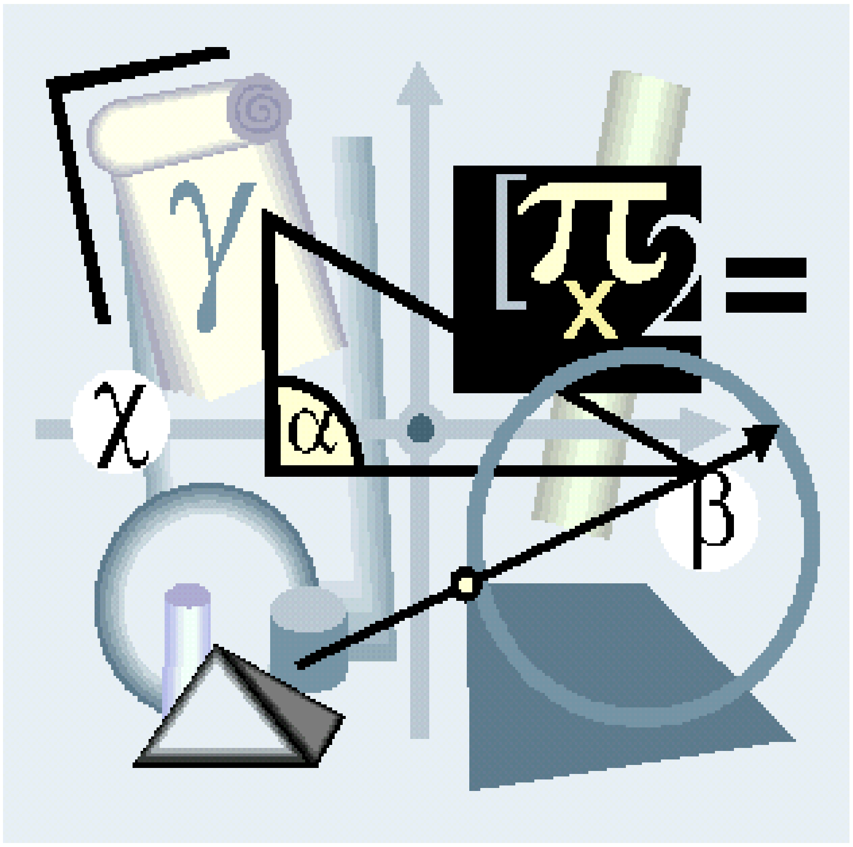
APM1513/1

UNISA university of south africa

DEPARTMENT OF MATHEMATICAL SCIENCES



APPLIED LINEAR ALGEBRA (APM1513)

STUDY GUIDE

August 2008 (version 1)
May 2009 (version 2)

TEAM:

Author:	Prof N.T. Bishop
ODL Learning Design:	Hentie Wilson, Curriculum
Mathematics Instruction Design:	Antonia Makina
Editor:	
Cover page layout:	Unisa Press, Graphic Artist
Title page layout:	Unisa Press, Graphic Artists
Document layout:	Hentie Wilson, CLD with MSWord2007, Special styles
Graphics:	Nigel Bishops
Printer and Binding:	Unisa Printers

Preface

This module is the first on the use of computers and computational methods in applied mathematics.

The other undergraduate modules are: COS2633 Numerical methods 1, APM2616 Computer algebra, APM3711 Numerical methods 2. This module, with COS2633 and APM3711, are about finding numerical solutions to problems; whereas APM2616 shows you how to solve problems symbolically.

The use of computers in mathematics has revolutionized the extent to which mathematical calculations can be applied to real-world problems. The modern world is full of such examples, such as Weather prediction, Drug design, Aircraft design, Electronic banking, and Telecommunication systems.

The teaching strategies imbedded include:

- Activity-based self-study.

The outcomes of this module are:

- To solve systems of linear equations with the use of Octave (or MATLAB)
- To perform basic matrix operations.
- To use iterative methods to find appropriate solutions for systems
- To know what is meant by the eigenvalue equations, to be able to calculate the eigenvalue of a matrics and its corresponding eigenvector and to know the Octave (or MATLAB) code to do it.
- To be able to solve linear programming problems by using the software.

The main sources used in writing this study guide were

- B.D. Hahn *Essential MATLAB for Scientists and Engineers* (Pearson Education South Africa, Cape Town, 2002)
- Previous Unisa study guides on (non-computational) applied linear algebra, originally written by G. Lemmer and revised by J. Hartney and R. Maritz.

Nigel Bishop

Table of Contents

Study Unit 1: Getting started with the computer software

TIME PERIOD: 10 hours approximately *RESOURCES: MATLAB/Octave programs*

LEARNING OUTCOMES

At the end of this Study Unit, you should be able to:

- Have Octave installed on your computer.
- Use Octave for simple calculations.
- Use Octave to produce simple graphs.
- Execute a simple Octave program by reading from a file.
- Produce Octave results as output, including graphs, to files.
- Use the Octave help facilities.

1.1 Installing and running the program (from TL101)

This module requires you to use a software programme and we decided to use Octave which is free of charge. MATLAB is another commercial software product similar to Octave, but that has to be purchased.

1.1.1 MATLAB or Octave?

Although there are occasional differences, the syntax of the two programming systems is almost identical. In some advanced, specialized applications, we found MATLAB was able to solve a problem but Octave was unsuccessful. However, for the introductory purposes of this module, the free Octave version is quite sufficient. You are welcome to use MATLAB rather than Octave if you wish, or if the computer that you are using for this module already has MATLAB installed. Be aware that there will be minor syntactical and layout differences between MATLAB and Octave; the notes in this study guide will only cover Octave so for a start we would suggest that you rather work with Octave.

1.1.2 Install the program, Octave

Before you can start work on this module, you need to *install the software* package onto your computer Desktop. (Information about how to do this is given in Tutorial Letter 101 because the software package is regularly updated and these details may change).

The installation process should show the **Octave icon** on your computer's Desktop. (If the icon is not there, go to the Start button, then **click** on **Programs**, then on **GNU Octave**, then on **Octave**. Otherwise, go back and redo the installation process).

NOTE: There may be minor differences in the layout in some cases because of version differences between the programs. The program version shown in this study guide was developed and tested using Octave version 3.0.0, running under the Windows operating system, and with *gnuplot* (as the graphics back-end).

Activity 1-1 Open the program

Double click on the program icon to open the program.

A new window should open that contains installation details. Details on the Octave Startup window should look like this:

```
GNU Octave, version 3.0.0
Copyright (C) 2007 John W. Eaton and others.
This is free software; see the source code for copying
conditions.
There is ABSOLUTELY NO WARRANTY; not even for
MERCHANTIBILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type
`warranty'.
Octave was configured for "i686-pc-msdosmsvc".
Additional information about Octave is available at
http://www.octave.org.
Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-
wanted.html
Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
helpful report).
For information about changes from previous versions, type
`news'.
 - Use `pkg list' to see a list of installed packages.
 - SciTE editor installed. Use `edit' to start the editor.
 - MSYS shell available (C:\Program Files\Octave\msys).
 - Graphics backend: gnuplot.
      octave-3.0.0.exe:1>
```

If you have problems to install the program, at first, try again. If you still have a problem contact the lecturer as this is a key step in starting.

1.1.3 Testing simple examples in the program

Now you are ready to investigate what Octave can do. In order to show what is the Octave program's INPUT and OUTPUT, we will in our directions type the Octave code in a different font, the `Courier font.`

Your input is indicated in **bold boxed,** so look for the *input fishbone prompt* **(>)**, then type your input.

For example,

Type the following calculation to use the program as a calculator.

```
>3.6+5.7
```

The programme should return the following OUTPUT:

```
ans = 9.3000
```

Activity 1-2 Test the program

Type in your own examples of calculations in the program. You are now investigating the ability of the program, Octave, to be used as a calculator.

Make a note of your own examples in your workbook/work file (you may make **Print Screen** copies, and **paste** them into a wordprocessor file, then **print** it out and paste it in your workbook as evidence of your work.)

FEEDBACK:
Were you able to do your own testing? If so, go to *myUnisa* Discussion Forums and share your problems with other students to see whether you get any reply from them. Your Tutor or Lecturer will monitor the discussions and support your study efforts.

If you have problems with the installation processor the testing activity, go to the Internet, with your Internet Explorer, to the Unisa student website, https://my.unisa.ac.za/portal.
Log in with your student number and password, then go to this module's website and its Discussion Forums, and ask for help there.
If you are unable to go online, phone 012 429-6202 for help from the department.

Remember, that in Octave the following symbols have specific meanings:

- * means `multiply`
- / means `divide`
- ^ means raise `to a power`, e.g. 3^2 means 3^2
- sqrt means `the square root`, e.g. sqrt(2) leads to 1.4142.

Test some more calculations of your own where you use the multiplication, division, power, and square root.

Now, let us look at an Example where we will use the program to do something interesting.

Activity 1-3 Solve equations

Use Octave to solve a pair of simultaneous equations, such as:

$1\, x_1 + 2\, x_2 = 5$, and $3\, x_1 + 4\, x_2 = 6$

FEEDBACK:
Let's do it together. In order to use Octave, we must first write the above equations in its matrix form, namely:

$A\, x = b$

(We will discuss such problems in more detail in Study Unit 3.)

We will now give you the Octave INPUT together with the OUTPUT.
(Remember that the part that you must type as INPUT is after the fishbone input prompt (>) and indicated in boxed **bold.)**

So, type the first row values; and separate rows with a colon, with or without a space after the ; and before the 3.)

```
> A=[1 2;3 4]
      A =
         1    2
         3    4
```

```
> b=[5;6]
      b =
         5
         6
```

```
> x=A\b
      x =
        -4.0000
         4.5000
```

Following this OUTPUT, you need to rewrite the required answer (in mathematical terms) as:

$x_1 = -4$, $x_2 = 4.5$

Activity 1-4 Plot a graph

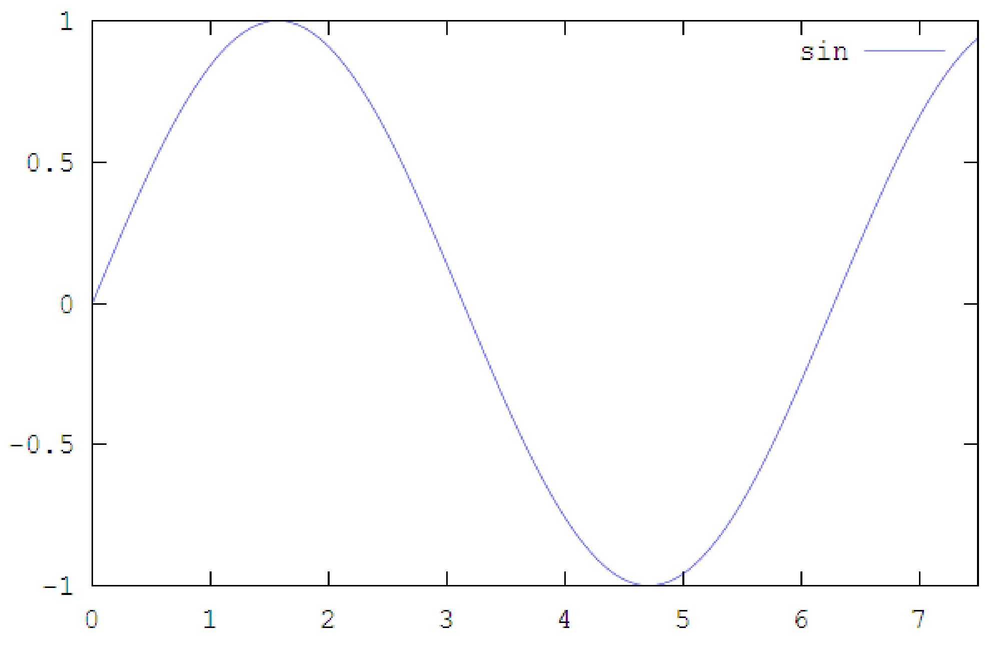
Use the program to plot a simple graph: sin(x) in the range 0<x<7.5

At the input prompt, type:

```
> fplot(@sin,[0,7.5])
```

The program produces a new window containing a graph of sin(x) in the range 0<x<7.5 (as shown below):

Table 1: Program-generated graph



1.1.4 Exit the program

Finally, exit your Octave session.

Activity 1-5 Exit or close the program session

Enter, at the input prompt, the command “quit”.

```
> quit
```

Now you know the basics about the program. Congratulations for getting this far.

1.2 Process to input files to the program, Octave

When you again open Octave, it assumes that the Home folder (that is the folder in which input and output files are located) is `C:\Program Files\Octave`. While it is possible to work in the Octave folder, it is better to use your own folder for your work, either in My Documents, or on the Desktop, or within your own structure.

1.2.1 Create a storage folder

The first thing to do is to **create the Home folder**, and once you have done so **open it** and ensure the path is given in the top of the window.

For example,

Suppose the Home folder is located in your My Documents folder at:

```
      C:\Documents and Settings\User\My Documents\apm113,
```

Then, to get Octave to use this as the Home folder, enter at the input prompt:

```
> cd "C:/Documents and Settings/User/My Documents/apm113"
```

Activity 1-6 Create storage folders

Create your own Home folder for your work in this module, “APM113” (upper or lower case are both accepted). For the first time, keep the file address as suggested above, in “Documents and Settings/User/My Documents”.

There are two important points to note about the above command that is different to the normal computer commands:

- The path appears inside quotation marks ("), so that the program will recognise the command;
- The symbol normally used (\) is replaced by forward slash (/).

Obviously, it can become somewhat tiresome to enter the above long address command every time you start an Octave session, so we will show you a shortcut.

Short cut:
Quit your Octave session to start a new session.
Confirm where the location of the default Home folder, by entering the command “**pwd**” (meaning, Print Working Directory).
Start editing by going to the SciTE-editor with the command “**edit**”.

Activity 1-7 Change the Home directory and name files

Find the working directory you are working in and change it to your own.
Ensure you work in the newly named Home folder, APM113.
Choose a file name, `xx.m` (where xx is **Test1**, or whatever you like). Enter, at the input prompt, **Test1.**

FEEDBACK:
Type the following:

```
> quit
```

Restart the program and type

```
> pwd
      ans = C:\Program Files\Octave
```

```
> edit
```

The result will be that a **new window** will appear (entitled, SciTE). In this window, enter just the one line to change to your new Home address directory you want to use, such as:

```
cd "C:/Documents and Settings/User/My Documents/apm113"
```

In this SciTE-editing window, in the Top bar, go to: **File**, then **Save As** (to save in the folder), `C:\Program Files\Octave`, (or \GNU Octave), (depending on the program version).

Save the file as Test1.m (or whatever name you choose). Now type in Octave:

```
> Test1
```

Test to see where you are:

```
> pwd
      ans = C:\Documents and Settings\User\My Documents\apm113
```

This illustrates a very important point: namely, that files can be created to execute a sequence of commands; through making "m-files". Read further to see the usage of m-files.

1.2.2 Using m files

Suppose that a file in what is currently the Home folder has an extension, ".m" (in other words, the file name is xx.m), and that the file contains Plain Text comprising a sequence of program commands. Then typing the file name without the extension (for Example xx) at the input prompt (>) will cause the sequence of Octave commands to be executed.

Use only Plain Text in the .m-file. So, use a plain text editor such as Notepad, or SciTE, or Wordpad (where the saved file type is "Text Document" - not as .rtf). (IMORTANT: Do not use a word processor, such as Word, as the file it produces keeps the formatting commands even though you cannot see it.)

Now, let us start using the program. Use the problem of two simultaneous equations in the next section as an example.

Activity 1-8 Save m-files

Open a SciTE-window (by entering "edit" at the Octave input prompt).
Enter the code for the problem to be solved (below).
Save the resulting file in the current Home folder, as your first example, `ex1.m`.

The content of the file `ex1.m` should be:

```
A=[1 2;3 4]
b=[5;6]
x=A\b
```

FEEDBACK:
Do the following INPUT in the program.

```
>edit
```

In the SciTE-window enter:

```
      A=[1 2;3 4]
      b=[5;6]
      x=A\b
Save the file with the name ex1.m. In Octave, type:
```

```
>ex1
```

When you enter `ex1` at the input prompt, you save the file in,

```
      C:\Documents and Settings\User\My Documents\apm113
```

The file name is called `ex1.m`.

Ensure your results or OUPUT is the same as below.

```
> ex1
      A =
         1   2
         3   4

      b =
         5
         6

      ans =
        -4.0000
         4.5000
```

Congratulations, now you know how to put information into the Octave program, so let's look at the next step.

1.3 Output from an Octave session

Having used a program to do some calculations, you will probably want to keep a record of the output in a file. There are several ways to do this such as using copy to Clipboard or diary.

1.3.1 Copy and paste to Clipboard

Click on the Octave icon in the top left corner of the Octave command window, for **Menu** to appear, click on **Edit**, then on **Mark**. Once you have done so, you can then ***highlight text*** in the Octave window in the usual way, and you copy it into the Clipboard by pressing the **Enter** key. **Paste** the copied text wherever you like.

For graphs, once you have produced a graph on the screen, **click Clipboard** icon on the far left in the Menu bar at the top of the graph window, and this will copy the plot to Clipboard. You can then **paste** the graph somewhere else, such as into a Word document.

Text in the Octave window is displayed using the `Courier font.` If you copy output into a Word document, and want to maintain the look of the original Octave session, then you should also use the Courier font in the Word document (as we have done in this guide).

1.3.2 Using a Diary file

From when you create a diary file, everything that appears in the Octave command window (both input and output) will be copied to the diary, keeping a record of your work.

Activity 1-9

At the Octave input prompt, enter the command "diary on".

```
> diary on
```

This has the effect of creating a file named diary in the Home folder. You can stop this at any stage by using the command

```
> diary off
```

You can then open the file diary with a text editor, such as the SciTE-editor, Notepad or Wordpad, and copy and paste parts of it to another document; but note that you must switch diary off, or end your Octave session, before you can edit it. Note also that, if the file diary already exists, the command "`diary on`" causes the new session to be appended to the end of the file, that is, you do not lose what is already there.

1.4 Using the Octave "Help" facility

The Octave system has a number of different facilities that you can use to get help. The simplest way is to use the command "`help`" but this only works if you know the exact name of the command about which you need more information, like "fplot".

You can get information that is much more detailed by using the command "`doc`". Scroll around in this mode using the arrow keys.

For example,

```
> help fplot
-- Function File:  fplot (FN, LIMITS)
 -- Function File:  fplot (FN, LIMITS, TOL)
 -- Function File:  fplot (FN, LIMITS, N)
 -- Function File:  fplot (..., FMT)
     Plot a function FN, within the defined limits.  FN an
be either a
```

```
     string, a function handle or an inline function.  The
limits of
     the plot are given by LIMITS of the form `[XLO, XHI]'
or `[XLO,
     XHI, YLO, YHI]'. TOL is the default tolerance to use
for the
     plot, and if TOL is an integer it is assumed that it
defines the
     number points to use in the plot.  The FMT argument is
passed to
     the plot command.
             fplot ("cos", [0, 2*pi])
             fplot ("[cos(x), sin(x)]", [0, 2*pi])
     See also: plot.
C:\Program Files\Octave\share\octave\3.0.0\m\plot\fplot.m
Additional help for built-in functions and operators is
available in the on-line version of the manual.  Use the
command
`doc <topic>' to search the manual index.
Help and information about Octave is also available on the
WWW
at http://www.octave.org and via the help@octave.org
mailing list.
-- less (100%) (f)orward, (b)ack, (q)uit
```

You exit the help mode by entering

```
q
```

Activity 1-10 Use Help

Go into the Help function. Scroll around. Exit it.

There is a complete user manual available in both html- and pdf-formats, located at:
C:\Program Files\Octave\doc\HTML\liboctave\index.html
C:\Program Files\Octave\doc\HTML\interpreter\index.html
C:\Program Files\Octave\doc\PDF\octave.pdf

The next activity focusses on getting you to use the Help function of the extensive program manual more readily, and to transfer knowledge into your everyday use of the program.

Activity 1-11 Write .m-file

Write a .m-file that uses Octave to solve the following problems.

1. Plot a graph of cos(x) in the range $-1<x<5$

2. Evaluate $9.2^{-0.5}$

3. Evaluate $\dfrac{\sqrt{7.3}}{2.4^3 + 3.1}$

4. Solve the simultaneous equations
   $2\,x_1 + 3\,x_2 = 10$
   $4\,x_1 + 5\,x_2 = 8$

5. Now, produce a Word document that contains the following:
   1) The content of the .m file;
   2) The output obtained from running the .m file;
   3) The graph.

6. Normally Octave outputs numbers with 5 significant figures.
   Use the Help facility with the keyword format to:
   1) find how to get output with 15 significant figures.
   2) Then evaluate $\sqrt{2}$ to 15 significant figures.

This skill is required throughout this module, so feel free to refer to the Help-function often for guidance. Keep evidence of your work (copy and paste your efforts to your workbook or make a printout from your diary file).

1.5 In conclusion

Well done you now have the program, Octave, working. You have used it to produce graphs and other simple calculations.

Now you are ready to start with the real work of the module. The next time you use the program you will find it becomes much easier, and easier.

Study Unit 2: Introduction to programming

TIME PERIOD: 30 hours approximately   *RESOURCES: Octave/MATLAB programs*

LEARNING OUTCOMES

At the end of this Study Unit, you should be able to use the computer software (such as Octave) to:

- Construct scalar, vector and matrix variables
- Manipulate variables to calculate new scalar, vector and matrix variables
- Use loops to repeat calculations
- Uses conditional tests, like “if” statements
- Construct user-defined functions

2.1 Introduction

This unit is a summary of the features of the Octave program that you will use throughout the module. If you would like to see more detail, you will find it in the extensive Octave manual.

If you are experienced in computer programming, you should not be tempted to skip this Study Unit. While you, as an experienced programmer, could go through this Study Unit quite quickly, you should not skip it altogether because there are important syntactical differences between Octave and other programming languages.

2.2 Scalar variables

In this section we deal with creating and naming variables.

2.2.1 Creating variables

A variable is created simply by assigning a value to it at the command line or in a program.

For example,

```
> a=17
      a =  17
```

This has the effect of creating a new variable “a” with value 17.

Alternatively, you can use the variable a also in another formula to test for “b”, such as:

```
> b=a^2
      b =  289
```

2.2.2 Variable names

Names of variables *must start* with a letter, and may include letters, numbers, as well as the underscore character (_). Names are case sensitive; therefore, A1 and a1 will represent different variables. It is important to note that assigning a variable name supersedes a meaning that may already exist. Thus, although it is permissible to define log=6.1 it would not be useful to do so because then you could no longer use the logarithmic function.

