

Apostolos Syropoulos
Richard W. D. Nickalls

The single biggest problem we face is that of visualisation.

Richard P. Feynman (1918–1988)

The Mathematical Gazette (1996); 80, 267

mathsPIC_{Perl} version 1.13

Apostolos Syropoulos,
Greek T_EX Friends,
366, 28th October Street,
GR-671 00 Xanthi, Greece.
asyropoulos@yahoo.com

<http://ocean1.ee.duth.gr/~apostolo>

and

Richard W. D. Nickalls,
Department of Anaesthesia,
Nottingham University Hospitals,
City Hospital Campus,
Nottingham, UK.
dick@nickalls.org

<http://www.nickalls.org/dick/>

as & rwdn

26 April 2010

T_EX Users Group: <http://www.tug.org/>
T_EX Usenet group: `comp.text.tex`

TUGboat: <http://www.tug.org/TUGboat/>
The PracT_EX Journal: <http://www.tug.org/pracjournal/>

CTAN (Comprehensive T_EX Archive Network)
<ftp://ftp.tex.ac.uk/> <http://www.tex.ac.uk/>
<ftp://ftp.ctan.org/> <http://www.ctan.org/>

mathsPIC_{perl}: CTAN: `/graphics/mathspic/perl/`

Typeset in
Times 10-point font
using L^AT_EX 2_ε

Cover figure by
František Chvála

Copyright © A Syropoulos & RWD Nickalls 26 April 2010

mathsPIC is released under the terms of the L^AT_EX Project Public License.
mathsPIC is distributed without any warranty or implied warranty of
merchantability or fitness for a particular purpose.

Contents

1	Introduction	1
2	Running <code>mathsPIC_{Perl}</code>	5
2.1	Installation	5
2.1.1	The <code>mathsPIC</code> package	5
2.1.2	Unix/Linux platform	6
2.1.3	MS-Windows platform	7
2.1.4	<code>mathspic.sty</code>	7
2.1.5	<code>mathspicX.pl</code>	7
2.2	Command-line syntax	7
2.3	Files	8
2.3.1	Filename extensions	8
2.4	Switches	9
2.5	Removing comment lines	9
2.6	Online help	9
2.7	The <code>mathsPIC</code> style-option	10
2.8	Error-messages	11
2.9	Log-file	12
3	The <code>mathsPIC</code> script file	13
3.1	<code>mathsPIC</code> style option	13
3.2	Headers and footers	13
3.3	Commands	15
3.4	Macros	16
3.4.1	Macro library	17
3.5	The plotting area	17
3.5.1	Axes	17
3.5.2	Second y-axis	20
3.5.3	Units	21
3.5.4	Tick-marks	22
3.6	Points	22
3.6.1	Point-name	22
3.6.2	Point-symbol	22
3.6.3	Line-free zone	23
3.6.4	Order of points	26
3.7	Lines	26
3.7.1	Line thickness	26
3.7.2	Recommendations	28

3.8	Text	29
3.9	Variables and constants	30
3.9.1	Scalar variables	30
3.9.2	Scalar constants	30
3.9.3	Mathematics	30
3.9.4	Scientific notation	31
3.10	The LOOP environment	32
4	mathsPIC commands	34
4.1	Mathematics	34
4.1.1	Macros	35
4.1.2	Making a macro library	36
4.2	Command definitions	37
4.2.1	Backslash	37
4.2.2	ArrowShape	38
4.2.3	beginLoop ... endLoop environment	38
4.2.4	beginSkip ... endSkip environment	38
4.2.5	Const	38
4.2.6	DashArray	39
4.2.7	DrawAngleArc	39
4.2.8	DrawAngleArrow	40
4.2.9	DrawArrow	40
4.2.10	DrawCircle	41
4.2.11	DrawCircumcircle	41
4.2.12	DrawCurve	41
4.2.13	DrawExcircle	42
4.2.14	DrawIncircle	42
4.2.15	DrawLine	42
4.2.16	DrawPerpendicular	42
4.2.17	DrawPoint	43
4.2.18	DrawRightangle	43
4.2.19	DrawSquare	43
4.2.20	DrawThickArrow	43
4.2.21	DrawThickLine	44
4.2.22	InputFile	44
4.2.23	LineThickness	45
4.2.24	Loop environment	45
4.2.25	Paper	46
4.2.26	Point	47
4.2.27	PointSymbol	49
4.2.28	Skip environment	49
4.2.29	Show...	50
4.2.30	System	51
4.2.31	Text	51
4.2.32	Var	52
4.3	Summary of mathsPIC commands	53
5	PiCTeX commands	56
5.1	Useful PiCTeX commands	57
5.2	Using the \$ symbol with PiCTeX	58

