toc
>> n=1001; A=rand(n); tic; [L,U,P]=LUpivot(A); toc
>> tic; [l u p]=lu(A); toc
```

5.4 Solution of linear equations

From §5.1.2, given a LU decomposition of A, the solution to $\mathsf{A}\mathbf{x} = \mathbf{b}$ can be obtained by first solving for $\mathbf{y}$ from

$$\mathsf{L}\mathbf{y} = \mathbf{b}\,, \quad \text{i.e.} \quad \begin{pmatrix} 1 & 0 & \cdots & 0 \\ L_{2\,1} & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ L_{n\,1} & \cdots & L_{n\,n-1} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}, \tag{5.11a}$$

and then solving for $\mathbf{x}$ from

$$\mathsf{U}\mathbf{x} = \mathbf{y}\,, \quad \text{i.e.} \quad \begin{pmatrix} U_{1\,1} & U_{1\,2} & \cdots & U_{1\,n} \\ 0 & U_{2\,2} & \cdots & U_{2\,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & U_{n\,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \tag{5.11b}$$

Equations (5.11a) and (5.11b) can be expressed in terms of suffices as

$$L_{i\,i} y_i = b_i - L_{i\,1} y_1 - \ldots - L_{i\,i-1} y_{i-1}\,, \qquad i = 1, 2, \ldots, n-1, n\,, \tag{5.12a}$$

$$U_{i\,i} x_i = y_i - U_{i\,i+1} x_{i+1} - \ldots - U_{i\,n} x_n\,, \qquad i = n, n-1, \ldots, 2, 1\,. \tag{5.12b}$$

A Matlab function to implement this algorithm is

```
function [ x ] = LUsolve( L, U, b )
%LUsolve: solve LUx=b
%
[n m]=size(b);
[s(1:2)]=size(L);
[s(3:4)]=size(U);
if s ~= n*ones(1,4), error('Matrices are not a consistent size.'), end
%
y=b;
for i = 1:n
  y(i,:) = y(i,:) - L(i,1:i-1)*y(1:i-1,:);
  y(i,:) = y(i,:)/L(i,i);
end
x=y;
for i = n:-1:1
  x(i,:) = x(i,:) - U(i,i+1:n)*x(i+1:n,:);
  x(i,:) = x(i,:)/U(i,i);
end
```

Comments on code. Here `n:-1:1` means loop from `n` to `1` in steps of `-1`, and it is assumed that `for` loops such as `for j=1:0` and `for j=n+1:n` are not executed.

07/11

Check on complexity. If we count multiplications we see that in this part of the code

$$\text{number of multiplications} = O(n^2). \tag{5.13}$$

It follows that for $n \gg 1$, the rate-limiting part of the code is the LU factorisation.

As a test of the code try

```
>> format compact
>> A=magic(5); b=rand(5,1);
>> [L,U]=LUfactor(A); X=LUsolve(L,U,b)
```

To check that the answer is correct, given that `X` is meant to be the solution of $\mathsf{A}\mathbf{x} - \mathbf{b} = \mathbf{0}$ try

```
>> A*X-b, sum(abs(A*X-b))
```

We can try different matrices

```
>> sa=@(b) sum(abs(b));
>> n= 5; A=hilb(n) ; b=rand(n,1); [L,U]=LUfactor(A); X=LUsolve(L,U,b); sa(A*X-b)
>> n=11; A=hilb(n) ; b=rand(n,1); [L,U]=LUfactor(A); X=LUsolve(L,U,b); sa(A*X-b)
>> n= 9; A=magic(n); b=rand(n,1); [L,U]=LUfactor(A); X=LUsolve(L,U,b); sa(A*X-b)
>> n=11; A=magic(n); b=rand(n,1); [L,U]=LUfactor(A); X=LUsolve(L,U,b); sa(A*X-b)
>> n= 8; A=magic(n); b=rand(n,1); [L,U]=LUfactor(A); X=LUsolve(L,U,b); sa(A*X-b)
```

In the last two cases pivoting might help. Recall that $\mathsf{A}\mathbf{x} = \mathbf{b}$, and hence that $\mathsf{LU}\mathbf{x} = \mathsf{PA}\mathbf{x} = \mathsf{P}\mathbf{b}$; so try

```
>> n=11; A=magic(n); b=rand(n,1); [l,u,p]=LUpivot(A); x=LUsolve(l,u,b(p))
>> sa(A*x-b)
>> n= 8; A=magic(n); b=rand(n,1); [l,u,p]=LUpivot(A); x=LUsolve(l,u,b(p))
>> sa(A*x-b), det(A)
```

Remark: pivoting does not help solve singular equations!

We can now solve systems with large determinants:

```
>> n=101; A=magic(n); b=rand(n,1); [l,u,p]=LUpivot(A); x=LUsolve(l,u,b(p));
>> sa(A*x-b), det(A)
```

and systems of large[ish] matrices, as long as we are willing to wait:

```
>> n= 501; A=rand(n); b=rand(n,1);
>> tic; [l,u,p]=LUpivot(A); x=LUsolve(l,u,b(p)); toc, sa(A*x-b)
>> tic; X=A\b; toc, sa(A*X-b)
>> n=1001; A=rand(n); b=rand(n,1);
>> tic; [l,u,p]=LUpivot(A); x=LUsolve(l,u,b(p)); toc, sa(A*x-b)
>> tic; X=A\b; toc, sa(A*X-b)
>> tic; X=inv(A)*b; toc, sa(A*X-b)
```

However, there are systems of equations that `LUpivot` and `LUsolve` cannot solve (although the failure is not always immediately clear):

```
if exist('XX','var') clear XX, end
if exist('F' ,'var') clear F , end
if exist('G' ,'var') clear G , end
sa=@(b) sum(abs(b)); format compact
%
for n=3:2:11
  A=hilb(n); b=rand(n,1); [l,u,p]=LUpivot(A); x=LUsolve(l,u,b(p)); X=A\b;
  fprintf(['n=%2d, |A*x-b|=%8.2e, |A*X-b|=%8.2e,\n'...