For example, a USELESS variable would be,

```
> log=6.1
      log =  6.1000
```

The following useful variables are pre-defined in Octave as mathematical constants:

e	2.7183
i	$\sqrt{-1}$
j	$\sqrt{-1}$
pi	3.1416

You can see the "currently defined variable" names by entering the command "who". Look at the last part in the program OUTPUT, the "local user variables", to see the variables you have defined yourself.

Activity 2-1 Check and remove variables

Determine the defined variable names in your program, both current and local user variables. Remove the definitions thereafter.

FEEDBACK:
Type in "who".

```
> who
      *** dynamically linked functions:
      __COM__       builtin: find  dispatch      getpwuid

      *** currently compiled functions:
      __default_graphics__  index                rindex
      edit                  ispc                 strcat
      fileparts             isunix               strrep
      findstr               lower
      fullfile              pkg

      *** local user variables:
      __nargin__  a           b           log
```

You can remove all these definitions, either by quitting and starting a new Octave session, or, by means of the command "clear".

```
> clear
```

It is a good idea to clear the workspace after finishing one problem and before starting another one, so as to avoid errors due to a variable having an unintended value.

2.3 Matrices and vectors

In this section, we deal with variables with more structure.

2.3.1 Construction of matrices and vectors

So far, we have defined variables that comprise just a single value, like scalars. However, the power of a programming language (like Octave or Matlab) is that it is easy to introduce and manipulate indexed data structures, such as vectors and matrices.

First, we need to construct vectors and matrices so, work through the following examples.

Activity 2-2 Construct matrices and vectors in the program

Explicitly construct:

1) a row vector with 3.3, 1.7, 2.1
2) a column vector with 3.3, 1.7, 2.1
3) a matrix , where the entries are known explicitly
4) a row vector, when the values change by equal increments
5) a row vector, when the increment is 1
6) a column vector when the values change by equal increments
7) A matrix, where some entries change by equal increments
8) A matrix, by combining an existing matrix with a column vector
9) A matrix by combining an existing matrix with a row vector
10) Transposition of a matrix

FEEDBACK:

Row vectors:

```
> v1=[3.3 1.7 2.1]
      v1 =
         3.3000    1.7000    2.1000
```

Column vectors:

```
> u1=[3.3; 1.7; 2.1]
      u1 =
         3.3000
         1.7000
         2.1000
```

A matrix:

```
> A1=[1 2 3;4 5 6;7 8 9]
      A1 =
         1   2   3
         4   5   6
         7   8   9
```

A row vector when the increment is 1:

```
> v3=1:4
      v3 =
         1.00000   2.00000   3.00000   4.00000
```

A row vector when the values change by equal increments (that are not 1):

```
> v2=1:-0.2:0.4
       v2 =
          1.00000    0.80000    0.60000    0.40000
```

A column vector when the values change by equal increments:

```
> u2=[1:-0.2:0.4]'
       u2 =
          1.00000
          0.80000
          0.60000
          0.40000
```

A matrix, where some entries change by equal increments:

```
> A2=[1:4;5:8;9.5:2:15.5;1 0 0 0]
       A2 =
           1.00000     2.00000     3.00000     4.00000
           5.00000     6.00000     7.00000     8.00000
           9.50000    11.50000    13.50000    15.50000
       .00000     0.00000     0.00000
```

A matrix, by combining an existing matrix with a column vector:

```
> A3=[A2 u2]
       A3 =
           1.00000     2.00000     3.00000     4.00000     1.00000
           5.00000     6.00000     7.00000     8.00000     0.80000
           9.50000    11.50000    13.50000    15.50000     0.60000
           1.00000     0.00000     0.00000     0.00000     0.40000
```

A matrix, by combining an existing matrix with a row vector:

```
> A4=[A2;v3]
       A4 =
           1.00000     2.00000     3.00000     4.00000
           5.00000     6.00000     7.00000     8.00000
           9.50000    11.50000    13.50000    15.50000
           1.00000     0.00000     0.00000     0.00000
           1.00000     0.80000     0.60000     0.40000
```

Transposition of a matrix:

```
> A5=A4'
       A5 =
           1.00000     5.00000     9.50000     1.00000     1.00000
           2.00000     6.00000    11.50000     0.00000     0.80000
           3.00000     7.00000    13.50000     0.00000     0.60000
           4.00000     8.00000    15.50000     0.00000     0.40000
```

2.3.2 General rules for construction of vectors and matrices

The above illustrates a number of general rules. NOTE: Refer to this when you are doing examples that are more complicated.

We use the following rules for the construction of vectors and matrices:

- Vectors and matrices are indicated by square brackets [ ]
- The elements of a row are separated by spaces
- The rows are separated by semi-colons (;)
- Using the colon (:) notation, the first entry in a row is the number before the (:), the last entry is the last number in the definition, and the entry between the two (:) is the increment. If the middle number is omitted, then the increment is 1.

2.3.3 Accessing elements of a vector or matrix

We can access the value of an element of a matrix or vector by the notation ---

- A(row index, column index) for a matrix
- v(index) **or** v(1,index) for a row vector
- u(index) **or** u(index,1) for a column vector

We illustrate this by some activities, with u2, v3 and A2 (defined as above).

Activity 2-3

Find the values of the following elements of a vector or matrix:
1) In a row vector, v3(2)
2) in a row vector, v3(3)
3) In a column vector, u2(4)
4) in a column vector, u2(3)
5) in a matrix, A2(2,3)
6) in a matrix, A2(3,2)

FEEDBACK:
For a row vector:

```
> v3(2)
       ans =  2
```

```
> v3(1,3)
       ans =  3
```

Note v3(1,3)=v3(3)

For a column vector:

```
> u2(4)
       ans =  0.40000
```

```
> u2(3,1)
       ans =  0.60000
```

Note that u2(3,1)=u2(3)

For a matrix:

```
> A2(2,3)
       ans =  7
```

```
> A2(3,2)
       ans =  11.500
```

2.3.4 *Accessing rows and columns of a matrix*

We can access the rows and columns of a matrix by the notation

- A(row index, :) for a row of a matrix
- A(:,column index) for a column of a matrix

Activity 2-4

Access the rows and columns of a matrix, with A2 defined as above, using the notation as above.

FEEDBACK:

```
> A2(2,:)
      ans =
         5    6    7    8
```

```
> A2(:,3)
      ans =
           3.00000
           7.00000
         13.50000
           0.00000
```

2.3.5 *Scalars as vectors and matrices*

We have already seen that Octave treats a row vector as being a matrix with a single row, and a column vector as being a matrix with one column (because v(1,index) and u(index,1) are valid statements).

In the same way, Octave treats a scalar quantity as being a matrix with only the (1,1) element, and this is equivalent to it being a vector with just one component. Therefore (in Octave), there is essentially no difference between scalars, vectors and matrices.

Activity 2-5

Clear your previous work.
Check that a; a(1); a(1,1) all have the same value

FEEDBACK:

```
> clear
```

```
> a=7.342
      a =  7.3420
```

```
> a
      ans =  7.3420
```

```
> a(1)
      ans =  7.3420
```

```
> a(1,1)
      ans =  7.3420
```

2.3.6 Special matrices

A number of special matrices are pre-defined in Octave, and here we give 4 cases that will be useful to you later.

a) ***CASE 1***

The statement `zeros(m,n)` defines a matrix with m rows and n columns whose entries are all 0.

For example,

```
> zeros(4,6)
      ans =

        0  0  0  0  0  0
        0  0  0  0  0  0
        0  0  0  0  0  0
        0  0  0  0  0  0
```

b) ***CASE 2***

The statement `ones(m,n)` defines a matrix with m rows and n columns whose entries are all 1.

For example,

```
> ones(3,2)
      ans =

         1   1
         1   1
         1   1
```

c) ***CASE 3***

The statement `rand(m,n)` defines a matrix with m rows and n columns whose entries are random numbers in the range 0 to 1.

For example,

```
> rand(4,5)
      ans =

         0.6785508   0.1797487   0.1222308   0.5802859   0.0345841
         0.9743568   0.2243425   0.1427010   0.3802661   0.5009699
         0.6864207   0.1635793   0.4904312   0.2338945   0.6085457
         0.5685229   0.1613623   0.9038765   0.8830253   0.0088788
```

d) ***CASE 4***

The statement `eye(n)` defines the square identity matrix with n rows and n columns.

For example,

```
> eye(5)
```

```
ans =

   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
   0   0   0   1   0
   0   0   0   0   1
```

2.4 Manipulation of matrices

We can produce new vectors and matrices from those that we have already constructed. We have already previously seen the following:

- **The transpose operator** ('), for example, in `A5=A4'`
- **Adding a row or a column to a matrix**, for example, in `[A2 u2]`, or `[A2;v3]`.

It is also straightforward to get the inverse and delete a row and column from matrices.

a) ***Construct the inverse of a square matrix using the operator inv***

For example,

```
> A6=rand(5,5)
A6 =
   0.228451   0.744584   0.311120   0.412804   0.415285
   0.752986   0.990980   0.539382   0.236439   0.684900
   0.079324   0.587490   0.386354   0.543607   0.880370
   0.052009   0.885506   0.522269   0.963522   0.262930
   0.392542   0.796100   0.162770   0.461631   0.082264
```

```
> A7=inv(A6)
A7 =
  -7.124120   1.774712   1.615726   0.305008   2.922358
   7.035698  -1.304923  -1.866377  -1.131128  -1.064458
   0.157018   2.020572  -2.009178   2.431010  -3.883385
  -6.063413   0.067611   2.244590   1.005264   2.812304
  -0.378068  -0.217589   1.731535  -0.960240   0.414733
```

b) ***Delete a row or a column from a matrix:***

For example, where you delete the 2nd row of A6 and the 3rd column of A7.

```
> A6(2,:)=[]
A6 =
   0.228451   0.744584   0.311120   0.412804   0.415285
   0.079324   0.587490   0.386354   0.543607   0.880370
   0.052009   0.885506   0.522269   0.963522   0.262930
   0.392542   0.796100   0.162770   0.461631   0.082264
```

```
> A7(:,3)=[]
A7 =
  -7.124120   1.774712   0.305008   2.922358
   7.035698  -1.304923  -1.131128  -1.064458
   0.157018   2.020572   2.431010  -3.883385
  -6.063413   0.067611   1.005264   2.812304
  -0.378068  -0.217589  -0.960240   0.414733
```

This enables you to construct any form of matrix or vector that you want or need. Now let's go further to see what else is important.

2.4.1 *Calculations with vectors and matrices using matrix algebra*

Octave as program recognizes two different meanings that can be given to arithmetic operations on vectors and matrices.

2.4.2 *Matrix algebra*

In the first case, the operators have the same meaning as used in matrix algebra.

a) ***Case 1***

Addition and subtraction, $A + B$, and $A - B$, are defined whenever A and B are matrices of the ***same size***, namely, they each have the same number of rows and columns.

Activity 2-6 Calculate the matrices with + and -

Calculate the matrices: A8=rand(2,3); B1=ones(2,3); A8+B1

FEEDBACK:

```
> A8=rand(2,3)
      A8 =
         0.35475   0.95425   0.70169
         0.18010   0.15871   0.60431

> B1=ones(2,3)
      B1 =
         1   1   1
         1   1   1

> A8+B1
      ans =
         1.3548   1.9542   1.7017
         1.1801   1.1587   1.6043
```

b) ***Case 2***

Multiplication, $A\ B$, is defined whenever A and B are matrices with the number of columns in A ***equal to the number of rows*** in B, or, if either A or B ***is a scalar***.

Activity 2-7 Calculate the matrices with x

Calculate the matrices: A9=rand(2,3); B2=rand(3,4); A9*B2

FEEDBACK:

```
> A9=rand(2,3)
      A9 =

         0.795470   0.401763   0.840127
         0.918606   0.521561   0.087270

> B2=rand(3,4)
```

```
B2 =

   0.59073   0.14314   0.74667   0.34705
   0.75876   0.39622   0.19848   0.20565
   0.57285   0.47794   0.56639   0.50184
```

```
> A9*B2
```

```
ans =

   1.25602   0.67458   1.14954   0.78030
   0.98839   0.37985   0.83885   0.46986
```

```
> 2.3*A9
```

```
ans =

   1.82958   0.92406   1.93229
   2.11279   1.19959   0.20072
```

c) ***Case 3***

Raising to a power, A^n, is defined whenever A is a square matrix, and it means A x A x A ... (n times).

Activity 2-8 Calculate the matrices with a A^n

Calculate the matrices: A10=rand(3,3); A10^2; A10^5

```
> A10=rand(3,3)
```

```
A10 =

   0.85898   0.12942   0.92354
   0.73955   0.88709   0.30950
   0.16907   0.29217   0.70776
```

```
> A10^2
```

```
ans =

   0.98971   0.49582   1.48702
   1.34363   0.97307   1.17662
   0.48096   0.48785   0.74750
```

```
> A10^5
```

```
ans =

   3.8196   2.7376   4.9474
   5.0490   3.5300   6.4821
   2.3903   1.6891   3.0186
```

d) ***Case 4***

Division is not straightforward in matrix algebra. Here we will introduce the Octave operators \ and / within the context of solving a system of simultaneous equations (In fact, the operators \ and / can be used more generally, but we will not discuss that).

For example,

Suppose that we have n by n square matrix A, and column vectors x and b both of length n, and that we want to solve for x in

$$A\,x = b.$$

Then,

$$x = A^{-1}\,b.$$

The Octave notation for the above is

```
x=A\b.
```

Of course, we could also write in Octave `x=inv(A)*b`, but this is less efficient so the \ operator is preferred.

For example,

Suppose instead that we want to solve

$$y\,A = c$$

with y and c now *row vectors* of length n. Then

$$y = c\,A^{-1}.$$

The Octave notation for the above is

```
y=c/A
```

Of course, we could also write in Octave `y=c*inv(A)`, but again this is less efficient.

Activity 2-9 Calculate the matrices with division

Calculate the matrices:	A11=rand(3,3);	b=rand(3,1);	c=rand(1,3)
	x=A11\b;	inv(A11)*b;	y=c/A11;
	c*inv(A11)		

```
> A11=rand(3,3)
      A11 =
         0.088404   0.120302   0.727375
         0.940880   0.350956   0.817928
         0.373165   0.451789   0.714572
```

```
> b=rand(3,1)
      b =
         0.093734
         0.363911
         0.892151
```

```
> c=rand(1,3)
      c =
         0.87188   0.98920   0.42356
```

```
> x=A11\b
      x =
        -0.39420
         2.73657
        -0.27583
```

```
> inv(A11)*b
      ans =
        -0.39420
         2.73657
        -0.27583
```

```
> y=c/A11
```

```
y =
  -2.147898   0.048056   2.724118
```

```
> c*inv(A11)
ans =
  -2.147898   0.048056   2.724118
```

2.4.3 Element-by-element calculations with arrays (vectors and matrices)

The second meaning that Octave recognizes for arithmetic operations on vectors and matrices is to do the calculations on an element-by-element basis. Actually, in this case, the objects with which we work are not vectors or matrices (because they do not satisfy the required rules), so it is better to call them ***arrays***, and talk about element-by-element operations on the arrays.

For example,

Suppose that we have a 1 x *n* array *v* and we want to construct a new array *u*, each element of which is the square of the corresponding element in *v*. In mathematical notation, what we want is

$$u_k = v_k^2, \text{ for each value of the index } k.$$

In Octave, if we write u=v^2 we will get an error message, so instead we use the operator (.^) to indicate that we are doing a calculation element-by-element, by writing u=v.^2.

Activity 2-10 Calculate the matrices of arrays

Calculate the matrices: v=rand(1,4); u=v^2; u=v.^2

```
> v=rand(1,4)
v =
   0.295962   0.194619   0.078277   0.839960
```

```
> u=v^2
error: for A^b, A must be square
error: evaluating binary operator `^' near line 53, column 4
error: evaluating assignment expression near line 53, column
2
```

```
> u=v.^2
u =
   0.0875937   0.0378766   0.0061273   0.7055324
```

NOTE: If you are doing matrix calculations there will be NO .^, but in element-by-element arrays calculations you will need the full stop (.).

Here are some more examples of valid element-by-element calculations on arrays.

Activity 2-11 Calculate more arrays

Calculate the arrays: A12=rand(4,3); B3=rand(4,3); A12.*B3.*1.8; A12.^B3

```
> A12=rand(4,3)
```

```
      A12 =
         0.95540   0.56061   0.65058
         0.54163   0.53082   0.11537
         0.14583   0.27776   0.53204
         0.22211   0.63209   0.37170
```

```
> B3=rand(4,3)
      B3 =
         0.787186   0.138921   0.080241
         0.390426   0.846420   0.913880
         0.561953   0.374361   0.883199
         0.039100   0.810616   0.554635
```

```
> A12.*B3.*1.8
      ans =
         1.353745   0.140185   0.093965
         0.380638   0.808729   0.189779
         0.147506   0.187165   0.845815
         0.015632   0.922290   0.371081
```

```
> A12.^B3
      ans =
         0.96472   0.92275   0.96609
         0.78710   0.58504   0.13895
         0.33893   0.61906   0.57274
         0.94287   0.68946   0.57758
```

```
> A12./B3./3
      ans =
         0.404565   1.345152   2.702612
         0.462424   0.209044   0.042080
         0.086500   0.247316   0.200800
         1.893515   0.259922   0.223388
```

From the above, we see that element-by-element calculations obey the following rules:

- The operators are ".+", ".-", ".*", "./", ".^".
- Each array in the expression must have the same size, i.e. all arrays must have the same number of rows *m* as well as the same number of columns *n*
- One of the operands for .* ./ .^ may be a scalar.

Activity 2-12 Construct vectors

Set up a row vector b with elements 1, 2, 3, 4, 5. Use array operations on b to set up the following vectors, each with 5 elements:

a) 2, 4, 6, 8, 10

b) ½, 1, 3/2, 2, 5/2

c) 1, ½, 1/3, ¼, 1/5

d) 1, 4, 9, 16, 25

FEEDBACK

Were you able to construct all the above vectors?

Here is a solution to c):

```
> 1./b
```

2.4.4 Extraction of some commonly-needed properties of matrices and vectors

The operators `size`, `length` and `sum`, give useful information about matrices and vectors.

- **Use** **`size(matrix_name)`**

This gives the number of rows, followed by the number of columns, in the matrix.

For example

```
> A13=ones(4,3)
       A13 =
          1   1   1
          1   1   1
          1   1   1
          1   1   1
```

```
> size(A13)
       ans =
          4   3
```

- **Use** **`length(vector_name)`**

  This is used to find the number of elements in a vector. Formally, it returns the greater of the number of rows or the number of columns in an array, but it is better practice to apply it only to vectors.

For example

```
> v=1:5
       v =
         1  2  3  4  5
```

```
> length(v)
       ans =  5
```

- **Use** **`sum(matrix_name)`**

  This adds up the elements in each column of a matrix - with the exception of a row vector in which case it adds up the elements in the row.

For example

```
> sum(v)
       ans =  15
```

```
> sum(A13)
       ans =
          4   4   4
```

2.5 Miscellaneous features of Octave

If you look at the Octave manual, you will see that there are many features and commands available. Here we describe just a few of them that you should find particularly useful.

2.5.1 Suppression of output with the semi-colon (;) terminator

The normal result of executing a statement in Octave – both in a .m file and in the command window – is that the answer is displayed in the command window. Of course, you always want to see the final answer of a calculation, but sometimes it is neater to suppress the output of intermediate steps. This is done using the semicolon statement terminator.

For example

```
> A14=ones(2,3);
> A14
      A14 =
         1   1   1
         1   1   1
```

2.5.2 *Multiple line statements with" ..."*

Sometimes an Octave statement can be quite long, and it would be convenient to split it over several lines. This is achieved by means of ... (as shown in the following example). The "..." construct can be used in both the command window as well as in a .m file.

For example

```
> A15=[1 2 3 4 5 6 7 8 9;12 1 3 14 15 16 17 18 19;...
> 21 22 23 24 25 26 27 28 29]
      A15 =
          1    2    3    4    5    6    7    8    9
         12    1    3   14   15   16   17   18   19
         21   22   23   24   25   26   27   28   29
```

2.5.3 *Several statements on the same line*

It is permissible to write several statements on the same line, separating the statements with either a semi-colon (in which case the output is suppressed), or with a comma (in which case the output is shown).

For example,

```
> c1=2,c2=c1^2;c3=c2^2
      c1 =  2
      c3 =  16
```

2.5.4 *The display command "disp"*

When Octave outputs the value of a variable, it appears in the form variable (=) value.

For example

```
> c1=2
c1 = 2
```

Sometimes, you want to suppress the "variable =" part, for example when producing a Table. You can also use the `disp` command to place text into the Octave output.

For example

```
> disp(c1)
 2
```

```
> disp("Place text here ...")
      Place text here ...
```

2.5.5 *Input statement*

The input statement causes something to be displayed on the screen in the Octave window, and then the system waits for the user to type in a statement in that window. Usually, this facility is used within a .m-file and asks the user to specify a parameter needed in the calculation.

Activity 2-13

Write a program that asks the user to enter a square matrix A, and then to enter a column vector b, and then to request a result using Return, to find or solve the system of equations *Ax=b.*

FEEDBACK:

File ex2.m

```
input("Enter a square matrix A  ");
input("Enter a column vector b  ");
input("Press Enter to find the result of solving Ax=b");
x=A\b
```

Octave window

```
> ex2
      Enter a square matrix A, A=[1 2;3 4]
      Enter a column vector b, b=[5;6]

      Press Enter to find the result of solving Ax=b
      x =
        -4.0000
         4.5000
```

2.5.6 *Complex numbers*

Handling complex numbers in Octave is easy! As was said earlier as part of the built-in functions, $\sqrt{-1}$ is represented by `i` or `j`.

Find the $\sqrt{2+3i}$

The variable `z` with real part 2 and imaginary part 3 is created by the statement,

```
> z=2+3*i.
```

All the arithmetic operators and most functions work with complex numbers, for example,

```
> sqrt(z)
      ans =  1.67415 + 0.89598i
```

There are some functions that are specific to complex numbers, and if applied to a real number, they simply treat it as a complex number with zero imaginary part.

For example

```
> real(3+2*i)
ans =  3
> imag(3+2*i)
ans =  2
> abs(3+2*i)
ans =  3.6056
> conj(3+2*i)
ans =  3 - 2i
```

If you are working with complex numbers, **be careful about the following:**

- DO NOT USE by `i` or `j` as variables anywhere in your program – the effect would be to change the default meaning from $\sqrt{-1}$
- For a matrix with complex elements, the transpose operator (`'`) means to take the *complex conjugate transpose*, that means, rows and columns are *interchanged* and also the signs of the imaginary parts are *changed*.

2.5.7 Operator precedence rules

The issue here is how the program assigns meaning to something like x=a*b+c

Activity 2-14

Will Octave treat this as `x=(a*b)+c` or as `x=a*(b+c)`?

FEEDBACK:
The answer is `x=(a*b)+c`, because the operator `*` has precedence over the operator `+`.