6	T_EX and L^AT_EX commands	60
6.1	Headers and footers	60
6.2	<code>thispagestyle{}</code>	61
6.3	<code>typeout{}</code>	61
6.4	The Color package	61
6.5	Other useful L ^A T _E X commands	62
7	Examples	63
7.1	Input- and output-files	63
7.2	Line modes	67
7.3	Arrows	70
7.4	Circles & colour	73
7.5	Functionally connected diagrams	77
7.6	Inputting the same data-file repeatedly	78
7.6.1	Using the <code>beginloop...endloop</code> environment	81
7.6.2	Using L ^A T _E X to cycle a loopcounter	82
7.7	Plotting graphs	83
7.8	Drawing other curves	86
7.9	Scaling	89
7.10	Using Perl programs	89
7.10.1	Example-1	89
7.10.2	Example-2	93
7.10.3	Commands for processing the files	99
8	Accessing T_EX parameter values¹	100
8.1	Useful T _E X commands	100
8.2	Outputting data to a file	101
8.3	The final code	103
9	Miscellaneous	106
9.1	Acknowledgements	106
9.2	Feedback	106
9.3	Development history	106
A	Tables	107
B	Arrows	110
C	Positioning the figures	113
C.1	Generating PS, EPS, PDF, JPEG files.	113
C.2	Using the <code>figure</code> environment	114
C.3	Using the PiCT _E X code	117
D	Installing Perl in MS-Windows	118
D.1	Perl	118
D.2	Text editors	121

¹Much of this chapter was originally presented by RWD Nickalls to the November 2001 ukTUG meeting.

E PiCTeX	122
E.1 The original files (1986)	122
E.2 The new updated files (1994)	123
E.3 Pictex2.sty	124
E.4 Errorbars.tex	124
E.5 DCpic	125
E.6 The PiCTeX Manual	125
References	127

Chapter 1

Introduction

MathsPIC_{Perl} is an open source Perl program for drawing mathematical diagrams and figures¹ (Nickalls, 1999; Syropoulos and Nickalls, 2000, 2005, 2007). More specifically, MathsPIC is a ‘filter’ program which parses a plain text input-file (known as the mathsPIC or .m file), and generates a plain text output-file containing T_EX, L^AT_EX and P_AC_TE_X commands for drawing a diagram. This output-file—which has by default an .mt filename-extension as it is really a T_EX file—can then be processed by T_EX or L^AT_EX in the usual way. It is anticipated that future versions will be able to generate output files in PostScript and SVG code.

Spaces and the comment % symbol are used by mathsPIC in the same way as T_EX. However, mathsPIC commands are *not* case-sensitive, and do not have a leading backslash (in order to distinguish them from P_AC_TE_X, T_EX and L^AT_EX commands, all of which can be freely used in the mathsPIC script file). MathsPIC also returns various parameter values in the output-file, e.g., angles, distances between points, center and radius of inscribed and exscribed circles, areas of triangles etc., since such values can be useful when making adjustments to a diagram.

The original motivation for mathsPIC arose from the need for an easy-to-use filter program for P_AC_TE_X, a versatile and freely available drawing package which offers the convenience of graphics code within the T_EX document itself (e.g., printer-independence). In this way, MathsPIC overcomes the main disadvantage of P_AC_TE_X, which is that P_AC_TE_X requires you to specify the coordinates of all the points. Consequently, this can make P_AC_TE_X extremely awkward to use when making complicated diagrams, particularly if several coordinates have to be re-calculated manually each time the diagram is adjusted.

For example, suppose it is necessary to draw a triangle ABC with AB 5 cm, AC 3 cm, and included angle BAC 40 degrees, together with its incircle. One such triangle is shown in Figure 1.1 and the P_AC_TE_X commands for drawing it are as follows (point A is at the origin (0,0) and the units are in cm).

¹The first version of mathsPIC (an MS-DOS version) was presented at the 1999 EuroT_EX conference in Heidelberg, Germany (Nickalls, 1999). A much improved Perl version was then developed, and a ‘work-in-progress’ presentation was given at the 2000 EuroT_EX meeting in Oxford, UK (Syropoulos and Nickalls, 2000). The first Perl version (1.0) was eventually uploaded to CTAN in 2005. The mathsPIC_{Perl} program is written in Perl 5.8.2 built for i86pc-Solaris. MathsPIC_{Perl} can be freely downloaded from CTAN (<http://www.ctan.org/tex-archive/graphics/mathspic/perl/>).

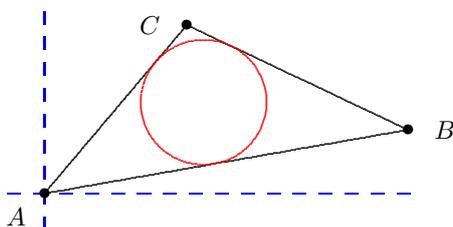


Figure 1.1:

```

\put {$\bullet$} at 0 0 % point A
\put {$\bullet$} at 4.924039 0.8682409 % point B
\put {$\bullet$} at 1.928363 2.298133 % point C
\plot 0 0 4.924039 0.8682409 1.928363 2.298133 0 0 /
\circulararc 360 degrees from 3.0086 1.2452 center at 2.1568 1.2452
\put {$A$} at -0.5 0
\put {$B$} at 5.424039 .8682409
\put {$C$} at 1.428363 2.298133

```

Although point A can be placed at the origin for convenience we have to resort to geometry and a calculator to determine the coordinates of points B and C , since in this example AB , AC , and the included angle are all defined (see above). It is then necessary to recall the coordinates of all the points in order to write the `\plot` command. Finally, the `\circulararc` command requires even more geometry and calculation to figure out the radius of the incircle, the coordinates of its center, and the coordinates of the starting point of the arc-drawing routine. Furthermore, if the initial diagram is not a suitable shape or size, the calculator has to be used again for any adjustments. In practice, therefore, $\text{P}\text{T}\text{E}\text{X}$ requires a certain amount of planning and calculation for all but the simplest of diagrams.