           '        |x-X|=%8.2e,     det=%8.2e\n\n'],...
          n,  sa(A*x-b),  sa(A*X-b),  sa(x-X), det(A))
end
n=21; A=hilb(n); b=rand(n,1); [l,u,p]=LUpivot(A); x=LUsolve(l,u,b(p));
fprintf('n=%2d, |A*x-b|=%8.2e\n', n, sa(A*x-b)), X=A\b;
%
for m=1:17
    n=m+1; XX(m)=n; A=hilb(n);
    E=eig(A); S=svd(A);
    F(m)=min(abs(E))/max(abs(E));
    G(m)=min(abs(S))/max(abs(S));
end
semilogy(XX,F, 'b'  ,'LineWidth',2); hold on;
xlabel('Size of matrix');
ylabel('Ratio of min(abs)/max(abs) eigenvalues/singular values');
semilogy(XX,G, 'r-.','LineWidth',2);
%
clear XX F G;
for m=1:8
    n=2*m+1; XX(m)=n; A=magic(n);
```

```
    E=eig(A); S=svd(A);
    F(m)=min(abs(E))/max(abs(E));
    G(m)=min(abs(S))/max(abs(S));
end
semilogy(XX,F, 'g'  ,'LineWidth',2);
semilogy(XX,G, 'k-.','LineWidth',2);
legend('Hilbert: eigenvalues','Hilbert: singular values',...
       'Magic (odd): eigenvalues','Magic (odd): singular values',...
       'Location','SouthWest'); hold off;
CurrentFig=gcf; figure(CurrentFig);
```

The underlying problem (highlighted by the *condition-number* warning) is the limited precision of about
07/12 16 significant digits.
07/13

6 The Nitty Gritty

The following is an outline. Details will be available in the *Part IB Computational Projects Manual*, which should be available online from sometime in July 2014 (or early August 2014). If any information differs between these notes and the *Manual*, it is the *Manual* that is definitive.

6.1 The Nature of CATAM Projects

The Part IB Computational Projects course is an elementary introduction to the techniques of solving problems in mathematics using computational methods. It is important to realise that the Computational Projects are about far more than making your code work.

- The projects are intended to be exercises in independent investigation somewhat like those a mathematician might be asked to undertake in the 'real world'.
- You need to understand why the underlying method works, and you often need to understand why and/or when the method does *not* work.
- In doing each project it is crucial that you think around the problem, and in your write-up make intelligent comments as a result of thinking around the problem.
- Indeed, the questions posed in the projects are more open-ended than standard Tripos questions: there is frequently not a single 'correct' response, nor often is the method of investigation fully specified. This is *deliberate*, and is intended to allow you both to demonstrate your ability to use your own judgement in such matters, and also to produce mathematically intelligent, relevant responses to imprecise questions.
- Many graduands say, with hindsight, that one of the most useful parts of the Tripos was CATAM. Please note the 'with hindsight', since many of you will initially find CATAM outside your comfort zone.

6.2 Examination Credit

After the lectures and practical sessions in the Part IA year, you work at your own speed.

- The credit earning part of the course is completed in the second year of the Mathematical Tripos.
- The course is examined entirely through the submission of project write-ups, i.e. coursework; there are no questions on the course in the written examination papers.
- The maximum credit obtainable is 160 marks and there are no alpha or beta quality marks.
- The credit obtained in the Computational Projects course is added directly to the credit gained in the written examination, and hence the maximum contribution to the final merit mark is 160 (which is roughly the same, averaging over the alpha weightings, as for a 16-lecture course).
- However, the total number of marks awarded for the Computational Projects course are often more than for a typical 16-lecture course. For instance, 15.9% of all 'raw' marks awarded to all students in Part IB in 2012 were CATAM marks. This percentage marginally increased to 16.1% if only the students who submitted CATAM projects were included.[22]

[22] For comparison, 10.7% of all 'raw' marks awarded to all students in Part II in 2012 were CATAM marks, with this percentage increasing to 12.5% if only the students who submitted CATAM projects were included. Note that since this is the first year in which there are no alpha or beta quality marks, as yet it is not possible to give meaningful percentages for the merit mark.

- Because there are significant marks available for CATAM, the Faculty of Mathematics takes any resort to *Unfair Means* or *Plagiarism*, or any breach of the *Guidelines for Collaboration* very seriously (see §9).

6.3 Timetable

The timetable below is given as a guide to the expected workload.

Part IA: end of Lent Term and beginning of Easter Term. Attend the introductory lectures and practical sessions. If you have no previous computing experience then you may need to spend extra time learning the basics; the summer vacation is a good opportunity to do this.

Summer between Part IA and Part IB. Do the *Introductory Project*[23]; let me repeat, **do the *Introductory Project***. While the *Introductory Project* carries no marks, doing it is the equivalent of doing practice Tripos questions before going in to the examinations. Make your mistakes on the *Introductory Project*, and learn from them before tackling the credit-earning projects. You might like to ask yourself whether you take examinations without doing [blind] practice questions. A model answer will be made available at the start of the Michaelmas Term.[24] Your College is at liberty to, indeed is encouraged to, arrange a supervision on the model project. As an innovation this year, there *may* be an *Examples Class* on the *Introductory Project* early in the Michaelmas Term.