The operator precedence rules in Octave, in order of increasing precedence, are:

```
+ - * / \ .\ .* ./ ^ .^
```

(For more full details, go to the Octave manual, under Expressions, Arithmetic Operators, Section 8.8 Operator Precedence.)

However, having said this, it is much better practice to write code that makes extensive use of parentheses `()` so as to avoid errors due to Octave interpreting your code differently to what you intended. Thus, for example, the entries in the left and right columns below are equivalent, but it is better to write code with parentheses as in the right column.

For example,

```
      x=a^b*c       x=(a^b)*c
      x=a*b+c       x=(a*b)+c
      x=a/b/c       x=a/(b*c)
```

2.5.8 Comments (%)

Comments can be included in Octave code using the `%` symbol, so everything to the right of the `%` is ignored by the Octave interpreter. Comments are usually made in .m files, and you can comment out a whole line or just part of a line. Below we show how we might include comments in the file `ex1.m` (from Study Unit 1).

For example,

```
%Octave script to solve the matrix equation Ax=b
A=[1 2;3 4] %The matrix A
b=[5;6] %The column vector b
x=A\b %The answer x
```

2.6 Control commands: Loops and branches

2.6.1 "For" loops

Often in programming, you want to repeat an action a fixed number of times. In Octave this is achieved by means of a "for" loop. We start with an example, and then give the general syntax.

Activity 2-15

In the following Octave code, find *n!* (n-factorial), with *n* between *1* and *10*, where
n! = 1 x *2* x 3 ... x *(n-1)* x *n*.

FEEDBACK:
In this case, you need to write the Octave code as a .m file.

File fac.m

```
n=10;
fact=1;s
for k=1:n
  fact=k*fact;
  factorials(k,:)=[k fact];
end
factorials
```

Ocatave window

```
> fac

factorials =
         1         1
         2         2
         3         6
         4        24
         5       120
         6       720
         7      5040
         8     40320
         9    362880
        10   3628800
```

The general form of the loop is:

```
for index=start_value:increment:end_value
  statements
end
```

(If the `increment` is omitted, it is assumed to be 1.)

Activity 2-16

Evaluate the following series:
$1^2 + 2^2 + 3^2 + \ldots + 100^2$

FEEDBACK
Did you get the right answer? It is 338350.

2.6.2 Matrix or array operations versus "for" loops

Often, the same effect can be achieved by using array or matrix arithmetic as by using a "for" loop.

Activity 2-17

Let us look again at the problem of constructing a vector, each element of which is the square of the corresponding element in a given vector.

FEEDBACK:
We saw that the following code solves the problem.

```
> v=rand(1,4)
      v =
         0.295962    0.194619    0.078277    0.839960

> u=v.^2
      u =
         0.0875937    0.0378766    0.0061273    0.7055324
```

Alternatively, using for loops, we could instead write the last statement as

```
> for k=1:4,u(k)=v(k)^2;end
> u
      u =
         0.0875937    0.0378766    0.0061273    0.7055324
```

Activity 2-18

Which should you rather use, array and matrix operations, or for loops?

This is a matter of programming style, and there is no definite answer. However, we can say that an important factor is that code should be written in a way that is easy for someone else to understand, which usually means that the simpler the code is the better. So, in the above example, we would prefer the array arithmetic option (`u=v.^2`), and that is usually the case (provided this can be done in a straightforward way). We suggest that the calculation of factorials is best done using a "for" loop.

2.6.3 If statements

The general form of an "if" statement is:

```
if (CONDITION)
  statements
elseif (CONDITION)
  statements
elseif (CONDITION)
  statements
...
else (CONDITION)
  statements
endif
```

Here is an example (as given in the online Help)

```
>          x = 1;
>          if (x == 1)
>            disp ("one");
>          elseif (x == 2)
>            disp ("two");
>          else
>            disp ("not one or two");
>          endif
           one
```

Let us look at another example.

Activity 2-19

A bank wants to calculate the interest due to its customers and add this to their accounts. This is complicated since the interest rate applicable depends on the initial balance, as follows:

Initial balance	Interest rate
Less than R1 000	0%
R1 000 to R5 000	5%
R5 000 to R10 000	8%
R10 000 to R20 000	9%
Above R20 000	10%

The Octave code is placed in the file bank.m, and the code is:

File bank.m

```
if (oldbalance < 1000)
  rate =0;
  elseif (oldbalance <5000)
    rate=0.05;
  elseif (oldbalance <10000)
    rate=0.08;
  elseif (oldbalance <20000)
    rate=0.09;
  else
    rate=0.1;
endif

newbalance=oldbalance*(1+rate)
```

Ocatave window

```
> oldbalance=30000
      oldbalance =  30000
> bank
      newbalance =  33000
> oldbalance=3000
      oldbalance =  3000
> bank
      newbalance =  3150
> oldbalance=300
      oldbalance =  300
> bank
      newbalance =  300
```

The coding of complicated conditions may best be done using a “`switch`” statement (you can find out more about this command from the online Help facility).

2.6.4 Evaluation of conditions

“If” statements (and other constructs that we will come across later), take a specified action *provided that a condition is true*.

A condition is something that can be either true or false, and is of the form

```
      a operation b
```

where `a, b` are scalars and where `operation` is one of the following:

Octave code	Mathematical meaning
`<`	$<$
`<=`	$\leq$
`>`	$>$
`>=`	$\geq$
`==`	$=$

If you want to test for equality, use == rather than =, for example

```
      > if(a==b)
```

However, you should test for equality with caution, using it only with integers and not with real numbers. The reason is because of round-off error. The exact result of a calculation may be `a=1` but it could well be that what is stored in the computer is `a=1+10`$^{-15}$, and a test `if(a==1)`would produce the result false. So instead use `if(abs(a-1)<10^(-10))` where `abs` is the absolute magnitude function.

In mathematical language you may write something like, if (a<x<b), but you should not write the equivalent in Octave, since there should be only one operation in each condition expression. In Octave, you achieve the desired effect by using the additional operators:

Octave code	Mathematical meaning
`&`	and
`|`	or

So the Octave equivalent of if (a<x<b) is

```
      if((a<x) & (x<b))
```

2.6.5 Conditional loops

In a “for” loop, the loop is executed a predetermined number of times. Often, you would want to exit the loop when some condition that is a result of the calculation being done is satisfied.

In Octave you achieve this by using `while` or `do until` loops. The general form of the these constructs is

```
            while(CONDITION)
              statements
            endwhile
      and
            do
              statements
            until(CONDITION)
```

The difference between the two constructs is that in the “`do until`” version the loop is executed at least once.

Activity 2-20

Suppose that you go to the bank with the variable interest rate described above. You have an amount of money to invest, and want to see how long you will have to wait until the investment grows to a target amount. The program is in the file `ex3.m`. Note, it makes use of the file `bank.m` .

FEEDBACK:

File ex3.m

```
startbalance
oldbalance=startbalance;
years=0;
while(oldbalance<targetbalance)
  bank;
  years=years+1
  oldbalance=newbalance;
endwhile
```

Ocatave window

```
> startbalance=4100
      startbalance =  4100
> targetbalance=6000
      targetbalance =  6000
> ex3
      startbalance =  4100
      newbalance =  4305
      years =  1
      newbalance =  4520.3
      years =  2
      newbalance =  4746.3
      years =  3
      newbalance =  4983.6
      years =  4
      newbalance =  5232.8
      years =  5
      newbalance =  5651.4
      years =  6
      newbalance =  6103.5
      years =  7
```

Activity 2-21

Evaluate the following series $\sum_{n=1}^{\infty} u_n$ in which u_n is not known explicitly but is given in terms of a recurrence relation. You should stop the summation when $|u_n| < 10^{-8}$.

$u_{n+1} = (u_n)^2$ with $u_1 = 0.5$

FEEDBACK
Did you get the right answer? It is 0.81642. If you have problems with this activity, go online and discuss it with your fellow students.

2.7 Functions

The general idea of a function is that it is a piece of code that reads in some form of data, manipulates it in some way, and produces some other form of data. Octave has many pre-defined functions, but more importantly it allows the user to create new functions, and proper use of this feature is essential in the development of well-structured programs in Octave.

2.7.1 Pre-defined functions

We have already come across a number of examples of functions.

Here is a list of some common mathematical functions that are implemented in Octave:

```
cos sin tan sec cosec cot log exp sinh cosh tanh abs sqrt
```

There are many, many predefined functions (see the Octave manual, Function Index, pages 591-602). These functions can be applied to a scalar or to an array, and when applied to an array the function operates on every element of the array.

For example, look at the next activity.

Activity 2-22

Find a matrix B whose elements are the exponential function (exp) applied to each element of a matrix A

FEEDBACK:

```
> A=rand(2,2)
       A =
          0.596494   0.308671
          0.073614   0.176858
```

```
> B=exp(A)
       B =

          1.8157   1.3616
          1.0764   1.1935
```

2.7.2 User-defined functions

You can define your own functions also. Some important points to note about function definitions include:

- You must choose actual names for the `return_name, function_name` and *inputs*, and the Octave statements in the function must calculate the `return_name` from the inputs.
- The code defining the function must be placed in a separate .m file called `function_name.m.`
- Once a function has been defined, you can call it from the Octave command window, from an Octave script (ordinary .m file), or from another function.
- A function can have several input slots, and the `return_name` can be a scalar or an array.
- From a technical point of view, data is passed to an Octave function by value rather than by reference, and variables used inside a function are hidden from the calling program. This means that, even if you have statement `input1=1` the value of the variable equivalent to input1 in the calling statement will not be reassigned to 1.

Activity 2-23

Define your own functions, using the syntax below.

SYNTAX:

```
function return_name=function_name(input1,input2,
...,inputn)
  Octave statements that use the inputs to calculate
return_name
endfunction
```

Here is an example.

Activity 2-24

Find a function that computes the roots of the quadratic equation

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

using the formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

FEEDBACK:

We create a function file quadratic.m and then call it from the Octave command window.

File quadratic.m

```
function x = quadratic(a,b,c)
  disc=sqrt(b^2-4*a*c);
  x(1)=(-b+disc)/(2*a);
  x(2)=(-b-disc)/(2*a);
endfunction
```

Ocatave window

```
> z=quadratic(1,-5,6)
      z =
         3    2

> disc
      error: `disc' undefined near line 21 column 1
```

The last statement demonstrates that, although disc must have been computed to have value 1 during the execution of quadratic, this value is hidden from the main Octave window.

2.7.3 Function handles

A function "handle" is a means of passing the name of a function, rather than a value (i.e. a real or complex number), to another function. It is really quite simple, and is often required when using one of Octave's pre-defined functions.

For example,

Suppose that we want to evaluate numerically the area under the curve *f(x)* in the range $a < x < b$, like,

$$\int_a^b f(x)dx$$

In Octave, this is achieved using the predefined function "`quad`". The "calling" syntax is

```
quad(@f,a,b)
```

So, the "@" symbol indicates the created "handle" of the function. Let's look at the following activity where the code illustrates this in another example.

Activity 2-25

Using a program, calculate, $\int\limits_0^{\pi/6} \sin(x)dx$

FEEDBACK:

```
> quad(@sin,0,pi/6)
      ans =  0.13397
```

Activity 2-26

Using a program, evaluate

$$\int\limits_1^3 \frac{dv}{1+v^{1.8}}$$

FEEDBACK:
The first step is to write a function file, say fv.m, that defines the integrand, and then use `quad` from the command window. Note that we use *array arithmetic*

operators (./ and .^) in `fv.m` because (for certain later applications) it is convenient that the function be able to operate on an array.

So, write an m-file and calculate `y=fv(x)`.

File fv.m

```
      function y=fv(x)
        y=1./(1+x.^1.8);
      endfunction
```

Ocatave window

```
> quad(@fv,1,3)
      ans =  0.50214
```

Finally, we look at how a function handle that is passed to another function can be used in that function. In other words, we look at how a function like `quad` would deal with the function passed to it. Actually, the internal workings of `quad` are fairly complicated, so instead let us look at the problem of calculating the average slope of a function *f(x)* over an interval *0< x <b*.

Activity 2-27

Calculate the average slope of a function *f(x)* over an interval *0< x <b,* defined as

$$\frac{f(b)-f(a)}{b-a}.$$

FEEDBACK:
We write a function file, "average_slope.m", and use it to find the average slope of the function fv defined above in the interval (1,3), as well as the average slope of the sine function in the interval (0,π).

NOTE: In the function file, `average_slope.m`, the program does not permit us to write, say, `f(b);` instead, we use `feval(f,b)`.

File, average_slope.m

```
      function x=average_slope(f,a,b)
        fb=feval(f,b);
        fa=feval(f,a);
        x=(fb-fa)/(b-a);
      endfunction
```

Ocatave window

```
> average_slope(@fv,1,3)
      ans = -0.18921
> average_slope(@sin,0,pi)
      ans = 3.8982e-017
```

2.8 Graphics

2.8.1 Plotting functions with "`fplot`"

You have already, in Study Unit 1, been introduced to this command. The usual syntax for this command is:

```
        fplot(@function_name,[lower_limit,upper_limit])
```

The example given in Study Unit 1 was:

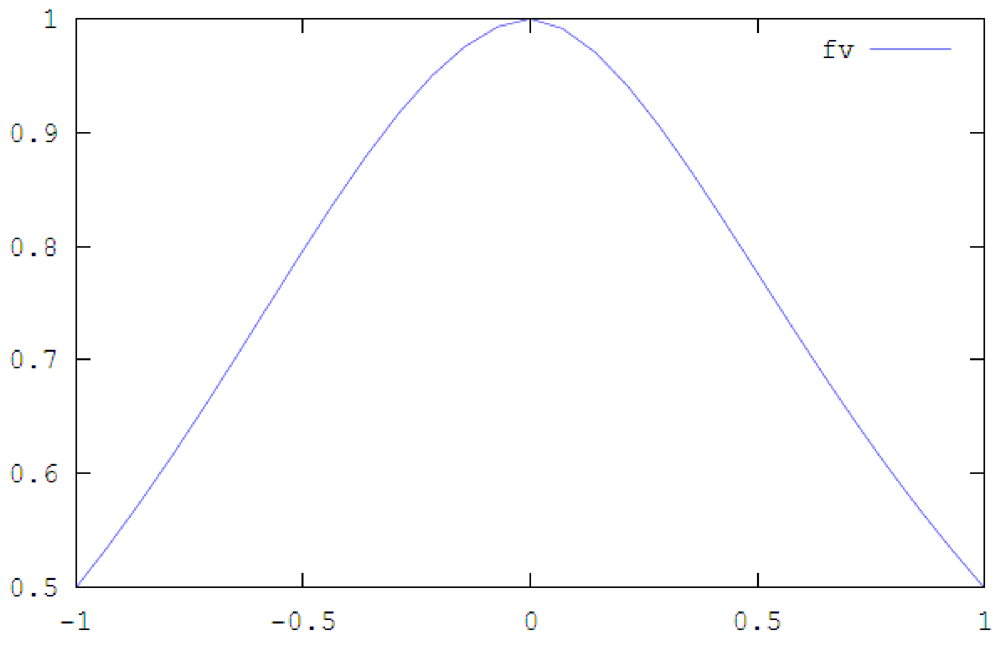
```
> fplot(@sin,[0,7.5])
```

You can also use `fplot` with a user-defined function, **provided** that the arithmetic operators used in defining the function are array operators, i.e. .* ./ .^ etc. Do the next example, together with the graph, for the function fv, defined above.

Activity 2-28

Plot the function fv in the range -1 <x < 1.

```
> fplot(@fv,[-1,1])
```



2.8.2 Plots in 2-dimensions with "`plot`"

The basic syntax is

```
        plot(x,y)
```

where `x,y` are row-vectors of the same length.

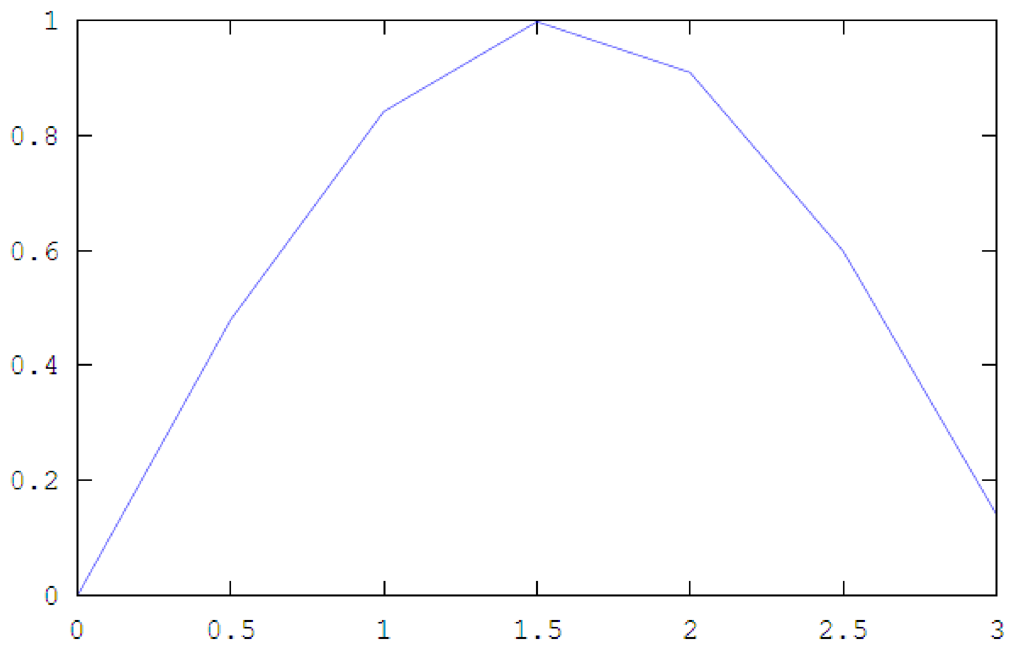
Activity 2-29

Plot a function in 2-dimenstions.

```
> x=0:0.5:3
        x =
       0.00000    0.50000    1.00000    1.50000    2.00000    2.50000
    3.00000
```

```
> plot(x,sin(x))
```

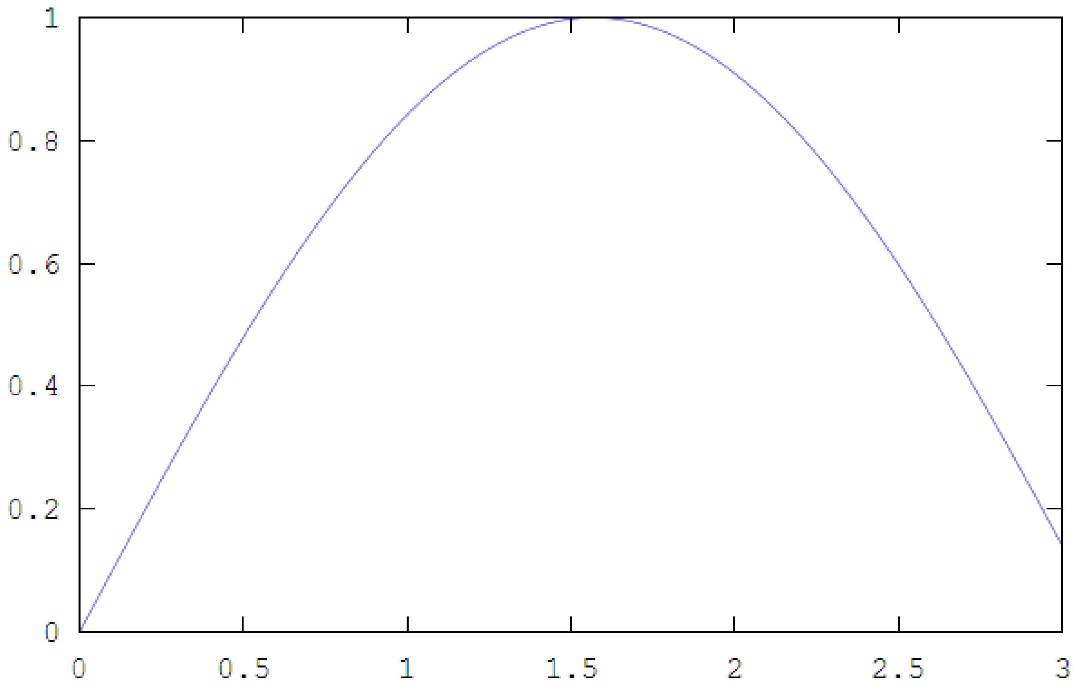
(See the plotted graphic, next.)



You can see that the effect of `plot` is to plot the points (x(1),sin(x(1))=(0,sin(0))=(0,0) up to (x(7),sin(x(7)) and then join these points by straight lines. The above graph is not smooth, but if you use a lot of points, the graph appears to be everywhere smooth.

For example,

```
> x=0:0.01:3;
> plot(x,sin(x))
```



There are many options available for controlling the appearance of a graph. Here are some of them:

- **Change the default colour of the graph** Give the desired colour of the graph as a third argument to either `plot` or `fplot`. The recognized colours are
  - "b" blue
  - "c" cyan
  - "g" green
  - "k" black
  - "m" magenta
  - "r" red
  - "y" yellow
- **Draw graphs of several functions in the same plot** You can do this very easily with both `plot` and `fplot`. All that you do is include additional arguments specifying the extra graphs in the argument list of `plot` or `fplot`.
- **Give the graph a title** Once the graph has been created, enter in the Octave window
  ```
  title("Name of graph");
  ```
- **Label the axes** Once the graph has been created, enter in the Octave window
  ```
  xlabel("Name of horizontal axis");
  ylabel("Name of vertical axis");
  ```
- **Give a legend to an curve** Once the graph has been created, enter in the Octave window
  ```
  legend("Name of curve");
  ```
  In the case that there are several functions on the same graph, the syntax is
  ```
  legend("Name of curve1","Name of curve2");
  ```
  where `curve1` and `curve2` refer to the order in which the curves were defined in the `plot` or `fplot` command

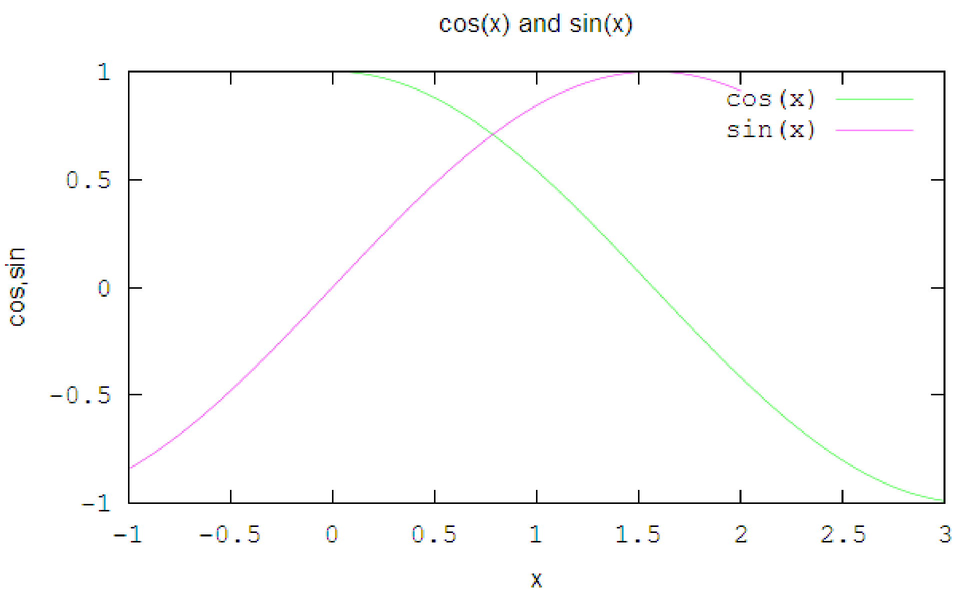
The following examples use all the above features.

