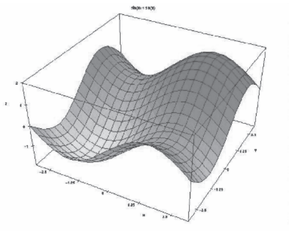
## DEPARTMENT OF MATHEMATICAL SCIENCES



# COMPUTER ALGEBRA:
Applied Mathematics

Only study guide for **APM2616**

J Munganga

University of South Africa
Pretoria

Printed and published by the
University of South Africa
Muckleneuk, Pretoria

Page-layout by the Department
CSET Mathematical

APM2616/1/2014

70100330

# CONTENTS

# STUDY UNIT 1

## 1.1 Introduction

This study unit is primarily about the practicalities of gaining access, and running, the software needed for the module. There is also a demonstration of the capabilities of a modern computer algebra system. The software that you will be using is

- Computer algebra: MuPAD.
- Scientific document preparation: LaTeX

Detailed information on how to

- access the software,
- install it,
- check that it is running correctly,

is given in Tutorial Letter 101.

You should now make sure that MuPAD and LaTex are installed and running correctly on your computer.

## 1.2 Examples of MuPAD's capabilities

Below, we reproduce a MuPAD session, which demonstrates the capabilities of the system. You should:

- Instantiate a MuPAD session, on your own computer.
- Repeat the input shown below.
- Then make your own variations so that you are using MuPAD to solve problems that are similar, but not identical, to those in the demonstration.

If the system objects to a command you type, you can use the online help (? command) to check the syntax required for the command.

The idea is for you to learn about the system by playing around with it. Have fun!

```
i1:=10!;
if1:=factor(i1);
                                   3628800
                                  8  4  2
                                 2  3  5  7
```

```
f1:=y^3+6*y^2*z+12*y*z^2+8*z^3;
ff:=factor(f1);
                         3      3         2      2
                        y  + 8 z  + 12 y z  + 6 y  z

                                          3
                                 (y + 2 z)
```

```
dfy:=diff(f1,y);
dfz:=diff(f1,z);
check1:=diff(dfy,z)-diff(dfz,y);
                                       2       2
                           12 y z + 3 y  + 12 z
                                       2       2
                           24 y z + 6 y  + 24 z
                                     0

d1:=simplify(diff(x^(sin(x^(3/b)))/sqrt(3*x^2-sin(a/x^2)),x));

/
|
|                 /  3 \
|      /  3 \     |  - |
|      |  - |     |  b |
|      |  b |  sin\ x  / + 2 /    2 /    2      / a  \ \1/2
| b sin\ x  / x              | 3 x  | 3 x  - sin| -- | |    -
|                            |      |           |  2 | |
\                            \      \           \ x  / /


      / a  \ /    2      / a  \ \1/2 \
   sin| -- | | 3 x  - sin| -- | |    | -
      |  2 | |           |  2 | |    |
      \ x  / \           \ x  / /    /
                     /  3 \
                     |  - |
                     |  b |
          / a  \  sin\ x  / /    2      / a  \ \1/2
   a b cos| -- | x          | 3 x  - sin| -- | |    -
          |  2 |            |           |  2 | |
          \ x  /            \           \ x  / /
           /  3 \
           |  - |                                        /  3 \
           |  b |                                        |  - |
        sin\ x  / + 4 /    2      / a  \ \1/2            |  b |
   3 b x              | 3 x  - sin| -- | |    + ln(x) cos\ x  /
                      |           |  2 | |
                      \           \ x  / /
```

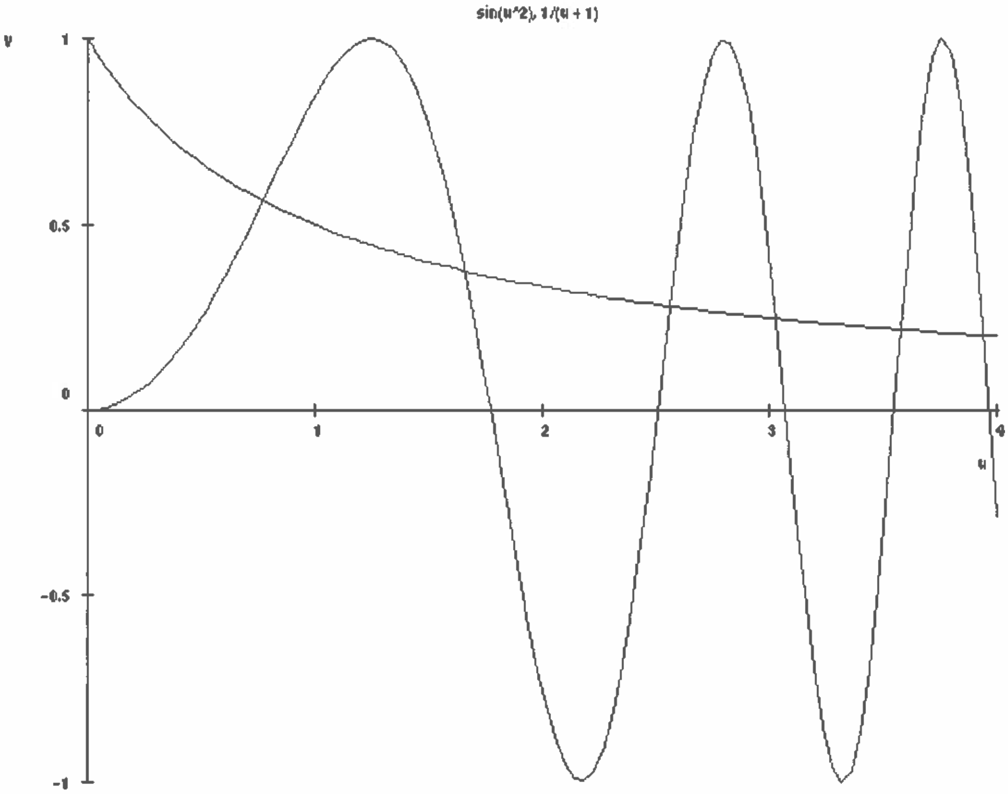
```
               /  3 \
               |  - |
               |  b |
    2 b + b sin\ x  / + 3
    ---------------------
              b           /    2 /    2      / a  \ \1/2
   x                      | 9 x  | 3 x  - sin| -- | |    -
                          |      |           |  2 | |
                          \      \           \ x  / /


                                         \
                                         |
                                         |
                                         |
                                         |
        / a  \ /    2      / a  \ \1/2 \ |   /    3 /    2      / a  \ \2 \
   3 sin| -- | | 3 x  - sin| -- | |    | | / | b x  | 3 x  - sin| -- | |  |
        |  2 | |           |  2 | |    | |   |      |           |  2 | |  |
        \ x  / \           \ x  / /    / /   \      \           \ x  / /  /

d1s:=subs(d1,b=2,a=PI,x=PI/2):
n1:=float(d1s);
                               -0.3045826756
f2:=x^2*sin(a*x);
                                 2
                                x   sin(a x)


i2:=int(f2,x);
                                            /  2      \
                    2 x sin(a x)            | x    2  |
                    ------------ - cos(a x) | -- - -- |
                          2                 | a     3 |
                         a                  \      a  /


d2:=diff(i2,x);
check2:=simplify(d2-f2);
```

```
                                  4


                                            /  2      \
                    2 sin(a x)              | x    2  |
                    ---------- + a sin(a x) | -- - -- |
                         2                  | a     3 |
                        a                   \      a  /
                                     0

f3:=1/x/(x+2)/(x+3);
i3:=int(f3,x);
                                     1
                             -----------------
                             x (x + 2) (x + 3)
                       ln(x)   ln(x + 2)   ln(x + 3)
                       ----- - --------- + ---------
                         6         2           3

i3d:=int(f3,x=1..infinity);
p3:=partfrac(f3);


                               ln(3)   ln(4)
                               ----- - -----
                                 2       3
                         1        1           1
                        --- - --------- + ---------
                        6 x   2 (x + 2)   3 (x + 3)

s3:=series(f3,x=1);
                                        2               3
       /   19 x          \   265 (x - 1)    3355 (x - 1)
1/12 + | - ---- + 19/144 | + ------------ - ------------- +
       \   144           /       1728           20736
                4                 5
   41041 (x - 1)    495859 (x - 1)             6
   -------------- - --------------- + O((x - 1) )
       248832           2985984


s3inf:=series(f3,x=infinity,10);

      1    5    19   65   211   665   2059   6305   19171    /  1  \
      -- - -- + -- - -- + --- - --- + ---- - ---- + ----- + O| --- |
       3    4    5    6    7     8      9     10      11     |  12 |
      x    x    x    x    x     x      x     x       x       \ x   /
```

```
e4:=expand(cos(2*y+b*z));

            2                                             2
      cos(y)  cos(b z) - 2 cos(y) sin(y) sin(b z) - sin(y)  cos(b z)

f5:=sin(6*x)/(2*x);
l0:=limit(f5,x=0);
linf:=limit(f5,x=infinity);
                                 sin(6 x)
                                 --------
                                   2 x
                                     3
                                     0
F:=x->sin(x)*x^2;
f6:=F(a-b);
e6:=expand(f6);
                              x -> sin(x)*x^2
                                   2
                            (a - b)  sin(a - b)

                                             2
2 a b cos(a) sin(b) - 2 a b cos(b) sin(a) - a  cos(a) sin(b) +
    2                  2                  2
   a  cos(b) sin(a) - b  cos(a) sin(b) + b  cos(b) sin(a)

solve(x^2-x+3*y=0,x),solve(x^2-x+3*y=0,y);

         {                 1/2            1/2       }  {      2 }
         {       (1 - 12 y)     (1 - 12 y)          }  { x   x  }
         { 1/2 - -------------, ------------- + 1/2 }, { - - -- }
         {             2              2             }  { 3   3  }

eqns1:={x^2+y=1,x-y=2};
solve(eqns1,{x,y});
                                           2
                          {x - y = 2, y + x  = 1}
{ --         1/2                1/2       --
{ |        13                 13           |
{ |  x = - ----- - 1/2, y = - ----- - 5/2  |,
{ --         2                  2         --
   --       1/2            1/2       -- }
   |      13             13           | }
   |  x = ----- - 1/2, y = ----- - 5/2  | }
   --       2                2       -- }
```

```
eqns2:={a*x+3*y-2*z=4,x=4*y-2*z+a,y-x=4*z-5};
solve(eqns2,{x,y,z});

         {x = a + 4 y - 2 z, y - x = 4 z - 5, 3 y - 2 z + a x = 4}
       { --                        2                  2       -- }
       { |      5 a + 33      4 a - 2 a  + 9      17 a - a  + 3  | }
       { |  x = --------, y = -------------, z = -------------  | }
       { --     7 a + 6           7 a + 6           14 a + 12   -- }

de1:=ode(y''(x)+y'(x)+2*y(x)=0,y(x));
solde1:=solve(de1);

           ode(2 y(x) + diff(y(x), x) + diff(y(x), x, x), y(x))

       {                 /    1/2 \                  /    1/2 \ }
       {       /   x \   | x 7    |        /   x \   | x 7    | }
       { C3 exp| - - | cos| ------ | + C4 exp| - - | sin| ------ | }
       {       \   2 /   \   2    /        \   2 /   \   2    / }

de2:=ode(y'(x)=(2+(x+y(x))^2)/cosh(y(x)),y(x));
solve(de2);

                     /                          2             \
                     |                (x + y(x))  + 2         |
                  ode| diff(y(x), x) - ---------------, y(x) |
                     \                    cosh(y(x))          /
                   /    /                          2             \ \
                   |    |                (x + y(x))  + 2         | |
           solve| ode| diff(y(x), x) - ---------------, y(x) | |
                   \    \                    cosh(y(x))          / /

F:=(x,Y)->[(2+(x+Y[1])^2)/cosh(Y[1])];
numeric::odesolve(0..1,F,[0]);
numeric::odesolve(0..2,F,[0]);
numeric::odesolve(0..3,F,[0]);

   (x, Y) -> [(2 + (x + Y[1])^2)/cosh(Y[1])]
                                 [2.475354047]
                                 [4.181702534]
                                 [5.158937655]
f7:=1/sqrt(½-sin(x)^2);
```

```
i7:=int(f7,x=0..PI/6);
```

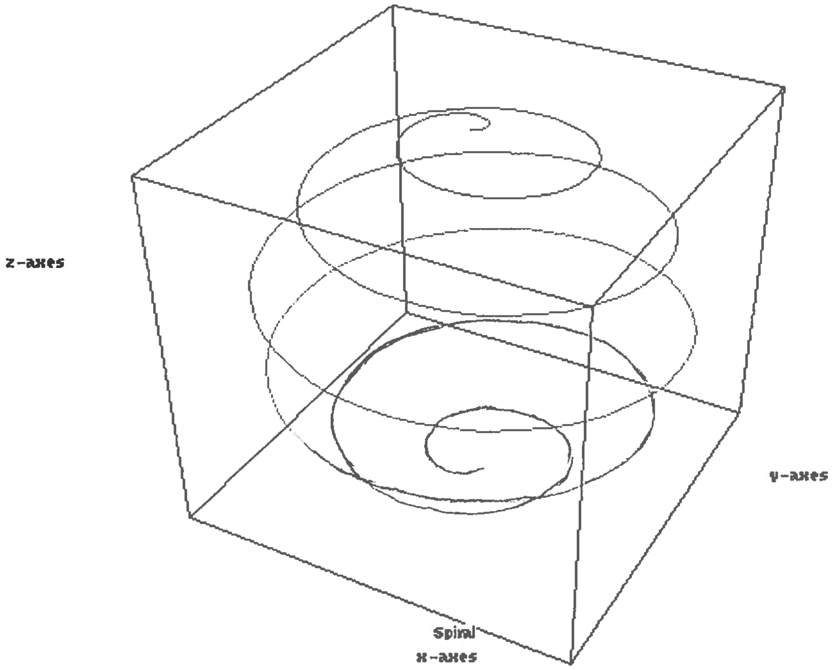


```
                         1
               ------------------
                              2 1/2
              (1/2 - sin(x) )
       /            1                    PI \
   int| ------------------, x = 0..-- |
       |                  2 1/2          6   |
       \ (1/2 - sin(x) )                     /
```

```
n7:=numeric::quadrature(f7,x=0..PI/6);
                                            0.8260178763
```

```
plotfunc2d(sin(u^2),1/(1+u),u=0..4);
```

```
plot3d(Axes = Box, Ticks = 0, Scaling = Constrained,
       Title = "Spiral", TitlePosition = Below,
[Mode = Curve,
[cos(12*u*PI)*sin(u*PI),
         sin(12*u*PI)*sin(u*PI),
         cos(u*PI)],
        u = [0, 1], Grid = [50], Smoothness = [5]
       ]);
```



```
A := matrix([[0, 1, a, 2], [1, a, 2, 0], [2, 0, 5, a],[1, 2, 0, a]]);
                              +-           -+
                              |  0, 1, a, 2  |
                              |              |
                              |  1, a, 2, 0  |
                              |              |
                              |  2, 0, 5, a  |
                              |              |
                              |  1, 2, 0, a  |
                              +-           -+
linalg::det(A);
                                  2    3
                           4 - 2 a  - a  - 13 a
Ai:=A^(-1);
                    array(1..4, 1..4,
                                              2
                                   - 4 a - 5 a
                      (1, 1) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                          2
                                 5 a + 2 a  - 20
                      (1, 2) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                           3
                                  - 2 a + a  + 8
                      (1, 3) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                            3
                                    12 a - a
                      (1, 4) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                       3 a
                      (2, 1) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                      2
                                     a  + 10
                      (2, 2) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                        2
                                     - a  - 4
                      (2, 3) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                       2
                                      a  - 2
                      (2, 4) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
```

```
                                         2
                                     2 a + a
                      (3, 1) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                     - a + 8
                      (3, 2) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                     3 a - 4
                      (3, 3) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                         5 a
                      (3, 4) = - --------------------,
                                           2    3
                                 13 a + 2 a  + a  - 4
                                     5 a - 2
                      (4, 1) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                    - 4 a - 5
                      (4, 2) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                          2
                                   2 a - a  + 2
                      (4, 3) = --------------------,
                                         2    3
                               13 a + 2 a  + a  - 4
                                        2
                                     2 a  + 1
                      (4, 4) = --------------------
                                         2    3
                               13 a + 2 a  + a  - 4
                    )
B:= subs(A,a=0);
                             +-            -+
                             |  0, 1, 0, 2  |
                             |              |
                             |  1, 0, 2, 0  |
                             |              |
                             |  2, 0, 5, 0  |
                             |              |
                             |  1, 2, 0, 0  |
                             +-            -+
linalg::eigenvalues(B);
{
{            3                1/2     1/3
{ -1, ---------------- + (I 18    + 3)    + 2,
{          1/2     1/3
{     (I 18    + 3)
            1/2     1/3
       (I 18    + 3)             1/2 /      1/2     1/3           3
   2 - ---------------- - 1/2 I 3    | (I 18    + 3)    - ----------------
              2                      |                         1/2     1/3
/                                     \                   (I 18    + 3)
   \           3                  1/2 /      1/2     1/3           3
   | - ------------------, 1/2 I 3    | (I 18    + 3)    - ----------------
   |          1/2     1/3             |                         1/2     1/3
   /   2 (I 18    + 3)                \                    (I 18    + 3)

            1/2     1/3                          }
   \   (I 18    + 3)              3              }
   | - ---------------- - ------------------ + 2 }
   |          2                  1/2     1/3     }
   /                      2 (I 18    + 3)        }
```

# STUDY UNIT 2

## 2.1 Introduction

We now start a (fairly) systematic study of the use of computer algebra to solve various mathematical problems. These notes are written as a guide to MuPAD. The syntax of Maple and REDUCE is, in many cases, identical to that of MuPAD; so, perhaps with the assistance of the online tutorials and help facilities, you should also be able to solve (simple) problems on these other systems.

Use of the MuPAD online help facility was described in study unit 1, and we need to emphasize the importance of this facility. These study notes are not going to repeat information available in the help facility, so you should go through the study material with your computer switched on and with MuPAD running. When various keywords are first introduced, they will appear in *italics*, and you should look for further details in the help facility.

## 2.2 Identifiers and assignments

As in most computer languages, the basic constructs are *identifiers* and statements, and we start by considering a particular type of statement, namely *assignment*. In MuPAD, identifiers are concatenations of the characters `a...z, A...Z, 0...9` and _, and must start with a letter (note that MuPAD distinguishes between lower and upper case letters). Statements are ended with

; or :

the difference being that the evaluation of the statement is echoed after ; but not after :. The syntax of an assignment statement is

identifier: = <expression>

The following are all valid assignment statements

```
a: = "My name":
b2: = (x^2)*3/(Hy^3);
A2b: = A*2*b;
```

In xmupad and mupadLight you do not have to end a statement with ; or : - the enter command causes the statement to be executed. However, since there are cases when the ; or : is needed, it is not good practice to omit the ; or :, and we will not do so in these notes.

In MuPAD, an identifier that has not been assigned a value, evaluates to itself. For example

```
x2;
x2
```

but

```
x2: = a + b:
x2;
a + b
```

If you wish to clear the value assigned to an identifier, so that it will again evaluate to itself, you do so by using the command *delete*. For example

```
x2: = a + b:
x2;
a + b
delete(x2):
x2;
x2
```

### 2.3 Input and output

In most cases in which computer algebra is being used to solve a real problem, you will need a record of the instructions given to the system, and the answers produced. You will probable find it convenient to use MuPAD interactively for problems requiring a few lines of code; but for larger problems it's often easier to compose off-line and then run the programme, as is done with general purpose programming languages.

MuPAD is very flexible in its interaction with the file system, but here we describe only the most commonly used features.

The default directory for file reading and writing is the directory from which the command to start MuPAD was given. In Linux, this can be whatever you choose, but if you are using MuPADLight this directory is fixed during the setup process, and will probably be c:\Program files\sciface\

Also in Linux it is possible to change the current directory, while running MuPAD, by the command

```
!cd
```

However, again this cannot be done in MuPADLight. Instead, you have two options (which also work in Linux).

Specify the full path of a file, e.g. c:\........
Specify a directory by setting the variables *WRITEPATH* and *READPATH* (note that these two must be set independently and could have different values).
The syntax for setting the variables is, for example,

```
WRITEPATH: = "c:\\My Files\\MuPAD\\Assignment1\\";
```

which would make the default directory for file-writing

c:\My Files\MuPAD\Assignment1

Why are there \\, and not \, in the above statement? Because, in a *string* (which is anything enclosed in "..") the character \ has special functions and \\ means \.

The directory specified by WRITEPATH must already exist - WRITEPATH is not an operating system command and cannot create a directory. Unfortunately, MuPAD does not report an error if you make a typing mistake and attempt to set WRITEPATH to a non-existent directory. Instead, when you write to file, MuPAD acts as though WRITEPATH had not been set.

When you write to a file (as described above), MuPAD concatenates WRITEPATH with the file name.

You can save a whole session (input and output) by means of the *protocol* command. The syntax is, for example,

```
protocol("ex.1");
```

With WRITEPATH set as above this would write all input and output (from the time that protocol is called) to

c:\My Files\MuPAD\Assignment1\ex.1

If a protocol is already running, the command

```
protocol( );
```

ends the protocol and closes the file. In xmupad and MuPADLight, you can also save a session by clicking on the "File" menu, and then on "SaveAs...".

The protocol output is very useful for you, or for another person, to read and understand what has been done. However, it cannot be read by MuPAD. In any programming environment the unexpected can happen, the system crashes and all current work is lost. You can save, in a form readable by MuPAD, the current status of your session by using the *write* command. For example,

```
write("temp.out");
```

will create a binary file c:\My Files\MuPAD\Assignment1\temp.out (assuming WRITEPATH set as above). The binary file is readable by MuPAD but not by a text editor. If you would also like to be able to read the file, you should enter

```
write(Text,"temp.out");
```

The resulting output will not use *prettyprint*, so it will not be as easy for you to read as protocol output, but nevertheless both you and MuPAD can read it.

Input to MuPAD is normally via the *read* command. The input file must consist of valid MuPAD statements and will be read and executed by the system. The file could have been obtained from a "write" statement, or it could have been composed offline by you (or it could be a combination of the previous two cases). For example

```
read("ex.2");
```

with READPATH set to c:\MuPAD\ would read and execute all statements in the file c:\MuPAD\ex.2

MuPAD can do much more than described above. It can input or output information about specific identifiers, input statements from a file one at a time (useful for debugging), and there are other options. You should look at the online help for entries about *fclose, fopen, fread, fprint, ftextinput.*

## 2.4 Types

Most modern computer languages require an explicit declaration of a variable (called an identifier in MuPAD) and of its type. Such a declaration is not needed in MuPAD. Allocating a type to an identifier is done implicitly and the type can change during a session; even so it is important to understand the different types used in MuPAD.

Until an identifier is a set by means of an assignment, it is regarded as being of type identifier or, to use the proper MuPAD notation, of type *DOM_IDENT*. If you type

```
domtype(a)
```

where a is an unassigned identifier, the system will return

```
DOM_IDENT
```

Now, type, for example

```
a: = 25*2 - 5;
45
domtype(a)
DOM_INT
```

An important difference between computer algebra systems and other programming languages is that there is no storage limit for an identifier, thus there is no largest integer that can be represented. For example, MuPAD can easily compute, exactly, 100! (! meaning factorial):

```
100!;

93326215443944152681699238856266700490715968264381621468592963895217
599993\
22991560894146397615651828625369792082722375825118521091686400000000
000000\
0000000000
```

You can change a from type integer to rational by typing

```
a: = a/6; domtype(a);

15/2

DOM_RAT
```

You can also change a to type float by doing some operation with a real number, e.g.

```
a: = a*1.0;domtype(a);
```

```
7.5

DOM_FLOAT
```

Again, as with integer arithmetic, there is no limit to the number of significant figures that can be used to represent a real number (in most other programming languages the limit is about 15 figures, due to the storage limit of 64 bits). The command *float* produces a real number approximation of an identifier or expression (provided this is meaningful), and the number of significant figures is determined by the variable *DIGITS* (the default value of which is 10). For example,

```
DIGITS:=100:float(PI);

3.141592653589793238462643383279502884197169399375105820974944592307
816406\
286208998628034825342117068
```

*I* is a system defined constant in MuPAD, that represents the square root of -1. Any combination of real, rational or integer with I leads to an identifier of type complex:

```
a: = 2 + 3*I: domtype(a);

DOM_COMPLEX

ac: = 2 - 3*I: b: = a*ac; domtype(b)
13
DOM_INT
```

It is clear that MuPAD has applied the result $I^2 = -1$.

A string is anything enclosed in quotation marks, and you have already used strings for writing to, and reading from, a file

```
a: = "I am a string": domtype(a);
DOM_STRING
```

Identifiers in MuPAD can be Boolean, and take the values TRUE or FALSE; such identifiers are of type *DOM_BOOL.*

A *list* is an ordered sequence in square brackets of any number of expressions of (in general) different types. For example

```
x: = [2.0, I,7, x^2]: domtype(x);
DOM_LIST
```

The elements of a list are indexed, starting from 1; for example

```
x[1]; x[3];
2.0
7
```

Once created, elements can be removed from a list, and new elements inserted (in any position). Details are given in the online help.

A *set* is a collection of different elements enclosed in curly brackets. Unlike a list, there is no concept of the order of the elements, and repeats are not possible. For example

```
a: = {1,2, x^2, x^3, 2, x^2}; domtype(a);
{1,2, x^2, x^3}
DOM_SET
```

The usual binary operators *union, intersect* and *minus* are defined for sets; for example

```
a: = {1, 2, x}: b: = {1,2, x^2}:
a union b;
{1, 2, x, x^2}
```

The function *contains* checks whether an element is a member of a set. Details are given in the online help.

The type used most often in MuPAD is expression, *DOM_EXPR*, for example

```
y: = 1 + x^2: domtype(y);
DOM_EXPR
```

The ability to manipulate expressions is the distinguishing feature of a computer algebra system, and most of this module will be concerned with this type.

Finally, we will also be using the data structures of array (DOM_ARRAY), table (DOM_TABLE) procedure (DOM_PROC) and polynomial (DOM_POLY), but they will not be introduced until later study units, when needed. There are other domain types in MuPAD, but they are not normally needed by users of the system - their use is mainly in system development.

## 2.5 Exercises

1. Use the protocol command and WRITEPATH variable to output a portion of a MuPAD session to a file **not** in the current directory.

2. Save the current status of a MuPAD session to a file **not** in the currect directory. Quit MuPAD, then restart and recover the previous session.

3. Create offline and not in the current directory, a file containing MuPAD commands. Start a MuPAD session and read and execute these commands.
4. With the same file as Qn. 3, read and execute only the first line of the file.

5. Create a file that can be re-read by MuPAD, that contains the value of only 1 identifier.

6. Use MuPAD to find the prime factors of an integer (hint: see *ifactor*).

7. Use MuPAD to find the greatest common divisor of a list of integers (hint: see *igcd*).

8. Use MuPAD to find the real and imaginary parts of various complex numbers (hint: see *Im, Re*).

9. Extract the numerator and denominator of a rational number (hint: see *denom, numer*).

10. Find out from the online help how to concatenate two strings, i.e. if

```
a: = "Hello": b: = "world"
```

create a new string c with value "Hello world".

11. Create a set, then create new sets with

(a) one member removed,
(b) one new member.

# STUDY UNIT 3

## 3.1 Introduction

This study unit continues the presentation of the basic features of compuer algebra that will be needed to solve real problems. The first tool described is that of simplification and substitution, and then we re-visit looping and branching (with which you should be familiar from other programming languages). As before these notes are not intended to be complete, but are rather a guide as to which topics you should be consulting in MuPAD's online help facility.

## 3.2 Simplification and substitution

The ability to simplify a given algebraic expression in a particular way is a key aspect of obtaining useful results from computer algebra. However, you will quickly learn that this can be more difficult than it first appears. Here we give only an introduction to the art of simplification of expressions in computer algebra, and other aspects will be considered in later Study Units.

First, it is important to understand how MuPAD stores identifiers. Until it is assigned a value, an identifier A is stored as, and evaluates to, itself. Once an assignement is made, the identifier A is given a value which the system computes from the right hand side of the assignment statement, and taking into account what is then known about the values of any identifiers appearing on the right hand side; thereafter, the identifier A keeps this value until it is changed by another assignment statement. If an identifier on the right hand side, say B, has its value changed by assignment, then an evaluation of A (by typing A;) will give a different result, but the value of A does not change. If this seems a bit confusing, work through the following MuPAD example, then open your own MuPAD session and enter commands different to those in the example and try to predict the output before looking at it.

```
a:=x+y;
                                x + y
x:=2: a;
                                y + 2
x:=3: a;
                                y + 3
y:=4: a;
                                  7
a:=x^2+y^2;
                                 25
x:=0: y:=0: a;
                                 25
```

MuPAD has an internal simplifier that is not controlled by the user. It automatically simplifies many objects of numerical type (integer, rational, real, complex, boolean), for example

```
sin(3*PI), exp(0), (2+I)*(2-I);
                              0, 1, 5
```

The only simplification that MuPAD makes to **expressions** is to change `0*a` to `0` (except in the case that `a` has the value `infinity`). The reason is that what is useful to do, or even correct, often depends upon information that the system does not have. Thus in MuPAD simplification of an expression has to be done explicitly by the user, and we now describe the tools most commonly used for this purpose.