Michaelmas Term and Christmas Vacation, Part IB. Carry out the **two core projects** and write them up. You are advised to finish the computing and the write-up of at least one of projects before you leave Cambridge at the end of the Michaelmas term.

Lent Term and Easter Vacation, Part IB. You have one week at the start of Lent Term to make **last-minute** changes to the core projects, which should then be submitted. Then do **two additional projects** (out of a choice of four) and write them up.

Easter Term, Part IB. You have one week to make **last-minute** changes to the additional projects. Then submit your work.

After the exams. **You must be available** in the last week of Easter term in case you are called

- either for a routine *Viva Voce Examination*,
- or for an *Examination Interview* or an *Investigative Meeting* if unfair means is suspected.

6.3.1 Planning your work

- You are strongly advised to complete all your computing work by the end of the Christmas and Easter vacations if at all possible, since the submission deadlines are early in Lent and Easter Terms.
- **Do not leave writing up your projects until the last minute.** When you are writing up it is highly likely that you will either discover mistakes in your programming and/or want to refine your code. This will take time. If you wish to maximise your marks, the process of programming and writing-up is likely to be **iterative**, ideally with at least a week or so between iterations.

[23] This will be made available later in the summer. It is likely to be identical to last year's *Introductory Project*, which is currently available online at

`http://www.maths.cam.ac.uk/undergrad/catam/IB/0pt1.pdf`

[24] It's relatively simple to find the answer before that, but it's educationally better to do the project blind.

- It is a good idea to write up each project as you go along, rather than to write all the programs first and only then to write up the reports; each year several students make this mistake and lose credit in consequence (in particular note that a program listing without a write-up, or vice versa, gains no credit). You can, indeed should, review your write-ups in the final week before the relevant submission date.

6.4 Programming language[s]

This year the Faculty is supporting MATLAB as the programming language. During your time in Cambridge the University will provide you with a free copy of MATLAB for your computer. Alternatively you can use the version of MATLAB that is available on the University and College Managed Cluster Service (MCS).

6.4.1 Your copy of MATLAB

All undergraduate students at the University are entitled to download and install MATLAB on their own computer that is running Windows, MacOS or Linux; your copy should be used for non-commercial University use only. The files for download, and installation instructions, are available at

`http://www.maths.cam.ac.uk/undergrad/catam/software/matlabinstall/matlab-personal.htm`.

This link is Raven protected. Several versions of MATLAB may be available; if you are downloading MATLAB for the first time it is recommended that you choose the latest version.

6.4.2 Programming manual[s]

The Faculty of Mathematics has produced a booklet *Learning to use MATLAB for CATAM project work*, that provides a step-by-step introduction to MATLAB suitable for beginners. This is available on-line at

`http://www.maths.cam.ac.uk/undergrad/catam/MATLAB/manual/booklet.pdf`

However, this short guide can only cover a small subset of the MATLAB language. There are many other guides available on the net and in book form that cover MATLAB in far more depth. Further:

- MATLAB has its own built-in help and documentation.
- The suppliers of MATLAB, *The MathWorks*, provide the introductory guide *Getting Started with MATLAB*. You can access this by 'left-clicking' on the `Getting Started` link at the top of a MATLAB '*Command Window*'. Alternatively there is an on-line version available at[25]

  `http://www.mathworks.co.uk/help/matlab/getting-started-with-matlab.html`

  A printable version is available from

  `http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf`

- *The MathWorks* provide links to a whole a raft of other tutorials; see

  `http://www.mathworks.co.uk/academia/student_center/tutorials/launchpad.html`

  In addition their *MATLAB* documentation page gives more details on maths, graphics, object-oriented programming etc.; see

[25] These links work at the time of writing. Unfortunately *The MathWorks* have an annoying habit of breaking their links.

http://www.mathworks.co.uk/help/matlab/index.html

- There is a plethora of books on MATLAB. For instance:
  (a) *Numerical Computing with MATLAB* by Cleve Moler[26] (SIAM, Second Edition, 2008, ISBN 978-0-898716-60-3). This book can be downloaded for free from

  http://www.mathworks.co.uk/moler/chapters.html

  (b) *MATLAB Guide* by D.J. Higham & N.J. Higham (SIAM, Second Edition, 2005, ISBN 0-89871-578-4).

  Further, Google returns about 24,300,000 hits for the search 'MATLAB introduction' (up from 13,400,000 hits last year), and about 3,210,000 hits for the search 'MATLAB introduction tutorial' (slightly down from 3,340,000 hits).

- The Engineering Department has a webpage that lists a number of helpful articles; see

  http://www.eng.cam.ac.uk/help/tpl/programs/matlab.html

6.4.3 To MATLAB, or not to MATLAB

Use of MATLAB is recommended, especially if you have not programmed before, but *you are free to write your programs in any computing language whatsoever*. Python, C, C++, C#, Java, Visual Basic, Mathematica and Maple have been used by several students in the past, and Excel has often been used for plotting graphs of computed results. The choice is your own, provided your system can produce results and program listings on paper.[27]

However, you should bear in mind the following points.

- The Faculty does *not* promise to help you with programming problems if you use a language other than MATLAB.

- Not all languages have the breadth of mathematical routines that come with the MATLAB package. You may discover either that you have to find reliable replacements, or that you have to write your own versions of mathematical library routines that are pre-supplied in MATLAB (this can involve a fair amount of effort). To this end you may find reference books, such as *Numerical Recipes* by W. H. Press *et al.* (CUP), useful. You may use equivalent routines to those in MATLAB from such works so long as you acknowledge them, and reference them, in your write-ups.