MathsPIC overcomes all these difficulties by providing an environment for manipulating named points and variables, which has the effect of making even very complicated mathematical diagrams easy to create. For example, the equivalent mathsPIC commands (which do not have a leading backslash) for drawing Figure 1.1 are as follows (the units are in cm as before).

```

point(A){0,0} % A is at origin
point(B){A,polar(5,10 deg)} % B is 5 cm from A; AB slope 10 deg
point(C){A,polar(3,50 deg)} % C is 3 cm from A; BAC = 40 deg
drawPoint(ABC) % put $\bullet$ at points A B C
drawline(ABCA) % draw the triangle
drawIncircle(ABC) % draw the incircle
var d = 0.5 % let d = 0.5 cm
text($A$){A, polar(-d,-140 deg)} % label for A
text($B$){B, shift(d,0)} % label for B
text($C$){C, shift(-d,0)} % label for C

```

Full mathsPIC file

The complete mathsPIC file for drawing Figure 1.1 is shown below. The style-option `mathspic.sty` automatically loads $\text{P}\text{T}\text{E}\text{X}$, and also implements the $\text{L}\text{A}\text{T}\text{E}\text{X}$ command `\thispagestyle{empty}` to stop a page number appearing in the figure.

```

% mpicpm01-1.m
\documentclass[a4paper]{article}
\usepackage{mathspic,color}
\begin{document}
%-----
\beginpicture
\setdashes
\color{blue}%
paper{units(1cm) xrange(-0.5,5), yrange(-0.5,2.5) axes(XY)}
\setsolid
point(A){0,0}
point(B){A, polar(5, 10 deg)}
point(C){A, polar(3, 50 deg)}
\color{black}%
drawpoint(ABC)
drawLine(ABCA)
\color{red}%
drawIncircle(ABC)
\color{black}%
var d = 0.5
text({A$}){A, polar(d,-140 deg)}
text({B$}){B, shift(d,0)}
text({C$}){C, shift(-d,0)}
\endpicture
\end{document}

```

All that remains is to generate the PS file using `dvips`, crop the image to the bounding box (using the `-E` option) and generate the EPS and PDF versions (see Section C.1). For example, if the mathsPIC file is, say, `triangle.m`, then the commands would be as follows:

```

mathspic triangle.m
latex triangle.mt
dvips triangle.dvi -o triangle.ps
dvips -E triangle.dvi -o triangle.eps
epstopdf triangle.eps

```

Unique facilities

MathsPIC facilitates the drawing of P_{CT}E_X diagrams because not only does it allow points to be defined in terms of other points (relative addressing), but it also allows the use of scalar variables which can be manipulated mathematically. Consequently, diagrams can be constructed in an intuitive way, much as one might with a compass and ruler; for example, constructing a point at a certain position in order to allow some other point to be constructed, perhaps to draw a line to. In other words, mathsPIC offers the freedom to create ‘hidden’ points having a sort of scaffolding function. In particular, this facility allows diagrams to be constructed in such a way that they remain functionally connected even when points are moved (see Section 7.5).

MathsPIC offers a number of other facilities, which allow the accurate drawing of extremely complicated diagrams. Since mathsPIC itself is a Perl program, both point

coordinates and variables can be defined using all the usual mathematical functions and syntax we associate with Perl. Files can be input recursively (using the `inputfile` command); there is a `do`-loop facility; and macros can be defined. In fact two sorts of macros can be used in the `mathsPIC` file, namely (a) special `mathsPIC` macros, and (b) the familiar TeX macros. Macros can also be stored in a library file (an ordinary ASCII text file) and input as and when necessary. Furthermore, the standard L^AT_EX packages can of course be used; e.g., the `Color` package and the `Rotation` package.

A powerful feature of `mathsPIC` is its facility for accessing the Perl command-line. This allows users to write their own dedicated Perl programs for writing configurable chunks of `mathsPIC` code on-the-fly to files which can then be input when required; for example, to draw particular elements of a diagram or even complete diagrams (see Section 7.10). The ability to use Perl programs in this way is, therefore, equivalent to having a powerful subroutine facility. It follows, therefore, that users can create their own library of useful Perl programs as well as libraries of `mathsPIC` macros.

Finally, note that `mathsPIC` can also be viewed as a handy tool for exploring geometry since its `show` commands return the values of various parameters; for example, angles, the distance between points, and areas of triangles.

Chapter 2

Running mathsPIC_{Perl}

MathsPIC_{Perl} is a Perl program and will therefore run on any platform on which Perl is installed. Apart from minor platform differences regarding filename conventions, commands for creating, editing and deleting ASCII files and so on, the practicalities of running and using mathsPIC will be essentially the same whichever platform is being used.

MathsPIC was developed using both GNU Linux and a Solaris x86 box, and consequently some of the command-line codes may reflect this perspective. For example, the following mathsPIC command to delete the file `temp.txt`

```
system("rm temp.txt")
```

uses the Unix¹ ‘remove’ command `rm`. Clearly this particular command will differ between platforms but it is assumed that users are familiar with their own local system commands. MS-Windows users may wish to look at Appendix D in which we address installing Perl on a MS-Windows platform. The authors welcome any relevant platform-related information so we can include it in updates to this manual.