The command *simplify* is a general purpose symbolic simplifier, and it will often produce the desired result. It can be called in a general form (e.g., `f:=simplify(f);`), or with the system instructed to restrict attention to a special class of possible simplifications. The possible additional arguments are: exp, ln, sin, cos, sqrt, logic, and an example of such a call is `f:=simplify(f,sqrt)`. The command *expand* is usually used to transform a product of sums into a sum of products. The command *factor* will often succeed in factoring expressions and polynomials (both the numerator and denominator). The command *normal* expands the numerator and denominator of an expression. The use of all these commands is illustrated in the following examples.

```
>> f := sin(x)^2 + cos(x)^2 + (exp(x) - 1)/(exp(x/2) + 1):
simplify(f);
                                    / x \
                                 exp| - |
                                    \ 2 /

>> simplify(f, exp);
                           2          2       / x \
                     cos(x)   + sin(x)   + exp| - | - 1
                                              \ 2 /

#COMMENT: In this case only simplifications involving exp were made
so the trigonometric part was not simplified#
>> simplify((a and b) or (a and (not b)), logic);

                                      a
>> factor(sqrt(2)*I*x^4 - sqrt(2)*I*x^2 - sqrt(2)*I*2);
                  1/2                1/2                1/2
              (I 2   ) (x + I) (x + 2   ) (x - I) (x - 2   )

>>
factor(7*(exp(x)^2 - 1)*sin(1)^3);
                                                      3
                    7 (exp(x) + 1) (exp(x) - 1) sin(1)
>> a0:=simplify((sqrt(2)*I*x^4 - sqrt(2)*I*x^2 - sqrt(2)*I*2)/(x+I));

                        4  1/2      2  1/2        1/2
                     I x  2    - I x  2    - 2 I 2
                     ---------------------------------
                                  x + I

#COMMENT: simplify does not recognise the common factor x+I; in order
to cancel it from a0 we now apply the factor command#
```

```
>> a1:=factor(a0);

                       1/2                 1/2          1/2
                 (I 2    ) (x - I) (x - 2    ) (x + 2    )

>> a2:=
expand((x + 1)^2*y/(y + z)^2);
                                                   2
                           y         2 x y        x  y
                       -------- + -------- + --------
                              2          2          2
                       (y + z)    (y + z)    (y + z)

>> a2:=simplify(a2);
                                                              2
                    y                  2 x y                 x  y
            --------------- + --------------- + ---------------
                       2    2                2    2                2    2
            2 y z + y  + z    2 y z + y  + z    2 y z + y  + z

>> a2:=factor(a2);
                                            2
                                     (x + 1)  y
                                     ----------
                                              2
                                      (y + z)
>> normal(a2);
                                                 2
                                  y + 2 x y + x  y
                                  ----------------
                                          2    2
                                  2 y z + y  + z
```

We can substitute a value for an identifier x in an expression f by means of the *subs* command. You can make several substitutions at once, i.e. within the same subs command, and it is also possible to substitute for a sub-expression rather than for an individual identifier. If you want to substitute for a system function such as sin, you must stop the system evaluating it by means of the *hold* command. On the other hand, we can make a substitution for an expression such as sin(x) without using hold. Note that sometimes the system only makes the substitution and does not evaluate the resulting expression, so that an explicit evaluation may be needed. The syntax is illustrated in the following examples

```
>> subs(a + b*a, a = 4);
                                    4 b + 4
>> subs([a * (b + c), sin(b +c)], b + c = a);
                                   2
                                 [a , sin(a)]
>>  subs(sin(x),  sin  =  cos),  subs(sin(x),  hold(sin)  =  cos),
subs(sin(x),sin(x)=x);
                              sin(x), cos(x), x
```

```
>> f := sin(z*(x + y)):
>> subs(f,x=1,y=-1);
                                  sin(0)

>> g:=subs(f,x=1,y=-1);
                                  sin(0)

>> g;
                                     0

>> subs(f, x + y = z);
                                       2
                                  sin(z )

>> subs(x + y + z, x + y = z);
                                 x + y + z
```

As seen above, using subs to replace sub-expressions does not always work. This depends on the way that the system stores the expression, although we will not explain this issue further. An alternative command for substituting sub-expressions is *subsex*, but particularly when substituting into long expressions, subsex can take a long time to execute (because it has many possible combinations to check). Thus subsex should be used only when subs does not work.

```
>> subsex(x + y + z, x + y = z);
                                    2 z
```

### 3.3 Looping and branching

The MuPAD language has the usual looping and branching constructs, with which you should be familiar from a first course in computer programming, although the command sysntax may be a little different. Because you should be familiar with these constructs, we just state them, and you should consult the online help facility for the detailed syntax.

! *for ... end_for*
! *while ... end_while*
! *repeat ... end_repeat*
! *if ... then .. end_if*
! *if ... then ... else ... end_if*
! *if ... then ... elif ... then ... ... else ... end_if*
! *case ... end_case*

Here are some examples using the most common constructs (for loop and if statement)

```
>> i := 20:
for i from 1 to 3 do
  a := i;
  b := i^2;
  print(a, b)
```

```
end_for:
                                 1, 1
                                 2, 4
                                 3, 9
>> i;
                                   4

>> for i from 2.2 downto 1 step 0.5 do
  print(i)
end_for:
                                  2.2
                                  1.7
                                  1.2

>> for i from 1 to 3 do
  print(i);
  if i = 2 then break end_if
end_for:
                                   1
                                   2

>> if FALSE then NO elif TRUE then YES_YES else YES_NO end_if;
                                YES_YES
```

There is one looping command that you have probably not seen before: *sum*, which is illustrated in the examples below. It computes the sum of an arithmetic expression with respect to an index in some given range.

```
>> sum(1/3^n,n=1..infinity);
                                  1/2

>> sum(a^2,a=1..5);
                                  55
>>  sum(b^2,b=1..k);
                                   2    3
                              k   k    k
                              - + -- + --
                              6   2    3
>> sum(b,b=1..k);
                                     2
                                k   k
                                - + --
                                2   2
```

## 3.4 Exercises

1. Write a MuPAD program to find all prime numbers between 10 and 100. The program should use the for loop and if constructs, and the only arithmetical operator allowed is multiplication (*). You may not use any of MuPAD's prime number testing functions. You may assume that prime numbers less than 10 are 2, 3, 5 and 7.

2. Use MuPAD to simplify $(5\ x^3 - 30\ a\ b\ x + 2\ x^4 + 20\ a\ b^2 + 10\ a\ x^2 + 4\ a\ x^3 - 15\ b\ x^2 + 10\ b^2\ x - 6b\ x^3 - 12\ a\ b\ x^2 + 8\ a\ b^2\ x + 4\ b^2\ x^2)/(x^2 - 3\ b + 2\ b^2)$.

3. Let

$$f = (1+x)^3 \sin(x) \cos(x) - \exp(3\ x) \cos^2(2\ x).$$

Use MuPAD to find a representation of f in the form $f = a + b\ x + c\ x^2$ (a, b, c numerical constants) valid for $x<<1$. You may assume

$$\exp(x) = 1 + x + x^2/2,\ \ \cos(x) = 1 - x^2/2,\ \ \sin(x) = x.$$

4. Use MuPAD to show that

$$(4^{(x-1)} + 3\ \ 2^{(2x-4)})/(2^{(2x+1)} + 5\ \ 2^{(2x-3)}) = 1/6.$$

5. Use MuPAD to find the sum of $(1/3)^n$ for n from 2 to infinity.

6. Use MuPAD to find the sum of $k^n$ for n from m to infinity and with $|k| < 1$. Hint - write k in a way that $1/k > 1$ or $1/k < -1$, and then use subs.

7. Use MuPAD to find constants a, b, c, d, e such that

$$\cos^2(x) + \sin(x) \cos(x) = a + b \sin(x) + c \cos(x) + d \sin(2\ x) + e \cos(2\ x).$$

8. Use MuPAD to prove

$$2 \cos(5\ x)/(\sin(2\ x) \cos^2(x)\ ) = -10 \sin(x) + \cos^2(x)/\sin(x) + 5 \sin^3(x)/\cos^2(x).$$

9. Use MuPAD to prove

$$(\sin^2(x) - \exp(2\ x)\ )/(\sin^2(x) + 2 \sin(x) \exp(x) + \exp(2\ x)\ ) = (\sin(x) - \exp(x)\ )/(\sin(x) + \exp(x)\ ).$$

10. Use MuPAD to prove

$$(\sin(2\ x) - 5 \sin(x) \cos(x)\ )/(\sin(x)\ (1+\tan^2(x)\ )\ ) = -9 \cos(x)/4 - 3 \cos(3\ x)/4.$$

# STUDY UNIT 4

## 4.1 Introduction

This study unit is concerned with the use of MuPAD in calculus, i.e. with differentiation and integration. Calculus uses functions of one or more variables, and so first we discuss the various ways in MuPAD for dealing with functions. As before these notes are not intended to be complete, but are rather a guide as to which topics you should be consulting in MuPAD's online help facility.

## 4.2 Functions

A function in MuPAD is a special case of a procedure (see *proc* in the online help), about which more will be said in later study units. The syntax for the definition, and simple useage, of a function is shown in the following example. Note that a function can be multivariate in both input and output parameters.

```
>> f1:=x->x^2*sin(a*x);
                             x -> x^2*sin(a*x)
>> f1(z^2);
                                4        2
                               z  sin(a z )

>> f2:=(x,y)->sqrt(x^2+y^2); #Example of 2-parameter input#
                         (x, y) -> sqrt(x^2 + y^2)

>> f2(3,4);
                                     5

>> f3:=x->[x,x^2,x^3]; #Example of 3-parameter output, as a list#
                            x -> [x, x^2, x^3]

>> f3(2*y);
                                      2     3
                             [2 y, 4 y , 8 y ]

>> f4:=y->{y,y^2,y^3}; #Example of output as a set#
                            y -> {y, y^2, y^3}

>> f4(0);
                                    {0}

>> f4(z);
                                     2   3
                                {z, z , z }

>> f5:=(y,z)->[y-z,z,-y];
#Example of 2-parameter input, 3-parameter list output#
                         (y, z) -> [y - z, z, -y]
```

```
>> f5(2,w);
                              [2 - w, w, -2]
```

Functions can be composed by means of the @ command; for example, if we define `h=f@g` then `h(y)` is equivalent to f(g(y)). You can also define repeated composition of a function with itself by means of the command `@@n` where the (integer) n is the number of repeated compositions. For example

```
>> f:=x->1+x^2; g:=x->1/x;
                               x -> 1 + x^2
                                 x ->  1/x

>> h:=f@g;
                        (x -> 1 + x^2)@(x ->  1/x)

>> h(y);
                                  1
                                  -- + 1
                                   2
                                  y

>> f4:=f@@4;
                              (x -> (;
                                1 + x^2;
                                ))@(x -> (;
                                1 + x^2;
                                ))@(x -> (;
                                1 + x^2;
                                ))@(x -> (;
                                1 + x^2;
                                ))

>> f4(z);
                            2     2     2     2
                        (((z  + 1)  + 1)  + 1)  + 1

>> normal(%);
    2        4        6        8       10       12      14    16
80 z  + 144 z  + 168 z  + 138 z  + 80 z   + 32 z   + 8 z   + z   + 26
```

In MuPAD there is a clear distinction between a function such as `f4`, which is of type `DOM_PROC`; and the result of applying the procedure `f4(z)`, which is of type `DOM_EXPR` (assuming that `z` is an identifier).

The above constructs are quite versatile for describing functions that are known explicitly. However, there are times when one needs an expression that contains an unknown function, or perhaps a function that satisfies certain properties but otherwise is not known. You will see later how this can be done with the -> construct, but here we introduce a simple alternative for representing unknown functions.

The notation

```
f(x), g(y, z)
```

means that `f(x)` is an expression with an unknown dependence on `x`; and that `g(y,z)` is another expression with an unknown dependence on `y` and `z`. Note that if, subsequently, we determine `f(x)`, then in MuPAD that does **not** determine, say, `f(y)`. For example,

```
>> f(x):=x*sin(x);
                                 x sin(x)

>> f(2*x);
                                  f(2 x)

>> f(y);
                                   f(y)

>> f(x);
                                 x sin(x)
```

### 4.3 Differentiation

The system function *diff* finds the derivative of an expression with respect to a specified variable. It can also be used to find higher order derivatives, including mixed partial derivatives. The function diff can operate on expressions that are known or unknown (like f(x)). The syntax `diff(f(x),x$n)` means the nth derivative of f(x) with respect to x.Its syntax and useage is illustrated in the following examples

```
>> f:=(x,y)->x^2*y^3;
                             (x, y) -> x^2*y^3

>> g:=x^2+y^2;
                                   2    2
                                  x  + y

>> #COMMENT Some first derivatives#
>> diff(f(x,z),z), diff(g,x), diff(h(x),x), diff(h(x),y);
                         2  2
                      3 x  z , 2 x, diff(h(x), x), 0

>> #COMMENT Some higher order derivatives#
>> diff(f(w,z),w,z), diff(f(w,z),z$3), diff(g,x,x);
                                   2     2
                              6 w z , 6 w , 2
```

The operator diff does not assume that, in general, partial derivatives commute; except when this is a consequence of the form of the expression being differentiated. For example

```
>> diff(f(x)*g(y),x,y)-diff(f(x)*g(y),y,x);
                                     0
```

```
>> diff(h(x,y),x,y)-diff(h(x,y),y,x);
                 diff(h(x, y), x, y) - diff(h(x, y), y, x)
```

If you want to find the derivative of an expression at a particular value, you should first find the derivative, then substitute the desired value - if you reverse the order, you will be finding the derivative of a constant, which is zero. For example

```
>> f:=x^3;
                                     3
                                    x

>> diff(subs(f,x=1),x);
                                     0

>> subs(diff(f,x),x=1);
                                     3
```

MuPAD has another differential operator, *D*. The difference between diff and D is that diff operates on expressions while D operates on functions. Thus, if f is a function, then D(f) is a new function, and D(f)(z) is the expression obtained by evaluating the function D(f) at z. These points are illustrated in the following examples (Note that *id* is the identity function, i.e. `id:=x->x;`).

```
>> D(sin), D(exp), D(ln), D(sin*cos), D(sin@ln), D(f+g);
                       1                2  cos@ln
             cos, exp, --, cos cos - sin , ------, D(f) + D(g)
                       id                    id

>> f := x -> (sin(ln(x))): D(f);
                                  cos@ln
                                  ------
                                    id

>> D(f)(1), D(f)(y^2), D(g)(0);
                                       2
                               cos(ln(y ))
                       cos(0), -----------, D(g)(0)
                                    2
                                   y
```

If a function f has more than one argument, then D([i],f) denotes the the partial derivative with respecct to the *i*th argument; and D([i,j,...],f) is the natural generalization to higher derivatives. For example,

```
>> f:=(x,y)->exp(a*x+b*y);
                         (x, y) -> exp(a*x + b*y)

>> D([1],f);
                        (x, y) -> a*exp(a*x + b*y)
```

```
>> D([1,1],f);
                       (x, y) -> a^2*exp(a*x + b*y)

>> D([1,2],f);
                       (x, y) -> a*b*exp(a*x + b*y)
```

Another notation used in MuPAD to denote the operator D is '. However, it will not be used in this study material, because the D notation is clearer.

### 4.4 Integration

You can use MuPAD to calculate the integral of an expression, either in the indefinite sense of an anti-derivative, or as a definite integral between limits. A limit of infinity (+ or -) is accpetable. The useage and syntax is straightforward, and is illustrated in the following examples.

```
>> f1:=1/(x*ln(x));
                                     1
                                  -------
                                  x ln(x)

>> int(f1,x);
                                 ln(ln(x))

>> int(f1,x=E..E^2);
                                   ln(2)
```

In the following example, the indefinite integral of a function is obtained, but the system does not calculate the definite integral, because it is unclear whether the definite integral exists. That depends upon the value of a and, if `2 <= a <= infinity`, then the definite integral is not defined. However, you can tell the system that a definite integral is continuous in the domain of integration by means of the Continuous parameter. **N.B.** You should apply the Continuous parameter only when the integral is continuous, because otherwise, although you will get an answer, **it will be wrong**.

```
>> int(x/(x-a)^4,x);
                                   a   x
                                   - - -
                                   6   2
                         -------------------------
                          3    3        2      2
                         x  - a  - 3 a x  + 3 a  x

>> factor(%);
                            (-1/6) (- a + 3 x)
                            ------------------
                                             3
                                (- a + x)

>> int(x/(x-a)^4,x=2..infinity);
```

```
Warning: Found potential discontinuities of the antiderivative.
Try option 'Continuous' or use properties (?assume).
[intlib::antiderivati\
ve]
                        /    x                      \
                     int| --------, x = 2..infinity |
                        |        4                  |
                        \ (x - a)                   /

>> int(x/(x-a)^4,x=2..infinity,Continuous);
                                   a
                                   - - 1
                                   6
                          - --------------------
                                 2           3
                            6 a  - 12 a - a  + 8
```

Another case where a definite integral appears to be undefined, but a meaningful value can be given to it, is if the integral has a Cauchy Principal value. However, that involves contour integration in the complex plane, which is a subject that you have not yet studied. Thus, the *PrincipalValue* option will not be discussed here.

Independently of whether MuPAD is able to find an analytic form for the indefinite integral, MuPAD can obtain a numerical approximation to a definite integral - provided the integral does not involve any unknowns. For example,

```
>> i1:=int(sin(x)^2*cos(x)^2,x=PI..3*PI);

                                    PI
                                    --
                                    4
>> float(i1);
                               0.7853981634

>> i2:=int(sin(x^2*exp(x^(1/3))),x=0..1);

                              2      1/3
                     int(sin(x  exp(x   )), x = 0..1)

>> float(i2);
                               0.4888639092

>> i3:=int(a*sin(x^2*exp(x^(1/3))),x=0..1);

                               2      1/3
                    int(a sin(x  exp(x   )), x = 0..1)

>> float(i3);
                               2      1/3
                    int(a sin(x  exp(x   )), x = 0..1)
```

The last integral, `i3,` could not be evaluated because it contains the unknown `a`.

MuPAD is able to find analytic forms for many indefinite integrals. However, you should be aware that MuPAD is definitely not perfect, and if the system says that it cannot evaluate an integral, this may or may not mean that a representation in terms of simple functions does not exist. For example,

```
>> int(sin(a*x)*sqrt(1+sin(a*x)),x);
                                                      1/2
                    int(sin(a x) (sin(a x) + 1)    , x)
```

However, when the same problem is given to Maple, we find

```
> int(sin(a*x)*sqrt(1+sin(a*x)),x);

             sqrt(1 + sin(a x)) (sin(a x) - 1) (2 + sin(a x))
         2/3 ------------------------------------------------
                                 cos(a x) a
```

Symbolic integration pakages are difficult to write because they have to allow for a large number of different cases; the same applies to differential equation solvers (which will be discussed in a later study unit). In these problems, if one computer algebra package does not work, then it is worth trying another package.

## 4.5 Exercises

1. Define (as MuPAD functions) $f(x) = x^2$ and $g(x) = \sqrt{x}$ . Compute f(g(g(3))) and f applied to y 10 times (i.e. f(f( ... f(y)...) by first defining appropriate MuPAD functions.

2. Given that a function f maps x to x sin(x), find the function that represents the derivative of f, and evaluate it at $\pi$.

3. Consider the function f: = x -> sin(x)/x. Compute the value of f(x) at x = 1.12, and find the derivative of f(x).

4. Use l'Hospital's rule to find

$$\lim_{x\to 0} \frac{x^3 \sin(x)}{(1 - \cos(x))^2}$$

5. Determine the first and second order partial derivatives of f(x,y) = sin(xy). Let x = x(t) = sin(t) and y = y(t) = cos(t). Compute the derivative of f with respect to t.

6. Use MuPAD to find the following indefinite integrals (with respect to x), and in each case check your answer by differentiating it:

(a) $x/(2ax - x^2)^{(3/2)}$,
(b) $(x^2-a^2)^{(1/2)}$,
(c) $1/(x(1+x^2)^{(1/2)})$.

7. Find an example, different to that given at the end of section 4.4, of an indefinite integral that is not solved by MuPAD but which nevertheless has a solution in terms of elementary functions.

8. Evaluate the following definite integrals. The integrals involve a parameter a, and you should state what conditions on a are needed for the integrals to exist.

(a) $x/((x-2)^2\ (x-3)^2)$, x = -infinity..a
(b) x ln(x), x = a..2
(c) $x/(x^4 - a^4)$, x = 0..infinity.

# STUDY UNIT 5

## 5.1 Introduction

This study unit is concerned with the use of MuPAD in linear algebra. As before these notes are not intended to be complete, but are rather a guide as to which topics you should be consulting in MuPAD's online help facility.

## 5.2 Data types

The MuPAD data types used in linear algebra are *table*, *array* and *matrix*. Of these, the table structure is by far the most flexible way for storing, and then later accessing, indexed data. A table can be constructed either explicitly or implicitly, as in the following examples.

```
>> T := table(a = 13, c = 42);
                                   table(
                                     c = 42,
                                     a = 13
                                   )

>> T[a], T[b], T[c];
                                 13, T[b], 42

>> delete T: T[4] := 7: T;
                                    table(
                                      4 = 7
                                    )

>> T1[(1,2)]:=a^2;
                                       2
                                      a

>> T1;
                                 table(
                                                2
                                   (1, 2) = a
                                 )
```

Once created, the elements of a table can be assigned new values in the usual way. The indices of a table can be anything - integers, algebraic expressions, etc. Note also that a table is a dynamic data structure, in that the new elements can be added to a table at any time.

The size of an array or a matrix is fixed when it is declared. An array is indexed by a list of integers, which can be negative, zero or positive; while a matrix is indexed by a list of positive integers. The command matrix can be used to transform an array into a matrix. Some declarations, and simple examples, of arrays and matrices follow.

```
>> A := array(-1..1, [2, sin(x), FAIL]);
                            +-               -+
                            | 2, sin(x), FAIL |
                            +-               -+

>> A[-1], A[2];
>> A[-1];
                                      2

>> A[2];
Error: Illegal argument [array]

>>  A := array(0..2, 1..2, (1, 2) = 42, (2, 1) = 1 + I);
                          +-                   -+
                          |  ?[0, 1], ?[0, 2]   |
                          |                     |
                          |  ?[1, 1],    42     |
                          |                     |
                          |   1 + I,  ?[2, 2]   |
                          +-                   -+

>> A:=array(0..1,-1..1, [[1, 2, 3], [4, 5, 6]]);
                               +-           -+
                               |  1, 2, 3   |
                               |            |
                               |  4, 5, 6   |
                               +-           -+

>> array(2..3, 1..3, 1..1,
      [
        [ [1], [2], [3] ],
        [ [4], [5], [6] ]
      ]);
                          array(2..3, 1..3, 1..1,
                            (2, 1, 1) = 1,
                            (2, 2, 1) = 2,
                            (2, 3, 1) = 3,
                            (3, 1, 1) = 4,
                            (3, 2, 1) = 5,
                            (3, 3, 1) = 6
                          )

>> A := matrix([[1, 5], [2, 3]]);
                                +-       -+
                                |  1, 5   |
                                |         |
                                |  2, 3   |
                                +-       -+
>> matrix(4, 4, [[1, 2], [2]]);
                                   +-             -+
                                   |  1, 2, 0, 0  |
                                   |              |
                                   |  2, 0, 0, 0  |
                                   |              |
                                   |  0, 0, 0, 0  |
                                   |              |
                                   |  0, 0, 0, 0  |
                                   +-             -+

>> a := array(1..3, 2..4,
  [[1, 1/3, 0], [-2, 3/5, 1/2], [-3/2, 0, -1]]
);
                                 +-                 -+
                                 |     1,  1/3,   0  |
                                 |                   |
                                 |    -2,  3/5, 1/2  |
                                 |                   |
                                 |  -3/2,   0,   -1  |
                                 +-                 -+

>> A := matrix(a);
                                 +-                 -+
                                 |     1,  1/3,   0  |
                                 |                   |
                                 |    -2,  3/5, 1/2  |
                                 |                   |
                                 |  -3/2,   0,   -1  |
                                 +-                 -+

>> b := array(1..4): b[1] := 2: b[4] := 0: b;

                                 +-                 -+
                                 | 2, ?[2], ?[3], 0 |
                                 +-                 -+

>> matrix(b);
Error:  unable  to  define  matrix  over  Dom::ExpressionField()
[(Dom::Matrix(D\
om::ExpressionField()))::new]
>> c:=array(0..1,0=a1,1=a2);
                                      +-      -+
                                      | a1, a2 |
                                      +-      -+
>> C:=matrix(c);
                                   +-    -+
                                   |  a1  |
                                   |      |
                                   |  a2  |
                                   +-    -+
```

We note the following differences between arrays and matrices:

- Th e indices of an array can be 0 or negative, but those of a matrix are between 1 and an upper limit.
- An array can have any number of dimensions (i.e., any number of indices), but a matrix always has 2 indices - this is even the case for the matrix C above, which appears as a 1-dimensional object, but is regarded as a matrix of size 2 rows and 1 column.
- An array can have uninitialized entries, but a matrix cannot; in a matrix the default value of an entry is 0.

Sometimes one would like to apply the same function to every element of a table, array or matrix. This is achieved in MuPAD by means of the *map* command. In the next example, we have a table that could represent prices, and we want to amend it so as to include VAT at 14%. The old table is not destroyed, and we just create a second table. The indices (left hand side) are not affected by map, only the entries (right hand side) are changed.

```
>>    T_VATexcluded := table(1 = 65, 2 = 28, 3 = 42):
>>    T_VATincluded:=map(T_VATexcluded, _mult, 1.14);
                                table(
                                  3 = 47.88,
                                  2 = 31.92,
                                  1 = 74.1
                                )
```

In the next example, we change every element x of a matrix to x+sin(x), by introducing a function f

```
>> f:=x->x+sin(x);
                               x -> x + sin(x)
>> A:=matrix([[y,z],[0,PI]]);
                                 +-        -+
                                 |  y,   z  |
                                 |          |
                                 |  0,  PI  |
                                 +-        -+
>> B:=map(A,f);
                          +-                          -+
                          |  y + sin(y), z + sin(z)   |
                          |                           |
                          |       0,         PI       |
                          +-                          -+
```

## 5.3 Packages

MuPAD contains a number of packages, also called libraries, which are collections of routines relevant to a particular topic. In this study unit we will be primarily concerned with the linear algebra package, but we start with some remarks about packages in general. Information about available packages can be obtained by using the *info* command: `info()` produces a list of all available packages, and `info(package-name)` gives a list of all routines available in the package. For example,

```
>> info(transform);
Library 'transform': library for integral transformations

-- Interface:
transform::fourier, transform::invfourier, transform::invlaplace,
transform::laplace
```

The general usage of a package routine can be cumbersome, because it has to be called by using the syntax `package-name::routine-name`. This can be avoided by using the *export* command, usually in the form `export(L)` which makes all routines in the package L available without the prefix `L::`. The effect of exporting L is cancelled by the command `unexport(L)`.

## 5.4 The linear algebra package, linalg

Full information on the routines available in the linear algebra package can be found by typing `info(linalg)`, and you will see that a lot of routines are available. You can then use the online help facility to find out the implementation details of a particular routine, e.g. by typing `?linalg::det`. These routines are defined only for data structures of type matrix, i.e. they are not defined for an array or a table. In some cases, the matrix must be square - actually, MuPAD assigns a special domain type to a square matrix, namely SquareMatrix, and this is done automatically when the matrix is initialized. The routines available are extensive, and can be used to solve a wide variety of problems in linear algebra, including the inverse, determinant, LU decomposition, eigenvalues and eigenvectors of a matrix; as well as many other features that are not covered in first year linear algebra. Some examples follow

```
>> export(linalg);

>> A:=matrix([[1,2,0],[2,3,0],[0,4,a]]);

                               +-         -+
                               |  1, 2, 0  |
                               |           |
                               |  2, 3, 0  |
                               |           |
                               |  0, 4, a  |
                               +-         -+
>> d:=det(A);
                         -a

>> 1/A;
                        +-            -+
                        |   -3,  2, 0  |
                        |              |
                        |   2,  -1, 0  |
                        |              |
                        |    8   4  1  |
                        |  - -,  -, -  |
                        |    a   a  a  |
                        +-            -+

>> eigenvalues(A);
                           1/2           1/2
                         {a, 5    + 2, 2 - 5    }

>> factorLU(A);

             -- +-          -+ +-           -+             --
             |  |  1,  0, 0  | |  1,  2, 0   |              |
             |  |            | |             |              |
             |  |  2,  1, 0  |,|  0, -1, 0   |, [1, 2, 3]   |
             |  |            | |             |              |
             |  |  0, -4, 1  | |  0,  0, a   |              |
             -- +-          -+ +-           -+             --

>> isPosDef(A);
                                 FALSE
```

The linear algebra package also contains routines connected with calculus in higher dimensions. These routines can be used to compute the gradient of a scalar field, as well as the curl and divergence of a vector field, in a variety of orthogonal coordinate systems (such as Cartesian, Cylindrical, Spherical, etc.). The Laplacian operator is not directly available, but is easy to construct as the divergence of the gradient. For further details see *linalg::grad, linalg::curl, linalg::divergence* and *linalg::ogCoordTab*.

Some examples follow.

```
>> f:=r^2*3*sin(2*theta)*sin(phi);
                                  2
                             3 r   sin(phi) sin(2 theta)

>> f1:=grad(f,[r,theta,phi],Spherical);
                             +-                         -+
                             |  6 r sin(phi) sin(2 theta)  |
                             |                           |
                             |  6 r sin(phi) cos(2 theta)  |
                             |                           |
                             |  3 r cos(phi) sin(2 theta)  |
                             |  -------------------------  |
                             |         sin(theta)          |
                             +-                         -+

>> Lf:=divergence(f1,[r,theta,phi],Spherical): Lf:=simplify(Lf);
                                             0
>> f:=matrix([y*z,x*z,x^2]);

                                              +-      -+
                                        |  y z   |
                                        |        |
                                        |  x z   |
                                        |        |
                                        |    2   |
                                        |   x    |
                                        +-     -+

>> curl(f,[x,y,z],Cartesian);

                              +-          -+
                             |     -x      |
                             |             |
                             |  - 2 x + y  |
                             |             |
                             |     0       |
                             +-           -+
```

## 5.5 Matrices over a specified algebraic structure

In general, the elements of a matrix are of type expression, but it is possible to give a more precise specification of the type of matrix. The elements of a matrix can be specified to be, for example, rational, real, integer, complex, or integer mod n. This specification will then also apply to any matrices that are derived from the original matrix, so that if a matrix is of type integer and the inverse would involve rational numbers, then if MuPAD is asked to compute the inverse the result would be FAIL. These features are illustrated in the following examples. First we define two constructors, `MatI7` and `MatQ` which define matrices over the integers mod 7, and the rational numbers, respectively. We then use these constructors to initialize matrices, using the same syntax as was used above for the constructor matrix,

```
>> MatQ := Dom::Matrix(Dom::Rational);

                        Dom::Matrix(Dom::Rational)

>>MatI7:= Dom::Matrix(Dom::IntegerMod(7));
                   Dom::Matrix(Dom::IntegerMod(7))

>> AQ:=MatQ([[1,2],[3,4]]);
                                +-      -+
                                |  1, 2  |
                                |        |
                                |  3, 4  |
                                +-      -+

>> AI:=MatI7([[1,2],[3,4]]);

                          +-                  -+
                          |  1 mod 7, 2 mod 7  |
                          |                    |
                          |  3 mod 7, 4 mod 7  |
                          +-                  -+


>> AI^2;
                          +-                  -+
                          |  0 mod 7, 3 mod 7  |
                          |                    |
                          |  1 mod 7, 1 mod 7  |
                          +-                  -+


>> AQ^2;
                               +-        -+
                               |   7, 10  |
                               |          |
                               |  15, 22  |
                               +-        -+

>> 1/AQ;
                              +-           -+
                              |   -2,   1    |
                              |              |
                              |  3/2, -1/2  |
                              +-           -+

>> 1/AI;
                          +-                  -+
                          |  5 mod 7, 1 mod 7  |
                          |                    |
                          |  5 mod 7, 3 mod 7  |
                          +-                  -+
>> eigenvalues(AQ);
                                     {}

>> eigenvalues(AI);
                                     {}

>> charpoly(AQ,x);
                                 2
                                x  - 5 x - 2
```

If the matrix A had been defined over the reals, then it is clear that the charateristic polynomial is soluble and thus eigenvalues would have been found. However, these eigenvalues are irrational, and thus for the matrix A defined over the rationals, or over the integers mod 7, no eigenvalues exist.