- If you choose a high-level programming language that can perform advanced mathematical operations automatically, then you should check whether use of such commands is permitted in a particular project. As a rule of thumb, do not use a built-in function if there is no equivalent MATLAB routine that has been approved for use, or if use of the built-in function would make the programming considerably easier than intended. For example, use of a command to test whether an integer is prime would not be allowed in a project which required you to write a program to find prime numbers. The *CATAM Helpline* (see §8 below) can give clarification in specific cases.

- Subject to the aforementioned limited exceptions, *you must write your own computer programs*. Downloading computer code, e.g. from the internet, that you are asked to write yourself counts as plagiarism (see §9).

[26] Cleve Moler is chief mathematician, chairman, and co-founder of MathWorks.
[27] There is no need to consult the *CATAM Helpline* as to your choice of language.

7 Project Reports

7.1 Project write-ups: examination credit

Each individual project carries the same credit. For each project, 40% of the marks available are awarded for writing programs that work and for producing correct graphs, tables of results and so on. A further 50% of the marks are awarded for answering mathematical questions in the project and for making appropriate mathematical observations about your results.

The final 10% of marks are awarded for the overall 'excellence' of the write-up. Half of these 'excellence' marks may be awarded for presentation, that is for producing good clear output (graphs, tables, etc.) which is easy to understand and interpret, and for the mathematical clarity of your report.

The assessors may penalise a write-up that contains an excessive quantity of irrelevant material (see below). In such cases, the 'excellence' mark may be reduced and could even become negative, as low as -10%.

Unless the project specifies a way in which an algorithm should be implemented, marks are, in general, not awarded for programming style, good or bad. Conversely, if your output is poorly presented — for example, if your graphs are too small to be readable or are not annotated — then you may lose marks.

No marks are given for the submission of program code without a report, or vice versa.

The marks for each project are scaled so that a possible maximum of 160 marks are available for the Computational Projects course. No quality marks (i.e. αs or βs) are awarded. The maximum contribution to the final merit mark is thus 160 and roughly the same (averaging over the α weightings) as for a 16-lecture course.

7.2 Project write-ups: advice

Your record of the work done on each project should contain all the results asked for and your comments on these results, together with any graphs or tables asked for, clearly labelled and referred to in the report. However, it is important to remember that the project is set as a piece of mathematics, rather than an exercise in computer programming; thus the most important aspect of the write-up is the **mathematical content**. For instance:

- Your comments on the results of the programs should go *beyond* a rehearsal of the program output and show an understanding of the mathematical and, if relevant, physical points involved. The write-up should demonstrate that you have noticed the most important features of your results, and understood the relevant mathematical background.
- When discussing the computational method you have used, you should distinguish between points of interest in the algorithm itself, and details of your own particular implementation. Discussion of the latter is usually unnecessary, but if there is some reason for including it, please set it aside in your report under a special heading: it is rare for the assessors to be interested in the details of how your programs work.
- Your comments should be pertinent and concise. Brief notes are perfectly satisfactory — provided that you cover the salient points, and make your meaning precise and unambiguous — indeed, students who keep their comments concise can get better marks. An over-long report may well lead an assessor to the conclusion that the candidate is unsure of the essentials of a project and is using quantity in an attempt to hide the lack of quality. Do not copy out chunks of the text of the projects themselves: you may assume that the assessor is familiar with the background to each project and all the relevant equations.

- Similarly you should not reproduce large chunks of your lecture notes; you will not gain credit for doing so (and indeed may lose credit as detailed in §7.1). However, you will be expected to reference results from theory, and show that you understand how they relate to your results. If you quote a theoretical result from a textbook, or from your notes, or from the WWW, you should give both a brief justification of the result and a *full reference*.[28] If you are actually asked to *prove* a result, you should do so briefly.

- Graphs will sometimes be required, for instance to reveal some qualitative features of your results. Such graphs, *including labels, annotations, etc.*, need to be computer-generated (see also § 7.3). Further, while it may be easier to print only one graph a page, it is often desirable (e.g. to aid comparison) to include *two or more* graphs on a page. Also, do not forget to clearly label the axes of graphs or other plots, and provide any other annotation necessary to interpret what is displayed. Similarly, the rows and columns of any tables produced should be clearly labelled.

- You should take care to ensure that the assessor sees evidence that **your programs do indeed perform the tasks** you claim they do. In most cases, this can be achieved by including a sample output from the program. If a question asks you to write a program to perform a task but doesn't specify explicitly that you should use it on any particular data, you should provide some 'test' data to run it on and include sample output in your write-up. Similarly, if a project asks you to 'print' or 'display' a numerical result, you should demonstrate that your program does indeed do this by including the output.

- **Above all, make sure you comment where the manual specifically asks you to.** It also helps the assessors if you answer the questions in the order that they appear in the manual and, if applicable, **number your answers** using the same numbering scheme as that used by the project. Make clear which outputs, tables and graphs correspond to which questions and programs.

The following are indicative of some points that might be addressed in the report; they are not exhaustive and, of course not all will be appropriate for every project. In particular, some are more relevant to pure mathematical projects, and others to applied ones.

- Does the algorithm or method always work? Have you tested it?