Activity 2-30

Plot the graph, give it a title, labels and add a legend.

FEEDBACK:
Type:

```
> plot(x,cos(x),"g",xx,sin(xx),"m")
> title("cos(x) and sin(x)");
> xlabel("x");
> ylabel("cos,sin");
> legend("cos(x)","sin(x)");
```



2.8.3 Plotting lines in 3-dimensions with" `plot3`"

The usage of `plot3` is very similar to that of `plot` in 2 dimensions. The basic syntax is

```
        plot3(x,y,z)
```

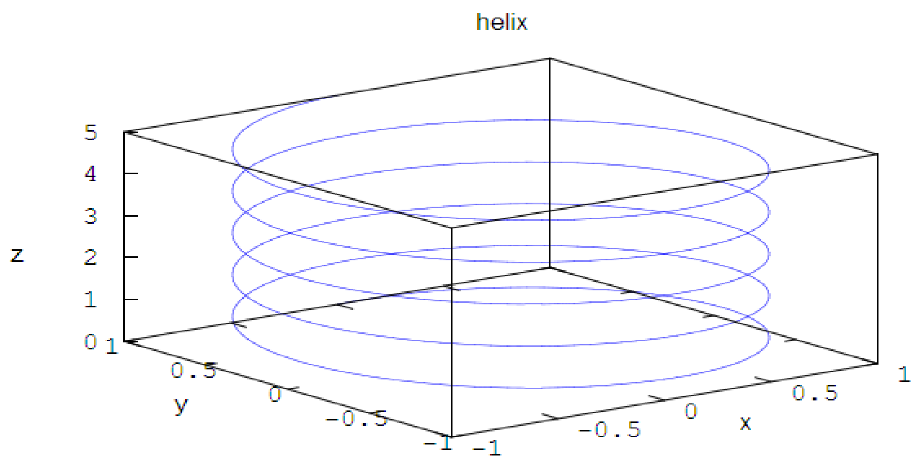
where `x,y,z` are row-vectors of the same length. As with `plot` you can give a title to the graph, label the axes, have multiple plots on the same axes, assign colours to each plot, and give a legend to each plot. Here are some examples.

Activity 2-31

Plot a 3-dimensional graph of the helix (x = sin(2πz), y = cos(2πz), z), with title, labels, colours and a legend.

FEEDBACK:
Type:

```
> z=0:0.01:5;
> plot3(sin(2*pi*z),cos(2*pi*z),z)
> xlabel("x");
> ylabel("y");
> zlabel("z");
> title("helix")
```

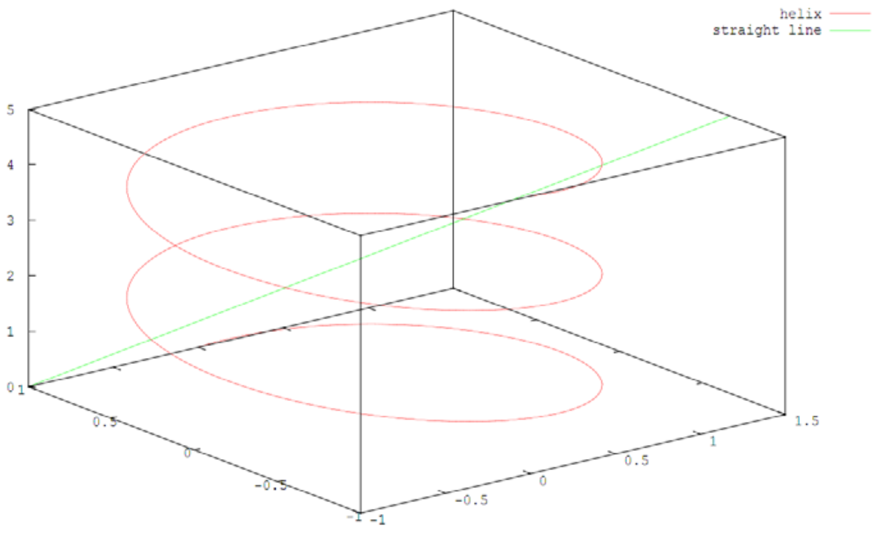


Activity 2-32

Plot a 3-dimensional graph for the helix (x = sin(2πz), y = cos(2πz), z), together with the straight line (x = z/2 – 1, y = 1 – z/3, z), with title, labels, colours and a legend.

FEEDBACK:
Type in the following:

```
> z=0:0.01:5;
> plot3(sin(pi*z),cos(pi*z),z,"r",z./2-1,1-z./3,z,"g")
> legend("helix","straight line");
```



NOTE: you can always rotate the view of a 3-dimensional graph; just place the mouse over the graph, hold down the left mouse button, and move the mouse.

e) ***Plotting surfaces 3-dimensions with "`mesh`"***

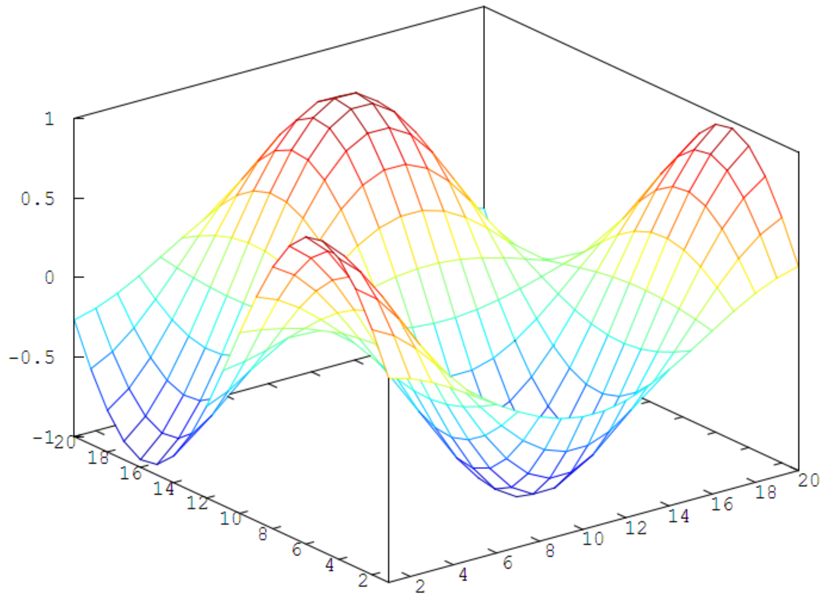
The simplest way to use `mesh` is to apply it to a matrix, and in this case, the coordinates on the horizontal axes are the indices of the matrix.

Activity 2-33

Use `mesh` to produce a 3-D graph for ...

Type:

```
> for a=1:20 for b=1:20
> A(a,b)=sin(0.3*a)*cos(0.3*b);
> end end
> mesh(A)
```



However, we would normally use 3 arguments as, `mesh(xx,yy,zz)`, with `xx,yy,zz` being matrices of the same size. We use `meshgrid` to generate `xx,yy` representing an evenly spaced grid in the *(x,y)* plane, and then use array operators to generate a surface *z=f(x,y)*.

Activity 2-34

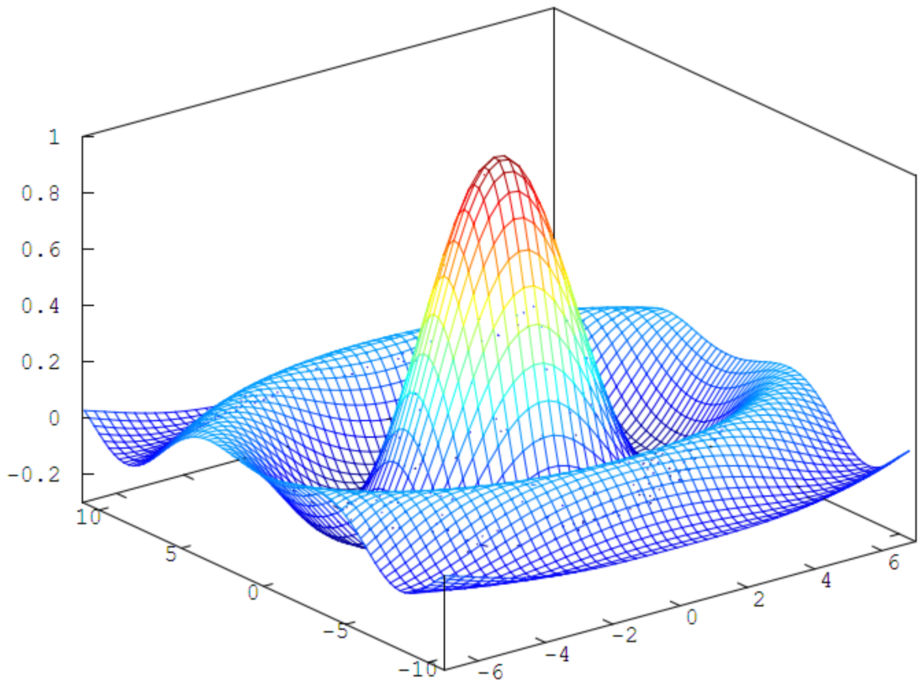
Plot the graph, with a mesh,

$$z = \frac{\sin\sqrt{x^2+y^2}}{\sqrt{x^2+y^2}}$$

in the range *-7 ≤ x ≤ 7, -11 ≤ y ≤ 11.*

FEEDBACK:
Type:

```
> [xx,yy]=meshgrid(-7:0.3:7,-11:0.3:11);
> zz=sin(sqrt(xx.^2+yy.^2))./sqrt(xx.^2+yy.^2);
> mesh(xx,yy,zz)
```



You can also obtain a contour map, or combine a surface plot with a contour map, using the commands `contour` and `meshc` respectively. The syntax is similar to that of `mesh`. As with other plotting routines, it is simple to give the plot a title and to label the axes. In the following examples we use `xx,yy,zz` (as defined above).

Activity 2-35

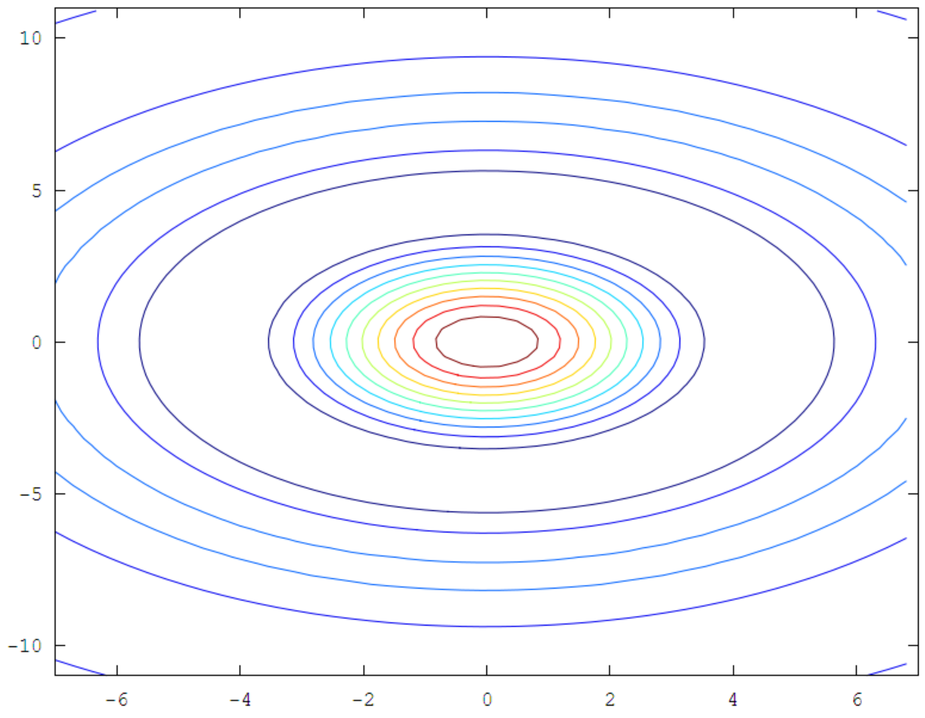
Plot the graph

$$z = \frac{\sin\sqrt{x^2 + y^2}}{\sqrt{x^2 + y^2}}$$

in the range *-7 ≤ x ≤ 7, -11 ≤ y ≤ 11,* with a countour map.

Type:

```
> contour(xx,yy,zz)
```



Activity 2-36

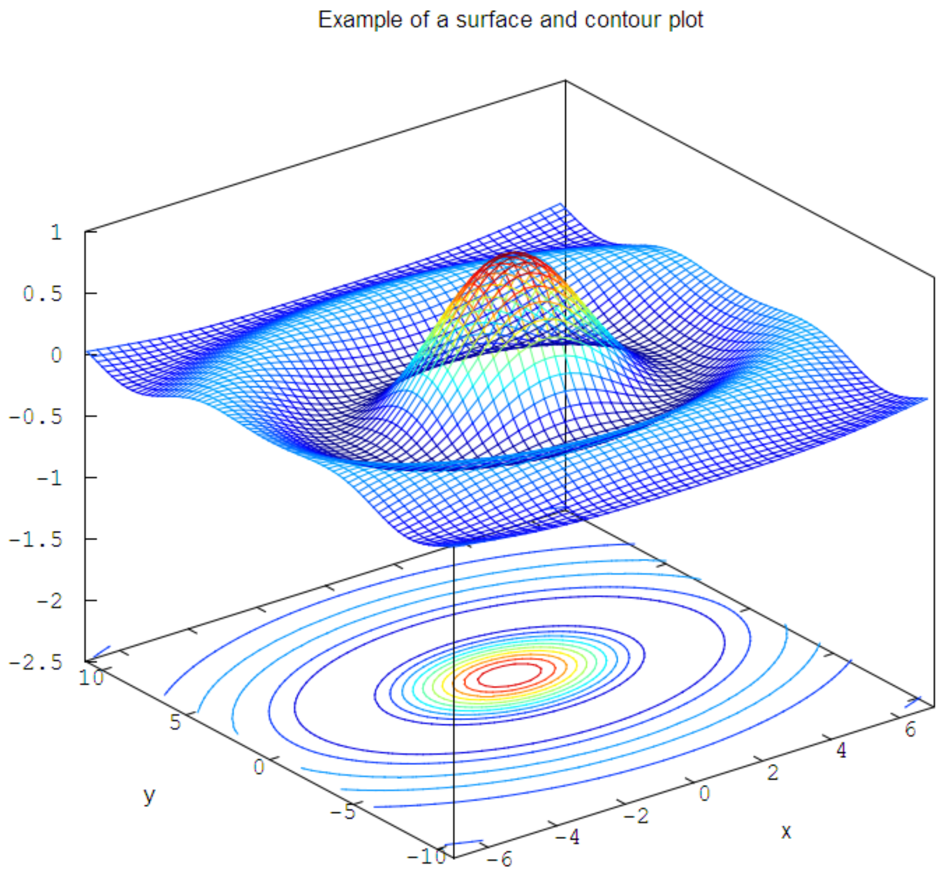
Plot the graph, with a mesh,

$$z = \frac{\sin\sqrt{x^2 + y^2}}{\sqrt{x^2 + y^2}}$$

in the range *-7 ≤ x ≤ 7, -11 ≤ y ≤ 11*

Type:

```
> meshc(xx,yy,zz)
> xlabel("x")
> ylabel("y")
> title("example of a surface and contour plot")
```



2.8.4 Other features of graphics

The above notes have given just a very brief introduction to the many graphical facilities available in Octave. The description includes everything that will be needed in this module, but you should be aware that it is easy to *use* Octave to construct other types of graph. (If you would like to find out more, read the manual, Section 15, pages 177 to 211.)

One feature that will be needed later is the plotting of discrete points that are not joined by lines. The discrete points plotted can be any of:

```
^ * + . o x
```

and the syntax is just to add the point style to the parameter list when calling `plot` or `fplot`.

Activity 2-37

Plot discrete points in a graph that are not joined by lines.

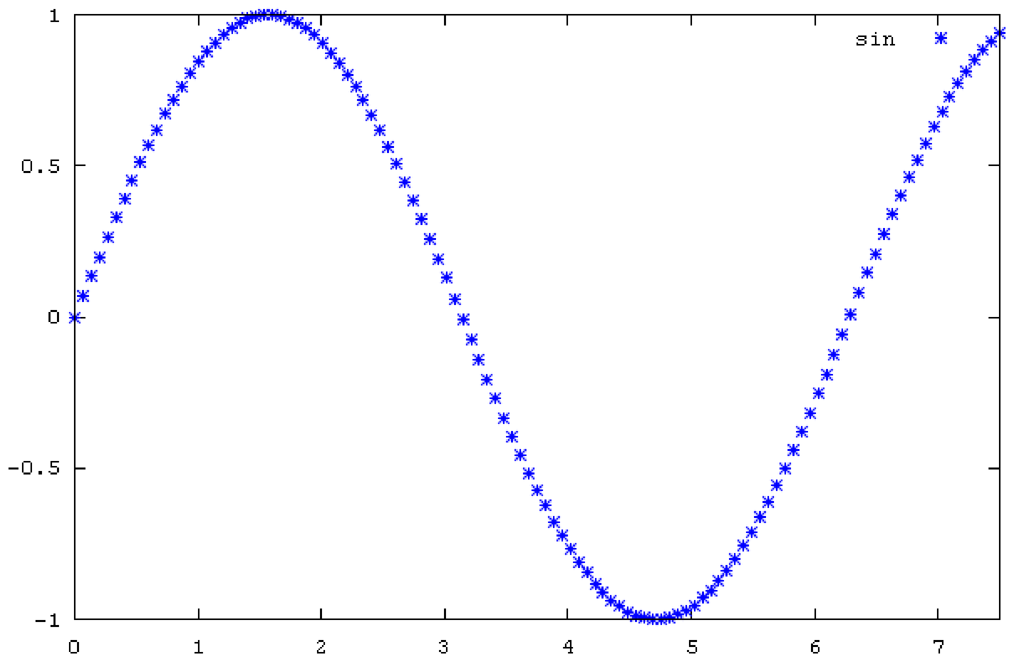
FEEDBACK:

Type:

```
> fplot(@sin,[0,7.5],"*")
```

or

```
> plot(x,sin(x),"o")
```



2.9 Additional Exercises

Activity 2-38

Do the exercises below. Keep evidence of your work (copy and paste your efforts to your workbook or make a printout from your diary file).

1. Evaluate the following series
   a. $1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\cdots+\frac{1}{1001}$
   b. $\frac{1}{1^2 3^2}+\frac{1}{3^2 5^2}+\frac{1}{5^2 7^2}+\cdots+\frac{1}{99^2 101^2}$

2. Evaluate the following series $\sum_{n=1}^{\infty} u_n$ in which u_n is not known explicitly but is given in terms of a recurrence relation. You should stop the summation when $|u_n| < 10^{-8}$

   $$u_{n+1} = (u_{n-1})^2 + (u_n)^2 \text{ with } u_1 = 0.5,\ u_2 = 0.6$$

3. Modify quadratic.m that solves the quadratic equation $a\,x^2 + b\,x + c = 0$ using the formula

   $$x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

   The function file should have (a, b, c) as input and should return the two solutions for x. The amendment required is that the function should check whether $a = 0$ and if so it should give the solution $x = -c / b$; if also $b = 0$, it should report an error message and exit the function without trying to divide by zero. Show that your code is correct by testing it on the cases
   a. $x^2 + 5\,x + 6 = 0$
   b. $2\,x + 4 = 0$
   c. $x^2 + 4 = 0$
   d. $0 = 0$

4. Formulas to find a numerical approximation to the first and second derivatives of a function $f(x)$ are

   $$f^{'}(x) = \frac{df}{dx} \approx \frac{f(x+h)-f(x-h)}{2h}, \quad f^{''}(x) = \frac{d^2 f}{dx^2} = \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$$

   with the approximation being better and better as $h \to 0$. Write function files deriv1.m (for the first derivative) and deriv2.m (for the second derivative) that implement these formulas. The inputs to each formula should be the function to be differentiated (remember the function handle construct @), the value of x, and the value of h. Use your code to estimate
   a. The first derivative of sin(x) at $x = \pi/4$ with $h = 10^{-5}$
   b. The second derivative of tan(x) at $x = \pi/6$ with $h = 10^{-3}$

5. Use the functions deriv1 and deriv2 constructed in question 5 to find the first and second derivatives of

$$f(x) = x \exp(x^2)$$

at $x = 0.5$. Evaluate both derivatives for $h = 10^{-2}, 10^{-3}, 10^{-4}, \ldots, 10^{-12}$, and then for each value of h evaluate the error. The exact answers are

$$f'(x) = (2x^2 + 1)\exp(x^2), \quad f''(x) = (4x^3 + 6x)\exp(x^2).$$

Then plot $\log_{10}$ (error) against $\log_{10}(h)$ (In Octave, the function $\log_{10}$ is `log10`). At what value of h is the error minimized?

6. Use Octave to draw the following graphs
   a. $f(x) = x \sin(x)$, $-3 \le x \le 8$
   b. $f(x) = x \cos(x^2)$, $0 \le x \le 5$

7. A formula for the population of the USA, using the logistic model, is

$$P(t) = \frac{197273000}{1 + e^{-0.03134(t-1913.25)}}$$

where t is the date in years. Some actual data is as follows

Date	Population
1800	5308000
1820	9638000
1840	17069000
1870	38558000
1900	75995000
1930	122775000
1950	150697000

Plot the graph of $P(t)$ against t as a continuous line, and the given data as discrete circles, i.e. do not join them with lines (Discrete circles are obtained by adding "o" to the `plot` parameter list).

8. A rather beautiful fractal picture can be obtained by plotting the points (x_i, y_i) generated by the following difference equations

$$x_{i+1} = y_i\,(1 + \sin(0.7\,x_i)) - 1.2\sqrt{|x_i|}$$

$$y_{i+1} = 0.21 - x_i$$

starting with $(x_1, y_1) = (0, 0)$.
Write a program to draw the picture (plot individual points and do not join them).