### 5.6 Exercises

1. Generate a table **telephoneDirectory** with the following entries

   John 452-9867 Jane 876-9021 Mary 231-0956 Susan 952-3478 Thomas 394-5623

   Use MuPAD to look up Jane's number, and also to find out whose number is 952-3478.

2. Using the table in 1. above, generate a list of all indices, as well as a list of all values.

3. Generate a Hilbert matrix, of dimension 15 x 15, with entries $H_{ij} = 1/(i + j-1)$ for i,j >=1.

4. Use the Hilbert matrix generated in 3. above, and also generate the vector **b** = H **e** where the vector **e** = (1,1, ....,1). Use MuPAD to find the exact solution to the system H **x** = **b** (of course, this should yield **x** = **e**). Convert all entries of H and **b** to floating-point values and solve the system of equations again. Compare the result to the exact solution.

5. Find the values of a, b and c for which the following matrix is not invertible

```
1 a b
1 1 c
1 1 1
```

6. Consider the following matrices

```
       1 3 0          7 -1
A =   -1 2 7     B =  2  3
       0 8 1          0  1
```

   Let $B^T$ represent the transpose of B. Compute the inverse of A + B $B^T$, both over the rational numbers and over the integers modulo 7.

7. Generate the 4 x 4 matrix

$A_{ij}$ = 0 for i = j,
= 1 for i<>j

Compute its determinant, its characteristic polynomial, and its eigenvalues. For each eigenvalue find the corresponding eigenvector.

# STUDY UNIT 6

## 6.1 Introduction

This study unit is concerned with the use of procedures in MuPAD. The procedure construct is found in virtually all computer languages, although other names, such as method or subroutine, are also used. The concept and properties of a MuPAD procedure are quite similar to those of other computer languages. As before these notes are not intended to be complete, but are rather a guide as to which topics you should be consulting in MuPAD's online help facility.

## 6.2 Basic syntax

You have already come across one way of implementing a procedure, as a function declaration using the construct ->. The general form for defining a procedure is by means of the command *proc*, and the basic syntax is illustrated in the following example. Both `f` and `g` have the same properties, the difference being that `f` is constructed using `proc` and `g` is constructed using ->.

```
>> f:=proc(x)
      begin
      return (x+sin(x^2))
      end_proc;
                             proc f(x) ... end

>> g:=x->x+sin(x^2);
                             x -> x + sin(x^2)

>> f(a)-g(a);
                                    0
```

The keywords that appear in the definition of a procedure are *proc*, *begin*, *return* and *endproc*. The effect of return is twofold. It assigns a value to the procedure and it also causes the system to exit the procedure. The use of return is not necessary: if a return statement is not present, the procedure will be exited once it comes to end_proc, and the value of the procedure will be the value of the last statement executed. Even so, it is good practice to use a return statement, because it makes the program easier for someone else to read and understand. The value of a procedure can be any type of MuPAD data structure, and not just an expression as in the above example. Thus we can use a procedure to construct a list, set, matrix, table etc. For example

```
>> delta:=proc(n)
        begin
        m:=matrix(n,n);
        for i from 1 to n do m[i,i]:=1 end_for;
        return(m)
        end_proc;
                           proc delta(n) ... end
```

```
>> I3:=delta(3);
                                          +-         -+
                                          |  1, 0, 0  |
                                          |           |
                                          |  0, 1, 0  |
                                          |           |
                                          |  0, 0, 1  |
                                          +-         -+
```

The above procedure `delta` constructs a unit square matrix of arbitrary size. Of course, the variable passed to delta must be a positive integer, otherwise an execution error would result.

### 6.3 Local and global variables

The MuPAD language has the common concept of the scope of a variable. Generally, i.e. unless declared otherwise, variables are global, which means that they can be seen, and changed, anywhere in the programme. However, variables can be declared at the start of a procedure to be *local* and then, even if there is a global variable with the same name, whatever happens inside the procedure does not affect the global variable. Variables passed to the procedure, i.e. the list of variables in brackets after `proc`, are automatically local. It is generally regarded as good programming practice to avoid using global variables, so that the input to a procedure is only via those variables passed to the procedure, and the output is only from the return statement. This means that if the procedure is copied from one programme to another, or even called at different times in the same programme, then its effect will be the same. There are cases where it is cumbersome to avoid using global variables; in such cases, at least make a comment at the start of the procedure that it uses global variables. The above example `delta` should be written with the variables `m` and `i` declared local, and we now show this.

```
>> delta:=proc(n)
       local m, i;
       begin
       m:=matrix(n,n);
       for i from 1 to n do m[i,i]:=1 end_for;
       return(m)
       end_proc;
                                    proc delta(n) ... end

>> I2:=delta(2);
                                           +-      -+
                                           |  1, 0  |
                                           |        |
                                           |  0, 1  |
                                           +-      -+

>> m;
                                             m
```

With the procedure `delta` defined as above, the identifier `m` evaluates to itself after execution of the procedure. It is also possible to define a procedure as local within another procedure, as illustrated in the following example

```
>> f:=proc(x)
local g;
begin
g:=y->y^2;
return(g(x))
end_proc;
                             proc f(x) ... end
>> f(a);
                                     2
                                    a
>> g(z);
                                   g(z)
```

The function `g` is not recognized in the main MuPAD session, i.e. it is not recognized outside `f`. There is one case where global variables cannot be avoided, because MuPAD does not permit a local variable to be used as an arbitrary variable such as the variable of integration. We can still achieve the effect of a local variable in the following way

```
>> f := proc(n)
save x;
begin
  delete x;
  int(x^n*exp(-x), x = 0..1)
end_proc:
>> x := 3: f(1), f(2), f(3);
               1 - 2 exp(-1), 2 - 5 exp(-1), 6 - 16 exp(-1)
>> x;
                                     3
```

The commmand *save* ensures that the value of `x`, on exiting the procedure, is the same as it was on entry; and the command `delete x` ensures that `x` does not have a value.

### 6.4 Output from a procedure

Normally output to the screen is suppressed from within a procedure, and it does not matter whether a statement ends with : or ;. If you want output to the screen during the excution of a procedure, this has to be instructed explicitly by using the print command; for example

```
>> f:=proc(x)
    begin
    print(Unquoted,"started");
    return(x^2)
    end_proc;
                                     proc f(x) ... end

>> f(b^3);
                                           started
                                             6
                                            b
```

One reason for producing output during the execution of a procedure is for debugging purposes, and the above example prints "started" to show that the first statement of the procedure has been executed. Using the commands described in study unit 2, it is also possible to write to, or read from, a file while executing a procedure.

### 6.5 The input variables to a procedure

The input variables to a procedure may be restricted to be of a specified type, by means of the syntax proc(a:Type,...). This construct can be very useful in error avoidance. For example, the procedure `delta` for constructing a unit matrix requires that the input parameter n should be a positive integer, so we make a (final )amendment to the procedure and obtain

```
>> delta:=proc(n:Type::PosInt)
        local m, i;
        begin
        m:=matrix(n,n);
        for i from 1 to n do m[i,i]:=1 end_for;
        return(m)
        end_proc;
                                  proc delta(n) ... end

>> delta(2);
                                        +-       -+
                                        |  1, 0   |
                                        |         |
                                        |  0, 1   |
                                        +-       -+
>> delta(0);
Error: Wrong type of 1. argument (type 'Type::PosInt' expected,
       got argument '0');
during evaluation of 'delta'
```

The library *Type* contains many different types that can be used - you should check out what is available through the online help facility (just enter ?Type in a MuPAD session).

There are some procedures in which the number of arguments passed to the procedure is not fixed for each call of the procedure, and we now describe how this can be implemented. First, note that it is not necessary for any arguments to be passed, but that brackets () must appear in both the definition and call of the procedure

```
>> hello:=proc()
          begin
          print(Unquoted,"Hello world");
          return()
          end_proc;
                           proc hello() ... end

>> hello();
                                Hello world
```

The empty return statement means that the procedure does not have a value after execution. During the execution of a procedure, the following are defined (see *args* in the online help for more detail)

! `args(0)` - A non-negative integer that is the number of arguments that were passed to the procedure

! `args(1) to args(args(0))` - The arguments that were passed to the procedure

! `args()` - An expression sequence, i.e. the sequence args(1) to args(args(0)), of the parameters that were passed to the procedure

The following example illustrates the use of this feature

```
>> f := proc() begin
  print(Unquoted, "number of arguments" = args(0)):
  print(Unquoted, "sequence of all arguments" = args()):
  if args(0) > 0 then
    print(Unquoted, "first argument" = args(1)):
  end_if:
  if args(0) >= 3 then
    print(Unquoted, "second, third argument" = args(2..3)):
  end_if:
end_proc:

>> f():
                          number of arguments = 0
                    sequence of all arguments = null()
```

```
>> f(a, b, c, d):
                          number of arguments = 4
                 sequence of all arguments = (a, b, c, d)
                            first argument = a
                      second, third argument = (b, c)
```

### 6.6 Evaluation within a procedure

In a MuPAD session, as discussed in study unit 3, evaluation is recursive: a variable defined in terms of other variables which are defined in terms of yet other variables, will be evaluated so that it involves only variables that evaluate to themselves. This does not happen inside a procedure, and these features are illustrated in the following example

```
>>p1:=proc(x,y,z)
begin
x:=y+z: y:=2: x:
return(x)
end_proc;
                         proc p1(x, y, z) ... end

>> p1(a,b,c);                                    b + c

>> delete a,b,c;
>> a:=b+c: b:=2: a:
>> a;                                    c + 2
```

Now, we reverse the order inside the procedure, so that the value of y is known before x is evaluated:

```
>> p1:=proc(x,y,z)
begin
y:=2: x:=y+z: x:
return(x)
end_proc;                         proc p1(x, y, z) ... end

>> delete a,b,c;
>> p1(a,b,c);                                    c + 2
```

The expected result is obtained. You need to be aware of this feature of MuPAD when writing procedures.

### 6.7 A procedure that evaluates to itself

Many system procedures return themselves unevaluated if the system is unable to perform an evaluation. For example,

```
>> diff(f(x),x);
                               diff(f(x), x)
```

The same effect can be achieved in a procedure that you write by using *procname*, usually in the form `return(procname(args()))`. The following example finds the inverse of a given input provided it is non-zero and of type numeric; otherwise the unevaluated function is returned.

```
>> f := proc(x)
begin
  if testtype(x,Type::Numeric)
    then if (x<>0)
      then return(float(1/x))
      else return(procname(args())) end_if
    else return(procname(args())) end_if
end_proc:
>> f(x), f(x + 1), f(3), f(2*I), f(0);
                f(x), f(x + 1), 0.3333333333, - 0.5 I, f(0)
```

### 6.8 Exercises

1. Write a short procedure called date that takes three integers day, month, year as input and prints the date in the usual way. For example, the call date(5,3,2001) should yield output on the screen 5/3/2001. Now amend the procedure so that if month is outside the range 1..12 the procedure reports an error.

2. Define the function f, from the positive integers to the positive integers

   f(x) = 3 x + 1 for odd x
   = x/2 for even x.

   Next, define the sequence $x_{i+1} := f(x_i)$. Write a procedure that, on input of a positive integer n, returns the smallest index i with $x_i = 1$.

3. Implement a procedure Quadrature whose input is a function f (of one variable) and a list X of numerical values with

   $x_0 < x_1 < \ldots < x_n$.

   The call Quadrature(f,X) should compute a numerical approximation of the integral

   int(f(x),x = $x_0..x_n$)

   by means of the formula

   sum( $(x_{i+1} - x_i)$ $f(x_i)$, i = 0..(n-1) ).

# STUDY UNIT 7

## 7.1 Introduction

This study unit is concerned with the solution of equations in MuPAD. You will learn how to solve (linear and nonlinear) algebraic equations, both single equations and systems of equations, as well as ordinary differential equations and recurrence relations. As before these notes are not intended to be complete, but are rather a guide as to which topics you should be consulting in MuPAD's online help facility.

## 7.2 Algebraic equations

The command that used equation solution is *solve*. The syntax can be quite varied, with a number of defaults defined. As can be seen in the online help facility, the variables to be found do not have to be specified, and you can solve an expression (assumed to be equal to zero) rather than an equation. However, I do not regard such options as promoting good programming practice (because of the possibility of ambiguity and therefore error). Thus, in these notes and examples, we will always specify an equation(s) as well the variable(s) to be found. The syntax for solving one equation is illustrated in the following example

```
>> solution:=solve(x^4 - 5*x^2 + 6*x = 2, x);
                              1/2         1/2
                         {1, 3    - 1, - 3    - 1}
```

The result of solve is an identifier of type set. If a system of equations is to be solved, then both the system and the unknowns are normally given as sets (although, as discussed in the online help, other data types can be used). For example

```
>> solve({x^2 + y = 1, x - y = 2}, {x, y});
{ --         1/2                1/2       --
{ |        13                 13           |
{ |  x = - ----- - 1/2, y = - ----- - 5/2  |,
{ --         2                  2         --
   --       1/2              1/2       -- }
   |      13               13           | }
   |  x = ----- - 1/2, y = ----- - 5/2  | }
   --       2                2         -- }
```

The same notation can be used for a single equation, but then the output appears in a different form:

```
>> solve({x^4 - 5*x^2 + 6*x = 2}, {x});
                                1/2               1/2
                {[x = 1], [x = 3    - 1], [x = - 3    - 1]}
```

The solution set may be infinite, as in

```
>> S := solve(sin(x*PI/7) = 0, x);
                            { 7*X4 |  X4 in Z_ }
```

You can pick out the solutions in a certain finite interval by intersecting S with that interval

```
>> S intersect Dom::Interval(-22, 22);
                        {-21, -14, -7, 0, 7, 14, 21}
```

MuPAD can also be used to solve inequalities, in which case the result is (normally) an interval. For example

```
>> solve(x^2 > 5, x);
              ]5^(1/2), infinity[ union ]-infinity, -5^(1/2)[
```

Note that MuPAD uses the notation ]a,b[ to mean (a,b), i.e all real numbers x such that a<x<b. Except in the case that the result is an interval, MuPAD looks for solutions throughout the complex plane. If, for example, you are only interested in real soultions, then this option can be passed to solve as a third parameter. For example,

```
>> solve (x^2+1=0,x);
                                  {- I, I}
>> solve (x^2+1=0,x,Domain=Dom::Real);
                                     {}
```

where {} means that the solution set is empty.

In cases where the equation depends upon a parameter(s), it may be that the form of the solution depends upon the value(s) of the parameter(s). MuPAD allows for this effect, as can be seen in the following example, which is the well known case of the solution of a quadratic equation

```
>> solve(a*x^2+b*x+c=0,x);

         /
         |
         |
         |
piecewise| C_ if a = 0 and b = 0 and c = 0,
         \
                                    {   c }
  {} if a = 0 and b = 0 and c <> 0, { - - } if a = 0 and b <> 0,
                                    {   b }
```

```
   {                  2 1/2                   2 1/2 }            \
   {   b   (- 4 a c + b )      b   (- 4 a c + b )   }            |
   { - - - ----------------  - - + ---------------- }            |
   {   2           2           2          2         }            |
   { ----------------------, ---------------------- } if a <> 0  |
   {           a                       a            }            /
```

The MuPAD construct *piecewise* indicates an object that is conditionally defined, i.e. its form depends on which condition evaluates to TRUE.

If MuPAD cannot solve the problem, it returns the original expression:

```
>> solve(exp(x)-cos(x)=0,x);
                        solve(exp(x) - cos(x) = 0, x)
```

Note that Maple does better than MuPAD - for the above problem, it finds

```
> a:=solve(exp(x)-cos(x)=0,x);
                                 a := 0
```

**7.3 Ordinary differential equations**

The process for solving an ordinary differential equation is that: first an identifier representing the equation is set by means of the *ode* command; then the problem is solved by using the solve command. In the simplest case the syntax is illustrated in the following example

```
>> eq := ode(x^2*diff(y(x),x) + 3*x*y(x) = sin(x)/x, y(x));
                 /            sin(x)     2                       \
              ode| 3 x y(x) - ------ + x  diff(y(x), x), y(x)    |
                 \               x                               /

>> solve(eq);
                               {   C1 + cos(x) }
                               { - ----------- }
                               {        3      }
                               {       x       }
```

In the above example, the input to ode was the differential equation together with the unknown function; the result obtained was the general solution involving one arbitrary constant C1. You can also use ode with initial condition(s) specified; and with a system of differential equations rather than a single equation, or equivalently with a differential equation of order greater than first.

```
>> ivp := ode({diff(f(t),t,t) + 4*f(t) = sin(2*t),
           f(0) = a, D(f)(0) = b}, f(t));
 ode({f(0) = a, D(f)(0) = b, 4 f(t) + diff(f(t), t, t) - sin(2 t)}, f(t))

>> solve(ivp);

           { sin(2 t)                b sin(2 t)   t cos(2 t) }
           { -------- + a cos(2 t) + ---------- - ---------- }
           {    8                         2            4     }
```

In the above example, an initial value problem is defined for a second-order equation. The initial conditions are specified as being part of a set that defines the differential equation. Next, we illustrate the solution of a system of differential equations.

```
>> sys := {diff(x(t),t) - x(t) + y(t) = 0, diff(y(t),t) - x(t) - y(t)
= 0};
    {y(t) - x(t) + diff(x(t), t) = 0, diff(y(t), t) - y(t) - x(t) = 0}

>> eq:=ode(sys,{x(t),y(t)});
ode({- x(t) + y(t) + diff(x(t), t), - x(t) - y(t) + diff(y(t), t)},
   {x(t), y(t)})

>> solve(eq);

{[y(t) = C7 exp((1 + I) t) + C8 exp((1 - I) t),
   x(t) = I C7 exp((1 + I) t) - I C8 exp((1 - I) t)]}
```

In order to test whether MuPAD has found the correct solution, we substitute the result obtained into the original system of differential equations. The first step is to extract the solution list from the result obtained by MuPAD which is a set containing the solution list. We do this by using the *op* command.Then we can use subs in the usual way.

```
>> res:=op(%,1);
[y(t) = C7 exp((1 + I) t) + C8 exp((1 - I) t),
   x(t) = I C7 exp((1 + I) t) - I C8 exp((1 - I) t)]

>> test:=subs(sys,res);
{diff(C7 exp((1 + I) t) + C8 exp((1 - I) t), t) -
   (1 - I) C8 exp((1 - I) t) - (1 + I) C7 exp((1 + I) t) = 0,
   (1 - I) C7 exp((1 + I) t) + (1 + I) C8 exp((1 - I) t) +
   diff(I C7 exp((1 + I) t) - I C8 exp((1 - I) t), t) = 0}

>> test:=simplify(test);
                                  {0 = 0}
```

Because sys is a set, the system shows test as {0=0} rather than {0=0,0=0}.

MuPAD peforms reasonably for one, first-order ordinary differential equation. However, in other cases (as we will illustrate below) its performance is quite limited. Thus if MuPAD does not integrate an ordinary differential equation, this may not mean that an analytic solution does not exist. In such a case, your options are

! Try another computer algebra system
! Obtain a numerical solution (this will be discussed in a later study unit)
! There are some additional procedures available in the library package *detools*.

We use MuPAD then Maple to try to solve Bessel's differential equation in standard form. MuPAD gives

```
>> eq:=x^2*diff(w(x),x,x)+x*diff(w(x),x)+(x^2-a^2)*w(x)=0;
                           2                           2    2
        x diff(w(x), x) + x  diff(w(x), x, x) + w(x) (x  - a ) = 0

>> deq:=ode(eq,w(x));
                           2                             2    2
    ode(x diff(w(x), x) + x  diff(w(x), x, x) + w(x) (- a  + x ), w(x))

>> solve(deq);
                             2                             2    2
solve(ode(x diff(w(x), x) + x  diff(w(x), x, x) + w(x) (- a  + x ), w(x)))
```

However, applying Maple to the same problem gives

```
> with(DEtools):
> eq:=x^2*diff(w(x),x,x)+x*diff(w(x),x)+(x^2-a^2)*w(x)=0;
                 / 2      \
               2 |d       |     /d      \     2    2
        eq := x  |--- w(x)| + x |-- w(x)| + (x  - a ) w(x) = 0
                 | 2      |     \dx     /
                 \dx      /
> dsolve(eq,w(x));
             w(x) = _C1 BesselJ(a, x) + _C2 BesselY(a, x)
```

### 7.4 Solving recurrence relations

A recurrence relation is something of the form

w(n+1) = f(w(n),w(n-1),w(n-2),...,n)

where n is an integer. A recurrence relation is the discrete analogue of a differential equation, and the solution process is similar to that for ordinary differential equations. First, the relation is defined by means of the command *rec*, and then solve is applied. The syntax is illustrated in the following example

```
>> r1:=rec(y(n + 1) = 2*y(n)*(n + 1)/n, y(n));
                  /             2 y(n)  (n + 1)          \
                rec| y(n + 1) - --------------, y(n), {} |
                  \                   n                  /
>> solve(r1);
                                        n
                                 {n C1 2 }
```

Initial conditions can be specified, but the syntax is different to that used for differential equations. We consider the above example, with y(1)=1

```
>> r1:=rec(y(n + 1) = 2*y(n)*(n + 1)/n, y(n),{y(1)=1});
               /            2 y(n)  (n + 1)                  \
            rec| y(n + 1) - --------------, y(n), {y(1) = 1} |
               \                  n                          /
>> solve(r1);
                                 {    n }
                                 { n 2  }
                                 { ---- }
                                 {   2  }
```

MuPAD can handle higher-order recurrence relations, as well as parameters appearing in the initial data or equations. For example

```
>> solve(rec(y(n + 2) - 2*y(n + 1) + y(n) = 2, y(n),
          {y(0) = -1, y(1) = m}));
                                      2
                              {m n + n  - 1}
```

If the form of the solution depends upon the value of a parameter, then a piecewise solution is found

```
>> solve(rec(a*y(n + 2) = y(n), y(n)));

          /               {    /  1   \n       /    1   \n   }            \
 piecewise| {0} if a = 0, { C5 | ---- |   + C4 | - ---- |   } if a <> 0 |
          |               {    | 1/2  |        |   1/2  |   }            |
          \               {    \ a    /        \   a    /   }            /
```

MuPAD cannot directly solve a system of recurrence relations: in order to do so, the system must be re-written as a single higher-order relation. For example,

y(n+1)=2*y(n)+z(n), z(n+1)=z(n)+y(n)

can be written as

y(n+2) = 2*y(n+1) + z(n+1), y(n+1) = 2*y(n) + z(n), z(n+1) = z(n) + y(n)

which system can be solved (with y(n+2), z(n+1) and z(n) regarded as unknowns) to find a 2nd order recurrence relation

y(n+2) = 3*y(n+1) - y(n)

and then solved by MuPAD. The complete programme is

```
>>eq:={y(n+2)=2*y(n+1)+z(n+1),y(n+1)=2*y(n)+z(n),z(n+1)=z(n)+y(n)};
{z(n + 1) = y(n) + z(n), y(n + 1) = 2 y(n) + z(n),
   y(n + 2) = 2 y(n + 1) + z(n + 1)}

>> s:=solve(eq,{y(n+2),z(n+1),z(n)});
{[z(n) = y(n + 1) - 2 y(n), y(n + 2) = 3 y(n + 1) - y(n),
   z(n + 1) = y(n + 1) - y(n)]}

>> r1:=op(op(s,1),2);
                       y(n + 2) = 3 y(n + 1) - y(n)

>> r:=rec(r1,y(n));
                rec(y(n) - 3 y(n + 1) + y(n + 2), y(n), {})

>> solve(r);
                {    /  1/2       \n      /        1/2 \n }
                {    | 5          |       |       5    |  }
                { C1 | ---- + 3/2 |  + C2 | 3/2 - ---- |  }
                {    \  2         /       \        2   /  }
```

Note the use of the *op* command to extract an element of a set or list.

### 7.5 Exercises

1. Solve the following algebraic equations, or systems of algebraic equations, and in each case verify your solution(s) by substitution into the original equations

   (a) $x^3 - 13x + 12 = 0$

   (b) $x + 2y = 3$, $y + 1/x = 1$

   (c) $\sin(x/2) = -2\sin(x/2)\cos(2x)$, with x real and $-2\pi \le x \le 2\pi$.

2. Solve the following inequalities, in each case restricting the solution to the real line

   (a) $1/|1-2x| > 3$

   (b) $3x \ge |x - 2|$

3. Explore the capabilities of MuPAD for solving non-polynomial equations, i.e. equations involving trigonometric functions, log, exp, $a^x$, etc.

4. Find an example of one first-order differential equation that MuPAD cannot solve.

5. Construct a system of three first-order differential equations and initial data, and use MuPAD to find a solution.

6. Solve the following system of recurrence relations

$$x(n+1) = 2x(n) + y(n) - z(n), \quad y(n+1) = x(n) -2y(n) -z(n), \; z(n+1) = x(n)/2 +y(n)/3 - z(n)/4.$$

# STUDY UNIT 8

## 8.1 Introduction

This study unit is concerned with the construction, manipulation and use of series in MuPAD. As before these notes are not intended to be complete, but are rather a guide as to which topics you should be consulting in MuPAD's online help facility.

## 8.2 Taylor series

The command *taylor* will compute the taylor series of a given expression about a given point. For example,

```
>> taylor(sin(a*x)/x,x=0);
                               3  2    5  4
                              a  x    a  x       5
                          a - ----- + ----- + O(x )
                                6      120
```

You need to specify the expression, the variable and the point about which the expansion is to be made. You can also specify to what order the expansion should be made - if this is not given the system will use the value of the environment variable *ORDER* whose default value is 6. Note that the order is calculated relative to the lowest degree term. Thus

```
>> taylor(sin(a*x)/x,x=0,10);
                     3  2    5  4    7  6    9  8
                    a  x    a  x    a  x    a  x        9
                a - ----- + ----- - ----- + ------ + O(x )
                      6      120    5040    362880
```

and

```
>> taylor(sin(a*x)*x,x=0,10);
                      3  4    5  6    7  8    9  10
                 2   a  x    a  x    a  x    a  x        11
              a x  - ----- + ----- - ----- + ------ + O(x  )
                       6      120    5040    362880
```

The taylor series may be computed about any point in the complex plane, including +infinity or -infinity (which are on the real line). For example,

```
>> s1:=taylor(sin(x),x=-I);
                                                        2
- I sinh(1) + cosh(1)  (x + I)  + 1/2 I sinh(1)  (x + I)   -
                   3                                             5
   cosh(1)  (x + I)                          4   cosh(1)  (x + I)
   ---------------- - 1/24 I sinh(1)  (x + I)  + ---------------- +
          6                                             120
            6
   O((x + I) )
```

```
>> s2:=taylor(z/(1-z),z=infinity);

                           1   1    1    1     / 1  \
                     - 1 - - - -- - -- - -- + O| -- |
                           z    2    3    4    |  5 |
                               z    z    z     \ z  /
```

Series may be manipulated using the usual arithmetical operations, and MuPAD will handle the error terms correctly. For example,

```
>> s1:=taylor(sin(x),x=PI,10);

                     3           5           7           9
             (x - PI)    (x - PI)    (x - PI)    (x - PI)              10
(- x + PI) + --------- - --------- + --------- - --------- + O((x - PI)   )
                 6          120         5040       362880
```

```
>> s2:=taylor(cos(x),x=PI);
                            2           4
                    (x - PI)    (x - PI)             6
              - 1 + --------- - --------- + O((x - PI) )
                        2          24
```

```
>> s1*s2;
                                3             5
                      2 (x - PI)    2 (x - PI)             7
           (x - PI) - ----------- + ----------- + O((x - PI) )
                           3             15
```

### 8.3 General series

In order for a Taylor series to exist, the expression and its derivatives (up to the given order) must exist (i.e., must be finite) at the point of expansion. When this is not the case, it may still be meaningful to construct a series representation of the expression, and this can be done by using the command *series*. For example

```
>> taylor(cos(x)/x,x=0);
Error: does not have a Taylor series expansion, try 'series' [taylor]

>> series(cos(x)/x,x=0);
                                     3
                            1   x   x       5
                            - - - + -- + O(x )
                            x   2   24
```

The syntax for using the command series is much the same as that for taylor, and we only describe features that are found in series but not in taylor. In general, series will return a result only if the expansion is valid in a neighbourhood of the complex plane about the point of expansion. However, it is possible to obtain an expansion valid only on the real line, or restricted to points on the real line to the right or left of the point of expansion, as indicated in the examples below. Note that the function sign(z) is defined for all z in the complex plane by

```
            z
 sign(z) = ---   if z<>0
           |z|
         =  0    if z=0
```

Further details are available under *sign* in the online help. The examples are

```
>> series(x*(1-sign(x)),x=0);
                                                7
                         x (- sign(x) + 1) + O(x )
>> series(x*(1-sign(x)),x=0,Real);
                                                7
                         x (- sign(x) + 1) + O(x )
>> series(x*(1-sign(x)),x=0,Right);
                                      7
                                   O(x )
>> series(x*(1-sign(x)),x=0,Left);
                                         7
                                2 x + O(x )
```

Using series, it is possible to construct an infinite symbolic sum - this process can be connsidered to be the converse of using the command sum with upper limit infinity, as discussed in an earlier study unit. The syntax is illustrated in the following example:

```
>> series(exp(-x), x, infinity);

                      /   n2     n2                    \
                      |  x   (-1)                      |
                   sum| ------------, n2 = 0..infinity |
                      \ n2 gamma(n2)                   /
```

where the *gamma* function is described fully in the online help, and note that n*gamma(n)=n!
The command *asympt* computes an asymptotic series, i.e. makes an expansion about infinity. The effect is the same as using series (or taylor, if permissible) with an expansion point of infinity, so we will not describe this command further.

**8.4 Extracting the coefficients of a series**
Once a series has been computed, it may be useful for subsequent calculations, to express the series as an expression without the error term, or to work with the coefficient of a particular power in the series. This can be done by using the commands *expr* and *coeff*. We will illustrate these commands for series, but you should be aware that they both have more general applicability (and in some cases,variations of the syntax)- see the online help.

```
>> s:=series(sin(a*x^2),x=0,8);
                                   3  6
                              2   a  x       8
                           a x  - ----- + O(x )
                                    6

>> ex:=expr(s);
                                       3  6
                                  2   a  x
                               a x  - -----
                                        6

>> c2:=coeff(s,2);
                                     a

>> c6:=coeff(s,6);
                                      3
                                     a
                                   - --
                                     6

>> c4:=coeff(s,4);
                                    0

>> coeff(s);
                                           3
                                          a
                            a, 0, 0, 0, - --, 0
                                          6
```

There are also available specialized coefficient commands, *lcoeff*, *tcoeff*and *nthcoeff*, but since the effect of these commands is contained in coeff, we will not describe them here and instead refer you to the online help.