- What is the theoretical running time, or complexity, of the algorithm? Note that this should be measured by the number of simple operations required, expressed in the usual $O\big(f(n)\big)$ or $\Omega\big(f(n)\big)$ notation[29], where n is some reasonable measure of the size of the input (say the number of vertices of a graph) and f is a reasonably simple function. Examples of simple operations are the addition or multiplication of two numbers, or the checking of the (p, q) entry of a matrix to see if it is non-zero; with this definition finding the scalar product of two vectors of length n takes order n operations. Note that this measure of complexity can differ from the number of MATLAB commands/'operations', e.g. there is a single MATLAB command to find a scalar product of two vectors of length n.

- What is the accuracy of the numerical method? Is it particularly appropriate for the problem in question and, if so, why? How did you choose the step-size (if relevant), and how did you confirm that your numerical results are reliably accurate (cf. §4.2)?

- How do the numerical answers you obtain relate to the mathematical or physical system being modelled? What conjectures or conclusions, if any, can you make from your results about the physical system or abstract mathematical object under consideration?

In summary, it is the candidate's responsibility to determine which points require discussion in the report, to address these points fully but concisely, and to structure the whole so as to present a

[28] See also the paragraph on *Citations* in §9
[29] For instance see http://en.wikipedia.org/wiki/Big_O_notation.

clear and complete response to the project. It should be possible to read your write-up without reference to the listing of your programs.

As an aid, for the **two core projects**, some brief additional comments are provided giving further guidance as to the form and approximate length of answer expected for each question. These also contain a mark-scheme, on which your marks for each question will be written and returned to you during the Lent Term. For the additional projects you are expected to use your judgement on the marks allocation.

7.2.1 Project write-ups: advice on length

The word *brief* peppers the last few paragraphs. However, each year some students just do not get it. To emphasise this point, in general **six sides of A4 of text**, *excluding in-line graphs, tables, etc.*, should be plenty for a clear concise report. Indeed, the best reports are sometimes shorter than this.

To this total you will of course need to add tables, graphs, printouts etc. However, *do **not** include every single piece of output you generate*: include a selection of the output that is a *representative* sample of graphs and tables. It is up to you to choose a selection which demonstrates all the important features but is reasonably concise. Remember that you are writing a report to be read by a *human being*, who will not want to wade through pages and pages of irrelevant or unimportant data. Twenty sides of graphs would be excessive for most projects, even if the graphs were printed one to a page.[30] Remember that the assessors will be allowed to **deduct** up to 10% of marks for any project containing an excessive quantity of irrelevant material. Typically, such a project might be long-winded, be very poorly structured, or contain long sections of prose that are not pertinent. Moreover, if your answer to the question posed is buried within a lot of irrelevant material then it may not receive credit, even if it is correct.

7.3 Project write-ups: technicalities

As emphasised above, elaborate write-ups are not required. However, *in a change from previous years*, you are required to submit your project reports *both* as hard-copy *and* electronically (both submissions must be identical). In particular, you will be asked to submit your write-ups electronically in Portable Document Format (PDF) form. Note that many word processors (e.g. LaTeX, *Microsoft Word*, *LibreOffice*) will generate output in PDF form. In addition, there are utility programs to convert output from one form to another, in particular to PDF form (e.g. there are programs that will convert plain text to PDF). Before you make your choice of word processor, you should confirm that you will be able to generate submittable output in PDF form. Please note that a PDF file generated by scanning a document is ***not*** acceptable; in particular, and for the sake of clarity, a PDF file generated by scanning a hand-written report is ***not*** acceptable.

In a very few projects, where a *sketch* (or similar) is asked for, a scanned hand-drawing is acceptable. Such exceptions will be noted *explicitly* in the project description.

If it will prove difficult for you to produce electronic write-ups, e.g. because of a disability, then please contact the *CATAM Helpline* as early as possible in the academic year, so that reasonable adjustments can be made for you.

Choice of Word Processor. As to the choice of word processor, there is no definitive answer. Many mathematicians use LaTeX (or, if they are of an older generation, TeX), e.g. this document is written in LaTeX. However, please note that although LaTeX is well suited for mathematical typesetting, it is absolutely acceptable to write reports using other word-processing software, e.g. *Microsoft Word* and *LibreOffice*. The former is commercial, while the latter can be installed for free for, *inter alia*, the Windows, MacOS and Linux operating systems from

`http://www.libreoffice.org/download/libreoffice-fresh/` and `http://www.libreoffice.org/download/libreoffice-stable/`.

[30] Recall that graphs should not as a rule be printed one to a page.

Both *Microsoft Word* and *LibreOffice* are available on the MCS.

LaTeX. If you decide to use a LaTeX, or you wish to try it out, LaTeX is available on the MCS. If you are going to use it extensively, then you will probably want to install LaTeX on your own personal computer. This can be done for free. For recommendations of TeX distributions and associated packages see

- `http://www.tug.org/begin.html#install` and
- `http://www.tug.org/interest.html#free`.

Front end. In addition to a TeX distribution you will also need a front-end (i.e. a 'clever editor'). A comparison of TeX editors is available on WIKIPEDIA; below we list a few of the more popular TeX editors.

TeXstudio. For Windows, Mac and Linux users, there is TeXstudio. The proTeXt distribution, based on MiKTeX, includes the TeXstudio front end.

TeXworks. Again for Windows, Mac and Linux users, there is TeXworks. The MiKTeX distribution includes TeXworks.

TeXShop. Many Mac aficionados use TeXShop. To obtain TeXShop and the TeXLive distribution see `http://pages.uoregon.edu/koch/texshop/obtaining.html`.

TeXnicCenter. TeXnicCenter is another potential front end for Windows users (see also `http://www.maths.cam.ac.uk/computing/win7/home_texniccenter.html`).