2.1 Installation

If you have a recent T_EXLive installation (2009 or later) then both mathsPIC and P₁CT_EX will already be installed. However, mathsPIC may not be immediately runnable from the command-line as this depends on the status and location of the Perl program `mathspicX.pl` and the associated BASH file `mathspic.sh`.

2.1.1 The mathsPIC package

The latest version of mathsPIC_{Perl} can be downloaded from the following directory in CTAN.

```
CTAN: /tex-archive/graphics/mathspic/perl/
```

¹The term “Unix” here is used as a synonym for both Unix systems (e.g., Solaris, TrueUnix) and Unix-like systems (e.g., Linux, FreeBSD).

List of files

grabtexdata.pl	% program used in Chapter 8
HELP.TXT	% text version of the Unix manpage
mathspicX.pl	% mathsPIC program version X (perl)
mathspic.sty	% style option
mathspic.1	% unix/Linux manpage file
mathspic.sh	% example BASH file for running mathsPIC
MATHSPIC.BAT	% example batch file (MS-Windows)
mathsPICmanual.pdf	% manual
mathsPICmanual.zip	% manual--all the LaTeX and figure files
mathsPICfigures.zip	% all the .m and .pl files described in the manual
mathspic.lib	% example library file of macros
README.txt	% this file
sourcecode.pdf	% source code
sourcecode.html	% source code
sourcecode.nw	% source code (noweb format)

2.1.2 Unix/Linux platform

Note that T_EXLive generally places non-executable mathsPIC scripts and programs (e.g., `mathspic13.pl` and `mathspic.sh`) in a ‘scripts’ directory, for example `/usr/local/texlive/2010/texmf-dist/scripts/mathspic/` and executable ones into `/usr/local/texlive/2010/bin/i386-linux/`

Consequently, this last directory needs to be included in the `$PATH`.

When installing mathsPIC manually, place the files as follows (and then update your system’s T_EX-file index, by invoking the `texhash` command, so the new files can be found by the system).

- **L^AT_EX package** (`mathspic.sty`)

This file needs to be placed either (a) with the T_EX style-options (typically in the directory `/usr/share/texlive/20XX/texmf-dist/tex/latex/mathspic/`) or possibly (b) among your ‘local’ packages (for example in: `/usr/local/texlive/texmf-local/tex/latex/local/mathspic/`).

- **The mathsPIC program** (`mathspicX.pl`)

This file should be placed where your shell can find it (typically in the directory `/usr/local/bin/`).

- **The mathsPIC BASH script** (`mathspic.sh`)

This file should be placed where your shell can find it (typically in the directory `/usr/local/bin/`).

- **Unix man page** (`mathspic.1`)

This useful quick-reference file should be placed in the directory where the man pages of your installation reside (typically in `/usr/share/man/man1/`). Following this, you will be able to access the mathsPIC man page by typing the command

```
$ man mathspic
```

2.1.3 MS-Windows platform

Since mathsPIC is a Perl program, if Perl is not already installed then see Appendix D for directions on how to install a free version of Perl.

2.1.4 mathspic.sty

Locate the directory containing all the L^AT_EX packages, and create a mathspic subdirectory. Copy into this new directory the file mathspic.sty.

2.1.5 mathspicX.pl

Locate the directory containing all the Perl .pl programs (typically the directory c:\perl\bin\ for the ActivePerl implementation of Perl) and copy into this directory the file mathspic.pl (see Appendix for details regarding installing Perl on MS-Windows platforms).

2.2 Command-line syntax

The general command-line syntax is as follows.

```
$ perl [<perl switches>] mathspic.pl [<mathspic switches>] <inputfile> [-o <outputfile>]
```

By default mathsPIC writes the output to a file having the same filename as the input file, but with the filename extension .mt. If you forget to type an input filename, then mathsPIC writes the following line to the screen.

```
mathspic version 1.13 Apr 26, 2010
Usage: mathspic [-h] [-b] [-c] [-o <out file>] <in file>
```

If you type \$ mathspic -h you get all the switches information as well

```
This is mathspic version 1.13 Apr 26, 2010
Type "man mathspic" for detailed help
Usage: mathspic [-h] [-b] [-c] [-o <out file>] <in file>
where,
  [-b]    enables bell sound if error exists
  [-c]    disables comments in output file
  [-h]    gives this help listing
  [-o]    creates specified output file
```

There are several ways of running mathsPIC from the commandline, as follows (at the time of writing the current version of the program is mathspic113.pl):

a—Invoking the Perl interpreter

The minimum command required to process the script file myfile.m is

```
$ perl mathspic113.pl myfile.m
```

b—Making the mathspicX.pl program executable

This is probably the most efficient approach, and is therefore the recommended method.