### 8.5 Example of the use of series

We conclude this unit by giving a real example of how series in computer algebra can be used to solve problems. The following integral arose, and neither MuPAD, nor any other system, could find an analytic solution

```
>> f:=1/(sin(x^2))^(1/4);
                                     1
                                ----------
                                      2 1/4
                                sin(x )

>> int(f,x=0..PI/6);
                          /     1               PI \
                       int| ----------, x = 0..-- |
                          |      2 1/4         6   |
                          \ sin(x )                /
```

In such a case, the next obvious approach to try is to find a numerical approximation (about which more will be said in a later study unit). However, that is going to be problematic in the above example, because the integrand is infinite at one of the end-points (x=0). It is not clear whether the integral exists, and even if it does numerical methods are unreliable in handling infinite quantities. We use series to solve the problem, as shown.

```
>> fs:=series(f,x=0,16);
                       7/2        15/2        23/2
                1     x       13 x        17 x             27/2
               ---- + ---- + -------- + -------- + O(x     )
                1/2    24       5760      107520
               x


>> fe:=expr(fs);
                             7/2        15/2        23/2
                      1     x       13 x        17 x
                     ---- + ---- + -------- + --------
                      1/2    24       5760      107520
                     x
```

```
>> fi:=int(fe,x=0..PI/6);

                     / PI \9/2   / PI \17/2      / PI \25/2
                     | -- |   13 | -- |       17 | -- |
          / PI \1/2  \ 6  /      \ 6  /          \ 6  /
        2 | -- |   + --------- + ------------- + -------------
          \ 6  /        108         48960          1344000


>> coeff(fi);
                              17/1344000, 13/48960, 1/108, 2

>> t1:=nthterm(fi,1);
                                         / PI \25/2
                                         | -- |
                                         \ 6  /

>> t2:=nthcoeff(fi,1);
                                         17/1344000

>> last_term:=float(t1*t2);
                                     0.00000000388626518

>> ans:=float(fi);
                                         1.44770718
```

The series expansion shows that the leading term near x=0, is x^(-1/2) so the function is clearly integrable. We use *nthterm* and *nthcoeff* to find the last term in our approximation to the integral, and then use float to find its numerical value. In this case it is clear by inspection that the series representing the integral is rapidly converging and so the value of the last term used gives an estimate of the error in ignoring the rest of the terms in the infinite series. The estimate obatined for the integral is 1.44770718.

### 8.6 Exercises

1. The order p of a root x of a function f is the order of the smallest non-vanishing derivative at the point x. For example, if f(x) = $x^2$, then the order of the root at x = 0 is 2. Find the order of the root at x = 0 of f(x) = exp(sin(x))-sin(exp(x)-1)-1.

2. Find the first five terms of the asymptotic expansion of $\sqrt{x+1}-\sqrt{x-1}$ .

3. Find the Taylor series, up to an error term O($z^8$) of

   $$f(z)=\sqrt{\cos(z)}$$

   about z = π. Create a 1-dimensional array F[i] that contains the terms of the series, with F[0] holding the $z^0$ term, F[1] the $z^1$ term, etc. up to F[7].

4. It is desired to solve the differential equation

$$\text{diff}(y(x),x) = y(x)\cos(x^2) - \sin(x^3), \text{ with } y(0) = 2.$$

MuPAD cannot do this directly, so instead find a series solution of the form

$$y(x) = 2 + y_1\, x + y_2\, x^2 + \ldots + y_9\, x^9 + O(x^{10})$$

Hint: Express $\cos(x^2)$ and $\sin(x^3)$ as series, and in the differential equation equate coefficients of $x^0, x^1, \ldots, x^8$.

# STUDY UNIT 9

## 9.1 Introduction

This study unit is concerned with the manipulation, transformation and simplification of algebraic expressions. Some aspects of expression simplification were discussed in study unit 3, specifically the commands *simplify, factor, expand, normal, subs* and *subsex*. These notes will be concerned with other methods of simplification. As before these notes are not intended to be complete, but are rather a guide as to which topics you should be consulting in MuPAD's online help facility.

We start by making some general comments about simplification in computer algebra.

- Algorithms for the simplification of expressions are computationally expensive (and the cost grows exponentially with the length of the expression). This is one reason that in MuPAD, as well as in most other computer algebra systems, simplification is user controlled.

- Another reason why simplification is user controlled is that the most appropriate form of an expression often depends upon the context. For example, consider

```
      cos(2 x)                2
1/2 - --------   =   sin(x)
         2
```

  The right hand side appears simpler, but for many subsequent manipulations, the left hand side is a more appropriate starting-point.

- MuPAD regards all identifiers as being elements of the complex plane, and makes only those simplifications that apply to every element of the complex plane. For example, `ln(exp(x))=x` is true only for `x>0` and if we want MuPAD to make such a simplification to an identifier that is real and positive, then we have to tell the system that this is the case.

## 9.2 The command *collect*

The command *collect* is used to re-express some expression as a polynomial expression in some specified unknown(s). For example,

```
>> p := x*y + z*x*y + y*x^2 - z*y*x^2 + x + z*x;
                                            2      2
                   x + x y + x z + x y z + x  y - x  y z
collect(p, [x, y]);
                                             2
                  x (z + 1) + x y (z + 1) + x  y (1 - z)
```

If there is only one unknown identifier then the square brackets can be omitted:

```
>> collect(p, x);
                                           2
                    x (y + z + y z + 1) + x  (y - y z)
```

We can also apply a MuPAD command to each of the coefficients of the resultant polynomial expression, as indicated in the following example with the command factor

```
>> collect(p, x, factor);
                                          2
                     x (y + 1) (z + 1) - x  y (z - 1)
```

### 9.3 The command *combine*

The command *combine* combines sub-expressions by using specified mathematical identities. If no identity is given, then it combines powers of the same base and powers with the same exponent. Its syntax and use is illustrated in the following examples

```
>> combine(sin(x) + x*y*x^(exp(1)));

                                       exp(1) + 1
                           sin(x) + y x
```

```
>> combine(sqrt(2)*sqrt(3));

                                     1/2
                                    6
```

Additional rules can be applied if a second argument is given to combine. This second argument must be one of arctan, exp, ln, sincos, sinhcosh, or it could be several of these in a list. A precise statement of the mathematical rules applied can be found in the online help.

```
>> combine(sin(a)*cos(b) + sin(b)^2, sincos);

                 sin(a + b)   cos(2 b)   sin(a - b)
                 ---------- - -------- + ---------- + 1/2
                     2            2            2
```

```
>> combine(exp(a)^2, exp);
                                  exp(2 a)
>> combine(ln(2)+ln(3)+sin(a)*cos(a), [ln, sincos]);

                                      sin(2 a)
                              ln(6) + --------
                                         2
```

The following example concerns ln(a)+ln(b), which is ln(a b) when a>0 and b>0, but not (in general) otherwise. Thus the behaviour below of MuPAD is correct. We will say more about the command assume later in this study unit.

```
>> combine(ln(a)+ln(b), ln);
                                ln(a) + ln(b)


>> assume(a>0): assume(b>0):
combine(ln(a)+ln(b), ln);
                                   ln(a b)
```

### 9.4 The command *rewrite*

The syntax of rewrite is the same as that of combine, although its effect is different. For example rewrite(f,exp) replaces all trigonometric and hyperbolic functions in f by exponential functions

```
>> rewrite(exp(a)^2, exp);
                                        2
                                   exp(a)


>> rewrite(tan(x), exp);
                                         2
                                I exp(I x)  - I
                              - ---------------
                                        2
                                  exp(I x)  + 1
```

As well as exp, the following targets can be used: cot, coth, diff, exp, fact, gamma, heaviside, ln, piecewise, sign, sincos, sinhcosh, tan, tanh. Unlike combine, several targets may **not** be specified in one rewrite command by placing them in a list. Instead, you would need to make a series of rewrite commands. A description of the substitution effect of each target is given in the online help, and we now illustrate the behaviour of rewrite with some examples

```
>> rewrite(D(D(y))(x), diff);
                             diff(y(x), x, x)

>> rewrite(fact(n), gamma), rewrite(gamma(n), fact);

                         gamma(n + 1), fact(n - 1)

>> rewrite(sign(x), heaviside), rewrite(heaviside(x), sign);

                                         sign(x)
                     2 heaviside(x) - 1, ------- + 1/2
                                            2
>> rewrite(cot(x), sincos), rewrite(sin(x), tan);

                                         / x \
                                    2 tan| - |
                           cos(x)        \ 2 /
                           ------, -------------
                           sin(x)     / x \2
                                   tan| - |  + 1
                                      \ 2 /
>> rewrite(arcsinh(x), ln);
                                     2     1/2
                            ln(x + (x  + 1)   )
```

### 9.5 The command *partfrac*

Given a rational expression f(x) = g(x) +p(x)/q(x), partfrac attempts to factorize q(x) and then expresses p(x)/q(x) as a partial fraction, that is as a sum of terms each containing one factor of q(x) in the denominator. See the online help for a more complete description. The first argument of partfrac is the function f that is to be expressed as a partial fraction; if the function f contains only one identifier then a second argument is not needed, but if it contains several identifiers then the second argument specifies the identifier with respect to which the partial fraction will be computed. The behaviour of partfrac is illustrated in the following examples

```
>> partfrac(23 + (x^4 + x^3)/(x^3 - 3*x + 2));

                       19            2             8
                x + --------- + ---------- + --------- + 24
                    9 (x - 1)            2   9 (x + 2)
                                3 (x - 1)
```

```
>> f := x^2/(x^2 - y^2): partfrac(f, x),  partfrac(f, y);
                     y           y          x           x
             1 - --------- - ---------, --------- - ---------
                 2 (y - x)   2 (x + y)  2 (x + y)   2 (y - x)
```

```
>> partfrac(1/(sin(x)^4 - sin(x)^2 + sin(x) - 1), sin(x));
                                                    2
                                   2 sin(x)   sin(x)
                                 - -------- - ------- - 2/3
                      1               3          3
                -------------- + --------------------------
                3 (sin(x) - 1)           2         3
                                   sin(x)  + sin(x)  + 1
```

Since partfrac uses factor, it looks for factors over the rationals and is unable to handle the next expression

```
>> partfrac(1/(x^2 + 2), x);

                                    1
                                  ------
                                   2
                                  x  + 2
```

However, we can make MuPAD find the factorization by multiplying through by 1/sqrt(-2)

```
>> partfrac(x/(sqrt(-2)*x^2 + 2*sqrt(-2)), x);

                      1/2 /       1                1        \
             - 1/2 I 2    | -------------- + -------------- |
                          |           1/2              1/2  |
                          \ 2 (x - I 2   )   2 (x + I 2   ) /
```

The use of partfrac is not to simplify expressions, but rather an expansion as a partial fraction is a necessary starting point for a number of computations, for example the integration of rational functions.

### 9.6 The command *rectform*

The commmand rectform is used to split an expression into its real and imaginary parts. This can also be done by means of the commands *Re* and *Im* which return the real and imaginary parts of an expression, but as we shall see in the examples below, rectform is more powerful than Re and Im in that it returns useful answers in some cases in which Re and Im fail. Also useful in this context is the command *conjugate* which returns the complex conjugate of the input expression.

```
>> delete z: r := rectform(sin(z));
            sin(Re(z)) cosh(Im(z)) + (cos(Re(z)) sinh(Im(z))) I


>> Re(r), Im(r);
              sin(Re(z)) cosh(Im(z)), cos(Re(z)) sinh(Im(z))


>> conjugate(r);
           sin(Re(z)) cosh(Im(z)) + (-cos(Re(z)) sinh(Im(z))) I


>> Re(ln(-4)) + I*Im(ln(-4));
                               I PI + ln(4)
```

The next example demonstrates that rectform is more powerful than Re and Im

```
>> delete z: f := exp(I*sin(z)):
Re(f), Im(f);
                   Re(exp(I sin(z))), Im(exp(I sin(z)))


>> r := rectform(f);

cos(sin(Re(z)) cosh(Im(z))) exp(-cos(Re(z)) sinh(Im(z))) +

   (sin(sin(Re(z)) cosh(Im(z))) exp(-cos(Re(z)) sinh(Im(z)))) I


>> Re(r);
         cos(sin(Re(z)) cosh(Im(z))) exp(-cos(Re(z)) sinh(Im(z)))


>> Im(r);
         sin(sin(Re(z)) cosh(Im(z))) exp(-cos(Re(z)) sinh(Im(z)))
```

The next example shows that the behaviour of rectform is sensitive to variable properties stated via the assume command

```
>> delete z: rectform(ln(z));

                          2        2
                   ln(Im(z)  + Re(z) )
                   ------------------- + I arg(Re(z), Im(z))
                            2

>> assume(z < 0): rectform(ln(z));

                                 ln(-z) + I PI
```

Finally, rectform can be applied to polynomials, series, lists, sets and arrays: in these cases it operates on each element of the data structure. For other data structures, rectform can be applied to each element by means of the *map* command

```
>> delete x, y: p := poly(ln(-4) + y*x, [x]):
rectform(p);

              poly((Re(y) + I Im(y)) x + (ln(4) + I PI), [x])

>> a := table("1st" = x, "2nd" = y):
rectform(a);
Error:  invalid  argument,  expecting  an  arithmetical  expression
[rectform::n\
ew]
>> map(a, rectform);

                        table(
                          "2nd" = Re(y) + I Im(y),
                          "1st" = Re(x) + I Im(x)
                        )
```

### 9.7 The command *radsimp*

The command radsimp(f), where f is an expression, is equivalent to simplify(f,sqrt), that is, it simplifies expressions containing radicals (square roots, or any other root). An example of its use is:

```
>> radsimp(2*2^(1/4) + 2^(3/4) - (6*2^(1/2) + 8)^(½));

                                        0
```

### 9.8 The command *assume*

As stated above, MuPAD regards all identifiers as being elements of the complex plane. If a particular identifier is known to belong to a subset of the complex plane, this information is (usually) passed to the system by means of the assume commnad. The syntax of assume statements, and the consequent use in simplifications, is illustrated in the following examples. Note that an assume command can be cancelled by the *unassume* command; i.e. unassume(z) means that, whatever has been stated before, the identifier z is now regarded as a general complex number. Normally, an assume command about an identifier cancels any previous properties set; unless the assume command contains the optional argument _and or _or, in which case the setting in the current command is combined with previous information. Information about identifiers is set, either implicitly or explicitly, by means of the Type library - to find out more about the options available, enter ?Type

```
>> assume(n, Type::Integer);
                               Type::Integer

>> simplify(sin(2*n*PI));
                                     0

>> assume(n, Type::NonNegInt);
                              Type::NonNegInt

>> assume(n, Type::NegInt, _or);
                               Type::Integer

>> assume(n, Type::Positive, _and);
                               Type::PosInt

>> assume(Re(z) > 0), assume(Im(z) < 0, _and);
                    Re(.) > 0, Re(.) > 0 and Im(.) < 0

>> abs(Re(z)), sign(Im(z));
                                 Re(z), -1

>> assume(x > y);
                                    < x

>> assume(y > 0, _and);
                           ]0, x[ of Type::Real
```

```
>> expand(ln(z1*z2));
                                   ln(z1 z2)

>> assume(z1 > 0): expand(ln(z1*z2));
                                ln(z1) + ln(z2)
```

Using an assume statement without specifying the identifier means that the assumption applies to **all** identifiers.

```
>> reset();
>> assume(Type::NonNegative);
                               Type::NonNegative

>> Re(x), Im(y), sign(1 + z^2);
                                    x, 0, 1
```

It is not permitted to apply an assume statement to an expression:

```
>> a:=b+c;

                                     b + c

>> assume(a>0);
Error: wrong type of first argument [property::assume]
```

### 9.9 The command *limit*

MuPAD provides the facility to find the limit of an expression, using the syntax

```
>> limit(f,x=x0 <,dirn>);
```

where f is an expression containing the identifier x, and x0 is the limit point. The optional argument dirn may have the values Left or Right and specifies the direction in which the limit is to be taken; if it is omitted, the system looks for a bi-directional limit and if one is not found limit returns the value undefined. The limit point x0 may have the value infinity or -infinity. The use of limit is illustrated in the following examples

```
>> limit((1 - cos(x))/x^2, x);

                                      1/2
```

```
>> limit((1 + 1/n)^n, n = infinity);
                                      exp(1)


>> limit(
   (exp(x*exp(-x)/(exp(-x) + exp(-2*x^2/(x+1)))) - exp(x))/x,
   x = infinity);
                                      -exp(2)


>> limit(1/x, x = 0);
                                     undefined


>> limit(1/x, x = 0, Left),
limit(1/x, x = 0, Right);
                                 -infinity, infinity


>> assume(n < 0): limit(x^n, x = infinity);

                                          0
```

### 9.10 Exercises

**N.B.** Throughout this set of exercises, **all variables are real, *except***

- a, b are real and strictly positive (i.e., > 0)
- j, k are integers (positive, negative or zero)
- m, n are strictly positive integers (i.e., > 0)
- w, z are complex

1. Transform the following expression into a polynomial in sin(x)

- $(9 + 8\sin(x)\,y - 5\sin^2(x)\,y)^3 + (1 + \sin(x))^2/y$

2. Change the following expression into a Fourier series, i.e all trigonometric terms must be of the form sin(nx) or cos(mx) (i.e. the answer must not contain any term of the form $\sin^2(nx)$, $\cos^2(mx)$, sin(nx) cos(mx), or higher powers)

- $(1 + 8\sin(x)y - 5\sin^2(x)\,y)^3 + (1 + \sin(x))^2$

3. Express the following in terms that do not contain trigonometric functions or their inverses

- $\arctan(w^2)\,\exp(\arcsin(z))$
- $(1 + 8\sin(w)\,y - 5\cos^2(w)\,y)^2$

4. Transform $(-5x^2 - 3x^3 + x^4 +15x +1)/(-5x -3x^2 + x^3 + 15)$ into a sum of terms each of which has a simple analytic integral.

5. Find the real and imaginary parts of

- $\sin^n(w) + \ln(1+a) + \ln(z)$
- exp(I arctan(nx))

6. Find the limits of the following expressions at the point indicated. If a limit does not exist, investigate the directional limits.

- $(t^{(1/7)}-3)/(t-2187)$ at $t = 2187$
- $x^n/\exp(x)$ at $x = \infty$
- $\sin(ax^3)/x^3$ at $x = 0$

# STUDY UNIT 10

## 10.1 Introduction

This study unit is concerned with the use of MuPAD's numerical packages. Many real problems do not admit a solution in analytic form, and the branch of mathematics called numerical analysis is concerned with finding approximate solutions to such problems - see APM213-X and APM311-Y. MuPAD, in common with other such systems, includes a library of numerical packages. However, before using them, a word of caution is needed: **such packages do not always give the correct answer**. The domain of applicability of a particular numerical method is the subject of numerical analysis, and will not be discussed in these notes, except that we will provide some examples of non-performance.

The numerical packages available in MuPAD can be seen by typing ?numeric. We will not discuss all of them, but only those applicable to the following types of problem (which have already been discussed from the point of view of obtaining a symbolic solution)

- Linear algebra
- Solution of an algebraic equation, or system of equations
- Definite integrals
- Ordinary differential equations

All floating-point computations in MuPAD are sensitive to the value of the environment variable *DIGITS*, whose default value is 10. This means that all floating-point identifiers are correct to 10 significant places. If a different level of accuracy is needed, this can be achieved with a statement of the form DIGITS:=20; (if 20 significant figures are desired).

## 10.2 Linear algebra

The packages listed under numeric in the online help that are relevant to linear algebra, and whose application you should understand, are: det, eigenvalues, eigenvectors, factorLU, inverse and matlinsolve. (Other linear algebra packages are: expmatrix, factorCholesky, factor QR, fmatrix, singularvalues, singulavectors, spectralradius). We will not describe each package, but will just give a couple of examples - the application of the packages is straightforward, and the detailed syntax is given in the online help. We start with two general comments:

- The numerical packages are much faster than the corresponding symbolic ones (for the case in which a matrix has only numerical entries).
- If a matrix is singular, then operations like inverse are not defined. A singular matrix is one whose determinant is zero, but unfortunately we cannot say that a matrix is almost singular if the determinant is very small. Matrices that are not singular, but are nearly singular, are called ill-conditioned and many numerical operations do not work accurately for such matrices. This issue is fully discussed in APM213. Here we will only note that a matrix is definitely well-conditioned if it is diagonally dominant, which means that the absolute value of the diagonal term is greater than the sum of the absolute values on the row or column through the diagonal term.

In the example below, we first define a procedure that constructs a square matrix of arbitrary size, and then use the matrix (m6) in some of the numeric packages. Specifically, we find the determinant, inverse, eigenvectors of m6 (in this case, the system returns the eigenvalues, then a matrix of the eigenvectors, and then an estimate of the error); and finally we construct a 6-vector B, whose entries are all 1, and solve m6 x = B to find x (The system also returns the kernel of the null space - since, here, it is zero this means that the solution found for x is unique). For reasons of space in the printed form of this study unit, we use examples in which the size of the matrix is 6 x 6, but you should try out the packages with larger matrices.

```
>> del:=proc(n)
     local m,i,j;
     begin
     m:=matrix(n,n);
      for i from 1 to n do for j from 1 to n do m[i,j]:=1/(1+4*(i-j))
end_for end_for;
     return(m)
     end_proc;
                                       proc del(n) ... end

>> m6:=del(6);
                       +-                                        -+
                       |    1,   -1/3,  -1/7,  -1/11,  -1/15,  -1/19  |
                       |                                          |
                       |   1/5,    1,   -1/3,   -1/7,  -1/11,  -1/15  |
                       |                                          |
                       |   1/9,   1/5,    1,    -1/3,   -1/7,  -1/11  |
                       |                                          |
                       |  1/13,   1/9,   1/5,    1,     -1/3,   -1/7  |
                       |                                          |
                       |  1/17,  1/13,   1/9,   1/5,     1,     -1/3  |
                       |                                          |
                       |  1/21,  1/17,  1/13,   1/9,    1/5,     1    |
                       +-                                        -+

>> numeric::det(m6);
                                           1.517285633
```

```
>> numeric::inverse(m6);
                                  array(1..6, 1..6,
                                    (1, 1) = 0.9105808586,
                                    (1, 2) = 0.2396265417,
                                    (1, 3) = 0.1597510278,
                                    (1, 4) = 0.1307053864,
                                    (1, 5) = 0.1213692874,
                                    (1, 6) = 0.137551859,
                                    (2, 1) = -0.2168049663,
                                    (2, 2) = 0.8558090776,
                                    (2, 3) = 0.2053941786,
                                    (2, 4) = 0.1348547637,
                                    (2, 5) = 0.1133669168,
                                    (2, 6) = 0.1213692874,
                                    (3, 1) = -0.07651939988,
                                    (3, 2) = -0.234927982,
                                    (3, 3) = 0.8457407355,
                                    (3, 4) = 0.1999023557,
                                    (3, 5) = 0.1348547637,
                                    (3, 6) = 0.1307053864,
                                    (4, 1) = -0.04120275378,
                                    (4, 2) = -0.08519366384,
                                    (4, 3) = -0.2385422587,
                                    (4, 4) = 0.8457407355,
                                    (4, 5) = 0.2053941786,
                                    (4, 6) = 0.1597510278,
                                    (5, 1) = -0.02517946065,
                                    (5, 2) = -0.04517845809,
                                    (5, 3) = -0.08519366384,
                                    (5, 4) = -0.234927982,
                                    (5, 5) = 0.8558090776,
                                    (5, 6) = 0.2396265417,
                                    (6, 1) = -0.01510767639,
                                    (6, 2) = -0.02517946065,
                                    (6, 3) = -0.04120275378,
                                    (6, 4) = -0.07651939988,
                                    (6, 5) = -0.2168049663,
                                    (6, 6) = 0.9105808586
                                  )
```

```
>> numeric::eigenvectors(m6);
   [0.9244813549 + 0.5550206611 I, 0.9244813549 - 0.5550206611 I,
[

    1.014569455 - 0.3412503189 I, 1.014569455 + 0.3412503189 I,


    1.060949191 + 0.1152572984 I, 1.060949191 - 0.1152572984 I],

    array(1..6, 1..6,
      (1, 1) = 0.3477101864 - 0.1665108884 I,
      (1, 2) = - 0.1665108884 + 0.3477101864 I,
      (1, 3) = - 0.5565285692 + 0.01897686 I,
      (1, 4) = 0.01897686 - 0.5565285692 I,
      (1, 5) = - 0.5059177529 - 0.3979217839 I,
      (1, 6) = - 0.3979217839 - 0.5059177529 I,
      (2, 1) = 0.1062202522 - 0.4708388878 I,
      (2, 2) = - 0.4708388878 + 0.1062202522 I,
      (2, 3) = - 0.06485973287 - 0.5353865522 I,
      (2, 4) = - 0.5353865522 - 0.06485973287 I,
      (2, 5) = 0.06578777068 + 0.3796940818 I,
      (2, 6) = 0.3796940818 + 0.06578777068 I,
      (3, 1) = - 0.2693482055 - 0.4214744462 I,
      (3, 2) = - 0.4214744462 - 0.2693482055 I,
      (3, 3) = 0.3263912189 + 0.003860733824 I,
      (3, 4) = 0.003860733824 + 0.3263912189 I,
      (3, 5) = - 0.06042207101 - 0.3976658933 I,
      (3, 6) = - 0.3976658933 - 0.06042207101 I,,
      (4, 1) = - 0.4364352658 - 0.1114749996 I,
      (4, 2) = - 0.1114749996 - 0.4364352658 I,
      (4, 3) = - 0.1937587833 + 0.2010783346 I,
      (4, 4) = 0.2010783346 - 0.1937587833 I,
      (4, 5) = - 0.2273927667 + 0.2604311944 I,
      (4, 6) = 0.2604311944 - 0.2273927667 I,
      (5, 1) = - 0.3058555127 + 0.166765253 I,
      (5, 2) = 0.166765253 - 0.3058555127 I,
      (5, 3) = - 0.2290160963 - 0.2845068304 I,
      (5, 4) = - 0.2845068304 - 0.2290160963 I,
      (5, 5) = 0.1375406695 - 0.1845522146 I,
      (5, 6) = - 0.1845522146 + 0.1375406695 I,
      (6, 1) = - 0.05738980624 + 0.2016341603 I,
      (6, 2) = 0.2016341603 - 0.05738980624 I,
      (6, 3) = 0.180637307 - 0.2202880929 I,
      (6, 4) = - 0.2202880929 + 0.180637307 I,
      (6, 5) = - 0.319636107 + 0.0272908651 I,
      (6, 6) = 0.0272908651 - 0.319636107 I
    )


    [4.650378999e-13, 4.65037861e-13, 5.072741357e-13, 5.072741409e-13,

    3.127416202e-13, 3.12741677e-13]
                                        ]

>> B:=array(1..6,[1 $ 6]);

                                +-                -+
                                | 1, 1, 1, 1, 1, 1 |
                                +-                -+

>> numeric::matlinsolve(m6,B);

                          -- +-              -+  +-   -+ --
                          |  |  1.699584961   |  |  0  |  |
                          |  |                |  |     |  |
                          |  |  1.213989258   |  |  0  |  |
                          |  |                |  |     |  |
                          |  | 0.9997558594   |  |  0  |  |
                          |  |                |, |     |  |
                          |  | 0.8459472656   |  |  0  |  |
                          |  |                |  |     |  |
                          |  | 0.7049560547   |  |  0  |  |
                          |  |                |  |     |  |
                          |  | 0.5357666016   |  |  0  |  |
                          -- +-              -+  +-   -+ --
```

### 10.3 Solution of an algebraic equation, or system of equations

There are a number of packages available, each of which is applicable under different circumstances. We list the packages with which you should be familiar, and give a brief description and an example of its use.

- numeric::linsolve is used to solve a linear system of equations. Of course, its internal functioning is very similar to that of numeric::matlinsolve, but the input and output formats are different

```
>> numeric::linsolve([x + y = 1, x + y = 2], [x, y]);

                                        FAIL

>> n := 100: x[0] := 0: x[n + 1] := 0:
eqs := [x[i-1] - 2*x[i] + x[i+1] = 1 $ i = 1..n]:


vars := [x[i] $ i = 1..n]:
numeric::linsolve(eqs, vars);

[x[1] = -50.0, x[2] = -99.0, x[3] = -147.0,

....................

   x[98] = -147.0, x[99] = -99.0, x[100] = -50.0]
```

- numeric::polyroots and numeric::polysysroots find the roots of a polynomial, or system of polynomials, respectively. The application is straightforward, as indicated in the following examples

```
>> numeric::polyroots(x^3 - 3*x - sqrt(2)=0);

                [-1.414213562, -0.5176380902, 1.931851653]

>> numeric::polyroots(x^5 - x^2=0);

  [- 0.5 - 0.8660254038 I, - 0.5 + 0.8660254038 I, 0.0, 0.0,   1.0]
```

```
>> numeric::polysysroots({x^2 + y^2 - 1, x^2 - y^2 = 1/2}, [x, y]);

{[x = -0.8660254038, y = -0.5], [x = -0.8660254038, y = 0.5],

   [x = 0.8660254038, y = -0.5], [x = 0.8660254038, y = 0.5]}
```

- numeric::fsolve is a general purpose equation solver, for one equation or for a system of equations. It can take as input any nonlinear equation, although MuPAD must be able to find a symbolic derivative. However, the generality of fsolve has certain consequences: it performs best with an initial estimate of the solution, and returns only one root. Thus, if the equations are linear or polynomial, the packages described above should rather be used. If the initial estimate is real, then fsolve looks for roots on the real line, but if the inital estimate is complex, it looks for roots over the whole complex plane.

```
>> numeric::fsolve(sin(x) = 0, x);

                              [x = -226.1946711]

>> numeric::fsolve(sin(x), x = 3);

                               [x = 3.141592653]

>> numeric::fsolve(sin(x), x = -4..-3);

                              [x = -3.141592654]

>> numeric::fsolve(sin(x), x = -4..-3.5);

                                     FAIL

>> numeric::fsolve([x^2 = exp(x*y), x^2 = y^2],
                 [x = 0..infinity, y]);

                   [x = 0.7530891649, y = -0.753089165]

>> numeric::fsolve(sin(x) + cos(x)^2 = 3, x);

                                     FAIL
```

```
  >> numeric::fsolve(sin(x) + cos(x)^2 = 3, x = I);

                           [x = 0.2972513613 + 1.128383965 I]
```

### 10.4 Definite integrals

There are two packages available, namely numeric::int and numeric::quadrature, and you can also use float(int(f(x),x = a..b)); as discussed in the online help, the effect of all three options is more or less the same.

```
>> numeric::quadrature(exp(x^2), x=-1..1);

                              2.925303492

>> numeric::quadrature(exp(-x^2), x=-2..infinity);

                              1.768308316

>> numeric::quadrature(numeric::quadrature(exp(x*y), x=0..y), y=0..1);

                             0.6589510757
```

The following examples demonstrate some of the limitations of numerical quadrature. In the first case, an analytic solution exists, but the package does not give an accurate answer (and says so). In the second case, analytically the integral is divergent

```
>> numeric::quadrature(x^(-9/10), x=0..1);
Warning: Precision goal not achieved after 10000 function calls!
Increase MaxCalls and try again for a more accurate result.
[numeric::quad\
rature]
                              9.998221196

>> int(x^(-9/10),x=0..1);
                                  10

>> numeric::quadrature(x^-1, x=0..1);
Warning: Precision goal not achieved after 10000 function calls!
```

```
Increase  MaxCalls  and  try  again  for  a  more  accurate  result.
[numeric::quad\
rature]
                              86.3557155
```

**10.5 Ordinary differential equations**

The numerical solution of ordinary differential equations is a huge subject. In these notes we will just describe the use of the method *numeric::odesolve* with the optional parameters taking the default values. First, we show how numeric::odesolve is used with a single first-order differential equation; in the example y'(t)=t sin(y), the intial condition is y(0)=2, and we want to find y(10):

```
>> f := proc(t, Y) begin [t*sin(Y[1])] end_proc: Y0:=[2]:

>> numeric::odesolve(0..10, f, Y0);

                            [3.141592654]
```

The domain of integration is the fist argument in numeric::odesolve; it is expressed in the form a..b with a the initial value of the independent variable (here, t) and b the final value. The second argument in numeric::odesolve is a procedure that returns the right hand side of the differential equation. The third argument is the initial value of the dependent variable (here, y), which we have named Y0 (although any name could be used). Note that both Y0 and Y are vectors (to allow for generalizations to a system of ordinary differential equations), and the syntax has to be as used in the above example.