2.10 In Summary

Now that you have completed this study unit, you can write your own code in Octave, and use it to for numerical calculations. You now have sufficient programming knowledge to be able to tackle interesting problems in linear algebra, and you will start to do so in the next study unit.

Study Unit 3: Use of Octave to solve linear systems of equations

Time period: 15 hours approximately

LEARNING OUTCOMES

At the end of this Study Unit, you should be able to write computer programs that:

- Solve linear systems of equations using the direct method of Gaussian elimination
- Solve linear systems of equations using the Jacobi iterative method
- Solve linear systems of equations using the Gauss-Seidel iterative method
- Identify systems of linear equations that are problematic for the above methods

3.1 Introduction

This Study Unit is concerned with systems of linear equations that have a unique solution.

The more general case of over-determined or under-determined systems will be discussed in Study Unit 4.

3.2 Gaussian elimination

There is no need for you to write a Gaussian elimination code, as this process is pre-defined in Octave using the construct `x=A\b` already introduced in Study Units 1 and 2 (Actually, Octave does not implement pure Gaussian elimination, but rather uses matrix factorization whose details are beyond the scope of a first level module).

Activity 3-1

Suppose that we want to solve

$$1\, x_1 + 2\, x_2 = 5$$
$$3\, x_1 + 4\, x_2 = 6$$

FEEDBACK:

The Octave code is

```
> A=[1 2;3 4];
> b=[5;6];
> x=A\b
x =

 -4.0000
  4.5000
```

Thus the required answer is

$x_1 = -4$, $x_2 = 4.5$

The above constructs can be used for any number of equations and unknowns (provided, of course, that the number of equations is the same as the number of unknowns).

3.3 Iterative methods

Here we discuss the Jacobi and Gauss-Seidel methods. However, first we should point out that, for a real problem, these methods perform better than `x=A\b` only for diagonally dominant, sparse matrices. We will define diagonally dominant later; in a sparse matrix, most of the entries are zero. Also, the speed of modern computers is such that it is, in practice, worthwhile to use an iterative method rather than `x=A\b` only for large matrices, which means matrices that are bigger than about 1000 x 1000. In practice, iterative methods should be used only for large, diagonally dominant, sparse matrices.

Iteration techniques are usually very easy to apply and can best be illustrated by examples.

Activity 3-2

Solve the following system of equations:

$$\begin{aligned} 20x_1 + \quad x_2 - \quad x_3 &= 17 \\ x_1 - 10x_2 + \quad x_3 &= 13 \\ -x_1 + \quad x_2 + 10x_3 &= 18. \end{aligned} \qquad (3.2.1)$$

FEEDBACK:

Before solving the system let us make two obvious observations:

(i) The system (3.2.1) can be solved very easily by Gaussian elimination to yield the unique solution $(x_1, x_2, x_3) = (1,-1,2)$ but the purpose of the examples is to illustrate a technique.

(ii) You will notice that, from a numerical point of view, the coefficient matrix of (3.2.1) viz

$$\begin{bmatrix} 20 & 1 & -1 \\ 1 & -10 & 1 \\ -1 & 1 & 10 \end{bmatrix}$$

is dominated by the diagonal entries in that their absolute values are much larger than the absolute values of the other entries.

Solution

We shall use two different iterative techniques and we begin with the *Jacobi method*. The first step is to 'solve' the first equation in (3.2.1) for x_1, the second for x_2 and the third for x_3. In fact, we don't really solve the equations, we merely re-write them in the form

$$\begin{aligned} x_1 &= \frac{17}{20} - \frac{x_2}{20} + \frac{x_3}{20} \\ x_2 &= -\frac{13}{10} + \frac{x_1}{10} + \frac{x_3}{10} \\ x_3 &= \frac{18}{10} + \frac{x_1}{10} - \frac{x_2}{10}. \end{aligned} \tag{3.2.2}$$

The second step is to assume that an approximate solution to the system is

$$\left(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}\right) = (0,0,0).$$

Note that it is not essential to choose (0, 0, 0) as the initial approximation. If we happen to know that, say, (1, 1, 1) is an approximate solution, then we could equally well begin with this approximation. In most cases, however, we do not know an approximate solution and it is customary to begin with (0, 0, 0).

The next step is to substitute the initial approximation (0, 0, 0) in (3.2.2) and we obtain the first approximate solution

$$\begin{aligned} x_1^{(1)} &= \frac{17}{20} = 0.85 \\ x_2^{(1)} &= -\frac{13}{10} = -1.3 \\ x_3^{(1)} &= \frac{18}{10} = 1.8. \end{aligned} \tag{3.2.3}$$

The first approximation $(0.85, \quad -1.3, \quad 1.8)$ is better than (0, 0, 0) and it is obvious that to obtain an even better approximation, we simply repeat the procedure with the solution (3.2.3) i.e. we iterate. If we substitute (3.2.3) in (3.2.2) we obtain the second approximation

$$\begin{aligned} x_1^{(2)} &= \frac{17}{20} + \frac{1.3}{20} + \frac{1.8}{20} = 1.005 \\ x_2^{(2)} &= -\frac{13}{10} + \frac{0.85}{10} + \frac{1.8}{10} = -1.035 \\ x_3^{(2)} &= \frac{18}{10} + \frac{0.85}{10} + \frac{1.3}{10} = 2.015. \end{aligned}$$

The iterations are obtained from an Octave program and are

```
ans =

  0.85000  1.00500  1.00250  1.00010  0.99997  1.00000  1.00000
 -1.30000 -1.03500 -0.99800 -0.99935 -0.99999 -1.00001 -1.00000
  1.80000  2.01500  2.00400  2.00005  1.99994  2.00000  2.00000
```

After seven iterations we therefore obtain a solution which, to six significant figures, coincides with the exact solution.

A slight variation of the Jacobi method is the so-called *Gauss-Seidel method.* The Gauss-Seidel method is usually (but not always) better than the Jacobi method and the only difference is that as better approximations become available, so we make immediate use of them in the next equation. (It's actually much easier than it sounds.)

As before, our initial approximation is $\left(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}\right) = (0,\ 0,\ 0)$ and if we substitute this in (3.2.2) then the first equation yields

$$x_1^{(1)} = \frac{17}{20} - \frac{0}{20} + \frac{0}{20} = 0.85.$$

For $x_2^{(1)}$, however, we *do not put*

$$x_2^{(1)} = -\frac{13}{10} + \frac{0}{10} + \frac{0}{10}$$

but instead we use the value $x_1^{(1)} = 0.85$ which we have just obtained from the first equation. In other words,

$$x_2^{(1)} = -\frac{13}{10} + \frac{0.85}{10} + \frac{0}{10} = -1.215$$

and we note that we have to use the value $x_3^{(0)} = 0$ in this equation because at this stage, this is all that we know about x_3. We now go ahead and compute $x_3^{(1)}$ from (3.2.2) using the values $x_1^{(1)}$ and $x_2^{(1)}$ as given by immediately above, i.e.

$$x_3^{(1)} = \frac{18}{10} + \frac{0.85}{10} + \frac{1.215}{10} = 2.0065.$$

We now do a second iteration in precisely the same way to obtain

$$x_1^{(2)} = \frac{17}{20} + \frac{1.215}{20} + \frac{2.0065}{20} = 1.0111.$$

The second and third equations of (3.2.2) yield respectively

$$x_2^{(2)} = -\frac{13}{10} + \frac{1.0111}{10} + \frac{2.0065}{10} = -0.99824.$$

and

$$x_3^{(2)} = \frac{18}{10} + \frac{1.0111}{10} + \frac{0.99824}{10} = 2.0009.$$

Further iterations are obtained from an Octave program and are

```
ans =

  0.85000  1.01107  0.99996  0.99999  1.00000
 -1.21500 -0.99824 -0.99991 -1.00000 -1.00000
  2.00650  2.00093  1.99999  2.00000  2.00000
```

After five iterations we therefore obtain a solution which, to six significant figures, coincides with the exact solution. In this particular example, therefore, the Gauss-Seidel method beats the Jacobi method by two iterations.

Octave program

We give here the Octave code for the Gauss-Seidel method. It is easy to modify this code so that it implements the Jacobi method instead, and that is left as an exercise for the student. The code is written as a function file. The user gives to the function the matrix `A`, the right hand side `b`, and the current estimate for the solution `xold`. The function performs one iteration of the Gauss-Seidel method returning the new estimate for the solution. Note the use of the temporary matrix `At` which has zeros in its diagonal; this can be avoided, but then the main loop defining `xnew(k)` would be rather more complicated. In the Octave window, we define `A`, `b`, and the initial estimate of the solution for the problem stated above, and then use a `for` loop to make a fixed number (in this case, five) of iterations of the Gauss-Seidel method.

Activity 3-3

Create the m-file (below) and run it in Octave.

FEEDBACK:

File gauss_seidel.m
```
function xnew=gauss_seidel(A,b,xold)
 n=size(A)(1);
 At=A;
 xnew=xold;
 for k=1:n
  At(k,k)=0;
 end
 for k=1:n
  xnew(k)=(b(k)-At(k,:)*xnew)/A(k,k);
 end
endfunction
```

Ocatave window
```
> A=[20 1 -1;1 -10 1;-1 1 10];
> b=[17 13 18]';
> x(:,1)=[0 0 0]';
> for k=1:5 x(:,k+1)=gauss_seidel(A,b,x(:,k)); end;
> x
```

```
      ans =

        0.85000   1.00500   1.00250   1.00010   0.99997   1.00000
      1.00000
       -1.30000  -1.03500  -0.99800  -0.99935  -0.99999  -1.00001  -
      1.00000
        1.80000   2.01500   2.00400   2.00005   1.99994   2.00000
      2.00000
```

3.4 Diagonal dominance

Now let us consider the system of equations

$$x_1 + x_2 = 3$$
$$x_1 - x_2 = 1.$$

The solution is $(x_1, x_2) = (2,1)$.

Suppose we try to solve this system by the Gauss-Seidel method. We put $(x_1^{(1)}, x_2^{(1)}) = (0,0)$ and we obtain

```
      x =

        0   3   1   3   1   3   1
        0   2  -0   2  -0   2  -0
```

Thus the result of iteration is that we go around in circles. In other words, this system cannot be solved by an iteration process and the question automatically arises: when does a system admit a convergent iterative solution? Incidentally, note that a solution converges if, after sufficiently many iterations, the exact and approximate solutions differ by as small an amount as we wish.

In this case it is easy to *state* the answer but the proof is beyond the scope of this course. Let us write the coefficient matrix of a system of equations in the usual notation (as referred to in MAT103N), that is:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & \\ & & \cdots & \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

Note that since we are assuming unique solutions throughout this Study Unit, the coefficient matrix must be square and of rank *n*. The matrix A is said to be *strictly diagonally dominant* if the absolute value of each diagonal entry is greater than the sum of the absolute values of the remaining entries in the same row, for example:

$$\begin{aligned} |a_{11}| &> |a_{12}| + |a_{13}| + \cdots + |a_{1n}| \\ |a_{22}| &> |a_{21}| + |a_{23}| + \cdots + |a_{2n}| \\ &\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ |a_{nn}| &> |a_{n1}| + |a_{n2}| + \cdots + |a_{nn-1}|. \end{aligned}$$

Clearly the coefficient matrix of (3.2.1) is strictly diagonally dominant since

$$|20| > |1| + |-1|,\ |-10| > |1| + |1|,\ |10| > |-1| + |1|.$$

Now it can be shown that if a coefficient matrix *A* is strictly diagonally dominant then both the Jacobi and Gauss-Seidel iterative solutions to the system of equations *A***x** = **b** are convergent. Note that the condition on *A* is a *sufficient condition - not a necessary condition*. In other words, there may be coefficient matrices which are NOT strictly diagonally dominant but the corresponding equations may nonetheless admit convergent Jacobi or Gauss-Seidel solutions.

However, the conditions on these matrices are too technical for this course and for our purposes we shall regard the condition as necessary as well as sufficient. There is one trivial exception to this which we discuss in the next example.

Activity 3-4

Consider the system

$$\begin{aligned} x_1 + 3x_2 - \ \ x_3 &= 6 \\ 4x_1 - \ \ x_2 + \ \ x_3 &= 5 \\ x_1 + \ \ x_2 - 7x_3 &= -9. \end{aligned}$$

FEEDBACK:
Can one use iterative techniques to solve this system? Clearly, the coefficient matrix is NOT strictly diagonally dominant since the first two equations do not satisfy the required condition. However, we know that we can interchange the first

two equations without affecting the solution to the system and if we do this, we obtain a coefficient matrix which IS strictly diagonally dominant, that is:

$$\begin{bmatrix} 4 & -1 & 1 \\ 1 & 3 & -1 \\ 1 & 1 & -7 \end{bmatrix}.$$

The system of equations is therefore

$$\begin{aligned} 4x_1 - x_2 + x_3 &= 5 \\ x_1 + 3x_2 - x_3 &= 6 \\ x_1 + x_2 - 7x_3 &= -9. \end{aligned}$$

and if we solve by Gauss elimination we obtain the solution

$$(x_1, x, x_3) = (1.3409, 2.1477, 1.7841)$$

Using the Gauss-Seidel method beginning with (0, 0, 0), we get

```
ans =

  0.00000  1.25000  1.22321  1.34683  1.33916  1.34108  1.34088
  0.00000  1.58333  2.15575  2.14053  2.14825  2.14761  2.14774
  0.00000  1.69048  1.76842  1.78391  1.78392  1.78410  1.78409
```

After six iterations we therefore obtain a solution which, to five significant figures, coincides with the exact solution. (An iteration is indicated here as a column of figures.)

3.4.1 Stopping criterion

The question which you are probably asking yourself at this point is: When does one stop? One way would be to simply limit the number of iterations to 5 or 10 or whatever. The problem here is that one doesn't know how fast the approximations are converging and after a considerable number of iterations one might still be quite a long way from a reasonable solution. A better way would be to place a restriction on the relative error. The difficulty here is that one doesn't know the exact solution - if one did, there wouldn't be much point in finding an approximate solution.

For iteration schemes, therefore, we define the relative error $\in_r^{(n)}$ by

$$\in_r^{(n)} = \left| \frac{x^{(n)} - x^{(n-1)}}{x^{(n)}} \right|.$$

Find the relative errors $\in_r^{(4)}$ in the activity above.

By definition, we have to compute

$$\left| \frac{x_i^{(4)} - x_i^{(3)}}{x_i^{(4)}} \right|.$$

for i = 1, 2, 3. We have

$$\left|\frac{x_1^{(4)}-x_1^{(3)}}{x_1^{(4)}}\right| = \left|\frac{1.3392-1.3469}{1.3392}\right| = 0.0057497$$

$$\left|\frac{x_2^{(4)}-x_2^{(3)}}{x_2^{(4)}}\right| = \left|\frac{2.1482-2.1405}{2.1482}\right| = 0.0035843.$$

$$\left|\frac{x_3^{(4)}-x_3^{(3)}}{x_3^{(4)}}\right| = \left|\frac{1.7839-1.7839}{1.7839}\right| = 0.$$

If we were required to iterate until all relative errors were less than, say 0.005, then we see that we would have to do at least one more iteration, and find $\in_r^{(5)}$.

Octave code with a stopping condition that depends upon the relative error uses the `until` construct. We write the code as a function file with inputs `A, b` (as usual); the initial estimate of the solution `xinitial`; the maximum permitted value for the relative error `TOL`; the maximum number of permitted iterations `max_it`; and the method to be used for obtaining an iterative solution `method`, which will be either `gauss_seidel` or `jacobi`. As we saw earlier, the iterative method may not converge, so one needs to specify a maximum number of iterations otherwise the program will never exit the conditional loop; recall that when evaluating a conditional statement `|` means or. The function returns the solution, and reports the number of iterations used, giving an error message if this number exceeds `max_it`. In the Octave window, we set `A, b, xinitial`; and then call `iterative_linear_solve` using the function handle construct `@` to specify the method. We run the program for two cases discussed earlier, one in which convergence occurs and the other in which it does not.

Activity 3-5

Create the m-file (below) and run it in Octave.

File iterative_linear_solve.m

```
function
xnew=iterative_linear_solve(A,b,xinitial,TOL,max_it,method)
 xold=xinitial;
 k=0;
 do
  xnew=feval(method,A,b,xold);
  err=max(abs((xnew-xold)./xnew));
  xold=xnew;
  k=k+1;
 until((err<TOL) | (k>max_it));
 k
 if (k>max_it)
  disp("ERROR: METHOD DID NOT CONVERGE");
  xnew=[];
 endif
endfunction
```

Ocatave window

```
> A=[20 1 -1;1 -10 1;-1 1 10];
> b=[17 13 18]';
> x0=[0 0 0]';
> y=iterative_linear_solve(A,b,x0,1/10^5,5,@gauss_seidel)
```

```
k = 5
y =

  1.0000
 -1.0000
  2.0000
```

```
> A=[1 1;1 -1];
> b=[3 1]';
> x0=[0 0]';
> y=iterative_linear_solve(A,b,x0,1/10^4,5,@gauss_seidel)
```

```
k = 6
ERROR: METHOD DID NOT CONVERGE
y = [](0x0)
```

Activity 3-6

Solve the following systems of equations, using the `A\b` construct, as well as the Gauss-Seidel method with a tolerance of 10^{-7}

$20\, x_1 - x_2 + x_3 = 20$

$2\, x_1 + 10\, x_2 - x_3 = 11$

$x_1 + x_2 - 20\, x_3 = -18$

Did you get the correct answer? It is $x_1 = x_2 = x_3 = 1$.

3.5 Exceptional cases (where the solution may not be reliable)

In this Study Unit we are solving linear systems of the form $A\, x = b$ where A is a n x n matrix. As shown (in MAT103N), we know that a unique solution exists provided the determinant of A is non-zero, or equivalently, provided that the rank of A is equal to n; such a matrix is called non-singular.

In Octave, it is very easy to check these conditions using the pre-defined functions `det` and `rank`.

Activity 3-7

Check these conditions in Octave using the pre-defined functions `det` and `rank`.

```
> A=[2 3;6 9]
```

```
A =
   2   3
   6   9
```

```
> det(A)
ans = 0
> rank(A)
ans = 1
```

Even though the matrix is singular and a unique solution does not exist, Octave nevertheless will construct a solution (with a warning message). For example, with `A` defined as above,

```
> b=[2 2]';
> A\b
```

```
      warning: matrix singular to machine precision, rcond = 0
      warning: attempting to find minimum norm solution
      ans =

       1.9725e+015
       -1.3150e+015
```

If you see such a warning message, you should check the rank or determinant of *A*, and note that the solution given by Octave is not unique and may not be a solution at all.

From a mathematical viewpoint, the determinant is either zero or not, and if it is nonzero the system has a unique solution. However, from the viewpoint of computing the solution, the situation is not so simple. Loosely expressed, problems can arise if we are in a situation in which the matrix is nearly singular (but identifying this situation is more involved than just looking at the numerical value of the determinant).

Activity 3-8

Consider the following examples

1) $1000\ x_1 + 2000\ x_2 = 5000$
2) $1000\ x_1 + 2000.001\ x_2 = 6000$

The determinant of the coefficient matrix is 1, but if we interpret the problem geometrically we see that we are trying to find the point of inter of two almost parallel lines, so that we are in a situation in which the matrix is nearly singular. We call such a matrix **ill-conditioned**, and measure how close it is to being singular by its **condition number**. In Octave, the condition number is a pre-defined function denoted by `cond` and defined for a square matrix `A` by

```
      cond(A) = norm(A) * norm(inv(A)),
```

where `norm(A)` is a measure of the average magnitude of the elements of `A`, and in this module we will not define it more precisely. The condition number is always at least one, and a singular matrix has an infinite condition number: the larger the condition number, the more ill-conditioned is the matrix. Do the examples below.

ACTIVITY

```
> A1=eye(6)/10^5
```

```
      A1 =

 1.0000e-005 0.0000e+000 0.0000e+000 0.0000e+000 0.0000e+000 0.0000e+000
 0.0000e+000 1.0000e-005 0.0000e+000 0.0000e+000 0.0000e+000 0.0000e+000
 0.0000e+000 0.0000e+000 1.0000e-005 0.0000e+000 0.0000e+000 0.0000e+000
 0.0000e+000 0.0000e+000 0.0000e+000 1.0000e-005 0.0000e+000 0.0000e+000
 0.0000e+000 0.0000e+000 0.0000e+000 0.0000e+000 1.0000e-005 0.0000e+000
 0.0000e+000 0.0000e+000 0.0000e+000 0.0000e+000 0.0000e+000 1.0000e-005
```

```
> cond(A1)
```

```
      ans = 1
```

```
> A2=ones(6,6)
       A2 =

         1   1   1   1   1   1
         1   1   1   1   1   1
         1   1   1   1   1   1
         1   1   1   1   1   1
         1   1   1   1   1   1
         1   1   1   1   1   1

> cond(A2)
       ans = Inf

> A3=A2+rand(6,6)/10^5;
> cond(A3)
       ans = 5.9902e+006

> format long
> A4=[1000 2000;1000 2000.001]
       A4 =

         1000.00000000000   2000.00000000000
         1000.00000000000   2000.00100000000

> det(A4)
       ans = 0.999999999976464
> cond(A4)
       ans = 10000004.0000005
```

You can use the condition number to estimate the accuracy at which Octave solves for x in $A\,x = b$. First, we define the **residual** which is $A\,x - b$. Normally, when we are using a direct method with the `A\b` construct, the residual is of the order of machine precision (which in Octave is the variable `eps`, and is usually about 2×10^{-16}), but for an iterative method it depends on the tolerance chosen and can be much larger. Then the accuracy at which we determine x will be no better than the residual multiplied by the condition number.