- 1 Rename the mathsPIC program (mathspic113.pl) to just *mathspic*;

```
$ mv mathspic113.pl mathspic
```

- 2 Now make the file executable, by typing the command:

```
$ chmod 755 mathspic
```

and then copy it to where the shell can find it (typically `/usr/local/bin/`). Now mathsPIC can be run from any directory by typing

```
$ mathspic myfile.m
```

Note that this is the approach adopted by T_EXLive, which places executable files in the directory,

```
/usr/local/texlive/2010/bin/i386-linux/
```

which therefore needs to be included in the \$PATH.

c—Using a batch file

Alternatively, mathsPIC can be run via a batch file, and an example BASH file is included in the package (mathspic.sh). Rename the BASH file to mathspic; make it executable; and then copy it to where the shell can find it (typically `/usr/local/bin/`). Now mathsPIC can be run from any directory just by typing

```
$ mathspic myfile.m
```

2.3 Files

2.3.1 Filename extensions

Throughout this manual we use the filename extension `.m` to denote a mathsPIC script file (input-file), purely in order to distinguish them from the other files which are generated by mathsPIC. In practice, though, the mathsPIC program will accept *any* filename-extension preferred by the user². However, mathsPIC by default uses the extension `.mt` for the T_EX output-file (a T_EX file containing P_IC_TE_X commands ready for T_EXing), and the extension `.mlg` for the log-file.

The mathsPIC command-line actions one input file and (optionally) one output filename (prefixed by the `-o` switch). Each command or switch must be separated by at least one space, as in the following example.

```
$ mathspic inputfile -o outputfile
```

²For example, there is an obvious clash with the extension used by the program *Mathematica*, so in this case a user could consider, perhaps, the alternative extension `.mp` for a mathsPIC script file.

If the `outfile` is not specified then `mathsPIC` will create an output-file having the same filename as the `infile` but with the filename extension `.mt`. For example, if you want an input-file called `myinfile.abc` to generate an output-file called `myoutfile.xyz` then use the following command.

```
$ mathspic myinfile.abc -o myoutfile.xyz
```

MathsPIC also writes to a log file (the `.mlg` file). This has the same filename as the input filename.

2.4 Switches

There are four command-line switches (`-h -c -b -o`) which are case-sensitive. If more than one switch is used then they must be separated by at least one space. The switches are as follows (this information is also given by typing `$ mathspic -h`).

- h Help—gives basic information
- b Beep—a beep is sounded if `mathsPIC` detects an error. If an audible beep does not sound in the presence of an error, then check the PC configuration to see if the PC beep is disabled.
- c Disables comment line generation in the output file.
- o Output file name

2.5 Removing comment lines

Once a diagram has been finalised, it is sometimes convenient to remove all the various commented lines from the final output-file, particularly if the file is a large one. This can be easily done using the `-c` switch, which will stop `mathsPIC` writing any comments to the output `.mt` file. For example, the following command disables the writing of comment lines to the output `.mt` file.

```
$ mathspic -c inputfilename
```

2.6 Online help

In Unix systems typing the command

```
$ man mathspic
```

will open the manpage help file. If this fails, then check that the man page file (`mathspic.1`) has been placed in the correct directory (typically `/usr/share/man/man1/`). MS-Windows users can read the equivalent file `HELP.TXT`.

2.7 The mathsPIC style-option

Since some \LaTeX commands are redefined by mathsPIC it is *always* necessary to use the mathsPIC style-option by using the following command in the preamble of the \LaTeX document.

```
\usepackage{mathspic}
```

If the color package is used, then it must be loaded *after* the mathspic package, as follows:

```
\usepackage{mathspic, color}
```

Since mathsPIC is only used to generate figures and diagrams, the style-option also includes the \LaTeX command `\thispagestyle{empty}` in order to stop page numbers appearing in a figure. Note also that if page numbering is not disabled when creating a diagram, then the page number will prevent the DVIPS BBox -E option (see Section C.1) from removing all the white space from between the figure and the number. However, there are instances when it may be useful to comment out the `\thispagestyle{empty}` command, as detailed in Section C.3.

The style-option also defines the macros for various forms of the logo (e.g., mathspIC, mathsPIC_p, mathsPIC_{perl}).

The style-option is as follows:

```
%%
%% This is file 'mathspic.sty',
%% April 26, 2010
%%
%% (c) copyright 2004-2010 A Syropoulos & RWD Nickalls
%% This program can be redistributed and/or modified under the terms
%% of the LaTeX Project Public License Distributed from CTAN
%% archives in directory macros/latex/base/lppl.txt; either
%% version 1 of the License, or any later version.
%%
%% Please report errors or suggestions for improvement to
%%
%% Apostolos Syropoulos          Dick Nickalls
%% apostolo@yahoo.com           dick@nickalls.org
%%
\wlog{Package 'mathspic' A Syropoulos & RWD Nickalls (April 2010)}%
\typeout{Loading mathsPIC package (c) RWD Nickalls & A Syropoulos%
        April 2010}%
\ProvidesFile{mathspic.sty}%
  [2010/04/26 v1.13 Package 'mathspic.sty'%
\def\fileversion{1.13}
\def\filedate{2010/04/26}
\ifx\fiverm\undefined
  \newfont\fiverm{cmr5}
\fi
\let\Linethickness\linethickness%
%-----mathsPIC logos-----
\newcommand{\mathsPIC}{\textsf{mathsPIC}}
```

```

\newcommand{\MathsPIC}{\textsf{MathsPIC}}
\newcommand{\mathsPICp}{\textsf{mathsPIC}%
    \kern-0.08em\raisebox{-0.15em}{\textit{\tiny P}}}
\newcommand{\MathsPICp}{\textsf{MathsPIC}%
    \kern-0.08em\raisebox{-0.15em}{\textit{\tiny P}}}
\newcommand{\mathsPICperl}{\textsf{mathsPIC}%
    \kern-0.08em\raisebox{-0.15em}{\textit{\tiny Perl}}}
\newcommand{\MathsPICperl}{\textsf{MathsPIC}%
    \kern-0.08em\raisebox{-0.15em}{\textit{\tiny Perl}}}
%%-----PICTEX stuff-----
\input prepictex
\input pictexwd
\input postpictex
%%-----
\thispagestyle{empty}% stop page numbers
\endinput
%%
%% End of file 'mathspic.sty'.