Next, we look at the first order system of differential equations, x' = x + y, y' = x - y, with x(0) = 1, y(0) = I (square root of -1); we use numeric::odesolve to find Y(2) = [x(2), y(2)].

```
>> f := (t, Y) -> [Y[1] + Y[2], Y[1] - Y[2]]: Y0 := [1, I]:

>> numeric::odesolve(0..2, f, Y0);

     [14.44977942 + 5.960812207 I, 5.960812207 + 2.528155005 I]
```

Note carefully the syntax used above, which can easily be generalized to a system of n differential equations.

A second-order (or higher) differential equation cannot be solved directly by numeric::odesolve. However, it can be solved by first transforming it into a system of first-order differential equations (which can always be done). The process is illustrated in the following example

- Given y"(x) = a^2 y(x) with y(0) = 1, y'(0) = -a (and take a = 1.2): Find y(10)
- Let Y[1] = y, Y[2] = y'; then the problem becomes
- Y[1]' = Y[2], Y[2]' = a^2 Y[1] with Y[1](0) = 1, Y[2](0) = -a: Find Y[1](10)

```
>> a:= 1.2: f := (x, Y) -> [Y[2], a^2*Y[1]]: Y0 := [1.0, -a]:

>> numeric::odesolve(0..10, f, Y0);

                    [0.000006144212335, -0.000007373054847]
```

The analytic solution to this problem is y(10) = exp(-12), and so there is no error in the solution found above. However, at x = 100, the analytic solution is exp(-120); but the system finds

```
>> numeric::odesolve(0..100, f, Y0);

                      [-4.728457546e32, -5.674149055e32]
```

This is just one example where a numerical method gives an answer that is completely wrong, and further the system has not detected that there may be a significant error.

### 10.6 Exercises

1. Define the n X n square matrix $A_n$ by the condition that the element in row i and column j is $2/(1 + (i-j)^2)$. Find the inverse, eigenvalues and eigenvectors of $A_9$. Also, if the vector b = (1,2,3, ...,9) solve $A_9$ x = b for x.

2. Solve the following system of equations

- $x(i-1) - (2- h^2)x(i) +x(i+1) = 0$, with i = 2, ..,49, and with x(1) = 1 and x(50) - x(49) = h, where h = 0.1

3. Find all roots of the following equations

- $x^4 - 5x^2 + 6x = 2$
- 2cos(x) + exp(x) = 2
- $x^2y^2 = 0$, x - y = 1
- sin(x+y) - exp(x)y = 0, $x^2 - y = 2$

4. Obtain a numerical estimate for each of the following definite integrals (given in the form: expression to be integrated, integration limits)

- $\exp(-x^3)$, $x = 0..1$
- $1/(1 + x^2)$, $x = 0..\infty$
- $1/\sin(x)^{(1/2)}$, $x = 0..1$

5. Obtain a numerical solution to each of the following initial value problems for an ordinary differential equation. In each case, also try to obtain an analytic solution and compare the values of the numerical and analytic solutions

- $y''(x) - 2\,y(x) = 0$, $y(2) = 1/2$, $y'(2) = -1/2$; find $y(1000)$
- $y'(t)(1+t) + y(t)\cos(t) = 1$, $y(0) = 2$; find $y(10)$

# STUDY UNIT 11

## 11.1 Introduction

This study unit is concerned with the use of MuPAD's graphical packages, and in particular with graphs in 2-dimensions. 3-dimensional graphs will be the subject of study unit 12. As before these notes are not intended to be complete, but are rather a guide as to which topics you should be consulting in MuPAD's online help facility.

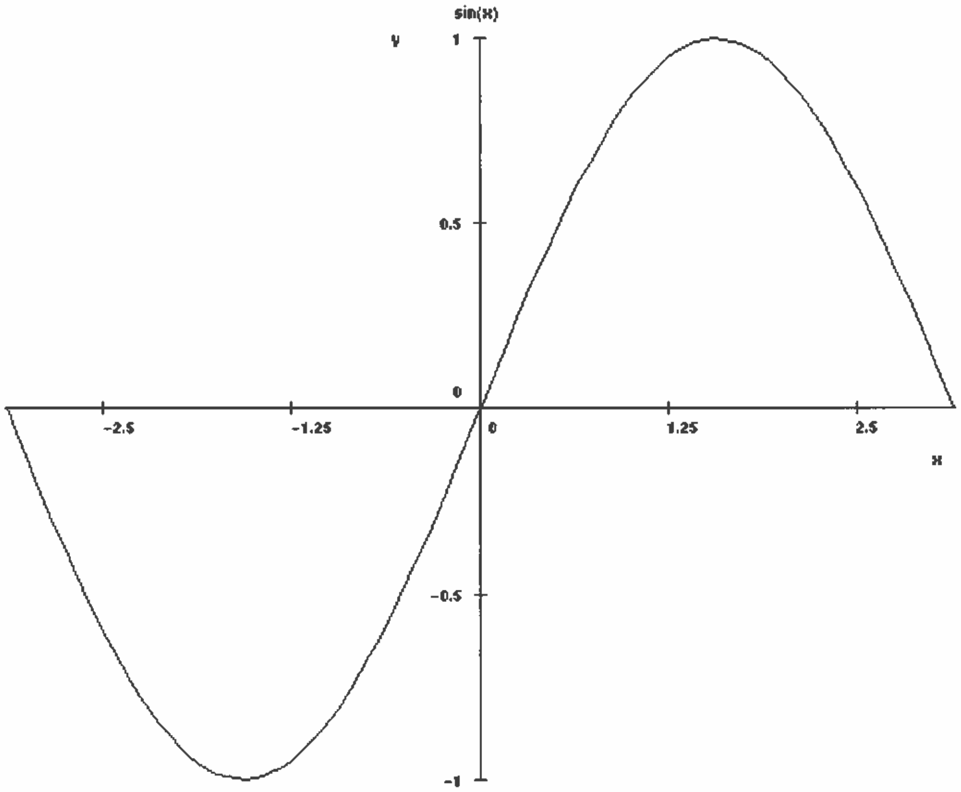
**N.B.** The printed version of these notes is in black and white, but the graphical images have been produced in colour. In order fully to appreciate this study unit, you should also download it (from http://einstein.unisa.ac.za/SU11.html) and view it on your computer screen.

## 11.2 Output

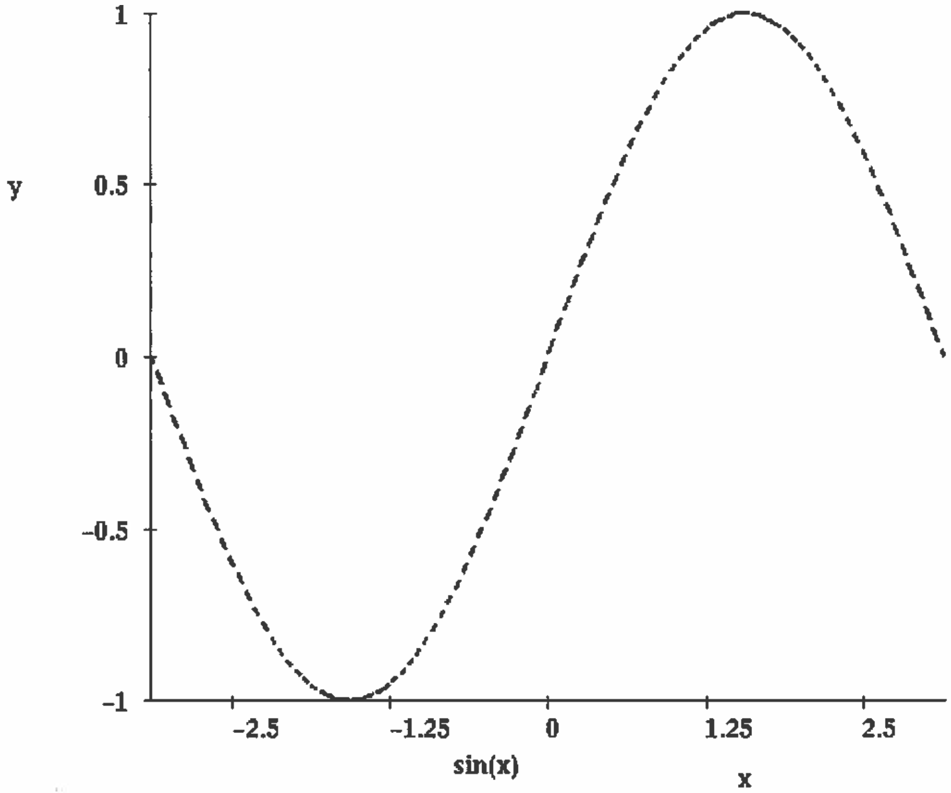
We start with a very simple example, that uses none of the optional features available, and instead plots a graph of sin(x), between -PI and PI using the default settings:

```
>> plotfunc2d(sin(x), x = -PI..PI):
```

This command produces the following graph



The graphical viewer supplied by MuPAD, Vcam, enables you to manipulate this graph in all sorts of ways. In the menus, object refers to the graph and scene refers to the axes etc. In the Linux version of MuPAD (which is the version on the server einstein), you can use the menus to change the size, colour, line style and thickness, font, etc., of both scene and graph. However, the fuctionality of MuPAD Light is more limited; in this case many of the options have to be set via MuPAD commands, and the syntax for doing this will be discussed later. Using the Vcam menus on einstein, the above graph was transformed to



You can use the Vcam menus to print an image, and to save an image to file (in a format that is readable only by Vcam). If you are using the Linux version of MuPAD, you can also export the image (export is an option under the file button on the Vcam viewer), which means that you can save it into a standard graphical format (.jpeg, .ps, etc.) which can be read by other programs; this option is not available in MuPAD light.

### 11.3 Creating a plot of a given function

Usually, the easiest way to plot a function of one variable is to use the command *plotfunc2d.* Assuming that the default settings are acceptable, the syntax for using plotfunc2d is straightforward

```
>> plotfunc2d(f,x=a..b);
```

where f is a MuPAD object that evaluates to a real number when x is given a real value, and the interval a..b is real, with a < b. You can use func2d to make two (or more) plots on the same axes simultaneously, for example

```
>> plotfunc2d(sin(x), cos(x), x = -PI..PI):
```

The default is that the vertical axis is called the y-axis and is scaled so that the maximum value of the function reaches the top of the axis. However, the user can also specify the name and range of the vertical axis, for example

```
>> plotfunc2d(sin(x), cos(x), x = -PI..PI, w = -2..2):
```

The system uses a default of 100 grid points to plot a graph, and draws straight lines between the plotted points. If desired, the user can change this default, e.g.,

```
>> plotfunc2d(sin(x), cos(x), x = -PI..PI, w = -2..2, Grid = 3):
```

(which produces graphs that looks nothing like sine and cosine curves). In addition, the user may specify a wide variety of Scen Options, as discussed in the next section.

### 11.4 Scene Options

A complete list of available parameters, and of the values they may be assigned as well as the default values, may be found in the online help under *plotOptions2d*. For example, suppose we would like to plot exp(x) with arrows on our axes, and with linear scaling on the horizontal axis and logarithmic scaling on the vertical axis

```
>> plotfunc2d(Arrows=TRUE,AxesScaling=[Lin,Log],exp(x), x = -PI..PI):
```

Some of the more commonly used SceneOptions with which you should be familiar are

- Arrows
- Axes
- AxesOrigin
- AxesScaling
- FontSize
- GridLines
- Labeling
- Labels
- LineStyle
- PointStyle
- Scaling
- Title

- TitlePosition

### 11.5 Parametric and other plots

The command plotfunc2d is very convenient to use for plotting a single-valued function. However, there are cases when something more general is required, for example if we want to plot a circle or an ellipse, or to show a set of data points on a graph. In these cases we can use *plot2d*. The syntax of plot2d is

```
>> plot2d( <SceneOption1, SceneOption2, ...,> object1 <,object2, ...>):
```

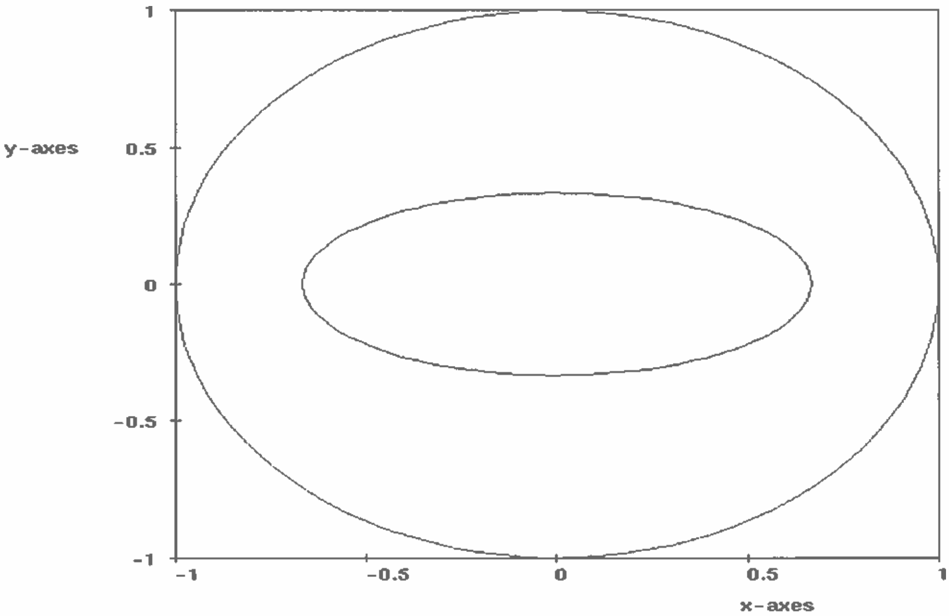
where the optional arguments SceneOption have been described above, and where object can be either

- Parametric curve with object of the form
  object:=[Mode=Curve, [x(u),y(u)], u=[umin, umax] <, Options>]
  where the parameter u lies in the range umin to umax, and where the parametric functions x(u) and y(u) are specified by the user. The available options are discussed in the online help (and the default settings may well be suitable).

- List of graphical primitives - we will not consider this case further in study unit 12.

As an example, we draw a circle (coloured magenta) with an ellipse (coloured blue) inside the circle. We use the option Scaling=Constrained so that the vertical and horizontal axis scalings are the same, and a circle appears circular. The code and graph follow.

```
>>    circle:=[Mode    =    Curve,    [cos(u),    sin(u)],    u    =    [0,
2*PI],Color=[Flat,RGB::Magenta]]:
```



```
>> ellipse:=[Mode = Curve, [2/3*cos(u), sin(u)/3], u = [0,
2*PI],Color=[Flat,RGB::Blue]]:
>> plot2d(Scaling = Constrained, FontSize=12, circle, ellipse);
```

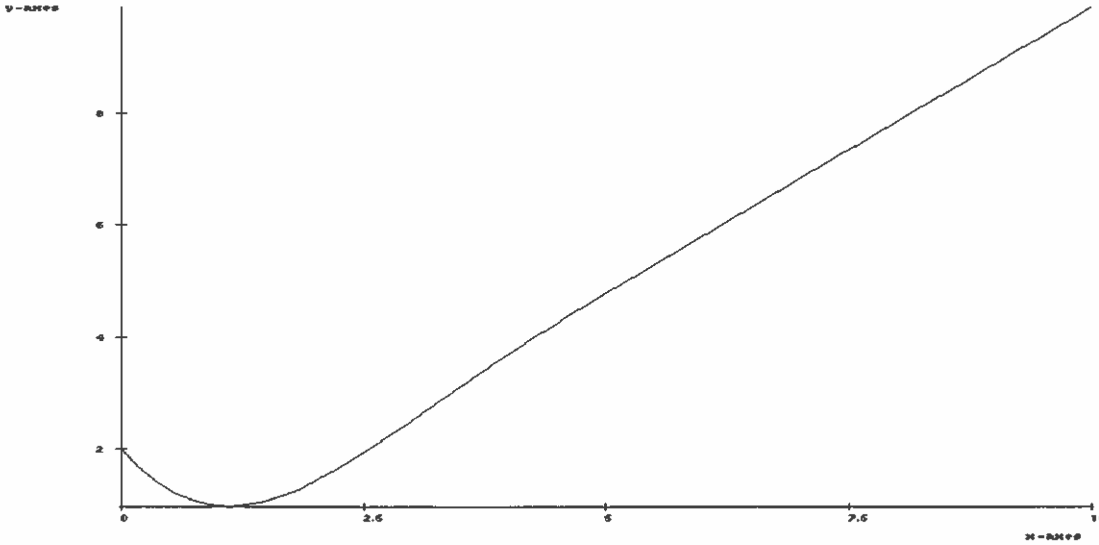
### 11.6 Other 2D plots

MuPAD has a library of plotting routines, and you can see a list of what is available by entering info(plot); and then use the online help to obtain information about any chosen routine. Here we will briefly discuss three of the routines, namely *plot:ode, plot::vectorfield* and *plot::implicit.*

The routine plot::ode uses internally numeric::odesolve, and you should be familiar with that package before trying to use plot::ode. We illustrate the use of plot::ode in its simplest form, i.e. with no options used

```
>> f := (t, Y) -> [t*Y[1] - Y[1]^2]:
Y0 := [2]:
G := (t, Y) -> [t, Y[1]]:
p := plot::ode([i $ i = 0..10], f, Y0,[G]):
plot(p);
```

The function f represents the right hand side of the differential equation, as in numeric::odesolve. The list [i $ i = 0..10] is the list of values on the horizontal axis (often x-values, but in this case t-values) for which the equation is to be solved numerically; the first value in the list is value of t at which Y[1] takes the value Y0. The function G provides a list of points to be plotted; in a logical sense, it is unnecessary in this example, but it would be needed in a 2nd order differential equation, expressed as a system of first-order equations, to tell the system whether to plot Y[1] or Y[2] (or perhaps some combination of these) against t. As discussed in the online help, there are a number of options that can be included with G. Probably the most useful is Style = Splines, which draws a smooth curve between the plotted points (the default is that they are joined by straight lines). Using this option, the above code becomes, with the output shown

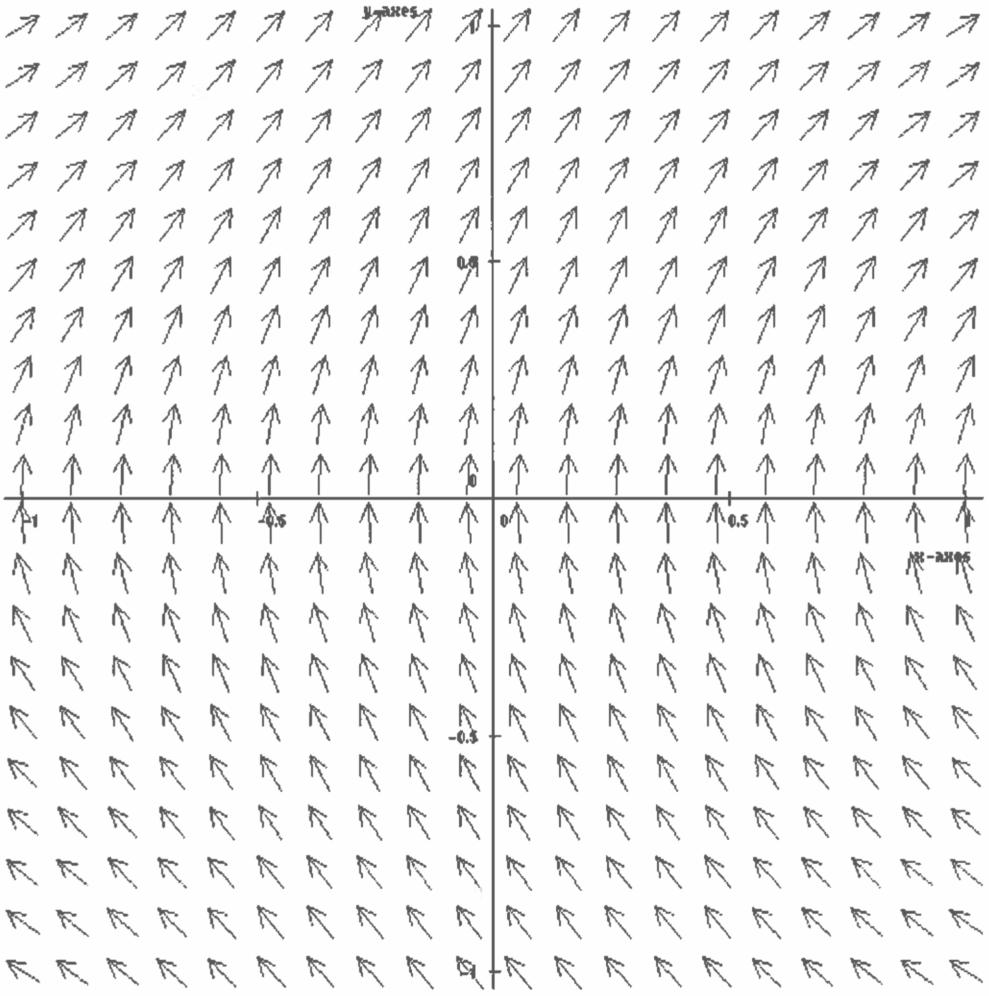


```
>> f := (t, Y) -> [t*Y[1] - Y[1]^2]:
Y0 := [2]:
G := (t, Y) -> [t, Y[1]]:
p := plot::ode([i $ i = 0..10], f, Y0,[G, Style = Splines]):
plot(p);
```

**If** you are using the Linux version of Mupad, the easiest way to change the scene (e.g., to change x-axes to t) is to use the Vcam display and edit the scene. With MuPAD light, you should read the online help to see how to pass the scene information to the system.

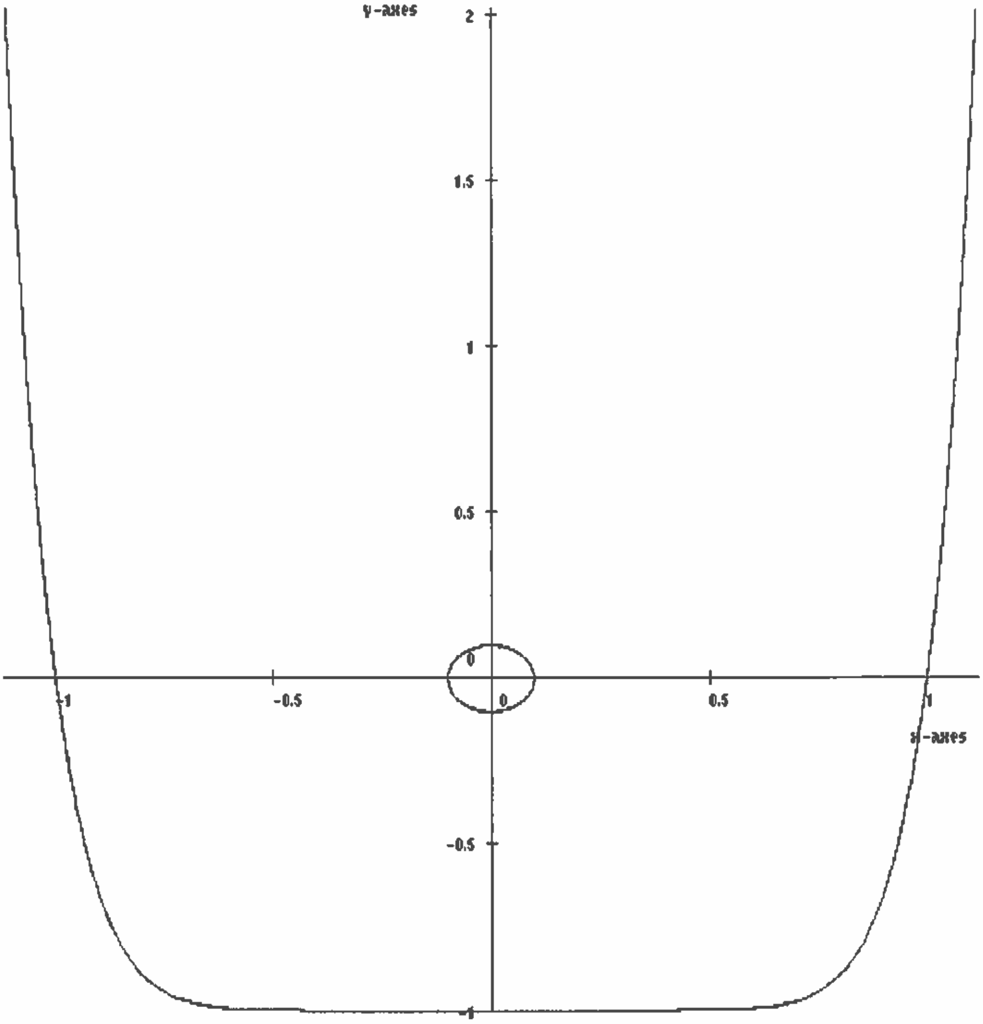
The routine plot::vectorfield requires as input the vectorfield and the range of the horizontal and vertical coordinates; other optional inputs are possible, as discussed in the online help. The use of the routine is straightforward, as illustrated in the following

```
>> p:=plot::vectorfield([sin(y),cos(x)],x=-1..1,y=-1..1):
>> plot(p):
```



The routine plot::implicit is used to plot f=0 for a functuion f from the real plane to the real line. The syntax and use of this routine are straightforward, as illustrated in the following example (for a rather complicated function)

```
>> p:=plot::implicit((1-0.99*exp(x^2+y^2))*(x^10-1-y),
x=-1.25..1.25,y=-1.1..2):
>> plot(p):
```

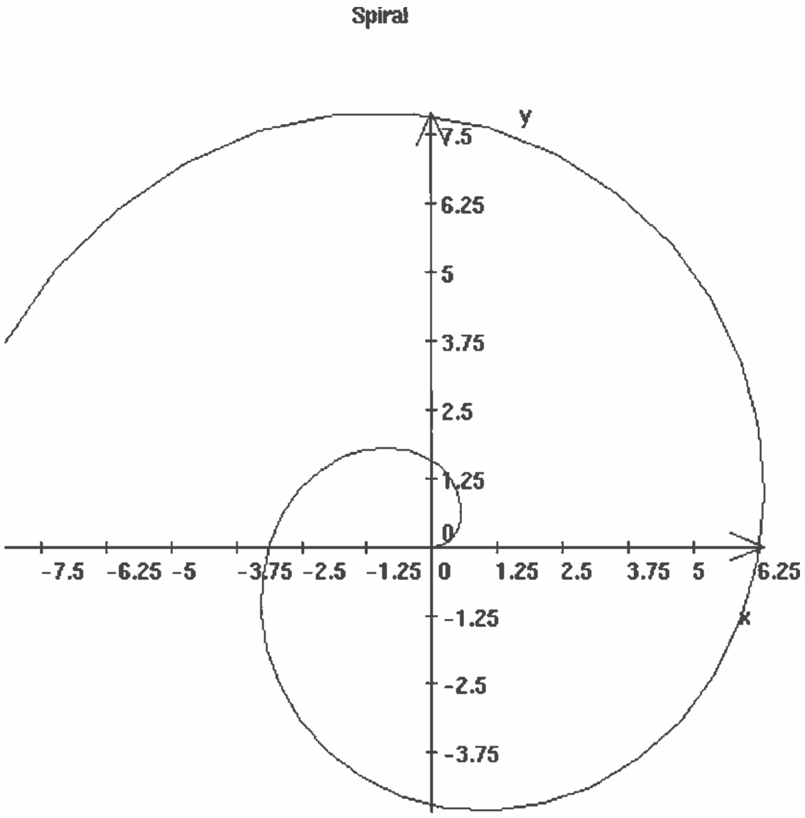


## 11.7 Exercises

**N.B.** Throughout these exercises, you are expected to set the various plotting options by means of MuPAD code, and not by using the Vcam edit menus.

1. Write MuPAD code to draw a graph that plots the functions f(t) = exp(sin(t)) and g(t) = $t^2/(1+t^2)$ in the range t = -1 to 5. The axes ahould be appropriately labelled, and the scale in the vertical and horizontal directions should be the same. The graph of f is to be blue and that of g green.

2. Write MuPAD code to plot a spiral (u cos(u), u sin(u)) coloured blue so as to produce the result given below



3. Write MuPAD code to produce a graph of two circles, radii 2 and 3, with the centre of the smaller circle at (x = -2, y = 1) and the centre of the larger circle at (x = 3, y = 1). The axes should be labelled appropriately, and there should be suitable titles for each circle, and for the scene. The colour of the smaller circle should be blue and that of the larger circle should be red.

4. Produce a plot demonstrating the behaviour of y(x) in the range x= 1 to 50 when y(x) satisfies the differential equation $x^2$ y"(x) +x y'(x) +$x^2$ y(x) = 0, with y(1) = 1 and y'(1) = 0.