3.6 Additional Exercises

1. Solve the following systems of equations, using the `A\b` construct, as well as the Gauss-Seidel method with a tolerance of 10^{-7} (in some cases convergence may not occur)

   a.
   $$\begin{aligned} 2\,x_1 - x_2 + 3\,x_3 &= 8 \\ 4\,x_1 + 2\,x_2 - 5\,x_3 &= -9 \\ 6\,x_1 + 3\,x_2 + x_3 &= 12 \end{aligned}$$

   b.
   $$\begin{aligned} 0.1\,x_1 + 0.05\,x_2 + 0.1\,x_3 &= 1.3 \\ 12\,x_1 + 25\,x_2 - 3\,x_3 &= 10 \\ -7\,x_1 + 8\,x_2 + 15\,x_3 &= 2 \end{aligned}$$

   c.
   $$\begin{aligned} 10\,x_1 + x_2 + 2\,x_3 &= 3 \\ x_1 + 10\,x_2 - x_3 &= 1.5 \\ 2\,x_1 + x_2 + 10\,x_3 &= -9 \end{aligned}$$

   d.
   $$\begin{aligned} 12\,x_1 - 3\,x_2 + 4\,x_3 - 2\,x_4 &= 12 \\ 2\,x_1 + 10\,x_2 - x_3 - 20\,x_4 &= 15 \\ x_1 - x_2 + 20\,x_3 + 4\,x_4 &= -7 \end{aligned}$$

$x_1 + x_2 - 20\,x_3 - 3\,x_4 = -5$

e. $10\,x_1 - x_2 + 7\,x_3 - 18\,x_4 = 2$
$2\,x_1 + 10\,x_2 – 4\,x_3 - 20\,x_4 = 1$
$-1.5\,x_1 + 6\,x_2 - 20\,x_3 - 2\,x_4 = 9$
$2\,x_1 + 5\,x_2 - 2\,x_3 + 30\,x_4 = -18$

2. Modify the function file gauss_seidel.m to produce a new function file jacobi.m that implements the Jacobi method. Now use the Jacobi method to solve example 3.2.1, i.e.

$$\begin{aligned} 20x_1 + x_2 - x_3 &= 17 \\ x_1 - 10x_2 + x_3 &= 13 \\ -x_1 + x_2 + 10x_3 &= 18. \end{aligned}$$

3. Write a function file that takes as input a matrix *A*, and tests whether or not the matrix is (a) square, and (b) diagonally dominant, reporting the answers on the screen. Show that your code is correct by testing it for the matrix in question 2, as well as for cases where the matrix is not diagonally dominant, and not square.

4. Modify the function file iterative_linear_solve.m to produce a new function file iterative_linear_solve 2.m, in which the stopping condition is that magnitude of the residual ($A\,x – b$) should be less than a given tolerance. Show that your code works by applying it to the problem in question 2, using the Gauss-Seidel method.

   In practice, this alternative stopping method is not often used. Why not?

5. The Hilbert matrix is a square $n \times n$ matrix defined by

$$H_{ij}^{(n)} = \frac{1}{i + j - 1}$$

   Define $b^{(n)}$ to be a column vector of dimension *n*, and with each element 1. Construct $b^{(n)}$ and $H_{ij}^{(n)}$, and then solve for $x^{(n)}$, $H_{ij}^{(n)}\; x^{(n)} = b^{(n)}$, in the cases $n = 4,7,10$ and 13. Comment on the results.

6. Define the 100×100 square matrix *A* and the column vector *b* by

$$A_{ij} = I_{ij} + \frac{1}{(i - j)^2 + 1},\; b_i = 1 + \frac{2}{i},\; 1 \le i, j \le 100$$

where I_{ij} is the 100×100 identity matrix (i.e. 1 on the main diagonal and 0 everywhere else). Solve $A\,x = b$ for *x* using both the Gauss-Seidel method and the `A\b` construct. Do not give the whole vector *x* in your output, but only x_2, x_{50} and x_{99}.

3.7 In conclusion

You have now used Octave to solve linear systems of equations, including large systems that are impossible to solve by hand. In these systems, the number of equations was equal to the number of unknowns, but what can be done if that is not the case? We will look at this question in the next study unit.

Study Unit 4: Overdetermined and underdetermined systems of linear equations

Time period: 15 hours approximately

LEARNING OUTCOMES

At the end of this Study Unit, you should be able to write Octave programs that

- Determine whether a system of equations has a unique solution, or whether it is overdetermined or underdetermined
- Use Octave to find the best "least squares" approximation to an overdetermined system
- Use Octave to find parameters that give the best "least squares" fit of a given form of curve to given data
- Use Octave to find the general solution of an underdetermined system

4.1 Identification of overdetermined and underdetermined systems

Suppose that we have *m* equations involving *n* unknowns. Writing the system in matrix form

$$A\,x = b$$

the matrix *A* has *m* rows and *n* columns, *x* is a column vector with *n* rows, and *b* is a column vector with *m* rows. The following situations can arise.

4.1.1 A is square (m = n) and det(A) ≠ 0

As discussed in Study Unit 3, such systems are neither overdetermined nor underdetermined and have a unique solution.

4.1.2 A is square (m = n) and det(A) = 0

Such systems are either underdetermined, as in

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} x = \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

which has infinitely many solutions, $x_1 = 1 + \alpha$, $x_2 = 1 - \alpha$, for any value of α; or inconsistent, as in

$$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} x = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

which has no solutions.

4.1.3 There are more equations than unknowns (m > n)

Usually such systems are overdetermined and a solution satisfying all the equations does not exist. However, it is possible that some of the equations are linear combinations of the others and that there are only *n* independent equations, as in

$$\begin{pmatrix} 1 & 1 \\ 2 & 3 \\ 3 & 3 \end{pmatrix} x = \begin{pmatrix} 2 \\ 5 \\ 6 \end{pmatrix}$$

which has the unique solution, $x_1 = 1$, $x_2 = 1$; or even that there are fewer than *n* independent equations so that there are infinitely many solutions, as in

$$\begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 3 \end{pmatrix} x = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

which has solutions, $x_1 = 1 + \alpha$, $x_2 = 1 - \alpha$, for any value of α.

4.1.4 There are more unknowns than equations (n > m)

Such systems are underdetermined, and usually have infinitely many solutions, as in

$$\begin{pmatrix} 1 & 2 & 1 \\ 1 & 1 & 0 \end{pmatrix} x = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

which has solutions, $x_1 = 1 + \alpha$, $x_2 = 1 - \alpha$, $x_3 = 1 + \alpha$ for any value of α. However, it is also possible for such systems to be inconsistent, as in

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \end{pmatrix} x = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

which has no solutions.

4.2 Overdetermined systems

Here we are concerned with situations in which there are more (independent) equations than unknowns, so that a solution to the problem $A\,x = b$ does not exist. So what more can be said? The answer comes down to what type of situation the system of equations is modeling. While it may be that non-existence of an exact solution is a sufficient answer, there are circumstances in which we would like to find an approximate solution. Consider the following example

Activity 4-1

A car is undergoing uniform acceleration. Measurements have been made of the car's displacement (*s*) from the origin against time (*t*) as follows:

t	s
1	6.7118
2	10.3049
3	16.1415
4	23.9482
5	33.8839
6	45.9405
7	60.1468
8	76.3638
9	94.6527
10	115.1413

Since the car is undergoing uniform acceleration, *s* and *t* are related by

$$s = s_0 + u_0\, t + \tfrac{1}{2}\, a\, t^2 .$$

Use the above data to find the best estimate of the constants s_0, u_0 and a.

We will solve the example later, but for now, we are interested in issues that are more general. Because of measurement inaccuracies, it makes sense that that there will not be values of s_0, u_0 and a, that exactly fit all the data points, and that a good approximate solution is all that can be expected. However, that still leaves the question as to what, precisely, is meant by a good approximate solution? We are no longer trying to solve $A\,x = b$ but instead we are seeking a column vector x_* such that the error in the approximation (also called the residual) ε is as small as possible, where

$$\varepsilon = (A\,x_* - b)$$

Now, ε is not a scalar quantity but is a column vector, so what do we mean by saying that we want ε to be as small as possible? There is not a clear answer to this question, and in subsequent courses you will learn about a number of different ways of defining the magnitude of a vector. Here, we use the definition (technically, the L_2 norm)

$$|\,\varepsilon\,| = \sqrt{\varepsilon_1^2 + \varepsilon_2^2 + \cdots + \varepsilon_n^2}$$

which can also be interpreted as the length of the vector.

In Octave, the norm of x is found by the call `norm(x)`. So, we want to find a vector x_* such that the value of $|\,\varepsilon\,|$ is minimized. Minimization is a simple problem in calculus, and also in this case is easy to work out. However, it requires applying calculus to vectors and matrices, which is a second level topic so here we just state the result. The condition that $|\,\varepsilon\,|$ is minimized leads to the **normal equations**

$$A'\,A\,x_* = A'\,b$$

where A' is the transpose of A (which was defined both mathematically and in Octave in Study Unit 2). The matrix $(A'\,A)$ is a square $n \times n$ matrix and $(A'\,b)$ is a n-vector, so (provided $\det(A'\,A) \neq 0$) there is a unique solution for x_*.

Here is a simple example on the use of the normal equations in Octave for you to work through.

Activity 4-2

Solve the system of equations

$$x_1 + 2\,x_2 = 0$$
$$3\,x_1 - x_2 = 4$$
$$2\,x_1 + x_2 = 1$$

Octave window

```
> A=[1 2;3 -1;2 1]
       A =

         1   2
         3  -1
         2   1

> b=[0;4;1]
       b =

         0
         4
         1

> At=A'
       At  =

         1   3   2
         2  -1   1

> An=At*A
       An  =

         14   1
         1   6

> det(An)
       ans  = 83.000
> bn=At*b
       bn  =

         14
         -3

> xn=An\bn
```

```
xn =

  1.04819
 -0.67470
```

It is also important to note that, if a system is overdetermined, Octave will automatically find the solution to the normal equations, as shown in the following addition to the above Octave session

```
> xs=A\b
xs =

  1.04819
 -0.67470

> err=A*xs-b
err =

 -0.30120
 -0.18072
  0.42169
```

So, here we have also used Octave to work out the residual (as `err`).

It is important to note that, in the above example, Octave *just gave the answer* to `A\b` as the solution of the normal equations *without any error* or warning message. So, if you use Octave to solve a system of equations, ***do not assume*** that the answer you get is the exact solution until you *check that the residual* (`A*xs-b`) is zero, or at least very small.

Also, note that it is important to check, as we did in the above example, whether or not the determinant of the normal matrix ($A'A$) is zero. While the procedure of transforming to normal form ensures that the normal equations are consistent, if the determinant is zero the solution found by Octave is not unique and there will be infinitely many solutions, as described in the next .

We are now ready to return to the first Activity

Activity 4-3

Find the solution to Activity 4-1

FEEDBACK:
We take the data that we are given and substitute it in to the assumed equation. We get:

$$t = 1: \quad s_0 + u_0\,1 + a\,0.5 \qquad = 6.7118$$
$$t = 2: \quad s_0 + u_0\,2 + a\,2 \qquad = 10.3049$$
$$\ldots\ldots$$
$$t = 10: \quad s_0 + u_0\,10 + a\,50 \qquad = 115.1413$$

This we can write in matrix form as

$$\begin{pmatrix} 1 & 1 & 0.5 \\ 1 & 2 & 2 \\ \vdots & \vdots & \vdots \\ 1 & 10 & 50 \end{pmatrix} \begin{pmatrix} s_0 \\ u_0 \\ a \end{pmatrix} = \begin{pmatrix} 6.7118 \\ 10.3049 \\ \vdots \\ 115.1413 \end{pmatrix}$$

The Octave code to solve the problem is quite straightforward. First, we type in the given data as a column vector *b*; and then generate the matrix *A*. Look carefully at how *A* is constructed, because you do something similar in all problems in which you try to estimate parameters by matching a given formula to given data. Then we check that the determinant of *A'A* is non-zero, and solve the problem using the `A\b` construct. We find:

`xs(1)` $= s_0 \quad = 5.13361$
`xs(2)` $= u_0 \quad = 0.50944$
`xs(3)` $= a \quad = 2.09808$

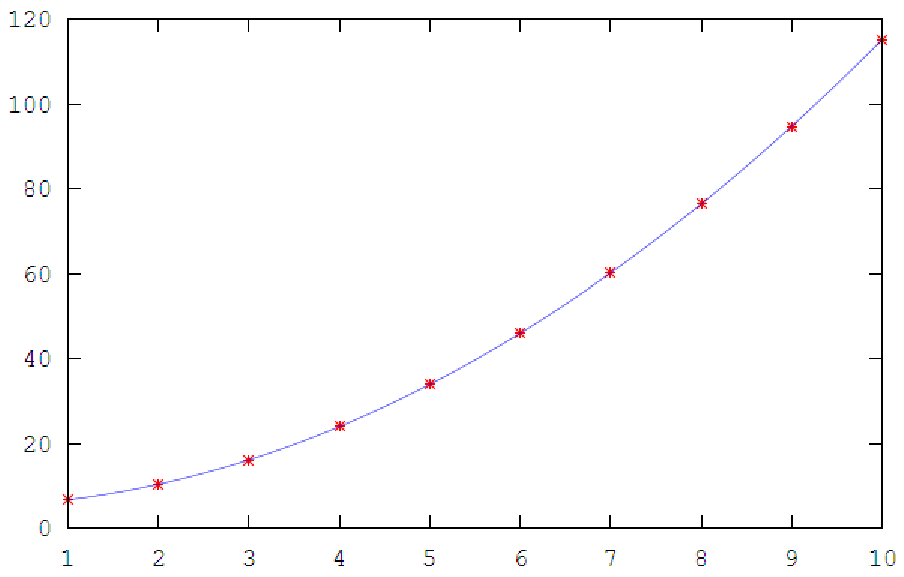
We plot the given data points (as *), as well as the function $s = s_0 + u_0\, t + ½\, a\, t^2$ with the parameters having the values just determined, as a continuous blue line. Finally, we work out the residual, as `err`. We see in the graph that, in this case, we have been able to construct a curve that fits the given data very well.

```
> b=[6.7118 10.3049 16.1415 23.9482 33.8839 45.9405 60.1468 ...
   76.3638 94.6527 115.1413]';
> t=[1:10]';
> A=[ones(10,1) t t.^2/2]; end;
> det(A'*A)
       ans = 1.0890e+005
> xs=A\b
       xs =

         5.13361
         0.50944
         2.09808
> tt=1:0.1:10;
> s=xs(1)+t*xs(2)+xs(3)*t.^2/2;
> plot(t,b,"*r",tt,s,"b")
> err=A*xs-b
       err =

        -0.0197373
         0.0437355
        -0.0382168
         0.0078059
         0.0229036
         0.0151764
        -0.0441759
        -0.0161532
         0.0380445
        -0.0093827
```

The graph output is:



In this case, the given curve fit the data quite well. Following below is another example of curve fitting, where the curve does not provide a particularly good fit.

Activity 4-4

Fit a curve of the form

$$y = a\sin(x) + c$$

to the data:

x	y
1.23	1.8934
1.98	1.9721
2.47	1.4022
5.64	0.2967

FEEDBACK:

Octave code

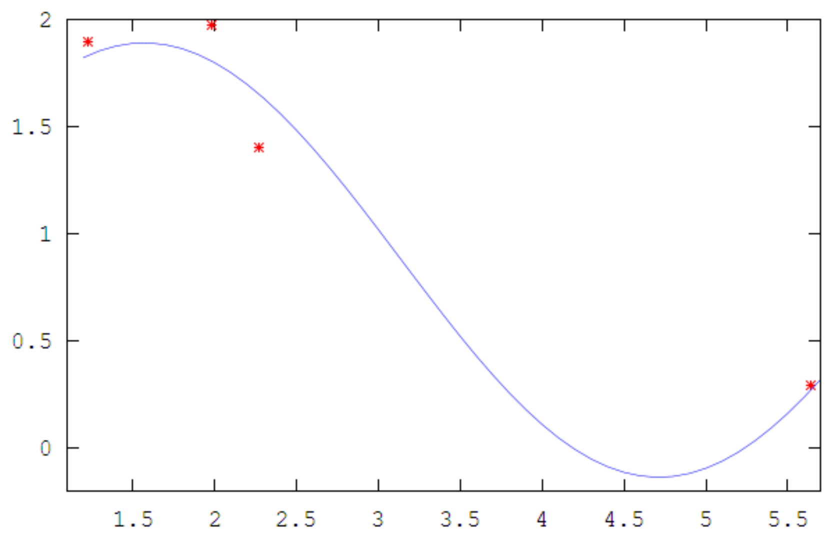
```
> x=[1.23 1.98 2.27 5.64]';
> b=[1.8934 1.9721 1.4022 0.2967]';
> A=[sin(x) ones(4,1)];
> xs=A\b
       xs =

         1.01307
         0.87810

> det(A'*A)
       ans = 6.5990
> err=A*xs-b
       err =

        -0.060496
        -0.164574
         0.251255
        -0.026185

> xx=1.2:0.1:5.7;
> plot(x,b,"*r",xx,xs(1)*sin(xx)+xs(2),"b")
```



We see here that the curve produced does not fit the given data points particularly well.

4.3 Underdetermined systems

A typical example of an underdetermined system is one in which there are more unknowns to find than there are equations given. In such situations, use of the simple `A\b` construct causes Octave to find a solution. Here is an example with 3 equations and 6 unknowns.

Activity 4-5

Find the unknowns in the following underdetermined system.

FEEDBACK:

```
> A=rand(3,6)
       A =
         0.890721  0.161542  0.413566  0.822816  0.132409  0.051145
         0.596242  0.327903  0.573553  0.478045  0.095813  0.322198
         0.566379  0.054946  0.522928  0.824055  0.903582  0.407719
```

```
> b=rand(3,1)
       b =
         0.266035
         0.134006
         0.067834
```

```
> x0=A\b
       x0 =
         0.1970789
         0.0050822
        -0.0092682
         0.1400378
        -0.1219441
        -0.1089772
```

```
> err=A*x0-b
       err =
        -2.2204e-016
        -1.1102e-016
        -2.7756e-016
```

The last statement evaluates the residual, as `err`, confirming that we have indeed found a solution of the problem. But is it the only solution? Theory tells us that if there are m equations in n unknowns with $m < n$, then in the general case we would expect a general solution to involve $n - m$ arbitrary parameters; so in this example we would expect three arbitrary parameters. We look for the additional solutions using a technique that is quite common in many areas of applied mathematics. We write the general solutions as

$$x = x_0 + \sum_i \alpha_i p_i$$

where x_0 is the particular solution already found, α_i are the arbitrary parameters, and p_i are n-vectors that we wish to find. Applying the equation $A\,x = b$, we immediately see that for each i the p_i satisfy

$$A\,p_i = 0.$$

This means that, in the language of linear algebra, we are looking for the **null space** of A. In this module, we will not say any more about the determination of the null space. In Octave, finding the null space is trivial: we just use the pre-defined function `null`, as shown below. The answer is returned as a matrix, the columns of which are the vectors p_i that we are seeking.

```
> N=null(A)
      N =
       -0.582073  0.271316  0.200055
        0.347654  0.687301 -0.360124
       -0.170757 -0.584404 -0.441237
        0.676688 -0.181640  0.069560
       -0.213805  0.280840 -0.251502
        0.086892  0.024746  0.753331
```

```
> A*N(:,1)
      ans =
       1.1102e-016
       0.0000e+000
       1.1102e-016
```

```
> A*N(:,2)
      ans =
       -5.5511e-017
       1.5266e-016
       2.7756e-016
```

```
> A*N(:,3)
      ans =
       1.3878e-017
       -6.9389e-018
       1.5266e-016
```

In the above code, we applied the matrix A to each column of N, and confirmed that it is indeed in the null space of A.

Here is another example of an underdetermined system of equations.

Activity 4-6

Solve,

$$x_1 + 2\,x_2 + 3\,x_3 = 1$$
$$4\,x_1 + 5\,x_2 + 6\,x_3 = 2.$$

Octave code

```
> A=[ 1 2 3;4 5 6]
> b=[1 2]'
> A\b
       ans =

         -0.055556
          0.111111
          0.277778
```

```
> null(A)
       ans =

          0.40825
         -0.81650
          0.40825
```

Thus the general solution to the problem is

$$x = \begin{pmatrix} -0.055556 \\ 0.111111 \\ 0.277778 \end{pmatrix} + \alpha \begin{pmatrix} 0.40825 \\ -0.81650 \\ 0.40825 \end{pmatrix}.$$

But, what happens if we try to solve an underdetermined and inconsistent system?

4.4 Underdetermined and inconsistent system

Sometimes we need to solve a more complex system where it is both underdetermined and inconsistent.

Activity 4-7

Solve:

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \end{pmatrix} x = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$$

Octave code

```
> A=[1 2 1;2 4 2];
> b=[4;2];
> x=A\b
```

```
warning: dgelsd: rank deficient 2x3 matrix, rank = 1
x =
  0.26667
  0.53333
  0.26667
```

```
> null(A)
```

```
ans =
  0.91287  0.00000
 -0.36515 -0.44721
 -0.18257  0.89443
```

Octave gives a warning that there may be a problem, and reports the rank of the matrix *A*. Here the rank is 1 which is less than the number of rows (2). In general, if the rank is less than the number of rows, the system may be inconsistent. What Octave does in such a case is to construct and solve the normal equations; in other words, the particular solution found is not an exact solution to the problem but is the *best approximate solution*. In such cases, we should check the residual, to see by how much the solution fails to satisfy the given system of equations.

Here we find:

```
> err=A*x-b
```

```
err =
 -2.4000
  1.2000
```

4.5 Additional Exercises

1. Find the best straight line ($y = m\,x + c$) fit to the data points
   (x, y) = (0, 1), (2, 0), (3, 1), (3, 2), (3, 1).
   Produce a graph showing the line, together with the given data points as discrete points.

2. Find the cubic polynomial that best fits the data points
   (x, y) = (-1, 14), (0, -5), (1, -4), (2, 1), (3, 22).
   Produce a graph showing the polynomial, together with the given data points as discrete points.

3. The sales figures for a business are as follows for the first six months of the year:
   R40 000, R44 000, R52 000, R64 000, R80 000, R84 000.
   The owner believes that the sales curve can be approximated by a quadratic function. Find the best quadratic fit to the data, and use it to estimate the projected sales for the rest of the year.

4. A formula for the population of the USA is
   $$P(t) = P_0 - ae^{-0.02(t-1800)}$$
   where *t* is the date in years. Some actual data is as follows

Date	Population
1800	5308000
1820	9638000
1840	17069000
1870	38558000
1900	75995000
1930	122775000

1950   150697000

Find values of P_0 and a that give a best fit of the formula to the data. Produce a graph showing the function $P(t)$ against time as a continuous line, together with the given data points as discrete points

5. Write a function file that generates a random solution to an underdetermined system of equations. The function should take a matrix A and a column vector b as inputs. It should check that the number of rows in A and b are the same, and that the number of rows in A is strictly less than the number of columns; if not, generate an error message and exit the function. The function should construct a solution x_0 to the problem, and also find the null space; and then generate a random solution by generating random numbers for the coefficients α_i of the null space vectors p_i. Finally, the function should check that the rank of A is equal to the number of rows; if not, generate a warning message (that the system may be inconsistent), and calculate and print out the residual. Test the function with different matrices to check that all the features work correctly.

6. Consider the system of equations

$$x_1 + x_2 + x_3 = 3$$
$$x_1 + 2\,x_2 + x_3 = 3$$
$$x_1 - x_2 + x_3 = 1$$

Use Octave to show that the system is inconsistent. Construct and solve the normal equations, find the null space, and then construct the best approximate general solution.

4.6 In conclusion

You have now used Octave to solve linear systems in which the numbers of unknowns and equations are not the same, and you have applied this to the problem (which is found in many areas of science) of fitting a given curve to given data.