```

2.8 Error-messages

A certain amount of syntax checking is performed by `mathsPIC`, and error-messages are written to the main output-file (`.mt` file) and also to the log-file (`.mlg` file).

A copy of the line containing an error is prefixed by `%% ***`; the associated error-message appears on the next line and is prefixed by dots (`%% ...`). Note that if the `-c` switch (`clean`) is used (see above), then error-messages appear only in the log file. If the `-b` switch is used then a beep is sounded if an error occurs during processing.

Runtime errors most commonly arise when an argument has been omitted, or division by zero has been attempted. Syntax errors arise when `mathsPIC` commands are written incorrectly (e.g. missing bracket, or a command being spelled incorrectly).

The following lines show some typical examples with respect to a `draw` command if a point has not been previously defined resulting in a 'point-name' error:

```

%% drawline(AB)
%% *** Line 15: drawline(AB
%% ***                )
%% ... Error: Undefined point B

```

Some other examples of error-messages are as follows.

```

%% drawline()
%% *** Line 22: drawline(
%% ***                )
%% ... Error: Wrong number of points

```

```

%% *** Line 49:
%% ***                pointt(K2){4,6}
%% ... Error: command not recognized

```

```
%% drawline(PQ)
%% *** Line 52 : drawline(PQ
%% ***           )
%% ... Error: points P and Q are the same
```

Since an error often has an effect on the processing of later commands in the script (mathsPIC file), a single error can result in a cascade of error messages. In other words, the number of error messages is not a good guide to the number of errors—there may only be one small typo causing all the error messages.

2.9 Log-file

mathsPIC outputs a log-file (.mlg file) which contains details of all errors, relevant line numbers and file names. The format was designed to match that of a standard \TeX log-file in order to be compatible with commonly used error-checking utilities.

For example, in the following code the errors in the 2nd, 3rd and 4th lines (combined with the fact that point B has not been defined) generate the error messages found in the associated log file (.mlg file) below.

```
%% test.m
point(A){5,5}
var d=
drawline(AB
var j=
text($B$){B, shift(d,0)}
```

The log file (.mlg file) shows the following:

```
-----
2011/05/29 14:42:41
mathspic (Perl version 1.13 Apr 26, 2010)
Copyright (c) 2005-2010 A Syropoulos & RWD Nickalls
Input file = test.m
Output file = test.mt
Log file   = test.mlg
-----
Line 3: var d=
          ***Error: Unexpected token
Line 4: drawline(AB
          ***Error: Undefined point B
Line 4: drawline(AB
          ***Error: Wrong number of points
Line 4: drawline(AB
          ***Error: Missing ) after arguments of
Line 5: var j=
          ***Error: Unexpected token
Line 6: text($B$){B
          , shift(d,0)}
***Error: undefined point/var
```

Chapter 3

The mathsPIC script file

All commands for generating a diagram (the script) are written to a plain ASCII file, known as the mathsPIC file, using a text editor in the usual way. This file is then processed by the mathsPIC program (`mathspic.pl`) as described in the previous chapter. While the mathsPIC file can of course have any filename and extension, in this manual all mathsPIC files have the filename extension of `.m` in order to distinguish them from the various derived files.

We distinguish three types of command, namely (a) mathsPIC commands, (b) P_{CT}E_X commands, and (c) T_EX or L^AT_EX commands. All these commands are detailed in the subsequent three chapters. Since mathsPIC currently only outputs a T_EX file then the mathsPIC file can also contain appropriate P_{CT}E_X, T_EX and L^AT_EX commands¹.

3.1 mathsPIC style option

Since the style option not only inputs some P_{CT}E_X files but also redefines some of the P_{CT}E_X commands, it is therefore necessary to always use the mathsPIC style option (`mathspic.sty`) by using the command

```
\usepackage{mathspic}
```

Note that in the file `mathspic.sty` the line `\let\Linethickness\linethickness%` is placed before inputting `pictexwd.tex` since both P_{CT}E_X and the L^AT_EX picture environment use the same command name, i.e. `\linethickness`.

3.2 Headers and footers

It is particularly useful to include in the mathsPIC file any T_EX or L^AT_EX headers and footers which would otherwise have to be added manually to the output-file before L^AT_EXing the file. For example, a typical format which allows for both L^AT_EX 2_ε and pdfL^AT_EX processing², might be as follows (this works nicely with L^AT_EX 2_ε, `dvips`, `ps2pdf`, and `pdfLATEX` in Linux). Note that it is important to load the `color` package (see Section 6.4) *after* `mathspic`.

¹However, it is anticipated that future versions of mathsPIC will be able output PostScript and SVG files, in which case only mathsPIC commands will be allowed.