LyX. LyX is not strictly a front end, but has been recommended by some previous students. LyX is available from

`http://www.lyx.org/`.

However, note that LyX uses its own internal file format, which it converts to LaTeX as necessary.

Learning LaTeX. A *Brief LaTeX Guide for CATAM* is available for download from

`http://www.maths.cam.ac.uk/undergrad/catam/LaTeX/Brief-Guide.pdf` .

- The LaTeX source file (which may be helpful as a template), and supporting files, are available for download as a zip file from

  `http://www.maths.cam.ac.uk/undergrad/catam/LaTeX/Guide.zip` .

  Mac and Unix users should already have an unzip utility, while Windows users can download 7-Zip if they have not.
- Please note that this introduction is still under development; you will be notified of updates via *CATAM News*.

Other sources of help. A welter of useful links have been collated by the Engineering Department on their *Text Processing using LaTeX* page; see

`http://www-h.eng.cam.ac.uk/help/tpl/textprocessing/LaTeX_intro.html`.

Program listings. At the end of each report you should include complete **printed listings** of every major program used to generate your results. You do *not* need to include a listing of a program which is essentially a minor revision of another which you have already included. Make sure that your program listings are the very last thing in your reports. Please do not mix program output and program listings together; if you do, the program output may not be marked as part of the report.

8 Information Sources

There are many ways of getting help on matters relating to CATAM.

The CATAM Web Page. The CATAM web page,

`http://www.maths.cam.ac.uk/undergrad/catam/`
contains much useful information relating to CATAM. There are on-line, and up-to-date, copies of the projects, and any data files required by the projects can be downloaded. There is also the booklet *Learning to use MATLAB for CATAM project work*.

CATAM News and Email. Any important information about CATAM (e.g. corrections to projects or to other information in the *Handbook*, availability of advisers, temporary closures of the *CATAM room*) is publicised via *CATAM News*, which can be reached from the CATAM web page. You **must read** *CATAM News* from time to time to check for these and other important announcements, such as submission dates and procedures.

As well as adding announcements to *CATAM News*, occasionally we will email students using the year lists maintained by the Faculty of Mathematics. You have a responsibility to read email from the Faculty, and if we send an email to one of those lists we will assume that you have read it.

After 1 October 2014 you can check that you are on the appropriate Faculty year list by referring to the `https://lists.cam.ac.uk/mailman/raven` webpage (to view this page you will need to authenticate using Raven if you have not already done so). You should check that the *Maths-IB* mailing list is one of your current lists.

If you are not subscribed to the correct mailing list, then this can be corrected by contacting the Faculty Undergraduate Office (email: `undergrad-office@maths.cam.ac.uk`) with a request to be subscribed to the correct list (and, if necessary, unsubscribed from the wrong list).

The CATAM Helpline. If you need help (e.g. if you need clarification about the wording of a project, or if you have queries about programming and/or MATLAB, or if you need an adviser to help you debug your programs), you can email a query to the *CATAM Helpline*: `catam@maths.cam.ac.uk`. Almost all queries may be sent to the *Helpline*, and it is particularly useful to report potential errors in projects. However the *Helpline* cannot answer detailed mathematical questions about particular projects. Indeed if your query directly addresses a question in a project you may receive a standard reply indicating that the *Helpline* cannot add anything more.

In order to help us manage the emails that we receive,

- please use an email address ending in `cam.ac.uk` (rather than a Gmail, etc. address) both so that we may identify you and also so that your email is not identified as spam;
- please specify, in the subject line of your email, 'Part IB' as well as the project number and title or other topic, such as 'MATLAB query', to which your email relates;
- please also **restrict each email to one question or comment** (use multiple emails if you have more than one question or comment).

The *Helpline* is available during Full Term and one week either side. Queries sent outside these dates will be answered subject to personnel availability. We will endeavour (but do not guarantee) to provide a response from the *Helpline* within three working days. However, if the query has to be referred to an assessor, then it may take longer to receive a reply. Please do not send emails to any other address.[31]

In addition to the *Helpline*, at certain times of the year, e.g. in the period immediately before submission, advisers may be available in the *CATAM room*. As well as answering queries about general course administration, programming and/or MATLAB, you are allowed to ask advisers to help you debug your programs. The times when they will be available will be advertised in *CATAM News*.

The CATAM FAQ Web Pages. Before asking the *Helpline* about a particular project, please check the *CATAM FAQ web pages* (accessible from the main CATAM web page). These list questions which students regularly ask, and you may find that your query has already been addressed.

[31] For example emails sent directly to the CATAM Course Director may be subject to a far longer delay in answering (and could end up either being missed altogether or consigned to `/dev/null`).

Advice from Supervisors and Directors of Studies. The general rule is that advice **must be general in nature**. You should not have supervisions on any work that is yet to be submitted for examination; however, you may have a supervision on the *Introductory Project*, and/or another non-examinable project, and/or any work set by your Director of Studies. A supervisor can also provide feedback on the *Core Projects* **after** they have been submitted (e.g. **after** your marks have been returned).

9 Unfair Means, Plagiarism and Guidelines for Collaboration

You **must** work *independently* on the projects, both on the programming and on the write-ups, i.e. you must write and test all programs yourself, and all reports must be written independently. However, it is recognised that some candidates may occasionally wish to discuss their work with others doing similar projects. This can be educationally beneficial and is accepted provided that it remains within reasonable bounds.

Acceptable collaboration. Acceptable collaboration may include an *occasional general discussion* of the approach to a project and of the numerical algorithms needed to solve it. Small hints on debugging code (note the *small*), as might be provided by an adviser, are also acceptable.