5. Plot the vectorfield (1, cos(2x) ) in the range $0 < x < 5$, $0 < y < 4$, and with the same scale on the x and y axes. Also plot, on the same axes, the curves $y = \sin(2\ x) + C$, for C=0, 1 and 2.

6. Plot contours of the function $f(u,v) = u^2 - 3v$ at f = 0, 1, 2, 3 and 4. The axes, contours and graph should be appropriately labelled

# STUDY UNIT 12

## 12.1 Introduction

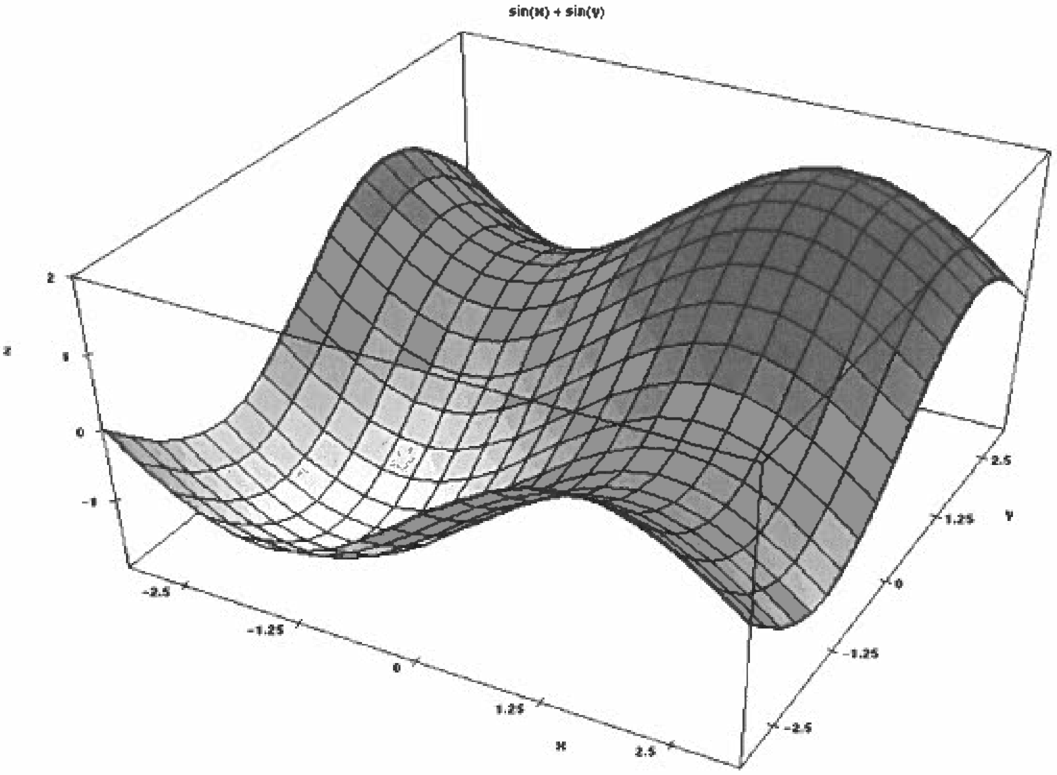
In this study unit we continue, and complete, our investigation of MuPAD's graphic facililites. In particular, we will look at the representation of functions in 3-dimensions, as well as at the creation of graphs from primitives such as line, point and polygon.

**N.B.** The printed version of these notes is in black and white, but the graphical images have been produced in colour. In order fully to appreciate this study unit, you should also download it (from http://einstein.unisa.ac.za/SU12.html) and view it on your computer screen.

## 12.2 Plots of functions in 3-dimensions

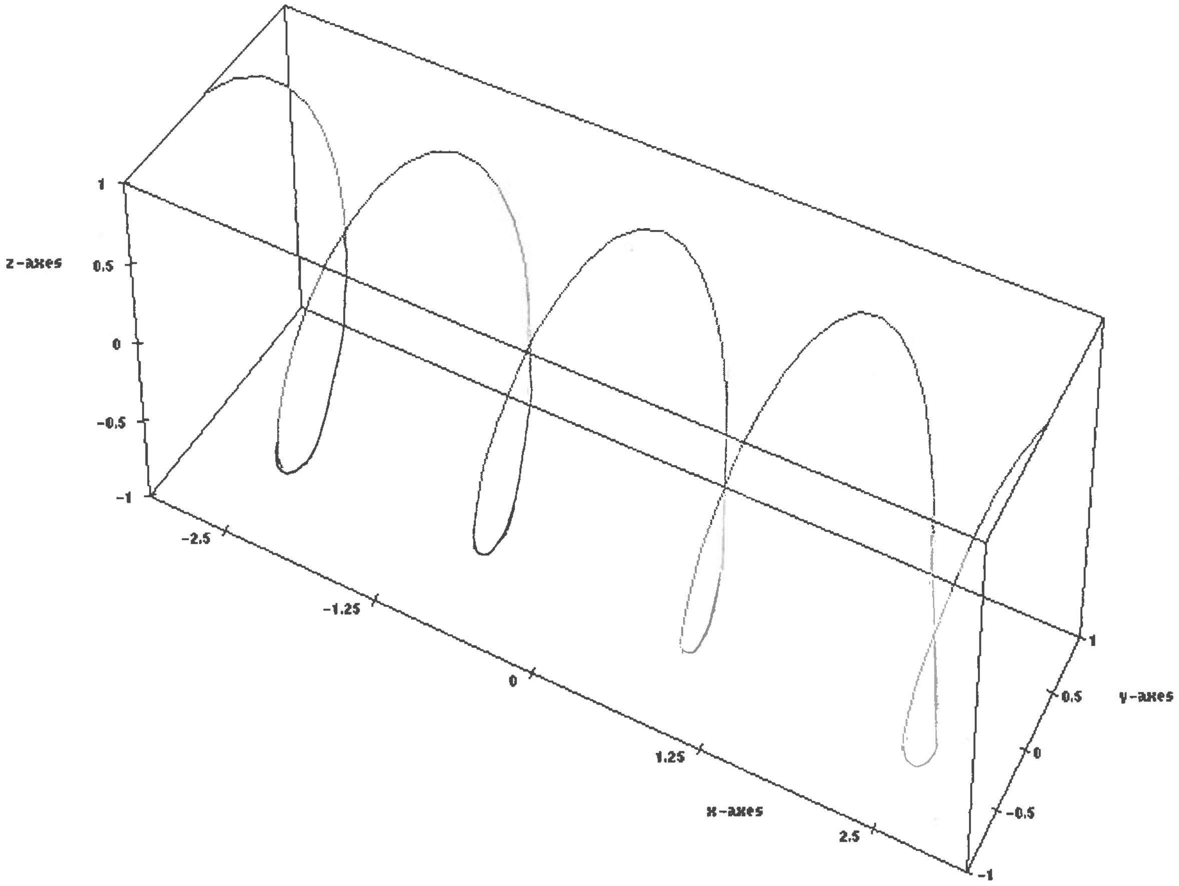
The syntax for plotting functions in 3-D is quite similar to the syntax in the 2-D case, at least if the various default options are suitable. For example, the following code, using *plotfunc3d*, plots the function sin(x) + sin(y) in the range x, y=-PI..PI

```
>> plotfunc3d(sin(x)+sin(y),x=-PI..PI,y=-PI..PI);
```



Similarly to the 2-D case, parametric plots are constructed using *plot3d.* We can construct a curve or surface (or a List, but that will be discussed later in this study unit), as in the following examples

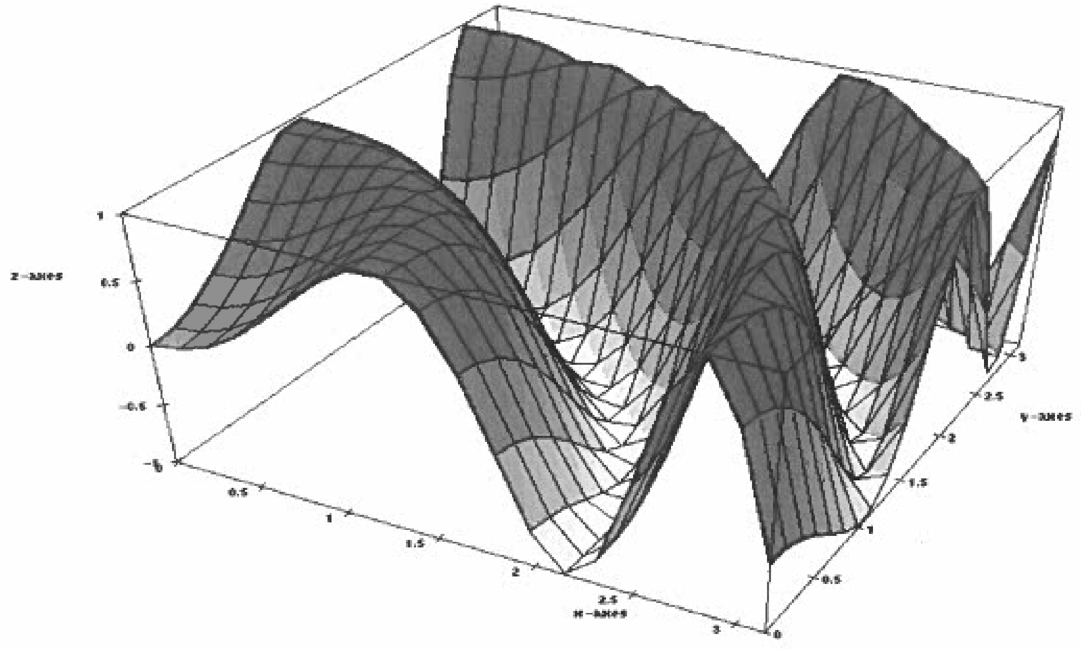
- We plot the curve x = u, y = sin(4*u), z = cos(4*u), with the parameter u in the range -PI to PI.



```
>> plot3d([Mode = Curve, [u,sin(4*u), cos(4*u)], u = [-PI, PI]]):
```

- We plot the parametric surface x = u, y = v, z = sin(u^2 + v^2), with both parameters u, v in the range 0..PI.

```
>> plot3d([Mode = Surface,[u, v, sin(u^2 + v^2)], u = [0, PI], v
= [0, PI]]):
```



### 12.3 Plotting options in 3-dimensions

The user has considerable flexibity in controlling the appearance of a 3-D graph. Full details are given in the online help under *plotOptions3d*, and here we describe only some of the most commonly used features.

You may use the Vcam viewer to adjust the view of the graph: zoom in or out, rotate the graph, etc. With the full version of MuPAD (i.e. MuPAD Professional or Linux), you can also edit the scene, or graphical object, using the menus provided (under Edit).

The alternative is to specify options using the following syntax, illustrated for the examples given above involving plotfunc3d and plot3d

```
>> plot3d(<SceneOption, ..., > [Mode = Surface, [u, v, sin(u^2 + v^2)],
u = [0, PI], v = [0, PI] <,SurfaceOption, ...> ]):
```

```
>> plot3d(<SceneOption, ..., > [Mode = Curve, [u,sin(4*u), cos(4*u)],
u = [-PI, PI] <,CurveOption, ...> ]):
```

```
>> plotfunc3d(<SceneOption, ..., > sin(x)+sin(y),x=-PI..PI,y=-PI..PI
<,Grid = [nx, ny]> ):
```

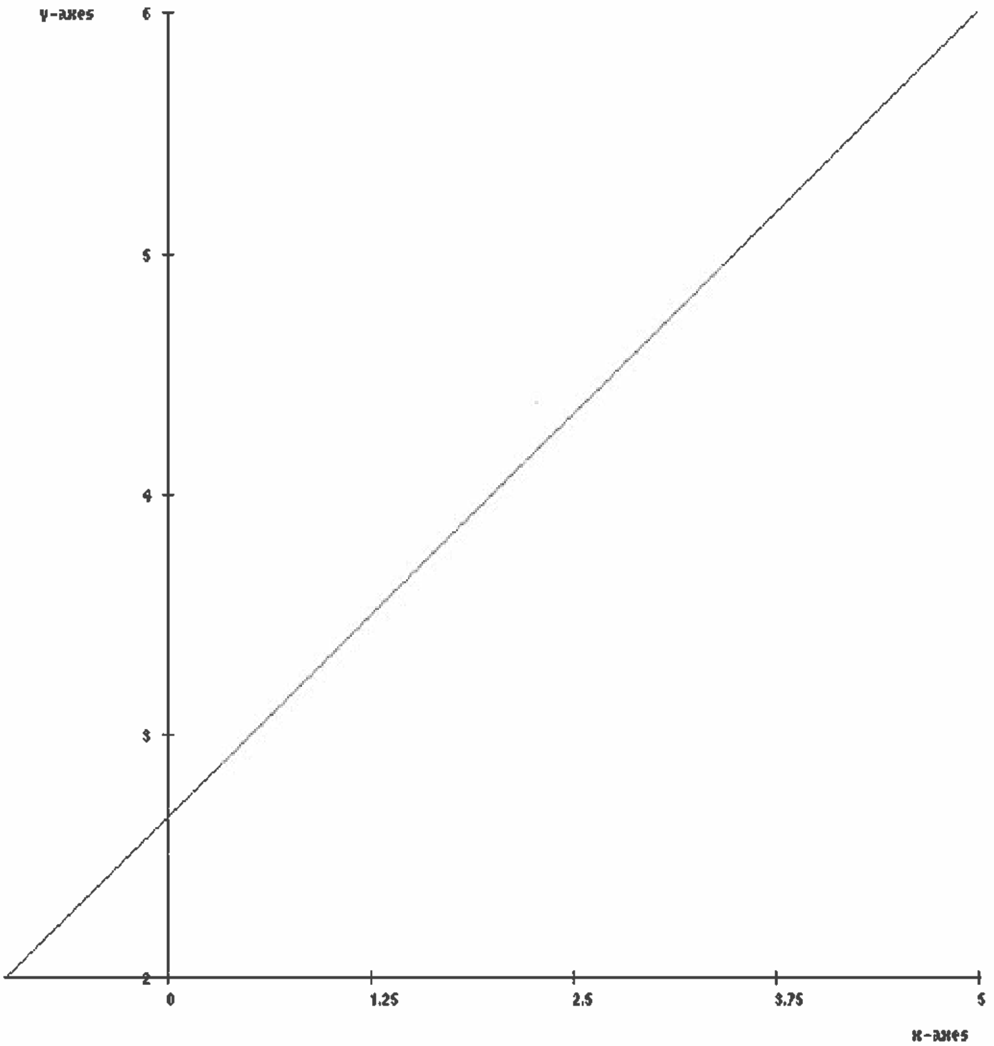
where the various SurfaceOptions and CurveOptions are described in the online help under ?plot3d; and where the meaning of Grid = [nx, ny] is described under ?plotfunc3d. The various SceneOptions, the values that can be used as well as the default value, are discussed in the online help under ?plotOptions3d. In most cases, the options are the same as, or a natural generalization of, what is available for 2-D plots.

### 12.4 Customized graphics

The contents of the MuPAD plotting library are found under info(plot). In general, these routines create a graphical primitive (which should be named), and the plot is then effected by the command plot(name). For example,

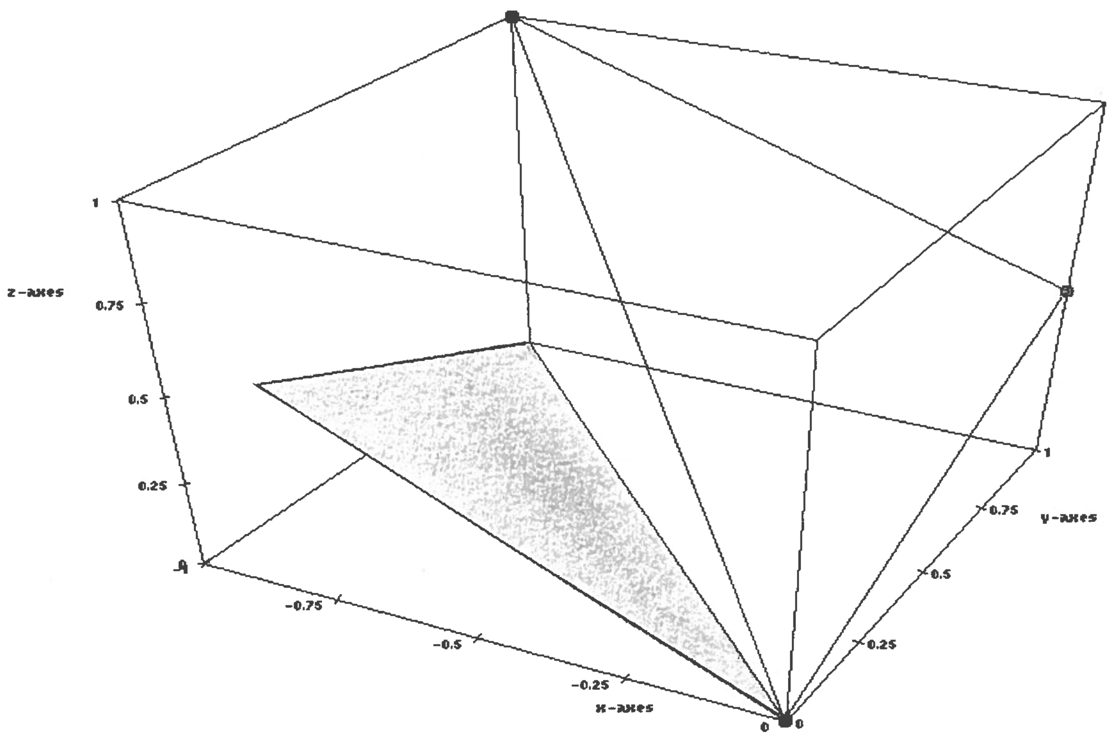
```
>> p:=plot::line([-1,2],[5,6]):
>> plot(p):
```

plots a straight line between the points defined above. The graphical primitives defined in the plot library that will be discussed here are *plot::Point*, *plot::line* and *plot::Polygon.* With these primitives, you can create points and lines, of various sizes and colours, as well as closed polygons that can be shaded in various ways. Details are given in the online help. The command *plot* then takes as input graphical primitive(s), together with scene options, and causes the graph to be plotted.

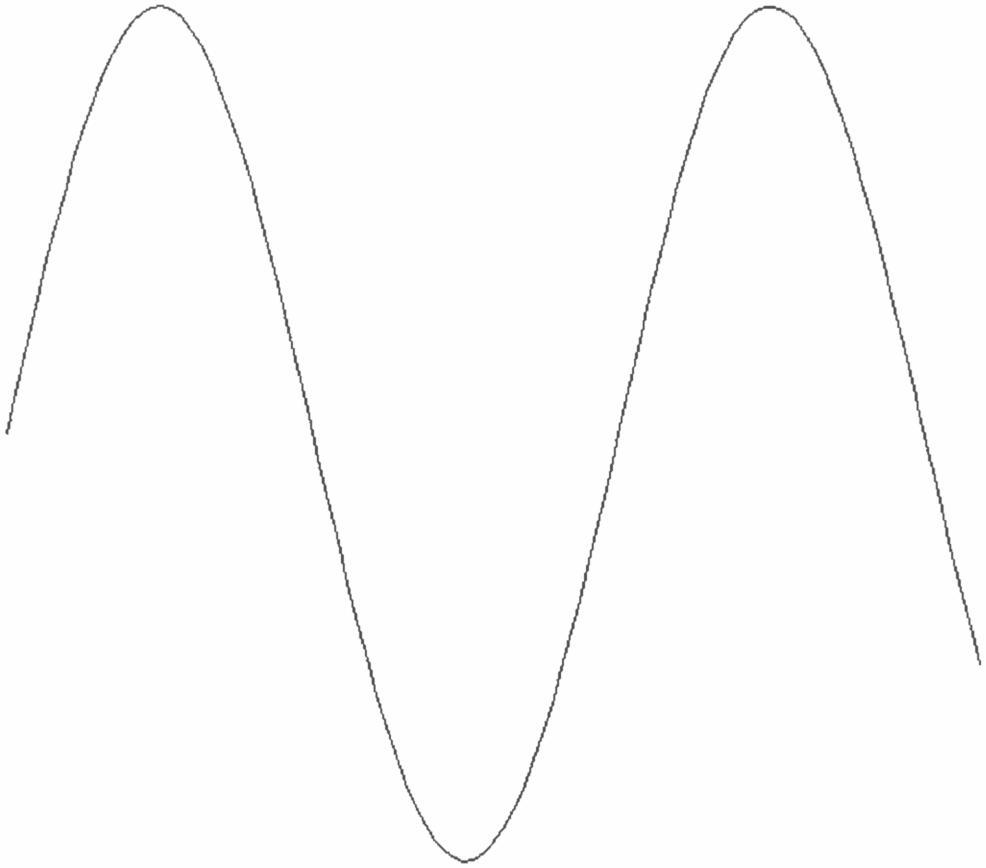


Alternatively, you may use plot2d or plot3d with Mode = List and provide a list of graphical primitives (with syntax as given in the online help, and **note** that in this case you should use *point* and *polygon* instead of plot::Point and plot::Polygon). These options are illustrated in the examples that follow.

```
>> p1 := point(0, 0, 0, Color = RGB::Red):
p2 := point(0, 1, 1/2, Color = RGB::Green):
p3 := point(-1, 1, 1, Color = RGB::Blue):
triangle1 := polygon(point(0, 0, 0), point(0, 1, 1/2),
point(-1, 1, 1), Closed = TRUE,
Color = RGB::Black):
triangle2 := polygon(point(0, 0, 0), point(-1, 0.2, 0.4),
point(-1, 1, 0), Closed = TRUE,
Filled = TRUE, Color = RGB::Antique):
>> object := [Mode = List, [p1, p2, p3, triangle1, triangle2]]:
>> plot3d(BackGround = RGB::White, ForeGround = RGB::Black,
PointWidth = 70, PointStyle = FilledCircles,
Axes = Box, object):
```



```
>> p:=plot::Polygon(plot::Point(i/10,sin(i/10)) $i=0..100,Color=RGB::Blue):
>> plot(p,Axes=None):
```



## 12.5 Exercises

**N.B.** Throughout these exercises, you are expected to set the various plotting options by means of MuPAD code, and not by using the Vcam edit menus.

1. Construct a scale diagram of the orbits of the inner planets of the solar system, using the following data

| Planet | Radius of orbit (AU) | Eccentricity of orbit |
|---|---|---|
| Mercury | 0.39 | 0.21 |
| Venus | 0.72 | 0.007 |
| Earth | 1.0 | 0.02 |
| Mars | 1.52 | 0.09 |

where 1AU is the radius of the Earth's orbit, and all orbits are elliptical with the eccentricity having the usual meaning.

2. Using smooth functions, construct a 3-D graph that looks like a wide-brimmed hat.

3. Find the eigenvalues and eigenvectors of the matrix

5 2 1
2 3 2
1 2 6

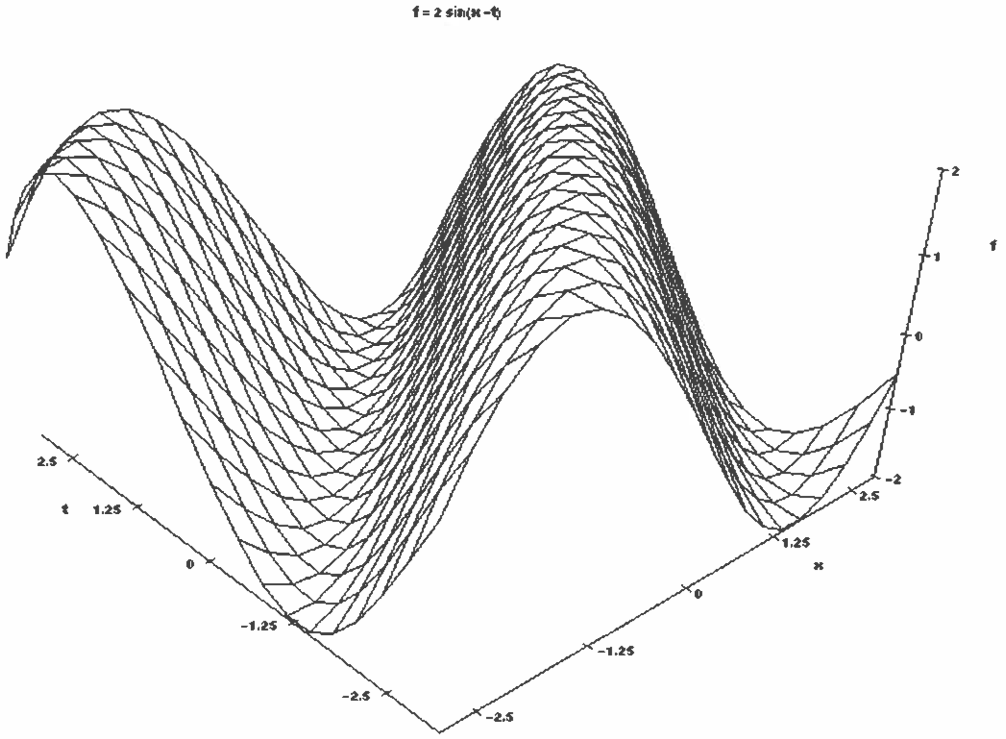
Draw a 3-D graph showing the eigenvectors, with each eigenvector having the magnitude of its corresponding eigenvalue. Each axis should have the same scale.

4. Plot the following on a 3-D graph

- Sphere, centred at the origin and radius 2
- Surface z = x - y +1 +x y/5
- Cylindrical surface $(x-10)^2 + y^2 = 81$, z any value

Estimate the points at which all three surfaces intersect, and then use this information as starting point in a numerical procedure to find accurately (to 10 significant figures) the coordinates of the points at which the surfaces intersect.

5. The following diagram is a plot of f = 2 sin(x - t). Write MuPAD code that reproduces the diagram (The graphical object is uniformly coloured blue)



# STUDY UNIT 13

## 13.1 Introduction

This study unit, together with study units 14 and 15, is concerned with the scientific word processing system LaTeX, which has become the standard system in mathematics and related disciplines. Here, we describe the basic use and structure of LaTeX. The later study units describe its use with mathematical expressions as well as with the incorporation of graphics into a LaTeX document.

## 13.2 Getting started with LaTeX

You will need to have

- Downloaded and installed a LaTeX system (Miktex) on your own computer

We now demonstrate an example that shows how to produce a simple document in LaTeX. There are some minor differences between Windows and Linux in how this is done: the following description is for Windows, and the text in *italics* shows the Linux variations.

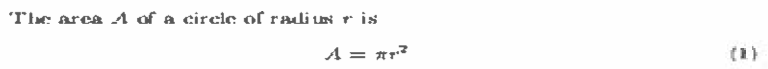
Create a file with content exactly that given below (**N.B.** this must be done with a plain text editor like edit, notepad or wordpad, and **not** by using a word processing system like MS-Word)

```
\documentclass{article}

\begin{document}

The area $A$ of a circle of radius $r$ is
\begin{equation}
A=\pi r^2
\end{equation}

\end{document}
```

Name the file test.tex (or whatever you like, provided the name ends with .tex). In a command window (MS-DOS prompt) make the current directory the one containing test.tex, and enter

```
latex test
```

The effect of the command is to create a number of new files, namely test.aux, test.dvi and test.log . The file test.log contains various messages generated by the LaTeX compiler, and you should look at it if something goes wrong The file test.aux contains information about cross-references, and usually its contents need not concern you. The file test.dvi contains the output of the LaTeX compilation, and you may view it by opening the miktex folder (under start, then programs) and click on dvi viewer, then on



The area $A$ of a circle of radius $r$ is

$$A = \pi r^2 \qquad (1)$$

open file, then find the directory containing test.tex and click on test.dvi (*Linux: enter at the command line, xdvi test*). You should then see a window containing

You can print directly from the dvi viewer window (*Linux: this is not possible*), and there are aslo other options. You can create postscript or pdf files, which are standard formats readable and printable by most computer systems. A postscript file, named test.ps, is created by the command

```
dvips test.dvi (Linux: dvips -f test.dvi > test.ps)
```

A pdf file, named test.pdf, is created directly from the file test.tex by the command

```
pdflatex test.tex
```

In a *Linux environment*, there are further options - but since they are not available to Windows users, we mention them only briefly. You can use latex to create a web document, in.html format, by the command

```
latex2html test.tex
```

This creates a directory test containing .html files as well as graphics files in .png format, with each equation or mathematical expression represented by means of a graphical file. There is also the very powerful and verstile facility convert, which can transform a .pdf or .ps file into an image file like .png, .jpeg etc. The syntax for using the command is, for example,

```
convert test.pdf jpeg:test.jpeg
```

Further details about what can be done with convert can be found by entering

```
man convert
```

and you will see that it can also make many other types of file transformation.

**13.3 Getting help with LaTeX**

The following sources are suggested

- The recommended text for this part of the module (LATEX: A document preparation system, by Leslie Lamport)
- Other books on LaTeX, found in the library at (or near) call number 686.22544
- Online help
    - Linux - enter, info latex
    - Windows - click on start, then programs, then miktex, then help, then Latex2e Reference

You should note that some books are about TeX rather than LaTeX, and that there is a difference between the two systems. LaTeX is built on top of TeX, i.e. the LaTeX system is, in a sense, a translator into TeX.

However, except for quite advanced applications that are outside the scope of this module, you would not need to know anything about the underlying TeX.

### 13.4 The structure of a LaTeX document

For the remainder of this study unit, we will look at some features of a sample LaTeX document. The sample document is su13-1.tex, which on compilation gives su13-1.pdf. Both files can be found at the end of this study unit.

- The comment symbol is % - everything to the right of % is ignored; if you want a % symbol, you must type \%
- The first statement in a LaTeX document defines the documentclass, and has the syntax

  ```
  \documentclass[options]{class}
  ```

  where class can be article, report, book, letter or slides, but we will not discuss the last three possibilities in these notes; and the options that we will consider are the font size (10pt (default) or 12pt) and the paper size (a4paper - the default is American letter, which is not an international standard).
- After the documentclass declaration, there can be various commands defining aspects of the page layout. However, with a couple of exceptions in later study units, we will not be concerned with these declarations and will use the defaults
- A LaTeX document consists of various environments defined by syntax of the form

  ```
  \begin{environment}
  ......
  \end{environment}
  ```

  In some cases, there are shortcuts that can be used to define an environment, as we shall see later. It is important that each \begin{environment} statement has a matching \end{environment} - otherwise the LaTeX compiler will complain. An environment that is found in every LaTeX file is document, and every file has a \begin{document} and ends with \end{document}.
- The next command in our document is \title{...}, and the text between the curly braces {} is the title of the document. The syntax \command{text} is commonly used in LaTeX. The title is composed of three parts, namely \title{}, \author{}, and \date{}, and then the command \maketitle (which must come after the other 3 commands). The \date command is optional, and if omitted, the system will insert the date on which the document is compiled; if there should be no entry at all here, just type \date{}, i.e. use \date but wih an empty text field. The command \\ within the author field causes a line-break - if omitted, the system would use its own algorithms for positoning the line-break, and so would break an author's name over two lines. Many LaTeX documents have a title, but its appearance is optional.
- Also optional is the abstract environment, produced by \begin{abstract} ... \end{abstract}; it is used to provide a summary of the document.
- The next command is \section{text} with text = Introduction; also notice the next statment \label{sec:intro}. The numbering of sections, subsections, equations, etc., is not done by the author, but is carried out automatically by the system. We refer to section numbers etc. using the following

syntax

```
...in Section \ref{text} ...
```

where text must match that given in the appropriate label statement.