²The `\ifx...\else...\fi` sequence is from `science.sty` (available on CTAN).

Note also that the L^AT_EX command `\thispagestyle{empty}` is included for convenience in `mathspic.sty` in order to stop a page number appearing in a figure (see Section 2.7 for details).

```
\documentclass[a4paper]{article}
\usepackage{mathspic,color}
\begin{document}
%-----
\beginpicture
....
....
\endpicture
\end{document}
```

For users of plain T_EX a typical format might be as follows. Note the need in this case to include the line redefining the `\linethickness` command (in `mathspic.sty`)

```
\input latexpic.tex
\let\Linethickness\linethickness% %% redefinition for mathsPIC
\input pictexwd.tex
\font\tiny=cmr5 %% used for drawing lines
\font\large=cmr12 %% used for drawing thicklines
%-----
\beginpicture
....
....
\endpicture
\bye
```

If it is necessary (or just convenient) to extend a T_EX or L^AT_EX command across several lines, then each additional line must be protected within the `mathsPIC` file using a leading `_...` sequence unless a line actually starts with a T_EX command. A typical example is the following macro (used in Figure 7.6) which defines a ‘display’ maths formula. The macro is split across several lines, as follows.

```
\newcommand{\formula}{%
\ $ \displaystyle \sum_{p \ge 0} \Delta_{jp} z^{(p+1)} $%
\ }%
text(\formula){B1}
```

Important: Note that when using T_EX or L^AT_EX commands *within* the P_IC_TE_X picture environment (`\beginpicture ... \endpicture`), it is very important to include the comment `%` symbol at the end of such lines, to prevent P_IC_TE_X accumulating additional `<space>` characters from the ends of non-P_IC_TE_X commands, since otherwise P_IC_TE_X incorporates such space characters into the horizontal distance used for representing *x*-coordinates, with the effect that all subsequent picture elements may be displaced slightly to the right.

This effect relates to how T_EX expands macros, as explained in the following texhax comment by Philip Ratcliffe (May 8, 2007).

... a reasonable rule-of-thumb is to put a comment (`%`) at the end of every line that would otherwise end with a right brace (`}`). That is, write

```
\typeout{Hello World}%
```

To be more explicit, a simple command with no parameters, such as `\par`, `\else`, `\centering`, will always eat the following space while, say, `\typeout{Hello World}`, will not (i.e., the space will remain as a physical space in your output). An the end-of-line character is always interpreted as a space.

For other \TeX and \LaTeX commands useful in `mathsPIC` see Chapter 6.

3.3 Commands

The idea underlying the `mathsPIC` file (input-file) is that it should be able to contain everything required to generate the proposed figure (i.e. all `mathsPIC` commands, comments, \TeX and \LaTeX commands including headers and footers, $\P\TeX$ commands, as well as lines to be copied verbatim) so that the output-file can be immediately \TeX ed to generate the graphic. Some general points relating to the `mathsPIC` file are as follows.

- `mathsPIC` commands are *not* prefixed by backslashes. They are therefore easily distinguished from \TeX , \LaTeX and $\P\TeX$ commands.
- Each `mathsPIC` command must be on a separate line. This is because `mathsPIC` frequently adds data to the end of a line in the output-file (see below).
- As with \TeX , spaces can be used to enhance readability. This is particularly useful when writing lists of points. For example the command `drawpoint(PQR1R2)` can be made easier to read by writing it as `drawpoint(P Q R1 R2)`.
- `mathsPIC` commands and point-names are *not* case sensitive. This allows the user to customise the commands to enhance readability. Thus the command `drawpoint` can therefore be written as `drawPoint` or `DrawPoint` etc.
- Delimiters have a hierarchical structure as follows:
Curved brackets contain the primary argument; e.g. `drawline(AB)`
Braces contain required supporting arguments; e.g. `point(A){5,6}`
Square brackets contain optional arguments;
e.g. `point(D){midpoint(PQ)}[symbol=\odot]`
- Logically distinct groups within brackets must be separated by commas.
e.g. `point(B2){A,polar(3,40deg)}[symbol=\odot, radius=2]`
- Comments are prefixed by the `%` symbol in the usual way. Lines having a leading `%` symbol are copied verbatim through to the output-file.
- Lines having a leading backslash command (i.e. where there is *no* inter-word space immediately following the backslash, e.g. `\setdashes` or `\begin{document}`) are copied verbatim through to the output-file. Consequently, all \TeX , \LaTeX and $\P\TeX$ commands can be used in the normal way providing the command is restricted to a single line. However, if such commands do run on to the subsequent lines, these lines will need to be prevented from being processed as `mathsPIC` commands, by prefixing them with `_` (see below)—unless of course they also start with a backslash command.

- Lines having a leading `_` (i.e. where the `\` is followed immediately by one or more inter-word spaces `_` e.g. `\ 25.3 16.8`) are copied verbatim through to the output-file *without* the leading backslash.
- Data-files containing mathsPIC commands can be input using the `inputfile()` command. Files can also be input *verbatim* using the `inputfile*()` command (useful for inputting files containing only \P\TeX commands and/or coordinate data; for example, a list of data points as part of a \P\TeX `\plot` command).
- The `system()` command gives access to the command-line.