Unacceptable collaboration. If a general discussion *either* is happening regularly *or* gets to the point where physical or virtual notes are being exchanged (even on the back of an envelope, napkin or stamp), then it has reached the stage of unacceptable collaboration. Indeed, assuming that you are interpreting the phrase *occasional general discussion* in the spirit that it is written, then if you have got to the stage of wondering whether a discussion has reached the limit of acceptable collaboration, or you have started a legalistic deconstruction of the term *acceptable collaboration*, you are almost certainly at, or past, the limit.

Unacceptable collaboration also *includes*

- copying any other person's program, either automatically or by typing it in from a listing;
- using someone else's program or any part of it as a model, or working from a jointly produced detailed program outline;
- copying or paraphrasing of someone else's report in whole or in part.

These comments apply equally to copying from the work of previous Part IB students, or another third party (including any code, etc. you find on the internet), as they do to copying from the work of students in your own year. Asking anyone for help, and this includes posting questions on the internet (e.g. StackExchange), that goes past the limits of *acceptable collaboration* as outlined above, constitutes unfair means.

Further, you should not allow any present or future Part IB student access to the work you have undertaken for your own CATAM projects, even after you have submitted your write-ups. If you knowingly give another student access to your CATAM work, whatever the circumstances, you may be penalised yourself.

Citations. It is, of course, perfectly permissible to use reference books, journals, reference articles on the WWW or other similar material: indeed, you are encouraged to do this. You may quote directly from reference works so long as you acknowledge the source (WWW pages should be acknowledged by a *full* URL). There is no need to quote lengthy proofs in full, but you should at least include your own brief summary of the material, together with a *full* reference (including, if appropriate, the page number) of the proof.

Programs. You must write your own computer programs. Downloading computer code, e.g. from the internet, that you are asked to write yourself counts as plagiarism even if cited.

University-wide Statement on Plagiarism. You should familiarise yourself with the University's *Statement on Plagiarism*. There is a link to this statement from the University's *Good academic practice and plagiarism* website http://www.admin.cam.ac.uk/univ/plagiarism/,

which also features links to other useful resources, information and guidance.

Faculty Guidelines on Plagiarism. You should also be familiar with the Faculty of Mathematics *Guidelines on Plagiarism*. These guidelines, which include advice on quoting, paraphrasing, referencing, general indebtedness, and the use of web sources, are posted on the Faculty's website at

http://www.maths.cam.ac.uk/facultyboard/plagiarism/.

Declarations. To certify that you have *read and understood* these guidelines, you will be asked to sign an electronic declaration at the start of the Michaelmas Term. You will receive an email with instructions as to how to do this at the start of the Michaelmas Term.

In order to certify that you have *observed* these guidelines, you will be required to sign a submission form when you submit your write-ups, and you are advised to read it carefully. You must list on the form *anybody* (students, supervisors and Directors of Studies alike) with whom you have exchanged information (e.g. by talking to them, or by electronic means) about the projects at any more than a *trivial* level: *any* discussions that affected your approach to the projects to *any* extent must be listed. Failure to include on your submission form any discussion you may have had *is* a breach of these guidelines.

However, declared exchanges are perfectly allowable so long as they fall within the limits of 'acceptable collaboration' as defined above, and you should feel no qualms about listing them. For instance, as long as you have refrained from discussing in any detail your programs or write-ups with others after starting work on them, then the limits have probably not been breached.

The assessors will not have any knowledge of your declaration until after all your projects have been marked. However, your declaration may affect your CATAM marks if the assessors believe that discussions have gone beyond the limits of what is acceptable. If so, or if there is a suspicion that your have breached any of the other guidelines, you will be summoned either to an *Examination Interview* in relatively minor cases, or to an *Investigative Meeting* in more serious cases.[32] Either case may lead to a change in the marks you receive for the Computational Projects.

Plagiarism detection. **The programs and reports submitted will be checked carefully both to ensure that they are your own work, and to ensure the results that you hand in have been produced by your own programs.**

Checks on submitted program code. The Faculty of Mathematics uses (and has used for many years) specialised software, including that of external service providers, which automatically checks whether your programs either have been copied or have unacceptable overlaps (e.g. the software can spot changes of notation). All programs submitted are screened.

The code that you submit, and the code that your predecessors submitted, is kept in *anonymised* form to check against code submitted in subsequent years.

Checks on electronically submitted reports. In addition, and as encouraged by the University, for the 2014-15 academic year the Faculty of Mathematics is planning to screen your electronically submitted reports using the *Turnitin UK* text-matching software. Further information will be sent to you at the beginning on the Michaelmas term. The electronic declaration which you will be asked to complete at the start of the Michaelmas term will, *inter alia*, cover the use of *Turnitin UK*.

Your electronically submitted write-ups will be kept in *anonymised* form to check against write-ups submitted in subsequent years.

[32] For example see
http://www.admin.cam.ac.uk/univ/plagiarism/examiners/nonresearchprocedure.pdf
and http://www.admin.cam.ac.uk/univ/plagiarism/examiners/investigative.pdf.

Example. As an instance to clarify the limits of 'acceptable collaboration', if an assessor reading two anonymous write-ups (without knowledge of the declared lists of discussions) were to see significant similarities in results, answers, mathematical approach or programming which would clearly not be expected from students working independently, then there would appear to be a case that the students have breached the limits. An *Examination Interview* or an *Investigative Meeting* would then be arranged.