- The command \vspace{x cm}, where x is a real number, causes the introduction of a vertical space of length x cm (other units can be used to specify the length, but we will not describe such options).

- References are cited by the command \cite{text} and then the labelling defined by text is used in the list of references, normally at the end of the document.

- The command \newpage forces a page-break.

- We can define sub-sections by the command \subsection{text} and also label it. We can also define \subsubsection{text}, but with default settings that is as far as you can go.

- The system takes no notice of single line-breaks. It uses various internal algorithms to decide where to place a line-break in the typeset text. You can ensure that a line-break is not placed at a given interword break by typing ~ between the words instead of leaving a space. A double line-break, so that there is a blank line between two pieces of text, is interpreted as a new paragraph.

- The environment defined by

  ```
  \begin{itemize} ...\end{itemize}
  ```

  causes a number of items to be displayed with leading bullets. Each item in the list is introduced by the command \item. The environment enumerate (created by \begin{enumerate} ... \end{enumerate}) is similar to itemize, except that the list of items is numbered. The environments itemize and enumerate can be nested.

- The next section is introduced by the command \section*{text} (rather than by \section{text}). The effect of the * is that the section heading does not have a number (so there is no point in having a \label{text} command), but otherwise is laid out in the same way as other section headings.

- The most useful feature of LaTeX is its ability to typeset mathematical formulas. Here we will only give a brief introduction, and the matter will be discussed in much more detail in the next study unit.

  - The syntax $text$ causes text to be printed in mathematical mode, and is used for short mathematical expressions within normal text.

  - The syntax $A^b$ causes b to be printed as a superscript to A, i.e. $A^b$

  - The syntax

    ```
    \begin{equation} expression \end{equation}
    ```

    causes expression to be printed as a displayed mathematical formula, and given an equation number; the equation can be labelled by means of a \label{text} statement just before the \end{equation} statement, and then referred to by, for example, Eq.(\ref{text}).

  - Displayed formulas, that are not numbered, are created by means of the syntax

    \begin{displaymath} expression \end{displaymath}

  - because this is so common, there is a shortcut notation, namely \[ expression \] .

- There is considerable flexibility in the appearance of the type-face being used, but here we will be concerned only with three simple variations from the default font:

  - The syntax {\it text} causes text to be printed in *italics*

  - The syntax {\bf text} causes text to be printed in **bold**

  - The syntax {\em text} causes text to be printed in a manner to emphasize it - normally this is italics, but that depends on the documentclass being used.

- The following syntax is used for the reference list

```
\begin{thebibliography}{99}
\bibitem{text - identical to that used by \cite} Reference details ...
...
\end{thebibliography}
```

  The figure 99 represents the maximum number of references that could appear - if there are 100 or more references, you would need a larger number here.

### 13.5 Exercise

1. Write LaTeX code to re–create su13-x1.pdf (see pages 121 – 122).
2. Write LaTeX code to re–create su13-x2.pdf (see pages 123).

```
%su13-1.tex
\documentclass[12pt,a4paper]{article}
\begin{document}

\title{su13-1.pdf: Cauchy boundaries in linearized gravitational theory}

\author{
         Bela Szilagyi,
      Roberto Gomez,
      Nigel T. Bishop \\
      and
         Jeffrey Winicour
      }

\maketitle

\begin{abstract}

We investigate the numerical stability of Cauchy evolution of linearized
gravitational theory in a 3-dimensional bounded domain.
\end{abstract}

\section{Introduction}
\label{sec:intro}

The computational evolution of 3-dimensional general relativistic  space-times
by means of Cauchy evolution is a potentially powerful tool to study
gravitational radiation

\vspace{2cm}
Historically, the first Cauchy codes were based upon the
Arnowitt-Deser-Misner (ADM) formulation~\cite{adm,york} of the Einstein
equations. This approach is discussed in Section \ref{sec:fda}.
Our two main
results are:
\newpage
```

```
\begin{itemize}
\item  ADM boundary
algorithms  are improperly posed. In
particular, this includes
\begin{enumerate}
\item  The metric
\item The extrinsic curvature
\end{enumerate}
\item We present a boundary evolution algorithm.
\end{itemize}
Our motivation is the difficulty experienced in
implementing CCM~\cite{vishu}.

\subsection{Finite difference algorithms}
\label{sec:fda}
We consider ADM evolution schemes. The possible finite
difference algorithms can be discussed in reference to the quadratic Eq.
(\ref{eq:wave}).


\section*{Background}

For simplicity we consider
\begin{equation}
a x^2 + b x +c =0
\label{eq:wave}
\end{equation}
which can also be written in text $a x^2 + b x +c =0$ or without an equation
number
\[a x^2 + b x +c =0\]

\section{Next}
We continue with {\bf bold}, {\it italic} and {\em emphasized} options.
```

```
\begin{thebibliography}{99}

\bibitem{adm} R. Arnowitt, S. Deser and C. Misner,
in {\it Gravitation: An Introduction to Current Research},
ed. L. Witten (1963).

\bibitem{york} J. W. York,
in  {\it Sources of Gravitational Radiation}, ed. L. L. Smarr (1979).

\bibitem{vishu} N.~T. Bishop, R. G\'omez, L. Lehner, R. Isaacson, B.
Szil\'{a}gyi
and J. Winicour, in {\it Black Holes, Gravitational Radiation and the
Universe}, eds. B. R. Iyer and B. Bhawal (1998).



\end{thebibliography}


\end{document}
```

# su13-1.pdf: Cauchy boundaries in linearized gravitational theory

Bela Szilagyi, Roberto Gomez, Nigel T. Bishop
and Jeffrey Winicour

March 12, 2002

**Abstract**

We investigate the numerical stability of Cauchy evolution of linearized gravitational theory in a 3-dimensional bounded domain.

## 1 Introduction

The computational evolution of 3-dimensional general relativistic space-times by means of Cauchy evolution is a potentially powerful tool to study gravitational radiation

Historically, the first Cauchy codes were based upon the Arnowitt-Deser-Misner (ADM) formulation [1, 2] of the Einstein equations. This approach is discussed in Section 1.1. Our two main results are:

- ADM boundary algorithms are improperly posed. In particular, this includes
  1. The metric
  2. The extrinsic curvature
- We present a boundary evolution algorithm.

Our motivation is the difficulty experienced in implementing CCM [3].

### 1.1 Finite difference algorithms

We consider ADM evolution schemes. The possible finite difference algorithms can be discussed in reference to the quadratic Eq. (1).

## Background

For simplicity we consider

$$ax^2 + bx + c = 0 \tag{1}$$

which can also be written in text $ax^2 + bx + c = 0$ or without an equation number

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

## 2 Next

We continue with **bold**, *italic* and *emphasized* options.

## References

[1] R. Arnowitt, S. Deser and C. Misner, in *Gravitation: An Introduction to Current Research*, ed. L. Witten (1963).

[2] J. W. York, in *Sources of Gravitational Radiation*, ed. L. L. Smarr (1979).

[3] N. T. Bishop, R. Gómez, L. Lehner, R. Isaacson, B. Szilágyi and J. Winicour, in *Black Holes, Gravitational Radiation and the Universe*, eds. B. R. Iyer and B. Bhawal (1998).

# su13-x1.pdf: **Third** *draft:* A massive particle in the characteristic gravity code

## 1 Introduction

The linear size of the massive particle will be such that it can be regarded as occupying just one grid-point. Thus, in order to avoid the particle from collapsing into a black hole, or from generating caustics, there is an upper limit on the mass of the particle.

As usual, we use units in which the mass of the central black hole is 1, and then the mass of the particle is $m < 1$. A precise estimate on the limitation imposed on $m$ is given in Section 4.

## 2 Notation

The notation used here will follow that in [1], and we define here only those features that do not appear in [1].

- $m$ is the mass of the particle.
- $z^i$ represents the $(r, q, p)$ coordinates of the particle on the null cone $u = u^n$.

## 3 Numerical procedure

We describe how to advance one timestep

1. Find $v^n$
2. Find $g^n$
3. Find $z^n$ from the formula
$$z^n = (v^n)^2 \tag{1}$$
4. Find $W$ from Eqn. (1).

## 4 Maximum value of the particle mass

In practice, and if the minimum value of $r$ is to be 2,

$$m < 1/10^4. \tag{2}$$

We should also consider whether the particle's extent in the $r-$direction is such that it could be forming a black hole.

## References

[1] N.T. Bishop, R. Gomez, L. Lehner, M. Maharaj, and J. Winicour, Phys. Rev. D **60**, 024005 (1999).

# su13-x2.pdf: New approach to calculating the News

N.T. Bishop

## 1 Introduction

The current approach to the news computation is described in [1], and it will be assumed that the reader is familiar with that description and notation. In summary

- We require $J = 0$ at $u = 0$
- Evolution equations for the following variables are solved
  1. (Eq. ([1]-36))
  2. (Eq. ([1]-31))
  3. (Eq. ([1]-40))
  4. $y^A$ (Eq. ([1]-39))
- Then, using the equations in [1]-Appendix, metric quantities are used to find the news.

## 2 Computer algebra calculation

Eq. ([1]-39), in the notation used here, is

$$y^A = (q, p) = 0 \tag{1}$$

Eq. (1) leads to the transformed metric satisfying the Bondi gauge condition

## References

[1] N.T. Bishop, R. Gomez, L. Lehner, M. Maharaj & J. Winicour, *Phys. Rev. D* **56** 6298-6309 (1997).

# STUDY UNIT 14

## 14.1 Introduction

This study unit continues the discussion of LaTeX. Here we describe the use of LaTeX in typesetting mathematical expressions, which is where LaTeX demonstrates its superiority over other wordprocessing systems.

The material in this study unit will be presented with reference to a sample document su14-1.tex, which on compilation gives su14-1.pdf. The .tex document is composed of numbered points (shown as %1, %2, etc.) and these same numbers appear in the LaTeX output, as well as in the discussion below. Both files can be found at the end of this study unit.

## 14.2 Mathematical expressions in LaTeX

N.B.: as stated in the previous study unit, mathematical expressions can be produced by the format $ ... $ (for an in-text expression), or \begin{equation} ... \end{equation} (for an expression that will be displayed and numbered), or \[ ... \] (for an expression that will be displayed but not numbered). Most of the features described here are valid only in mathematical expressions, i.e. only within one of the environments stated in the previous sentence.

1. Superscripts and subscripts are generated by ^ and _ respectively. The system assumes that the command applies only to the next character: if there should be several characters in the subscript or superscript, enclose them in {}, for example $e^{-2at}$. You can also make a subscript or superscript to a character that is already a subscript or superscript.

2. Fractions are generated by the command \frac{numerator}{denominator}. Both the numerator and denominator can be general expressions, and (multiply) nested fractions are permitted.

3. The nth root of an expression is produced by the command \sqrt[n]{expression}; the field [n] is optional and can be omitted when a square root is intended.

4. A series of dots can be produced by \ldots or \cdots .

5. A wide variety of special mathematical symbols is available in LaTeX, and a table showing LaTeX code and the resulting symbol can be found in the file su14-2.pdf (which, in the printed version of these notes is at the end of this study unit). Note also the command \not which places a slash through the symbol following it. The commands listed \arccos ... \tanh are needed because otherwise, for example, \tan would be printed as though it meant t times a times n.

6. Capital letters (and only capital letters) can be printed in caligraphic style, using, for example, the syntax \mathcal{A}. You can express any symbol in bold face by means of the syntax \mathbf{symbol(s)}. There are also the type style commands \mathit{}, \mathrm{}, \mathsf{} and mathtt{}, but they will not be considered further in these notes.

7. Many mathematical expressions contain parentheses of various forms. A complete list of options is given in su14-2.pdf; in these examples we use only the most common forms, i.e. (a+b),[a+b], {a+b} - in LaTeX \{ a+b \} , |a+b| and || a+b || - in LaTeX \| a+b \| . It is important that parentheses match, and that they are an appropriate size for the expression. This is achieved by using the commands \left and \right followed immediately by the symbol for the required bracket. The \left and \right commnads must come in matching pairs, but the type of the parentheses need not be the same. Thus, if desired, you can produce something of the form ( ... [ . Further, you can make one of the pair invisible by typing \left. or \right. so in this way you can arrange that the output, i.e. the typeset expression, does not have matching parentheses.

8. Usually the default spacing produced by LaTeX is satisfactory, but sometimes you will need to adjust it. N.B. in a mathematical environment spaces between symbols are ignored, so if you need to adjust the spacing you will have to use one of the following commands: \, \: \; and \  produce spacings of increasing magnitude (The last option is the symbol \ followed by one typed space). If you need to reduce the spacing between symbols, use \! which results in a small negative space.

9. You can put overlines or underlines in formulas, and they can be nested, by means of the syntax \overline{expression} or \underline{expression}. You can also use the commands \overbrace{expression} or \underbrace{expression} to put a curly brace over or under an expression; in this case you can also put a superscript or subscript to the brace.

10. An array environment can be declared within a mathematical environment. The syntax is \begin{array}{tuv..} .... \end{array}, where each of t, u, v, ..., has the value c, l or r describing the positioning of items in the respective column: c for centred, l for left flush and r for right flush. Within the body of the environment, adjacent rows are separated by a \\ command and adjacent items within a row are separated by the & character. There must be no & after the last item in a row, and no \\ after the last row. Often useful within an array is the command \vdots that places a sequence of vertical dots. Using \left and \right, as discussed above, places appropriately sized parentheses around an array.

11. LaTeX will not automatically split a long expression over two or more lines: it has to be explicitly instructed to do so by the user. The syntax is \begin{eqnarray} ... \end{eqnarray}. Note the following features of the eqnarray syntax:

- Each line-break is indicated by \\ - but do not put \\ at the end of the formula.
- The default is that each line has an equation number - if you do not want a number for a particular line, put \nonumber after it, and before the \\ if present.
- If you want no equation numbers at all, use instead the environment \begin{eqnarray*}... \end{eqnarray*}.

- Alignment is arranged by putting & before and after the symbol on each line to be in vertical allignment; the symbol can be a blank space. For example, in a sequence of equalities, it would be ususal to align the = signs by typing & = & .

12. The command \mbox{text} is used to create regular text within a mathematical expression.

### 14.3 MuPAD and LaTeX

If you would like to include MuPAD code and/or output in a LaTeX document, it is important that the spacing is exactly preserved. This is done by using the environment \begin{verbatim} text \end{verbatim}, in which case material is typeset exactly as input.

You can also use MuPAD to generate LaTeX by means of the command generate::TeX. The syntax is illustrated below

```
>> t1:=generate::TeX(sqrt(x^2)/x);

                "\\frac{\\sqrt{x^2}}{x}"
```

and the result is translated into a form usable by LaTeX by printing t1 with the Unquoted option

```
>> print(Unquoted,t1);
                \frac{\sqrt{x^2}}{x}
```

Another example is given in su14-1.tex. Note that in the verbatim environment the system does not enter any line-breaks (although it will enter page-breaks), and so the input may cause a line to go outside the normal output region, or even off the page.

### 14.4 LaTeX packages

A number of specialized symbols and commands are defined in packages, and the system must be told to look in a particular package for these constructs. The required syntax is to place a command \usepackage{name} after the \documentclass declaration and before the \begin{document} statement. The file su14-1.tex contains the command \usepackage{amssymb}. Comment this statement out (by placing % in front of it), and you will see that the LaTeX compiler objects to the \eth symbol (which represents a derivative on the unit sphere).

### 14.5 Exercise

1. Write LaTeX code to re–create su14-x1.pdf (see page 138).
2. Write LaTeX code to re–create su14-x2.pdf (see page 139).

```
%su14-1.tex

\documentclass[12pt,a4paper]{article}
\usepackage{amssymb}
\begin{document}

\title{su14-1.pdf}
\author{}
\date{}
\maketitle


\section*{14.2 Mathematical expressions in \LaTeX}

\begin{enumerate}
%1
\item  $x^2$, $e^{-2at}$, $e^{-2at^2}$, $V_i$ and $B_{(a,b)}^{x^{2a}}$
%2
\item \[ x= \frac{1+y}{2-y} \]
    \[ a = \frac{c+d^{2e}}{c+\frac{x^2}{y^3}} \]
%3
\item $\sqrt{x+y}$, $\sqrt[3]{4}$ and
    \[ \sqrt[n]{\frac{x}{y}} \]
%4
\item $x_1, \ldots ,x_n$ or $x_1 + \cdots + x_n$
%5
\item If $\lambda \not< \Lambda$ then $\lambda \geq \Lambda$
\begin{equation}
\sum_{i=1}^n \Omega_i = \int_0^1 \frac{\Xi}{\Delta} d\theta
\end{equation}
$\lim_{n \rightarrow \infty}\Gamma =0$ is different when displayed
\[ \lim_{n \rightarrow \infty}\Gamma =0 \]
\[ tan\alpha \neq \tan\alpha \]
$A \cup B = \emptyset$ $\Leftrightarrow$ $A=\emptyset$ and $B=\emptyset$.
%6
```

```
\item If $\mathcal{F}>0$ and $\mathcal{G}>0$ then $\mathcal{FG}>0$.
%7
\item We write
\[ d = c \left(\sqrt{  \frac{a}{1+b}} -e^x \left[ a+b \right] \right) \]
but, if we really wanted to, we could produce
\[ d = c \left(\sqrt{  \frac{a}{1+b}} -e^x \left[ a+b \right\{ \right] \]
or even
\[ d = c \left(\sqrt{  \frac{a}{1+b}} -e^x \left[ a+b \right\{ \right. \]
%8
\item
\[ \sqrt{2} x \rightarrow \sqrt{2} \, x \]
\[ n/ \log n \rightarrow n/ \! \log n \]
\[ \int \int z dx dy \rightarrow \int \!\!\! \int z\, dx\, dy  \]
%9
\item You can have nested overlining $\overline{\overline{x}^2+1}$ as well
as over- and under-braces
\[ \underbrace{a+ \overbrace{b + \cdots +y}^{24} + z}_{26} \]
%10
\item
\begin{equation}
 M=\left(
\begin{array}{clr}
a+b+c & uv & 27 \\
a-b & u+v & 134 \\
a & 3u - vw & 2,975
\end{array} \right)
\end{equation}
Solve
\[
\left(
\begin{array}{ccc}
a_{11} & \cdots & a_{1n} \\
\vdots &  & \vdots \\
a_{n1} & \cdots & a_{nn}
\end{array} \right)
\left( \begin{array}{c}
     X_1 \\ \vdots \\  X_n
   \end{array} \right)
= \left( \begin{array}{r}
     1 \\ \vdots \\  n
   \end{array} \right)
\]
%11
\item
\begin{eqnarray}
A & = & \frac{x^6 - y^6 - 4 x y^5 + 4 x^5 y - 5 x^2 y^4 + 5 x^4 y^2}
          {x^5 + y^5 + 5 x y^4 + 5 x^4 y + 10 x^2 y^3 + 10 x^3 y^2}
          \nonumber \\
 & = & \frac{(x-y)(x+y)^5}{(x+y)^5} \nonumber \\
 & = & x-y \\
 & > & x-y-1
\end{eqnarray}
%12
\item
\[ \log xy = \log x + \log y, \mbox{ if $x>0$ and $y>0$} \]
\end{enumerate}

\section*{14.3 MuPAD and \LaTeX}

We use the verbatim environment to preserve the spacing in MuPAD output,
so that it reads correctly
\begin{verbatim}
>> expand((x^2+2*y)^3);

                   3    6      4        2 2
                8 y  + x  + 6 x  y + 12 x  y
\end{verbatim}
We write the following MuPAD code:
\begin{verbatim}
```

```
>> assume(a>0):
>> t1:=generate::TeX(hold(int)(1/sqrt(a^2-x^2),x)):
>> t2:=generate::TeX(int(1/sqrt(a^2-x^2),x)):
>> print(Unquoted, t1 =t2);

  \int \frac{1}{\sqrt{a^2 - x^2}} d x = \arcsin\left(\frac{x}{a}\right)
\end{verbatim}
The above result is compiled by \LaTeX to give
\[\int \frac{1}{\sqrt{a^2 - x^2}} d x = \arcsin\left(\frac{x}{a}\right) \]


\section*{14.4 \LaTeX packages}
The next equation uses a symbol defined in the amssymb package.
\begin{equation}
J= \eth^2 \alpha
\end{equation}
\end{document}
```

# su14-1.pdf

## 14.2 Mathematical expressions in LaTeX

1. $x^2$, $e^{-2at}$, $e^{-2at^2}$, $V_i$ and $B^{x^{2a}}_{(a.b)}$

2.
$$x = \frac{1+y}{2-y}$$
$$a = \frac{c+d^{2e}}{c+\frac{x^2}{y^3}}$$

3. $\sqrt{x+y}$, $\sqrt[3]{4}$ and
$$\sqrt[n]{\frac{x}{y}}$$

4. $x_1, \ldots, x_n$ or $x_1 + \cdots + x_n$

5. If $\lambda \not< \Lambda$ then $\lambda \geq \Lambda$
$$\sum_{i=1}^{n} \Omega_i = \int_0^1 \frac{\Xi}{\Delta} d\theta \qquad (1)$$
$\lim_{n\to\infty} \Gamma = 0$ is different when displayed
$$\lim_{n\to\infty} \Gamma = 0$$
$$tan\alpha \neq \tan\alpha$$
$A \cup B = \emptyset \Leftrightarrow A = \emptyset$ and $B = \emptyset$.

6. If $\mathcal{F} > 0$ and $\mathcal{G} > 0$ then $\mathcal{FG} > 0$.

7. We write

$$d = c\left(\sqrt{\frac{a}{1+b}} - e^x\left[a+b\right]\right)$$

but, if we really wanted to, we could produce

$$d = c\left(\sqrt{\frac{a}{1+b}} - e^x\left[a+b\{\right]\right.$$

or even

$$d = c\left(\sqrt{\frac{a}{1+b}} - e^x\left[a+b\{\right.\right.$$

8.

$$\sqrt{2}x \to \sqrt{2}\,x$$

$$n/\log n \to n/\!\log n$$

$$\int\int z dx dy \to \iint z\,dx\,dy$$

9. You can have nested overlining $\overline{\overline{x}^2+1}$ as well as over- and under-braces

$$\underbrace{a+\overbrace{b+\cdots+y}^{24}+z}_{26}$$

10.

$$M = \begin{pmatrix} a+b+c & uv & 27 \\ a-b & u+v & 134 \\ a & 3u-vw & 2,975 \end{pmatrix} \tag{2}$$

Solve

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ n \end{pmatrix}$$

11.

$$\begin{aligned} A &= \frac{x^6 - y^6 - 4xy^5 + 4x^5y - 5x^2y^4 + 5x^4y^2}{x^5 + y^5 + 5xy^4 + 5x^4y + 10x^2y^3 + 10x^3y^2} \\ &= \frac{(x-y)(x+y)^5}{(x+y)^5} \\ &= x - y \qquad (3) \\ &> x - y - 1 \qquad (4) \end{aligned}$$

12.

$$\log xy = \log x + \log y, \text{ if } x > 0 \text{ and } y > 0$$

## 14.3 MuPAD and LaTeX

We use the verbatim environment to preserve the spacing in MuPAD output, so that it reads correctly

```
>> expand((x^2+2*y)^3);

                          3    6      4         2  2
                       8 y  + x  + 6 x  y + 12 x  y
```

We write the following MuPAD code:

```
>> assume(a>0):
>> t1:=generate::TeX(hold(int)(1/sqrt(a^2-x^2),x)):
>> t2:=generate::TeX(int(1/sqrt(a^2-x^2),x)):
>> print(Unquoted, t1 =t2);

   \int \frac{1}{\sqrt{a^2 - x^2}} d x = \arcsin\left(\frac{x}{a}\right)
```

The above result is compiled by LaTeXto give

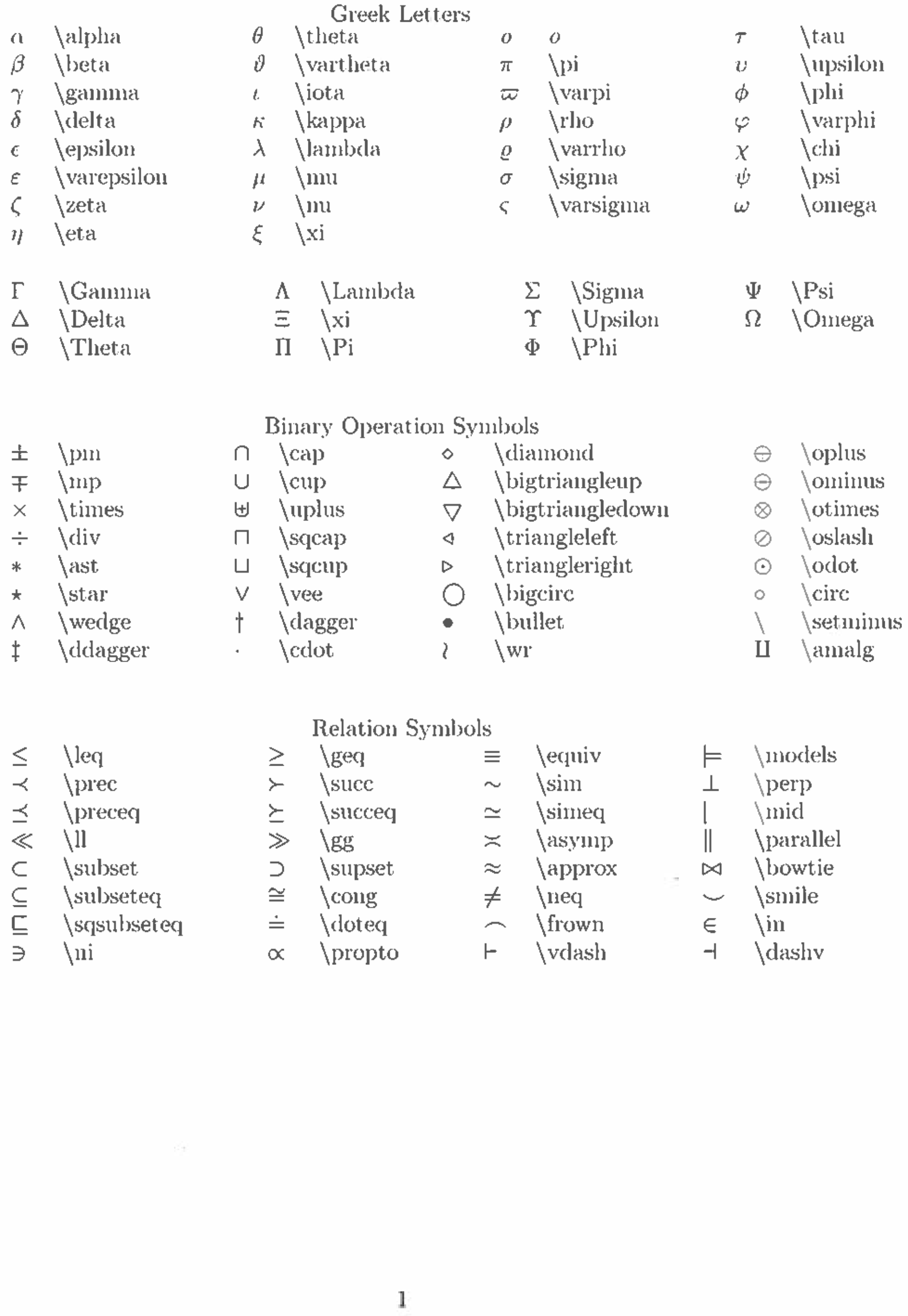
$$\int \frac{1}{\sqrt{a^2 - x^2}} dx = \arcsin\left(\frac{x}{a}\right)$$

## 14.4 LaTeXpackages

The next equation uses a symbol defined in the amssymb package.

$$J = \eth^2\alpha \tag{5}$$

## su14-2.pdf



Greek Letters

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\alpha$ | \alpha | $\theta$ | \theta | $o$ | $o$ | $\tau$ | \tau |
| $\beta$ | \beta | $\vartheta$ | \vartheta | $\pi$ | \pi | $\upsilon$ | \upsilon |
| $\gamma$ | \gamma | $\iota$ | \iota | $\varpi$ | \varpi | $\phi$ | \phi |
| $\delta$ | \delta | $\kappa$ | \kappa | $\rho$ | \rho | $\varphi$ | \varphi |
| $\epsilon$ | \epsilon | $\lambda$ | \lambda | $\varrho$ | \varrho | $\chi$ | \chi |
| $\varepsilon$ | \varepsilon | $\mu$ | \mu | $\sigma$ | \sigma | $\psi$ | \psi |
| $\zeta$ | \zeta | $\nu$ | \nu | $\varsigma$ | \varsigma | $\omega$ | \omega |
| $\eta$ | \eta | $\xi$ | \xi | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\Gamma$ | \Gamma | $\Lambda$ | \Lambda | $\Sigma$ | \Sigma | $\Psi$ | \Psi |
| $\Delta$ | \Delta | $\Xi$ | \xi | $\Upsilon$ | \Upsilon | $\Omega$ | \Omega |
| $\Theta$ | \Theta | $\Pi$ | \Pi | $\Phi$ | \Phi | | |

Binary Operation Symbols

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\pm$ | \pm | $\cap$ | \cap | $\diamond$ | \diamond | $\ominus$ | \oplus |
| $\mp$ | \mp | $\cup$ | \cup | $\bigtriangleup$ | \bigtriangleup | $\ominus$ | \ominus |
| $\times$ | \times | $\uplus$ | \uplus | $\bigtriangledown$ | \bigtriangledown | $\otimes$ | \otimes |
| $\div$ | \div | $\sqcap$ | \sqcap | $\triangleleft$ | \triangleleft | $\oslash$ | \oslash |
| $\ast$ | \ast | $\sqcup$ | \sqcup | $\triangleright$ | \triangleright | $\odot$ | \odot |
| $\star$ | \star | $\vee$ | \vee | $\bigcirc$ | \bigcirc | $\circ$ | \circ |
| $\wedge$ | \wedge | $\dagger$ | \dagger | $\bullet$ | \bullet | $\setminus$ | \setminus |
| $\ddagger$ | \ddagger | $\cdot$ | \cdot | $\wr$ | \wr | $\amalg$ | \amalg |