Study Unit 5: Eigenvalues, eigenvectors and matrix diagonalization

Time period: 15 hours approximately

Learning outcomes
At the end of this Study Unit, you should be able to write programs that:

- Find the eigenvalues and eigenvectors of a square matrix A using the program function `eig`.
- Find, when it exists, a matrix P that diagonalizes A.
- Use the power method to find the dominant eigenvalue and corresponding eigenvector of A.

Some of this knowledge you might have already encountered in MAT103N. More detail will also follow in MAT211R. In this module the focus is a statement (without proof), for mathematical background.

5.1 Summary of mathematical results

Here we state (without any proof) the key definitions and results about eigenvalues and eigenvectors. (Further details are given in other modules such as MAT103N and MAT211R.)

Definition

Given an $n \times n$ matrix A, λ and $\mathbf{v}$ ($\neq \mathbf{0}$) are said to be an **eigenvalue** and associated **eigenvector** if

$$A\,\mathbf{v} = \lambda\,\mathbf{v}$$

5.1.1 Calculation of eigenvalues and eigenvectors

The eigenvalues are found by rewriting the above definition as

$$(A - \lambda I_n)\,\mathbf{v} = \mathbf{0} \qquad (1)$$

where I_n is the $n \times n$ unit matrix, and observe that a non-zero solution for $\mathbf{v}$ can exist only if

$$\det(A - \lambda I_n) = 0$$

which is called the **characteristic equation**. It is a polynomial equation in λ of degree n. By the fundamental theorem of algebra, the equation has n roots. In general, some of the roots may be complex, and it may happen that some roots are repeated. Given an eigenvalue, Eqn. (1) can be solved to find the associated eigenvector (and if the eigenvalue is complex, then in general so is the eigenvector). The result found for an eigenvector is not unique, because you can always multiply an eigenvector by a non-zero scalar and get another eigenvector with the same eigenvalue. Thus often, the solution found for an eigenvector is given in **normalized** form, which means that the vector $\boldsymbol{v}$ is rescaled so that it is a unit vector. Explicitly, the construction is

$$\boldsymbol{v} \rightarrow \boldsymbol{u} = \boldsymbol{v}\,|\boldsymbol{v}|^{-1} \quad \text{with} \quad |\boldsymbol{v}| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

(In Octave, for a vector $\boldsymbol{v}$, $|\boldsymbol{v}|$ is coded as `norm(v)`). Note however that even a normalized eigenvector is not uniquely defined: if it is real, it is not unique up to multiplication by -1, and if it is complex up to multiplication by any complex number c with $|c|$=1.

5.1.2 Some useful properties of eigenvalues and eigenvectors

- In the general case, there are no repeated roots and there are n distinct eigenvalues. It can then be shown that the corresponding eigenvectors form a **complete** set of linearly independent vectors. This means that any n-vector $\boldsymbol{x}$ can be expressed uniquely as a linear sum of eigenvectors,

  $$\boldsymbol{x} = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i$$

  for some (unique) scalar coefficients α_i.

- Now suppose that the characteristic equation has repeated roots, and that there are r distinct eigenvalues with $r < n$. Suppose that there are m linearly independent eigenvectors, then m will be somewhere in the range $r \le m \le n$. If $m = n$ then the set of eigenvectors is complete, and if $m < n$ we say that the set of eigenvectors is **degenerate**.

- If the matrix A is real, and if (λ, $\boldsymbol{v}$) are a complex eigenvalue-eigenvector pair, then their complex conjugates (λ^*, $\boldsymbol{v}^*$) are also an eigenvalue-eigenvector pair.

5.2 The Octave command `eig`

It is very easy in Octave to find the eigenvalues and eigenvectors of a matrix A, using the command `eig`. The syntax is illustrated in the following code

```
> A1=[3 -1 0;-1 2 -1;0 -1 3]
      A1 =
         3  -1   0
        -1   2  -1
         0  -1   3

> [P1 L1]=eig(A1)
      P1 =
        4.0825e-001  -7.0711e-001  -5.7735e-001
        8.1650e-001  -1.4076e-017  5.7735e-001
        4.0825e-001  7.0711e-001  -5.7735e-001

      L1 =
         1   0   0
         0   3   0
         0   0   4
```

We see that the diagonal entries of *L* are the eigenvalues, and the columns of *P* are the corresponding normalized eigenvectors. Here are some more examples

- Complex eigenvalues and eigenvectors

```
> A2=rand(3,3)
      A2 =

         0.618515   0.539806   0.305781
         0.774175   0.111187   0.676873
         0.686104   0.076203   0.474557
```

```
> [P2 L2]=eig(A2)
P2 =

   0.60969 + 0.00000i   0.55806 - 0.17883i   0.55806 + 0.17883i
   0.61906 + 0.00000i  -0.35045 + 0.39556i  -0.35045 - 0.39556i
   0.49502 + 0.00000i  -0.61425 + 0.00000i  -0.61425 - 0.00000i

L2 =

   1.41489 + 0.00000i   0.00000 + 0.00000i   0.00000 + 0.00000i
   0.00000 + 0.00000i  -0.10532 + 0.15067i   0.00000 + 0.00000i
   0.00000 + 0.00000i   0.00000 + 0.00000i  -0.10532 - 0.15067i
```

- Repeated eigenvalues with a degenerate set of eigenvectors

```
> A3=[-3 2;-2 1]
      A3 =

        -3   2
        -2   1
```

```
> format long
> [P3 L3]=eig(A3)
P3 =

   0.707106781186548+0.000000000000000i   0.707106781186548-0.000000000000000i
   0.707106781186547+0.000000002257377i   0.707106781186547-0.000000002257377i

L3 =

  -1.000000000000000+0.000000006384826i   0.000000000000000+0.000000000000000i
   0.000000000000000+0.000000000000000i  -1.000000000000000-0.000000006384826i
```

```
> rank(P3)
      ans =  2
> cond(P3)
      ans = 6.2649e+008
```

Analytically, it is easy to work out the characteristic equation,

$$\lambda^2 + 2\lambda + 1 = 0$$

which clearly has a repeated eigenvalue $\lambda = 1$. Then working out the eigenvectors, we find that there is only one, namely (in normalized form)

$$v = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix}$$

Octave finds the eigenvalues correctly to 10 significant figures, giving an erroneous small imaginary part to each eigenvalue. As a consequence of this error, Octave incorrectly finds that there are two linearly independent eigenvectors (although the difference between the two eigenvectors is very small). Thus it shows that the rank of the eigenvector matrix is 2 when the correct value is 1. However, the large condition number found for the matrix of eigenvectors indicates that the value found for the rank may not be reliable.

This is an **important point**: In cases in which equal, or nearly equal, eigenvalues are found, you should check the condition number of the eigenvector matrix, and a large condition number indicates that the eigenvectors found are not all linearly independent.

- Repeated eigenvalues with a complete set of eigenvectors

```
> A4=[1 -3 3;3 -5 3;6 -6 4]
       A4 =

          1   -3   3
          3   -5   3
          6   -6   4
```

```
> [P4 L4]=eig(A4)
       P4 =

         -0.40825  -0.40825  -0.43794
         -0.40825   0.40825  -0.81576
         -0.81650   0.81650  -0.37781

       L4 =

          4.00000   0.00000   0.00000
          0.00000  -2.00000   0.00000
          0.00000   0.00000  -2.00000
```

```
> cond(P4)
       ans =  4.1103
```

5.3 Matrix diagonalization

Matrix diagonalization is trivial in Octave, because the eigenvector matrix *P* returned by Octave is also the diagonalization matrix as we can check by evaluating $P^{-1}AP$. For example, using the matrices defined in the previous section,

```
> inv(P1)*A1*P1
       ans =

         1.0000e+000  -1.2035e-017  -9.3493e-016
         -5.3668e-017  3.0000e+000  -1.0221e-015
         -1.2821e-016  -3.4044e-017  4.0000e+000
```

```
> inv(P4)*A4*P4
       ans =

         4.0000e+000  -6.3448e-016  -6.6104e-016
         3.9926e-016  -2.0000e+000  -5.3424e-017
         6.2439e-016  -8.4134e-017  -2.0000e+000
```

5.4 The power method

In certain real-world problems we may have to deal with large diagonalizable matrices whose characteristic equation is time-consuming to solve. In Octave, the function `eig` takes only a few seconds to find the eigenvalues and eigenvectors of a matrix of size 500 × 500, so in practice, in this context, large means larger than about 500 × 500. Moreover, the main feature of interest in many real-world problems is the so-called **dominant eigenvalue** i.e. the eigenvalue whose absolute value is larger than the absolute values of all the other eigenvalues. The dominant eigenvalue is particularly important in circumstances where we are interested in the behaviour of $\boldsymbol{x}_p = A^p \boldsymbol{x}_0$ for large p.

We now describe an iterative process to determine an approximation to the dominant eigenvector (the eigenvector corresponding to the dominant eigenvalue). From the approximation to the dominant eigenvector, it is a trivial matter to compute an approximation to the dominant eigenvalue. The procedure which we follow is known as the power method and is quite straightforward.

Now, we describe the algorithm for the power method, and then explain why it works.

5.4.1 Algorithm

Algorithm is defined as a logical step-by-step procedure for solving a mathematical problem in a finite number of steps, often involving repetition of the same basic operation.

Given an $n \times n$ matrix A, we choose an arbitrary non-zero n-dimensional column vector $\boldsymbol{x}_0$ and proceed to construct

$$\boldsymbol{x}_1 = A\,\boldsymbol{x}_0,\ \boldsymbol{x}_2 = A\,\boldsymbol{x}_1,\ \ldots,\ \boldsymbol{x}_i = A\,\boldsymbol{x}_{i-1},\ \ldots,\ \boldsymbol{x}_p = A\,\boldsymbol{x}_{p-1},$$

Then each $\boldsymbol{x}_i$ is an estimate of the (non-normalized) dominant eigenvector, with the estimate becoming better and better as i increases. At each value of i we can construct an estimate μ_i of the dominant eigenvalue using the **Rayleigh quotient** formula

$$\mu_i = (\boldsymbol{x}_{i-1} \,.\, \boldsymbol{x}_i\,) \,/\, (\boldsymbol{x}_{i-1} \,.\, \boldsymbol{x}_{i-1})$$

We estimate the relative error ε_i in $\boldsymbol{x}_i$ using a definition similar to that given in Study Unit 3, section 2,

$$\varepsilon_i = \left| \frac{x_i}{|x_i|} - \frac{x_{i-1}}{|x_{i-1}|} \right|.$$

Our estimate of the relative error is the norm of the difference between successive estimates of $\boldsymbol{x}_i$ normalized. We continue the computation until ε_i is less than a given tolerance, at, say, $i = p$. Then we estimate the dominant eigenvalue as $\lambda = \mu_p$, and the normalized dominant eigenvector as $\boldsymbol{v} = \boldsymbol{x}_p \,/\, |\,\boldsymbol{x}_p\,|$.

5.4.2 Justification/why

We suppose that A has a complete set of eigenvectors, so that the initial estimate $\boldsymbol{x}_0$ can be written as

$$\boldsymbol{x}_0 = \alpha_1\,\boldsymbol{v}_1 + \alpha_2\,\boldsymbol{v}_2 + \ldots + \alpha_n\,\boldsymbol{v}_n$$

where $\boldsymbol{v}_1 \ldots \boldsymbol{v}_n$ are the eigenvectors, and suppose that $\lambda_1 \ldots \lambda_n$ are the corresponding eigenvalues.

Suppose also that λ_1 is the dominant eigenvalue, so that

$$\left|\frac{\lambda_2}{\lambda_1}\right|, \left|\frac{\lambda_3}{\lambda_1}\right|, \ldots, \left|\frac{\lambda_n}{\lambda_1}\right| < 1.$$

We now multiply $\boldsymbol{x}_0$ by A, and do this p times, getting

$$A^p \boldsymbol{x}_0 = \alpha_1 \lambda_1^p \boldsymbol{v}_1 + \alpha_2 \lambda_2^p \boldsymbol{v}_2 + \ldots + \alpha_n \lambda_n^p \boldsymbol{v}_n$$

$$= \alpha_1 \lambda_1^p \left(\boldsymbol{v}_1 + \left(\frac{\alpha_2}{\alpha_1}\right)\left(\frac{\lambda_2}{\lambda_1}\right)^p \boldsymbol{v}_2 + \left(\frac{\alpha_3}{\alpha_1}\right)\left(\frac{\lambda_3}{\lambda_1}\right)^p \boldsymbol{v}_3 + \ldots + \left(\frac{\alpha_n}{\alpha_1}\right)\left(\frac{\lambda_n}{\lambda_1}\right)^p \boldsymbol{v}_n\right).$$

Now as $p \to \infty$, $\left(\frac{\lambda_2}{\lambda_1}\right)^p, \left(\frac{\lambda_3}{\lambda_1}\right)^p, \ldots, \left(\frac{\lambda_n}{\lambda_1}\right)^p \to 0$, so that

$$A^p \boldsymbol{x}_0 \to \alpha_1 \lambda_1^p \boldsymbol{v}_1,$$

as required.

Usually the power method converges quite rapidly, although there are exceptions

- If $\boldsymbol{x}_0$ is chosen such that $\alpha_1 = 0$, then the method cannot converge to $\boldsymbol{v}_1$. In order to avoid this possibility, we usually set $\boldsymbol{x}_0$ randomly, i.e. using Octave's random function, and run the power method for two different initial vectors $\boldsymbol{x}_0$.
- If the largest eigenvalue of A is complex, then its complex conjugate is also an eigenvalue. In this case, although $\lambda_1 \neq \lambda_2$, $|\lambda_1| = |\lambda_2|$ and the power method will probably fail.
- If the largest eigenvalue of A is a multiple root of the characteristic equation, with or without degenerate eigenvectors, then the power method will probably fail.
- Note, however, that although the theorem guarantees convergence only if there is a complete set of eigenvectors, in practice convergence usually does occur in the case that there are degenerate eigenvectors **provided** the associated eigenvalue has an absolute value strictly less than that of the largest eigenvalue.

We now give a function file power_method.m that implements the power method, using a conditional loop that exits once the relative error is below a given tolerance, or the maximum number of iterations has been exceeded. The inputs to the function are the matrix `A`, the tolerance `TOL`, and the maximum number of iterations `max_it`.

File power_method.m

```
function [e_vec lam]=power_method(A,TOL,max_it)
  k=0;
  n=size(A)(1);
  e_vec_old=rand(n,1);
  do
    e_vec_new=A*e_vec_old;
    lam=(e_vec_new'*e_vec_old)/(e_vec_old'*e_vec_old);
    err=norm(e_vec_new/norm(e_vec_new)-
e_vec_old/norm(e_vec_old));
    e_vec_old=e_vec_new;
    k=k+1;
  until((err<TOL) | (k>max_it));
  k
  e_vec=e_vec_new/norm(e_vec_new);
  if (k>max_it)
    disp("ERROR: METHOD DID NOT CONVERGE");
    e_vec=[];
    lam=[];
  endif
endfunction
```

The following extract from the Octave command window shows some examples of the use, and failure, of the power method. First, we apply the method to the various matrices **`A1`** to **`A4`** specified in Section 2.

Feedback: Octave window output

```
> [P L]=power_method(A1,1/10^6,100)
      k =  38
      P =

         0.57735
        -0.57735
         0.57735

      L =  4.0000
```

```
> [P L]=power_method(A2,1/10^6,100)
      k =  8
      P =

         0.60969
         0.61906
         0.49502

      L =  1.4149
```

```
> [P L]=power_method(A3,1/10^6,100)
      k =  101
      ERROR: METHOD DID NOT CONVERGE
      P = [](0x0)
      L = [](0x0)
```

```
> [P L]=power_method(A4,1/10^6,100)
k =  20
P =

   0.40825
   0.40825
   0.81650

L =  4.0000
```

```
> A5=rand(1000,1000);
> [P L]=power_method(A5,1/10^6,100);
      k =  5
> L
      L =  499.93
> [P L]=eig(A5);
> L(1,1)
      ans =  499.93
```

In the last example we generated a large (1000 × 1000) random matrix, and we see that both `eig` and `power_method` give the same result for the largest eigenvalue. Also, if you run the program, you will see that `eig` takes much more time than `power_method`. Note that, because we used a random generator, if you run the program you would get different numerical values.

5.5 Additional Exercises

1. Find the eigenvalues and eigenvectors of the following matrices, using both `eig` and `power_method` (for the dominant eigenvalue and eigenvector). If the power method fails, discuss why. For those matrices that are diagonalizable, give the diagonalization matrix

   a. 2.781344 -1.921334 0.493612 1.367198 -1.014289
      0.015050 -0.205731 0.903377 1.780261 -0.824057

```
-0.087144   0.606003   2.977860  -0.140473  -0.750938
0.212440  -2.477599   0.980236   4.233562  -1.207581
-0.136646  -1.168924   0.453692   0.915245   1.712964
```

b.

```
-1.54575  -3.47002  -1.70112  -2.58917
-3.28104  -2.07998  -1.45597  -2.75629
0.55497   0.94078   2.02863   0.46100
8.94120   9.67047   4.47796   9.09710
```

c.

```
-1.6731385  -3.6381454  -1.8272855  -2.7022868
-1.7080530  -0.0039461   0.1019678  -1.3595453
2.4426950   3.4321965   3.8982919   2.1372124
5.7672696   5.4815125   1.3344020   6.2787928
```

d.

```
4.9541    6.6650    8.1445    2.9998
10.1406   17.3006   14.2773    9.3552
8.9200   10.6881    8.4619    7.7253
-25.6960  -40.6289  -38.0172  -21.7166
```

2. Modify the power method so that the stopping condition is changed to

$$\left|\frac{Ax_n - \mu_n x_n}{|x_n|}\right| < \text{tolerance}$$

where x_n, μ_n are the current estimates of the eigenvector and eigenvalue. Demonstrate the validity of your code by running it on the matrices in question 1.

3. Modify the power method so that it finds the smallest eigenvalue and corresponding eigenvector. You do this by evaluating

$$x_{n+1} = A^{-1} x_n$$

rather than $x_{n+1} = A\, x_n$. When convergence occurs, it is to the eigenvector corresponding to the smallest eigenvalue, and to the inverse of the smallest eigenvalue. Demonstrate the validity of your code by running it on a number of test cases.

4. Consider a fictional species, and suppose that the population can be divided into three different age groups: babies, juveniles and adults. Let the population in year n in each of these groups be

$$\boldsymbol{x}_{(n)} = \begin{pmatrix} x_{b(n)} \\ x_{j(n)} \\ x_{a(n)} \end{pmatrix}$$

The population changes from one year to the next according to $\boldsymbol{x}_{(n+1)} = A\, \boldsymbol{x}_{(n)}$, where the matrix A is

$$A = \begin{pmatrix} \tfrac{1}{2} & 5 & 3 \\ \tfrac{1}{2} & 0 & 0 \\ 0 & \tfrac{2}{3} & 0 \end{pmatrix}$$

In the long term, what will be the relative distribution of the population amongst the age groups?

5. spread of an infectious disease in a town can be modeled (when the number of infected people is much less than the total population) as $\boldsymbol{x}_{(n+1)} = A\,\boldsymbol{x}_{(n)}$, where *n* refers to the month, and

$$\boldsymbol{x}_{(n)} = \begin{pmatrix} x_{s(n)} \\ x_{m(n)} \\ x_{r(n)} \\ x_{d(n)} \end{pmatrix}$$

The disease is spread by mosquito bites, and $x_{m(n)}$ refers to the number of mosquitoes that carry the disease; $x_{s(n)}$ is the number of sick people; $x_{r(n)}$ is the number of people who recover; and $x_{d(n)}$ is the number of people who die. The matrix *A* is

$$A = \begin{pmatrix} 0.7 & 0.8 & 0 & 0 \\ 1.6 & 0 & 0.1 & 0 \\ 0.2 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0 \end{pmatrix}$$

In the long term, at what rate will the incidence of the disease increase every month? Consequently as a public health initiative, the mosquito breeding areas are being sprayed with insecticide, and the matrix *A* changes to *B* where

$$B = \begin{pmatrix} 0.7 & 0.3 & 0 & 0 \\ 0.6 & 0 & 0.1 & 0 \\ 0.2 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0 \end{pmatrix}$$

Show that the effect will be that the disease will be eradicated

5.6 In conclusion

You have now successfully used Octave to find the eigenvalues and eigenvectors of a matrix, and applied this to some real-world problems.

Study Unit 6: Linear programming

Time period: 15 hours approximately

LEARNING OUTCOMES

At the end of this Study Unit, you should be able to:

- Understand the basic concepts of linear programming.
- Use the Octave package `glpk` to solve problems in linear programming
- Express realistic problems in mathematical terms, and then use the Octave package `glpk` to solve them

The focus here is the solving of the linear programming problems with some touching on realistic problems.

6.1 The basic ideas of linear programming

You will recall that in MAT111N we briefly discussed linear programming, which is the problem of trying to maximize or minimize a certain linear function, called the *objective function*, in which the variables were subject to constraints in the form of inequalities. We restricted our attention to problems involving two variables only and this enabled us to use geometric methods in solving them. In essence, the constraint inequalities enabled us to determine a region of the xy-plane in which all the inequalities were satisfied. We called this the *feasible region* of the problem. The objective function could be represented as a plane sitting above the xy -plane and clearly we were only interested in that part of the plane above the feasible region. It was a fairly simple matter to show that the maximum or minimum of the objective function occurred at one of the so-called *extreme points* (i.e. corners) of the feasible region. We therefore simply evaluated the objective function at the various extreme points and this gave us the height of the plane above these points. The largest or smallest height gave us the maximum or minimum value of the objective function subject to the various constraints.

Example 6.1.1
Maximize the function

$$L = 40\,x_1 + 60\,x_2$$

subject to the constraints

$$\begin{aligned}&2\,x_1 + x_2 \le 70\\&x_1 + x_2 \le 40\\&x_1 + 3\,x_2 \le 90\\&x_1,\ x_2 \ge 0\end{aligned}$$

This is a 2D problem so it can easily be solved using the geometric method, as shown in Fig. 6.1.1. The maximum of the function L is found by evaluating it at the various extreme points and we find that the maximum occurs at $C = (15, 25)$ and that the maximum value is $L = 2100$.