Sanctions. The Computational Projects are considered to be a single piece of work within the Mathematical Tripos: therefore, if it is concluded that you have used unfair means for one or more individual write-ups, you may stand to lose your marks for the entire Computational Projects course. Unfortunately, there have been cases in recent years where *some individuals have been penalised by the loss of significant numbers of marks*, which has lead to the drop of a class.

The Faculty of Mathematics wishes to make it clear that any breach of these guidelines will be treated very seriously.

However, it is emphasised that the great majority of candidates have had no difficulty in keeping to these guidelines in the past; if you find them unclear in any way you should seek advice from the *CATAM Helpline*.

9.1 Oral examinations

Viva Voce Examinations. A number of candidates may be selected, either randomly or formulaically, for a *Viva Voce Examination* after submission of either the core or the additional projects. This is a matter of routine, and therefore a summons to a *Viva Voce Examination* should not be taken to indicate that there is anything amiss. You will be asked some straightforward questions on your project work, and may be asked to elaborate on the extent of discussions you may have had with other students. So long as you can demonstrate that your write-ups are indeed your own, your answers will not alter your project marks.

Examination Interviews. For relatively minor cases of suspected plagiarism, collusion or other unfair means the Chair of Examiners may summon a particular candidate or particular candidates for interview on any aspect of the written work of the candidate or candidates not produced in an examination room which in the opinion of the Examiners requires elucidation.[33] One purpose of an *Examination Interview* might be to confirm that you have fully declared all collaboration.

Investigative Meetings. For more serious cases of suspected plagiarism, collusion or other unfair means, the Chair of Examiners may summon a candidate to an *Investigative Meeting.* If this happens, you have the right to be accompanied by your Tutor (or another representative at your request). The reasons for the meeting, together with copies of supporting evidence and other relevant documentation, will be given to your Tutor (or other representative).[34]

Timing. Viva Voce Examinations, *Examination Interviews* and *Investigative Meetings* are a formal part of the Tripos examination, and if you are summoned then you must attend. In the case of the core projects these will usually take place during Lent Full Term (although they may take place exceptionally during Easter Full Term), and in the case of the additional projects these will usually take place during the last week of Easter Full Term. Easter Term *Viva Voce Examinations* are likely to take place on the Monday of the last week, while *Examination Interviews* and *Investigative Meetings* may take place any time that week. If you need to attend during the last week of Easter Full Term you will be informed in writing just after the

[33] See http://www.admin.cam.ac.uk/univ/plagiarism/examiners/nonresearchprocedure.pdf.
[34] For more information see
http://www.admin.cam.ac.uk/univ/plagiarism/examiners/index.html and http://www.admin.cam.ac.uk/univ/plagiarism/examiners/investigative.pdf.

end of the written examinations. **You must be available** in the last week of Easter Term in case you are summoned.

10 Submission and Assessment

In order to gain examination credit for the work that you do on this course, you must write reports on each of the projects that you have done. As emphasised earlier it is the quality (not quantity) of your written report which is the most important factor in determining the marks that you will be awarded.

10.1 Submission form

When you submit a hard-copy of your project reports you will be required to sign a submission form detailing which projects you have attempted and listing all discussions you have had concerning CATAM (see §9, *Unfair Means, Plagiarism and Guidelines for Collaboration*). Further details, including the definitive submission form, will be made available when the arrangements for electronic submission of reports and programs (see below) are announced.

10.2 Submission of written work

In order to gain examination credit, you must:

- submit electronic copies of your reports and programs (see §10.3);
- complete and sign your submission form;
- submit, with your submission form, your written reports and program printouts for every project for which you wish to gain credit;
- sign a submission list.

Please note as part of the submission process your work will be placed into plastic wallets, with the individual wallets being sent to different examiners; hence each project should have its own wallet. You can provide your own wallets (which will speed up submission) or use the wallets provided on the day. If a project will not fit into a single wallet, then re-read section §7.2.1 on *Project write-ups: advice on length*.

The location for handing in your work will be announced via *CATAM News* and email closer to the time.

10.3 Electronic submission

You are also required to submit electronically copies of both your reports and your program source files. The electronic submission must be identical to the hard-copy submission. Electronic submission enables the Faculty to run automatic checks on the independence of your work, and also allows your programs to be inspected in depth (and if necessary run) by the assessors.

As regards your programs, electronic submission applies whether you have done your work on your own computer, on the MCS, or elsewhere, and is regardless of which programming language you have chosen.

Full details of the procedure will be announced about one week before the submission deadlines via *CATAM News* and email, so please do *not* make enquiries about it until then.

However please note that you will need to know your UIS password in order to submit copies of your report and program source files.

If you cannot remember your UIS password you will need to ask the Computing Service to reset it.[35] This may take some time, so check that you know your UIS password well before submission day.

Naming convention. To make submission and marking easier, please put all your source files related to different projects in separate directories/folders. Further, please name each directory/folder using a convention whereby the first few characters of the directory/folder name give the project number, with the dot replaced by either a minus sign or underscore (`_`). For example, all the programs written for project 2.3 should be placed in a directory/folder with a name beginning with `2-3` or `2_3`.

10.4 (Non)-return of written work

We regret that students' submitted work cannot be returned to them after the examination; it must be retained in case of a query or an appeal at a later stage. You are recommended to keep at least an electronic version of your work (or even make a photocopy before submitting the hard-copy).

Since the manuals will be taken off-line after the close of submission, you might also like to make a hard copy of the projects you have attempted.

Please note that all material that you submit electronically is kept in *anonymised* form to check against write-ups and program code submitted in subsequent years.

[35] E.g. see `http://www.ucs.cam.ac.uk/docs/faq/accounts/n3`.