Relation Symbols

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\leq$ | \leq | $\geq$ | \geq | $\equiv$ | \equiv | $\models$ | \models |
| $\prec$ | \prec | $\succ$ | \succ | $\sim$ | \sim | $\perp$ | \perp |
| $\preceq$ | \preceq | $\succeq$ | \succeq | $\simeq$ | \simeq | $\mid$ | \mid |
| $\ll$ | \ll | $\gg$ | \gg | $\asymp$ | \asymp | $\parallel$ | \parallel |
| $\subset$ | \subset | $\supset$ | \supset | $\approx$ | \approx | $\bowtie$ | \bowtie |
| $\subseteq$ | \subseteq | $\cong$ | \cong | $\neq$ | \neq | $\smile$ | \smile |
| $\sqsubseteq$ | \sqsubseteq | $\doteq$ | \doteq | $\frown$ | \frown | $\in$ | \in |
| $\ni$ | \ni | $\propto$ | \propto | $\vdash$ | \vdash | $\dashv$ | \dashv |

## Arrow Symbols

| | | | | | |
|---|---|---|---|---|---|
| $\leftarrow$ | \leftarrow | $\longleftarrow$ | \longleftarrow | $\uparrow$ | \uparrow |
| $\Leftarrow$ | \Leftarrow | $\Longleftarrow$ | \Longleftarrow | $\Uparrow$ | \Uparrow |
| $\rightarrow$ | \rightarrow | $\longrightarrow$ | \longrightarrow | $\downarrow$ | \downarrow |
| $\Rightarrow$ | \Rightarrow | $\Longrightarrow$ | \Longrightarrow | $\Downarrow$ | \Downarrow |
| $\leftrightarrow$ | \leftrightarrow | $\longleftrightarrow$ | \longleftrightarrow | $\updownarrow$ | \updownarrow |
| $\Leftrightarrow$ | \Leftrightarrow | $\Longleftrightarrow$ | \Longleftrightarrow | $\Updownarrow$ | \Updownarrow |
| $\mapsto$ | \mapsto | $\longmapsto$ | \longmapsto | $\nearrow$ | \nearrow |
| $\hookleftarrow$ | \hookleftarrow | $\hookrightarrow$ | \hookrightarrow | $\searrow$ | \searrow |
| $\leftharpoonup$ | \leftharpoonup | $\rightharpoonup$ | \rightharpoonup | $\swarrow$ | \swarrow |
| $\leftharpoondown$ | \leftharpoondown | $\rightharpoondown$ | \rightharpoondown | $\nwarrow$ | \nwarrow |
| $\rightleftharpoons$ | \righleftharpoons | | | | |

## Miscellaneous Symbols

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\aleph$ | \aleph | $\prime$ | \prime | $\forall$ | \forall | $\infty$ | \infty |
| $\hbar$ | \hbar | $\emptyset$ | \emptyset | $\exists$ | \exists | $\imath$ | \imath |
| $\nabla$ | \nabla | $\neg$ | \neg | $\jmath$ | \jmath | $\surd$ | \surd |
| $\flat$ | \flat | $\triangle$ | \triangle | $\ell$ | \ell | $\top$ | \top |
| $\natural$ | \natural | $\clubsuit$ | \clubsuit | $\wp$ | \wp | $\bot$ | \bot |
| $\sharp$ | \sharp | $\diamondsuit$ | \diamondsuit | $\Re$ | \Re | $\Vert$ | \ |
| $\backslash$ | \backslash | $\heartsuit$ | \heartsuit | $\Im$ | \Im | $\angle$ | \angle |
| $\partial$ | \partial | $\spadesuit$ | \spadesuit | | | | |

## Variable-sized Symbols

| | | | | | |
|---|---|---|---|---|---|
| $\sum$ | \sum | $\bigcap$ | \bigcap | $\bigodot$ | \bigodot |
| $\prod$ | \prod | $\bigcup$ | \bigcup | $\bigotimes$ | \bigotimes |
| $\coprod$ | \coprod | $\bigsqcup$ | \bigsqcup | $\bigoplus$ | \bigoplus |
| $\int$ | \int | $\bigvee$ | \bigvee | $\biguplus$ | \biguplus |
| $\oint$ | \oint | $\bigwedge$ | \bigwedge | | |

## Log-like Functions (all preceded by \)

| | | | |
|---|---|---|---|
| arccos | cos | csc | exp |
| ker | lim sup | min | sinh |
| arcsin | cosh | deg | gcd |
| lg | ln | Pr | sup |
| arctan | cot | det | hom |
| lim | log | sec | tan |
| arg | coth | dim | inf |
| lim inf | max | sin | tanh |

Delimiters

<table>
<tr><td>$($</td><td>(</td><td>$)$</td><td>)</td><td>$\uparrow$</td><td>\uparrow</td><td>$[$</td><td>[</td></tr>
<tr><td>$]$</td><td>]</td><td>$\downarrow$</td><td>\downarrow</td><td>$\{$</td><td>\ {</td><td>$\}$</td><td>\ }</td></tr>
<tr><td>$\updownarrow$</td><td>\updownarrow</td><td>$\lfloor$</td><td>\lfloor</td><td>$\rfloor$</td><td>\rfloor</td><td>$\Uparrow$</td><td>\Uparrow</td></tr>
<tr><td>$\lceil$</td><td>\lceil</td><td>$\rceil$</td><td>\rceil</td><td>$\Downarrow$</td><td>\Downarrow</td><td>$\langle$</td><td>\langle</td></tr>
<tr><td>$\rangle$</td><td>\rangle</td><td>$\Updownarrow$</td><td>\Updownarrow</td><td>$/$</td><td>/</td><td>$\backslash$</td><td>\backslash</td></tr>
<tr><td>$\vert$</td><td>|</td><td>$\Vert$</td><td>\|</td><td></td><td></td><td></td><td></td></tr>
</table>

Math Mode Accents

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\hat{a}$ | \hat{a} | $\acute{a}$ | \acute{a} | $\bar{a}$ | \bar{a} | $\dot{a}$ | \dot{a} |
| $\check{a}$ | \check{a} | $\grave{a}$ | \grave{a} | $\vec{a}$ | \vec{a} | $\ddot{a}$ | \ddot{a} |
| $\breve{a}$ | \breve{a} | $\tilde{a}$ | \tilde{a} | | | | |

## su14-x1.pdf

We can now determine the metric variables $_{[\mathcal{R}]}$, to first order in $v$, from the conditions:

$$
\begin{aligned}
g^{01}_{[\mathcal{R}]} = -e^{-2\beta_{[\mathcal{R}]}} \quad &\rightarrow \quad \beta_{[\mathcal{R}]1} = \beta_{[\mathcal{R}]0} + v\beta_{[\mathcal{R}]v} = -\frac{1}{2}\log - g^{01}_{[\mathcal{R}]1} \\
\frac{g^{11}_{[\mathcal{R}]}}{g^{01}_{[\mathcal{R}]}} = -1 - \frac{W_{[\mathcal{R}]}}{r_{[\mathcal{R}]}} \quad &\rightarrow \quad W_{[\mathcal{R}]1} = W_{[\mathcal{R}]0} + vW_{[\mathcal{R}]v} = -r_{[\mathcal{R}]}\left(\frac{g^{11}_{[\mathcal{R}]1}}{g^{01}_{[\mathcal{R}]1}} + 1\right) \\
\frac{g^{1A}_{[\mathcal{R}]}}{g^{01}_{[\mathcal{R}]}} = -U^{A}_{[\mathcal{R}]} \quad &\rightarrow \quad U_{[\mathcal{R}]1} = U_{[\mathcal{R}]0} + vU_{[\mathcal{R}]v} = -q_{[\mathcal{R}]A}\frac{g^{1A}_{[\mathcal{R}]1}}{g^{01}_{[\mathcal{R}]1}}. \qquad (1)
\end{aligned}
$$

The Maple calculation uses the usual rules for defining the spin-weighted quantities $U_{[\mathcal{A}]}$, $J_{[\mathcal{A}]}$ and $K_{[\mathcal{A}]}$, and for replacing $q_{[\mathcal{A}]}-$ and $p_{[\mathcal{A}]}-$derivatives by $\eth_{[\mathcal{A}]}$ and $\bar{\eth}_{[\mathcal{A}]}$.

The Maple output for the matrix $g^{\alpha\beta}_{[\mathcal{A}]}$ was

```
[exp(-2 B) Va ,    exp(-2 B) Uq ,    exp(-2 B) Up ,    exp(-2 B)]
[                                                              ]
[                    2                          2              ]
[                  fA  (2 K - J - Jb)       I fA  (J - Jb)     ]
[exp(-2 B) Uq , 1/8 ------------------ , 1/8 -------------- , 0]
[                           2                       2          ]
[                          r                       r           ]
[                                                              ]
[                      2                  2                    ]
[                  I fA  (J - Jb)       fA  (J + Jb + 2 K)     ]
[exp(-2 B) Up , 1/8 -------------- , 1/8 ------------------ , 0]
[                         2                      2             ]
[                        r                      r              ]
[                                                              ]
[exp(-2 B) ,              0 ,               0 ,              0]
```

## su14-x2.pdf

The coordinate transformation $x^a_{[\mathcal{A}]} \rightarrow x^a_{[\mathcal{R}]}$ is written in the form

$$\begin{aligned} r_{[\mathcal{R}]} &= r_{[\mathcal{A}]} + vA^r + v^2C^r \\ x^A_{[\mathcal{R}]} &= x^A_{[\mathcal{A}]} + vA^A + v^2C^A \\ u &= H^u + vA^u + v^2C^u, \end{aligned} \tag{1}$$

where, in general, $A^a$, $C^a$ and $H^u$ are functions of $r_{[\mathcal{A}]}$ and $x^A_{[\mathcal{A}]}$; and where $u$ (in the retarded system) and $v$ (in the advanced system) are written without suffices because there is no ambiguity in so doing.

We find

$$\begin{aligned} \frac{\partial}{\partial r_{[\mathcal{R}]}} &= F(-A^u + \frac{1}{2}(\bar{A}\eth_{[\mathcal{A}]}H^u + A\bar{\eth}_{[\mathcal{A}]}H^u))\frac{\partial}{\partial r_{[\mathcal{A}]}} \\ &\quad -FH^u_{,r_{[\mathcal{A}]}}A^q\frac{\partial}{\partial q_{[\mathcal{A}]}} \\ &\quad -FH^u_{,r_{[\mathcal{A}]}}A^p\frac{\partial}{\partial p_{[\mathcal{A}]}} \\ &\quad +FH^u_{,r_{[\mathcal{A}]}}\frac{\partial}{\partial v}; \end{aligned} \tag{2}$$

Tha computer algebra script begins as follows:

```
xA:=array(0..4,[r,q,p,v]);
xR:=array(0..4,[rR,qR,pR,u]);
```

# STUDY UNIT 15

## 15.1 Introduction

This study unit completes the discussion of LaTeX. Here we describe the use of LaTeX in typesetting documents that include graphics and tables

The material in this study unit will be presented with reference to sample documents su15-1.tex, which on compilation gives su15-1.pdf. The files su15-1.tex and su15-1.pdf can be found at the end of the study unit.

## 15.2 Tables in LaTeX

LaTeX provides three environments for representing tabular material.

1. The environment \begin{table} ... \end{table} is used to contain the other two environments, if so desired. A table environment with other forms of input, produces output that is not usually useful. It has two features

- It cannot be broken by an automatically generated page break
- It can be captioned, by the command \caption{Name}. It can also be labelled (by the command \label{name}) and referred to elsewhere in the document by the label name; in this case, \caption must be before \label .

2. It can be tricky to persuade LaTeX to position the table where you want it. The syntax for specifying the location is \begin{table}[x] where x can be any of h, t, b, p; x can also be a combination of these, indicating your priorities. The default for x is tbp which indicates that the table should be placed at the top or bottom of a text page, or an a separate page consisting only of figures and/or tables. If you want the table to appear where you have entered it in the document use \begin{table}[h] . There is aslo a table* environment, in which the tables produced are not numbered.

The environment \begin{tabbing} ... \end{tabbing} has the following features

- First, tabbing stops must be set, and this is done by the \= command. In LaTeX spaces (or more precisely, spaces after one space) are ignored, so often tabbing stops are set by writing a line with \= and other text, and then making it produce no output by ending it with \kill. If the line defining the tabbing stops does not end with \kill, then it must end with \\ .

- The command \> causes the output to move to the next tab stop

- Lines of input, except the last line, are ended by \\ .

3. The environment \begin{tabular} ... \end {tabular} provides a fairly versatile facility for all sorts of tabular material. In many ways it is similar to the array environment, but differs in that it is used in normal text mode rather than mathematical text mode. It has the following features

- The declaration statement is

  \begin{tabular} {abc ... n}

  where each of abc ... n can be one of the following symbols: | l r c. The symbol | causes a vertical line to be drawn through the whole table, l means a left-justified column, r means a right-justified column and c means a centre-justified column.

- Each row of the table ends with \\ , apart from the last row (with one exception for the last row, noted below)

- A command \hline at the beginning of the environment, or after \\ , draws a horizontal line across the full width of the environment.; if a horizontal line is wanted at the bottom of the table, then there is a \\ after the last row (and then \hline).

- The command \cline{i-j}, where i and j are positive integers, draws a horizontal line across columns i through j inclusive.

- A single item in the table that spans several columns is made with a \multicolumn command, having the form

  \multicolumn{n} {p} {text}

  where the positive integer n is the number of coulmns to be spanned, text is the text to be printed, and p is a single l r or c (and may also contain | symbols)

- The default is that the resultant table is left-justified. If you want the table to be centred, you need to place it in a center environment, by means of the syntax

  \begin{center} \begin{tabular} ... \end{tabular} \end{center}

  The environment center will be discussed further in 15.4 below.

- Outside the table environment, some of the bounding horizontal and vertical lines may not work.

### 15.3 Miscellaneous features of LaTeX

1. We have already described how to cite references (which are listed at the end of the document), and there are also occasions when you want to put a footnote on the page in which you refer to it. You do this by means of the command ...\footnote{text} ...

2. Usually the default layout of LaTeX is satisfactory, but there are times when some text (including the contents of tables etc.) needs to be centred. This is achieved by means of the center environment, with syntax

```
\begin{center} ... \end{center}
```

Similarly, you may declare a flushleft or flushright environment.

3. If desired, you can temporarily adjust the font size by using the same syntax as for **bold** or *italic* typeface. The size options available are (in increasing magnitude)

```
\tiny \scriptsize \footnotesize \small \normalsize \large \Large \LARGE \huge \Huge
```

The syntax for producing some text in, for example, Large print is

```
{\Large text}
```

If you submit your assignments via SOL, you are asked to include information about your student number, module code and assignement number on each page. You can do this in LaTeX by placing the following code between the \documentclass declaration and \begin{document}

```
\pagestyle{myheadings}
\markboth{text} {text}
```

where text is the information to be given at the top of each page.

### 15.5 Exercise

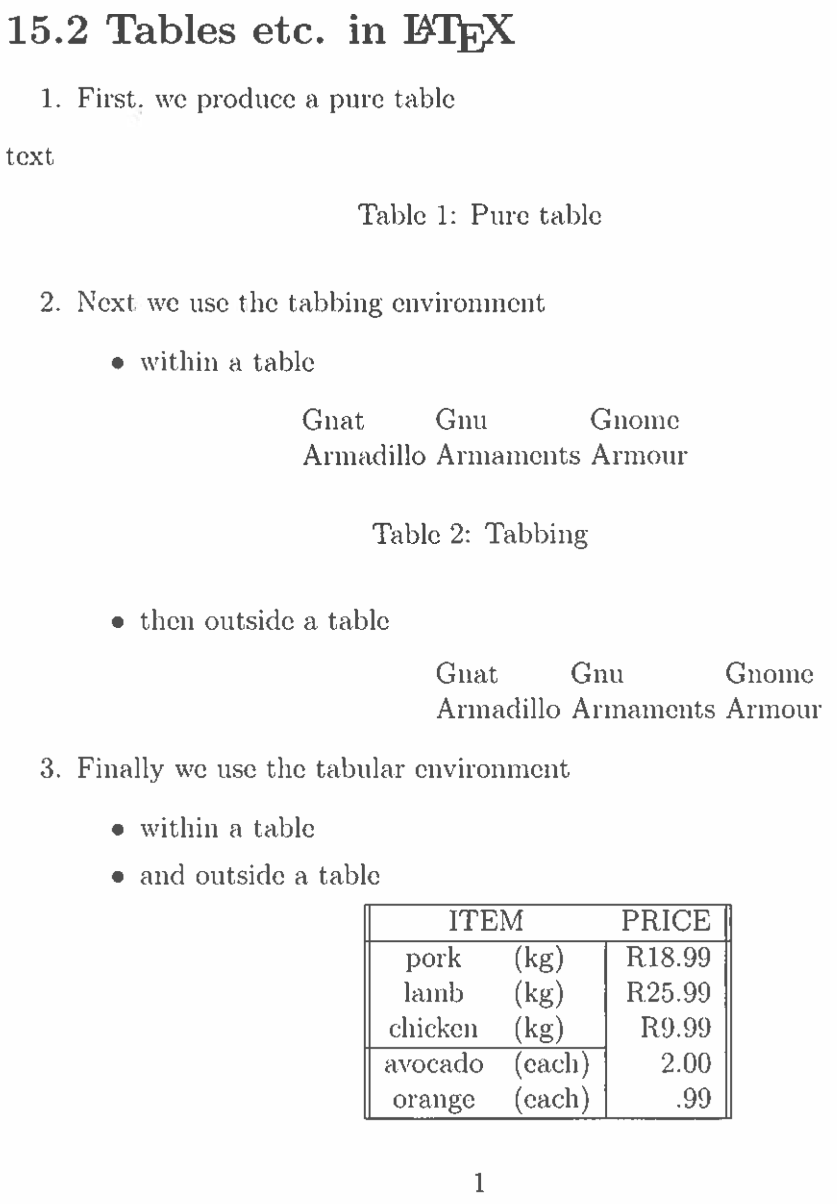
1. Write LaTeX code to re–create su15-x1.pdf (see page 149).
2. Write LaTeX code to re–create su15-x2.pdf (see page 150).

```
%su15-1.tex

\documentclass[12pt,a4paper]{article}
\pagestyle{myheadings}
\markboth{Student number, APM216, Assignment n}
{Student number, APM216, Assignment n}
\usepackage{graphics}

\begin{document}
\title{su15-1.ps}
\author{}
\date{}
\maketitle


\section*{15.2 Tables etc. in \LaTeX}
\begin{enumerate}
\item First, we produce a pure table
\begin{table}[h]
text
\caption{Pure table}
\end{table}
\item Next we use the tabbing environment
\begin{itemize}
\item within a table
\begin{table}[h]
\begin{tabbing}
xxxxxxxxxxxxxxxxxxx \= Armadillo \= Armaments \=  \kill
 \> Gnat \> Gnu \> Gnome \\
 \> Armadillo \> Armaments \> Armour
\end{tabbing}
\caption{Tabbing}
\end{table}
\item then outside a table
```

```
\begin{tabbing}
xxxxxxxxxxxxxxxxxxx \= Armadillo \= Armaments \=  \kill
 \> Gnat \> Gnu \> Gnome \\
 \> Armadillo \> Armaments \> Armour
\end{tabbing}
\end{itemize}
\item Finally we use the tabular environment
\begin{itemize}
\item within a table
\begin{table}[h]
\begin{center}
\begin{tabular}{|| c l | r||} \hline
\multicolumn{2}{||c}{ITEM} & \multicolumn{1}{c||}{PRICE} \\ \hline
pork & (kg) & R18.99 \\
lamb & (kg) & R25.99 \\
chicken & (kg) & R9.99  \\ \cline{1-2}
avocado & (each) &2.00 \\
orange & (each) & .99 \\ \hline
\end{tabular}
\end{center}
\caption{Tabular}
\end{table}
\item and outside a table
\begin{center}
\begin{tabular}{|| c l | r||} \hline
\multicolumn{2}{||c}{ITEM} & \multicolumn{1}{c||}{PRICE} \\ \hline
pork & (kg) & R18.99 \\
lamb & (kg) & R25.99 \\
chicken & (kg) & R9.99  \\ \cline{1-2}
avocado & (each) &2.00 \\
orange & (each) & .99 \\ \hline
\end{tabular}
\end{center}
\end{itemize}
\end{enumerate}
```

```
\section*{15.3 Miscellaneous features of \LaTeX}

\begin{enumerate}
\item It is easy to make a footnote\footnote{Here it is} in a \LaTeX
document.
\item It is also easy to arrange \\
\begin{center}
to centre a piece of text.
\end{center}
\item Text can be in {\Large various} {\tiny different} sizes.
\end{enumerate}

\end{document}
```

# su15-1.pdf



## 15.2 Tables etc. in LaTeX

1. First, we produce a pure table

text

Table 1: Pure table

2. Next we use the tabbing environment

- within a table

| | | |
|---|---|---|
| Gnat | Gnu | Gnome |
| Armadillo | Armaments | Armour |

Table 2: Tabbing

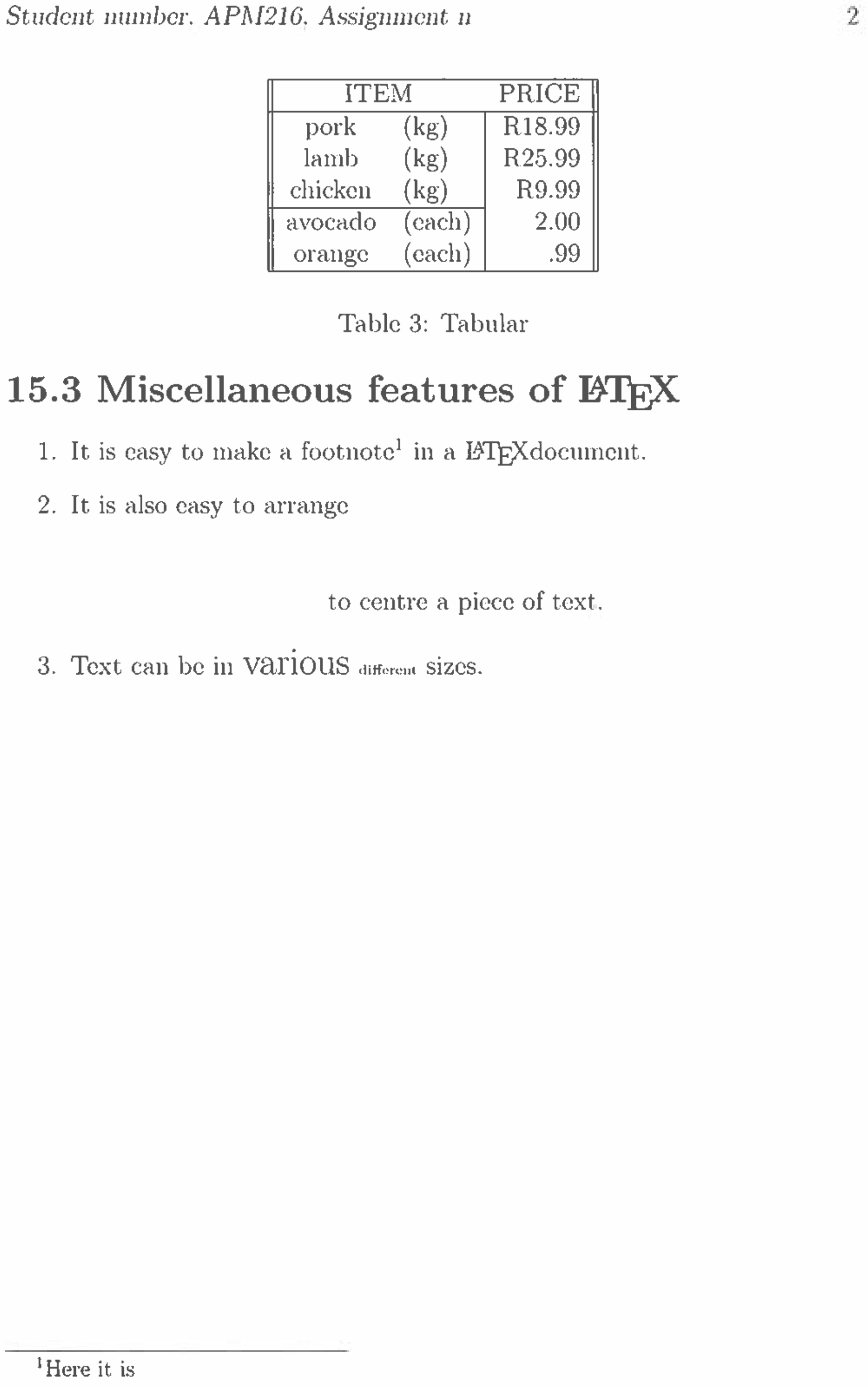
- then outside a table

| | | |
|---|---|---|
| Gnat | Gnu | Gnome |
| Armadillo | Armaments | Armour |

3. Finally we use the tabular environment

- within a table
- and outside a table

| ITEM | | PRICE |
|---|---|---|
| pork | (kg) | R18.99 |
| lamb | (kg) | R25.99 |
| chicken | (kg) | R9.99 |
| avocado | (each) | 2.00 |
| orange | (each) | .99 |



*Student number. APM216. Assignment n* 2

| ITEM | | PRICE |
|---|---|---|
| pork | (kg) | R18.99 |
| lamb | (kg) | R25.99 |
| chicken | (kg) | R9.99 |
| avocado | (each) | 2.00 |
| orange | (each) | .99 |

Table 3: Tabular

## 15.3 Miscellaneous features of LaTeX

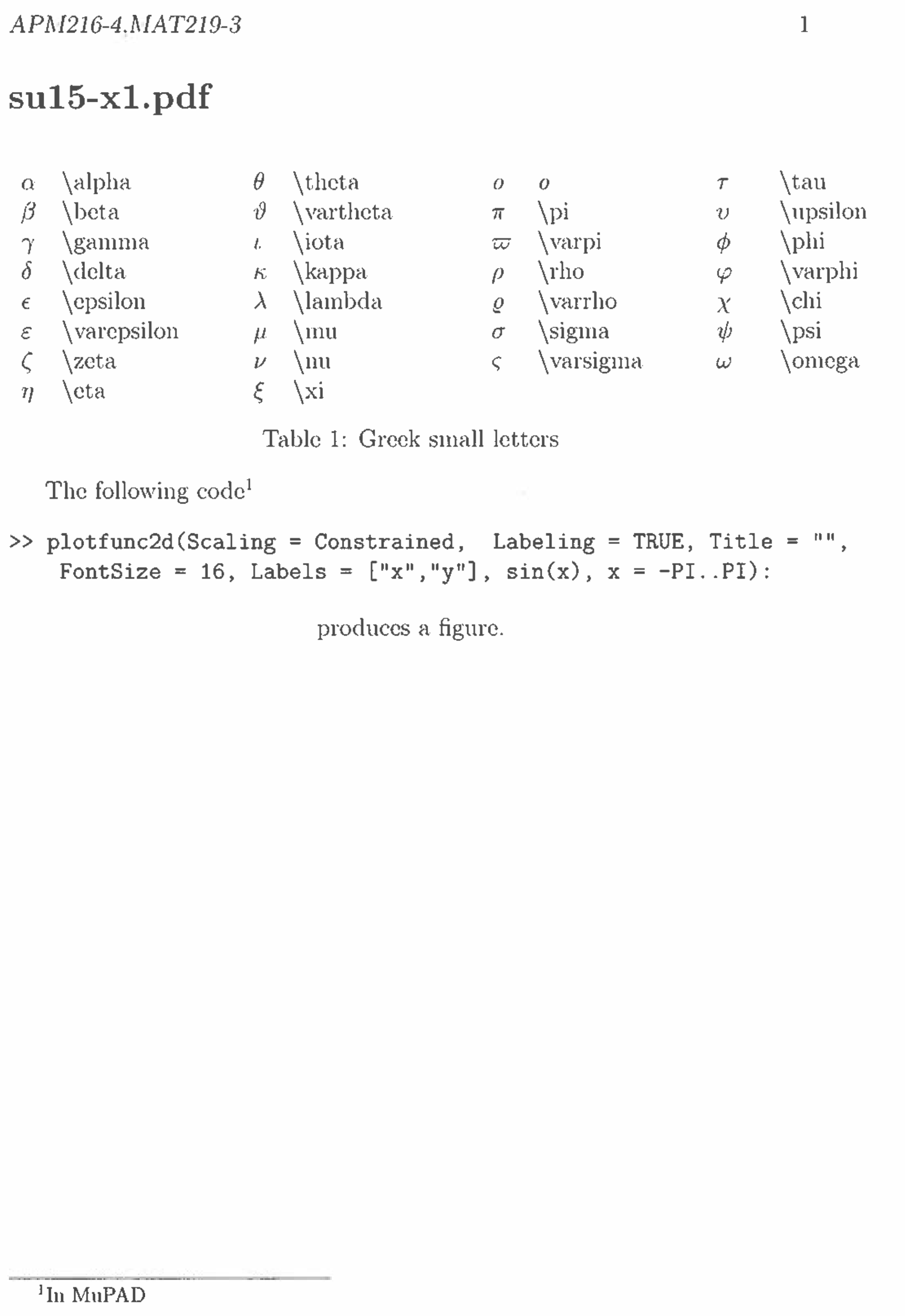
1. It is easy to make a footnote[1] in a LaTeXdocument.

2. It is also easy to arrange

   to centre a piece of text.

3. Text can be in various different sizes.

[1] Here it is



# su15-x1.pdf

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\alpha$ | \alpha | $\theta$ | \theta | $o$ | $o$ | $\tau$ | \tau |
| $\beta$ | \beta | $\vartheta$ | \vartheta | $\pi$ | \pi | $\upsilon$ | \upsilon |
| $\gamma$ | \gamma | $\iota$ | \iota | $\varpi$ | \varpi | $\phi$ | \phi |
| $\delta$ | \delta | $\kappa$ | \kappa | $\rho$ | \rho | $\varphi$ | \varphi |
| $\epsilon$ | \epsilon | $\lambda$ | \lambda | $\varrho$ | \varrho | $\chi$ | \chi |
| $\varepsilon$ | \varepsilon | $\mu$ | \mu | $\sigma$ | \sigma | $\psi$ | \psi |
| $\zeta$ | \zeta | $\nu$ | \nu | $\varsigma$ | \varsigma | $\omega$ | \omega |
| $\eta$ | \eta | $\xi$ | \xi | | | | |

Table 1: Greek small letters

The following code[1]

```
>> plotfunc2d(Scaling = Constrained,  Labeling = TRUE, Title = "",
    FontSize = 16, Labels = ["x","y"], sin(x), x = -PI..PI):
```

produces a figure.

[1] In MuPAD

# STUDY UNIT 16

This study unit comprises a project based on all the previous study units. The project varies from year to year, and details will be given in a Tutorial Letter.