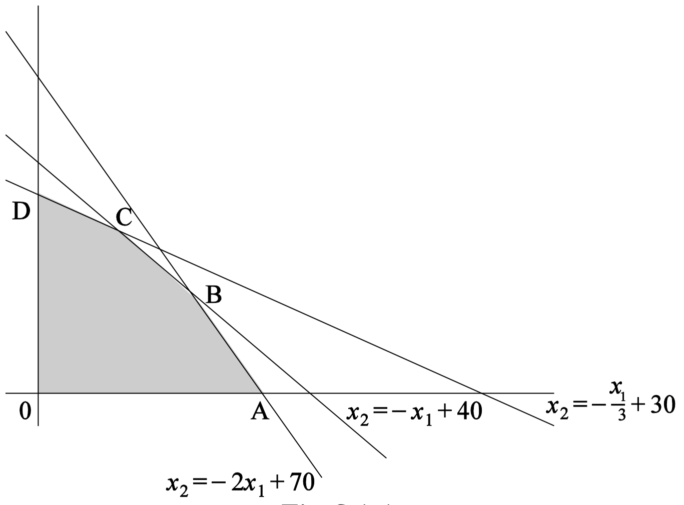


Fig 6.1.1

In practice, linear programming is a widely applied field of mathematics, because it can be used to guide business decisions to maximize profits. For example, the variables represent the different products made by a business, the objective function is a linear sum of the profits made on the different products, and the constraints represent a variety of factors such as factory capacity, market limitations, supply limitations, etc. In practice, the number of variables and constraints can be large, and it is impossible to use a geometric method (which can only be used when there are two, or perhaps three, variables). Thus we have to resort to algebraic methods to solve the problem.

However, there are geometric results that will be useful. The following are easy to show when there are two variables, and in fact hold whatever the number of variables (but we will not prove this)

- The maximum or minimum of the objective function occurs at one of the extreme points (i.e. corners or vertices) of the feasible region
- The feasible region is *convex* (which means that any straight line between two points in the region lies entirely within the region)

The consequence of convexity is that we do not have to worry about local minima or maxima: if an extreme point is maximum (or minimum) relative to neighbouring extreme points, then it is the solution to the problem. For example, suppose we want to maximize $L = x_2$ over the shaded region in Fig. 6.1.2. In both cases (a) and (b), this occurs at the point D; however, in case (a) the region is non-convex and there is another local maximum at the point B, whereas in case (b) the region is convex and there are no other local maxima.

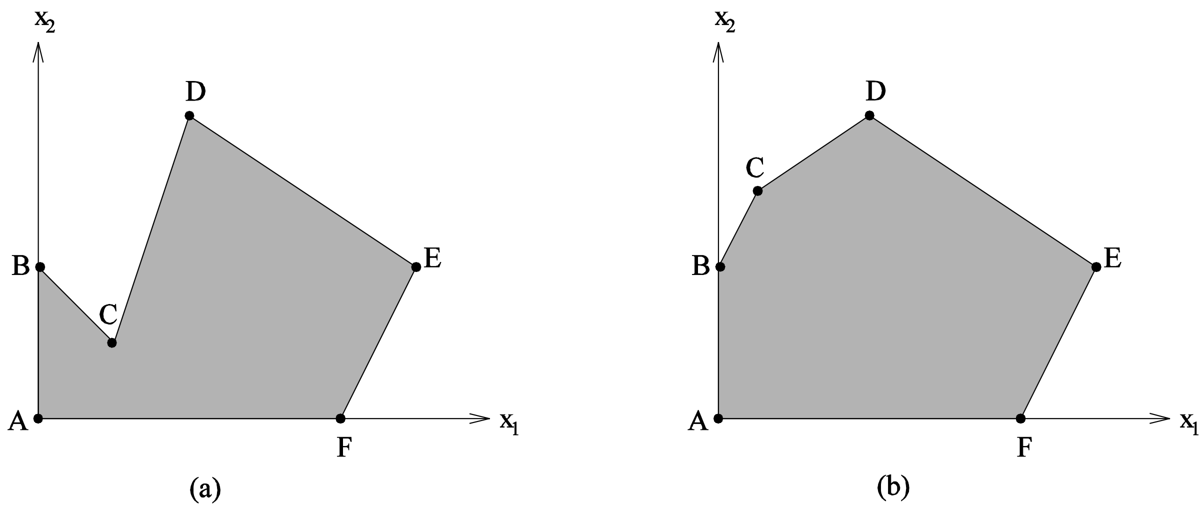


Fig. 6.1.2

In addition, we will assume (as in Example 6.1.1) that all variables are *non-negative*, i.e.

$$x_i \geq 0, \ \ i = 1 \cdots n$$

The reasons for doing so are that, in practice, the variables usually represent quantities like production runs which cannot be negative, and also making the assumption does simplify matters.

You might think that the simplest way to tackle a linear programming problem, given the power of modern computers, is just to find all the extreme points (which are just solutions of linear systems of equations), and then evaluate the objective function at these points. However, for a large system, that problem is far too complex. For example, if there are 50 variables and 100 constraints, the number of points that must be examined is

$$\frac{100!}{50!\,50!} \approx 10^{29}$$

Modern computers run at about 10^9 operations per second, so the above computation would take much longer than the age of the Universe. So instead, an iterative approach is used.

6.2 The simplex method

Although implemented algebraically, the method has a straightforward geometric interpretation. The idea is that we start with a simple, known, extreme point usually the origin. Then we identify neighbouring extreme points and evaluate the objective function there. We choose as our next iterate that extreme point with the largest (or smallest) value of the objective function. The process is continued until moving to another extreme point does not increase (or decrease) the objective function. The method is guaranteed to work because the feasible set is convex, and will converge to a solution after a number of steps of the order of the number of variables plus the number of constraints. The solution found could, in principle, be infinite, but in practice that would mean that the constraints have not been completely specified.

The justification of the algebraic implementation of the simplex method is beyond the scope of this first level module, and instead we simply describe the use of the Octave package `glpk` that implements the simplex algorithm.

After this summary/background, lets see how we can use a program to solve such problems.

6.3 The Octave `glpk` "package"

Octave has a "package" `glpk` to solve problems in linear programming. We illustrate its use by means of applying it to example 6.1.1, and then discuss more generally how the package is used.

The Octave code to solve the problem is

```
> C=[40;60]
       C =

          40
          60

> A=[2 1;1 1;1 3]
       A =

          2   1
          1   1
          1   3

> b=[70;40;90]
       b =

          70
          40
          90

> lb=[]
       lb = [](0x0)
> ub=[]
       ub = [](0x0)
> ctype="UUU"
       ctype = UUU
> vartype="CC"
       vartype = CC
> s=-1
       s = -1
> [xmax,Lmax]=glpk(C,A,b,lb,ub,ctype,vartype,s)
       xmax =

          15.000
          25.000

       Lmax =  2100
```

We now examine the above code and describe how it relates to the mathematical specification of the problem.

- `C` is a column vector such that $L{=}Cx'$ (with x a column vector, x' a row vector) is the function to be minimized or maximized

- `A` is a matrix such that Ax expresses the left hand side of the constraints
- `b` is a column vector that specifies the right hand side of the constraints
- `lb` is a column vector that specifies the lower bound of each element in x. If this lower bound is always 0, then, according to the Octave manual, we do not need to specify `lb` and simply define it as the empty matrix. **However**, in some cases we have found that Octave is unstable when `lb` is not set explicitly to 0
- `ub` is a column vector that specifies the upper bound of each element in x. If this upper bound is always infinity, then we do not need to specify `ub` and simply define it as the empty matrix
- `ctype` is a string of characters containing the sense of each constraint in the constraint matrix. Each element of the string may be one of the following values
  - U An inequality constraint with an upper bound ($\le b_i$).
  - S An equality constraint ($= b_i$).
  - L An inequality constraint with a lower bound ($\ge b_i$).

  Here, there are 3 constraints each of which is an upper bound
- `vartype` is a string of characters with length equal to the size of x. For our purposes, each entry will always be C
- `s` is either -1 meaning that L must be maximized, or +1 meaning that L must be minimized
- `xmax` is the solution found for x that maximizes (or minimizes) L
- `Lmax` is the maximum (or minimum) value found for L that satisfies all the constraints

Here are some more examples

ACTIVITY 6.2.1

Maximize the function

$$L = 2x_1 + 5x_2 + 4x_3$$

subject to the constraints

$$x_1 + 2x_2 + x_3 \le 4$$
$$x_1 + 2x_2 + 2x_3 \le 6$$
$$x_1,\ x_2,\ x_3 \ge 0.$$

The Octave code is

```
> C=[2;5;4]
       C =

          2
          5
          4
```

```
> A=[1 2 1;1 2 2]
       A =

          1   2   1
          1   2   2
```

```
> b=[4;6]
       b =

          4
          6
```

```
> lb=[]
        lb = [](0x0)
> ub=[]
        ub = [](0x0)
> ctype="UU"
        ctype = UU
> vartype="CCC"
        vartype = CCC
> s=-1
        s = -1
> [xmax,Lmax]=glpk(C,A,b,lb,ub,ctype,vartype,s)
        xmax =

           1.6934e-316
           1.0000e+000
           2.0000e+000

        Lmax =  13
```

Example 6.2.2

Minimize the function

$$L = -2\,x_1 + x_2$$

subject to the constraints

$$2\,x_1 + x_2 \le 20$$
$$x_1 - x_2 \le 4$$
$$-x_1 + x_2 \le 5$$
$$x_1,\ x_2 \ge 0$$

The Octave code is

```
> C=[-2;1]
        C =

          -2
           1

> A=[2 1;1 -1;-1 1]
        A =

           2   1
           1  -1
          -1   1

> b=[20;4;5]
        b =

           20
            4
            5

> lb=[]
        lb = [](0x0)
> ub=[]
        ub = [](0x0)
> ctype="UUU"
        ctype = UUU
> vartype="CC"
        vartype = CC
> s=1
        s =  1
```

```
> [xmin,Lmin]=glpk(C,A,b,lb,ub,ctype,vartype,s)
      xmin =

         8
         4

      Lmin = -12

      octave-3.0.0.exe:48> C=[2.5 2 1.5 3]';
      octave-3.0.0.exe:49> A=[1 1 1 1;0 1 0 1;1 1 0 0];
      octave-3.0.0.exe:50> b=[100000 50000 55000]';
      octave-3.0.0.exe:52> lb=[0 10000 0 0]';
      octave-3.0.0.exe:53> ub=[50000 100000 40000 100000]';
      octave-3.0.0.exe:54> ctype="UUU";
      octave-3.0.0.exe:55> vartype="CCCC";
      octave-3.0.0.exe:56> s=-1;
      octave-3.0.0.exe:57>
      [xmax,Lmax]=glpk(C,A,b,lb,ub,ctype,vartype,s)
      xmax =

         45000
         10000
          5000
         40000

      Lmax =  260000
```

6.4 More realistic linear programming examples

We now give some real-life examples of linear programming problems. In each case, the first step is to translate the description into a mathematical formulation, i.e. the specification of the variables, the constraints and the objective function. We then solve the problems using the `glpk` package.

Activity 6.3.1

As an example: The Milko Dairy can receive no more than 100 000 litres of milk per day. Due to a long-term contract, at least 10 000 litres of each day's milk must be used for cheese manufacture. The balance can be used for bottled milk, butter or yoghurt. At today's market prices, the contribution to profit and fixed cost of each litre of milk, when put to these uses, is as follows

Butter	R2.50
Cheese	R2.00
Bottled milk	R1.50
Yoghurt	R3.00

The butter equipment can handle up to 50 000 litres of milk per day, and the milk equipment up to 40 000 litres. Part of the yoghurt and cheese processing uses the same equipment, and this imposes a limit on the combined usage of 50 000 litres per day. The butter and cheese packaging equipment can handle a combined usage of at most 55 000 litres per day.

What mix of products should the company produce so as to maximize profit?

Solution/Feedback
We start by defining our variables:

x_1	Number of litres of milk used for butter
x_2	Number of litres of milk used for cheese
x_3	Number of litres of milk used for bottled milk
x_4	Number of litres of milk used for yoghurt

Then the objective function is

$$L = 2.5\,x_1 + 2.0\,x_2 + 1.5\,x_3 + 3.0\,x_4$$

The constraints are

$$x_1 + x_2 + x_3 + x_4 \le 100\,000$$
$$x_2 \ge 10\,000$$
$$x_1 \le 50\,000$$
$$x_3 \le 40\,000$$
$$x_2 + x_4 \le 50\,000$$
$$x_1 + x_2 \le 55\,000$$

Octave code

```
> C=[2.5 2 1.5 3]';
> A=[1 1 1 1;0 1 0 1;1 1 0 0];
> b=[100000 50000 55000]';
> lb=[0 10000 0 0]';
> ub=[50000 Inf 40000 Inf]';
> ctype="UUU";
> vartype="CCCC";
> s=-1;
> [xmax,Lmax]=glpk(C,A,b,lb,ub,ctype,vartype,s)
      xmax =

         45000
         10000
          5000
         40000

      Lmax =  260000
```

Comments
Thus the Milko Dairy will maximize its trading profit for the day at R260 000 by using 45 000 litres of milk for butter, 10 000 for cheese, 5 000 for bottled milk and 40 000 for yoghurt. In the Octave code, we have expressed constraints that involve only a single variable by setting lower bounds or upper bounds on the variables by specifying the vectors `lb` and `ub`. In the vector `ub`, we then set the upper bound as `Inf`, meaning infinity, for those variables (x_2 and x_4) that do not

have explicit upper bounds. Then the constraint matrix A and right hand side b represent only the three constraints comprising two or more variables.

Example 6.3.1

The ABC clothing manufacturing company has a number of different product lines, T-shirts, jeans, dresses, jackets, coats, etc. Let us call these products P_1, P_2, P_3 and P_4. The manufacture of the clothing requires raw materials including different types and quality of raw cloth, dyes, accessories like buttons, buckles, zips, etc., packaging materials, etc. and we call the raw materials R_1, R_2, R_3 , R_4 and R_5. Each product line requires different quantities of raw materials, and the following table shows how much of each raw material should be available in order to produce one unit of each product line

	R_1	R_2	R_3	R_4	R_5
P_1	0.1	0.09	0.095	0.03	0.02
P_2	0.23	0.17	0.132	0.19	0.03
P_3	0.17	0.19	0.201	0.02	0.021
P_4	0.11	0.08	0.123	0.04	0.07

The manufacturing process involves different departments in the factory, dyeing, cutting and sewing, quality inspection, and packaging, and we call these departments D_1 to D_4. Each product line requires time (in hours) in the various departments as follows

	D_1	D_2	D_3	D_4
P_1	0.7	1.1	0.8	0.2
P_2	1.1	0.85	0.95	0.3
P_3	0.6	0.9	0.6	0.4
P_4	0.9	1.2	0.4	0.2

Management needs to set a production schedule for the next month. The buying department has obtained quotations for the various raw materials, although some items are in short supply and there are supply limitations

	R_1	R_2	R_3	R_4	R_5
Price (Rands)	3050	3550	4800	3970	3020
Supply limitations	60		70		

The sales department advises current market rates for the product lines, together with any upper limits that should not be exceeded to avoid market over-supply and price depression.

	P_1	P_2	P_3	P_4
Price (Rands)	6200	7000	8000	7300
Sales limit	100	200	200	200

In addition, due to an outstanding contract, at least 10 units of P_1 must be produced.
Each department can provide 400 hours per month. How much of each product line should be produced so as to maximize the company's trading profit?

Solution
We start by defining our variables:

$x_1 = P_1, x_2 = P_2, \; x_3 = P_3, x_4 = P_4, x_5 = R_1, x_6 = R_2, x_7 = R_3, x_8 = R_4, x_9 = R_5$

The objective function is the income expected from sales of each product line, less the cost of the raw materials,

$$L = 6200\,x_1 + 7000\,x_2 + 8000\,x_3 + 7300\,x_4 - 3050\,x_5 - 3550\,x_6 - 4800\,x_7 - 3970\,x_8 - 3020\,x_9$$

The constraints due to the time available in each department are

$$0.7\,x_1 + 1.1\,x_2 + 0.6\,x_3 + 0.9\,x_4 \leq 400$$
$$1.1\,x_1 + 0.85\,x_2 + 0.9\,x_3 + 1.2\,x_4 \leq 400$$
$$0.8\,x_1 + 0.95\,x_2 + 0.6\,x_3 + 0.4\,x_4 \leq 400$$
$$0.2\,x_1 + 0.3\,x_2 + 0.4\,x_3 + 0.2\,x_4 \leq 400$$

The amounts of raw materials required for each product line translate into equality constraints

$$-\,0.1\,x_1 - 0.23\,x_2 - 0.17\,x_3 - 0.11\,x_4 + x_5 = 0$$
$$-\,0.09\,x_1 - 0.17\,x_2 - 0.19\,x_3 - 0.08\,x_4 + x_6 = 0$$
$$-\,0.095\,x_1 - 0.132\,x_2 - 0.201\,x_3 - 0.123\,x_4 + x_7 = 0$$
$$-\,0.03\,x_1 - 0.19\,x_2 - 0.02\,x_3 - 0.04\,x_4 + x_8 = 0$$
$$-\,0.02\,x_1 - 0.03\,x_2 - 0.021\,x_3 - 0.07\,x_4 + x_9 = 0$$

Variables x_2 to x_9 are subject to a lower bound of 0, and for x_1

$$x_1 \geq 10$$

Many of the variables are subject to an upper bound

$$x_1 \leq 100, x_2 \leq 200, x_3 \leq 200, x_4 \leq 200, x_5 \leq 60, x_7 \leq 70$$

The Octave code is written in a .m file

File ex6_3_2.m
```
ctype="UUUUSSSSS";
lb=[10 0 0 0 0 0 0 0 0];
ub=[100 200 200 200 60 Inf 70 Inf Inf];
s=-1;
b=[400 400 400 400 0 0 0 0 0]';
vartype="CCCCCCCCC";
A=[0.7 1.1 0.6 0.9 0 0 0 0 0;
1.1 0.85 0.9 1.2 0 0 0 0 0;
0.8 0.95 0.6 0.4 0 0 0 0 0;
0.2 0.3 0.4 0.2 0 0 0 0 0;
-0.1 -0.23 -0.17 -0.11 1 0 0 0 0;
-0.09 -0.17 -0.19 -0.08 0 1 0 0 0;
-0.095 -0.132 -0.201 -0.123 0 0 1 0 0;
-0.03 -0.19 -0.02 -0.04 0 0 0 1 0;
-0.02 -0.03 -0.021 -0.07 0 0 0 0 1];
C=[6200 7000 8000 7300 -3050 -3550 -4800 -3970 -3020]';
[xmax,Lmax]=glpk(C,A,b,lb,ub,ctype,vartype,s)
```

Feedback: sOctave window

```
> ex6_3_2
      xmax =

         10.000
         38.411
        200.000
        146.959
         60.000
         57.187
         64.296
         17.476
         15.839

      Lmax = 2.1918e+006
```

Comments

Thus the company will maximize its trading profit for the month at R2 191 800 by producing 10 units of product P_1, 38 units of product P_2, 200 units of product P_3, and 147 units of product P_4. The company will need to order 60 units of raw material R_1, 57 units of raw material R_2, 64 units of raw material R_3, 17 units of raw material R_4, and 16 units of raw material R_5.

6.5 Additional Exercises

1. Find the maximum value as well as the point at which the maximum occurs of
   $L = x_1 + 2\,x_2 + 3\,x_3$
   subject to the constraints
   $x_1 + x_2 + 2\,x_3 \leq 8$
   $3\,x_1 + 3\,x_2 + x_3 \leq 9$
   $x_1, x_2, x_3 \geq 0$

2. Find the minimum value as well as the point at which the minimum occurs of
   $L = -3\,x_1 - 4\,x_2 + x_3$
   subject to the constraints
   $-x_1 + x_2 + 2\,x_3 \leq 5$
   $2\,x_1 + x_2 + x_3 \leq 20$
   $x_1, x_2, x_3 \geq 0$

3. Find the minimum value as well as the point at which the minimum occurs of
   $L = -2\,x_1 - 5\,x_2 + x_3$
   subject to the constraints
   $x_1 + 2\,x_2 - x_3 \leq 6$
   $x_2 + 2\,x_3 \leq 6$
   $2\,x_2 + x_3 \leq 4$
   $x_1, x_2, x_3 \geq 0$

4. Find the maximum value as well as the point at which the maximum occurs of
   $L = 2\,x_1 + 3\,x_2 + 4\,x_3 + 3\,x_4$
   subject to the constraints
   $1.5\,x_1 + 2\,x_2 + 1.5\,x_3 + x_4 \leq 30$
   $1\,x_1 + 2\,x_2 + 1\,x_3 + 3\,x_4 \leq 45$
   $5\,x_1 + 4\,x_2 + 7\,x_3 + 2\,x_4 \leq 65$

$6\,x_1 + 3\,x_2 + 7\,x_3 + 4\,x_4 \leq 60$
$8\,x_1 + 4\,x_2 + 8\,x_3 + 2\,x_4 \leq 70$
$x_1, x_2, x_3, x_4 \geq 0$

5. The Suitcase Manufacturing Company produces a number of different types of suitcase of varying qualities, which are called S_1, S_2, S_3, S_4 and S_5. The manufacturing process involves different departments in the factory, and we call these departments D_1 to D_6. Each suitcase requires time (in minutes) in the various departments as follows

	D_1	D_2	D_3	D_4	D_5	D_6
S_1	10	15	10	12	5	5
S_2	15	20	16	20	8	5
S_3	21	25	20	20	20	8
S_4	26	21	28	25	25	10
S_5	33	28	30	29	34	15

The contribution to gross profit (i.e., the selling price less the cost of raw materials) of each type of suitcase is given in the following table, which also shows the minimum number of each type of suitcase that must be produced together with the maximum number (in terms of contracts with retail stores)

	S_1	S_2	S_3	S_4	S_5
Profit (Rands)	120	150	235	300	350
Minimum number	200	100	100	100	100
Maximum number	500	300	300	300	300

In addition, there is, this month, a supply limitation on the locks used on the higher quality suitcases (S_3, S_4, and S_5), and the total production of these suitcases cannot exceed 600.

Each department can provide 24000 minutes per month, except Department D_6, which can only offer 15000 minutes. How much of each product line should be produced so as to maximize the company's trading profit?

6. A company receives orders to deliver its goods to three different cities as follows

City	A	B	C
Order	22	21	25

where the quantity of the order is in truckloads. The company has sufficient stock in its warehouses and a truckload of goods can be delivered from any warehouse to any city. However, there are a limited number of trucks available at each warehouse

Warehouse	P	Q	R
Trucks	17	31	26

The variable costs (in Rands) per truckload to deliver goods from each warehouse to each destination are

	City	A	B	C
Warehouse				
	P	6000	5000	4000
	Q	5000	5500	6000
	R	9000	8500	8000

What is the cheapest delivery schedule and what is its cost?

[Hint: Choose the variables as x_1 to x_9 where x_1 is the number of truckloads from P to A, x_2 is the number from P to B, ... , , x_9 is the number from R to C]

6.6 In conclusion

You have now successfully used Octave to solve linear programming problems that cannot be tackled by the simple graphical method. You have also successfully solved some real-world linear programming problems.