3.4 Macros

MathsPIC currently allows macros consisting of a single command, either with or without parameters. (see Section 4.1.1). MathsPIC macros are subject to a number of rules as follows:-

- Macros are created using the `%def` command, and destroyed using the `%undef` command.
- When a macro is used in a command then the macro-name *must* have a `&` prefix (to distinguish it as a macro).
- Macro names are case-sensitive (unlike all other mathsPIC command-names which are *not* case sensitive)
- Macros must evaluate to a ‘numerical expression’ (see Section 4.2) (i.e. not to strings).

It is strongly recommended that a `%` is placed at the end of the macro definition (as is done with \LaTeX commands) in order to prevent \P\TeX from including additional horizontal whitespace.

No parameters

Examples of macros which do not take any parameters are the following two commands which create the two macros `fancydashes` and `plainedashes`.

```
%def fancydashes() dasharray(1pt,2pt,3pt,4pt)%
%def plainedashes() dasharray(1pt,1pt)%
```

Note that in the macro-definition command the curved bracket (the parameter bracket) at the end of the word `fancydashes()` remains empty if there are no parameters. This pair of curved brackets marks the end of the command-name and the beginning of the macro definition (i.e. some mathsPIC commands).

To use the macro (as a command) it is necessary to use the `&` prefix. The `()` brackets are only necessary if the macro takes parameter(s), as follows.

```
...
&fancydashes
drawline(AB)
&plainedashes
drawline(PQ)
...
```

The macro `fancydashes()` is deleted using the following command.

```
%undef fancydashes()
```

Single parameter

An example of a macro taking a single parameter is as follows:

```
%def thick(t) linethickness(t pt)%
```

Here the command `&thick(2)` is equivalent to the command `linethickness(2pt)`.

Multiple parameters

An example of a macro taking multiple parameters is as follows:

```
%def mypoint(P,x,y) point(P){x,y}%
```

If we now write the command `&mypoint(Q7,3,5)` this command will be processed to generate the point $Q7$ as shown in the following comment in the output file.

```
%% point(Q7){3,5} Q7 = (3, 5)
```

It is anticipated that the macro facility will be upgraded in subsequent versions of `mathsPIC` to allow a multi-line macros facility.

3.4.1 Macro library

It may be useful to create a file for storing frequently used `mathsPIC` macros, say, the text file `mathspic.lib` (see more on macros in Section 4.1). For example, we could put together some of the macros described above into one file as follows:

```
%%--- mathspic.lib---
%def fancydashes() dasharray(1pt,2pt,3pt,4pt)%
%def plaindashes() dasharray(1pt,1pt)%
%%--- end of library ---
```

and input the file at the start (for processing by `mathsPIC`) by placing the command `inputfile(mathspic.lib)` in the `mathsPIC` file just after `\begin{document}`.

3.5 The plotting area

3.5.1 Axes

When drawing a new figure it is often useful to have a graduated ruled frame to guide placement of picture elements. This task is greatly simplified by using `mathsPIC`'s one-line `paper` command, which has optional `axes` and `ticks` parameters³. The axis-codes used in the `axes()` option are L (Left), R (Right), T (Top), B (Bottom), X (X-axis), Y (Y-axis). For example, the following `paper` command generates a drawing area 5 cm \times 5 cm with a ruled frame on four sides as shown in Figure 3.1a.

³Since `PTEX` uses the name 'axis', `mathsPIC` recognises both spellings ('axis' and 'axes').

```
paper{units(1mm),xrange(0,50),yrange(0,50),axes(LRTB),ticks(10,10)}
```

This particular paper command⁴ is converted by mathsPIC into the following P_{CT}E_X code in the output-file (.mt file).

```
\setcoordinatesystem units <1mm,1mm>
\setplotarea x from 0 to 50 , y from 0 to 50
\axis left ticks numbered from 0 to 50 by 10 /
\axis right ticks numbered from 0 to 50 by 10 /
\axis top ticks numbered from 0 to 50 by 10 /
\axis bottom ticks numbered from 0 to 50 by 10 /
```

For graphs it is more usual for the axes to be centered on the origin (0,0), and this is provided for by the XY options. For example, Figure 3.1b was generated using the following paper command,

```
paper{units(1cm),xrange(-2,3),yrange(-2,3),axes(XY),ticks(1,1)}
```

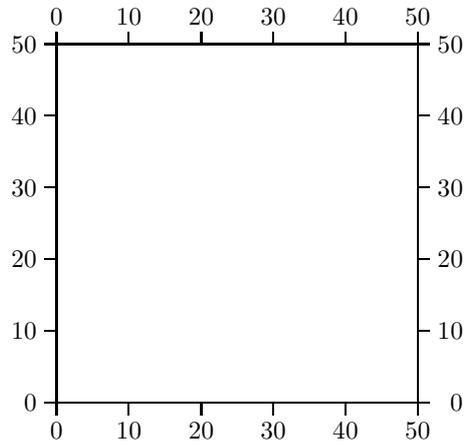
which is converted by mathsPIC into the following P_{CT}E_X code in the output-file.

```
\setcoordinatesystem units <1cm, 1cm>
\setplotarea x from -2 to 3, y from -2 to 3
\axis left shiftedto x=0 ticks numbered from -2 to -1 by 1
      from 1 to 3 by 1 /
\axis bottom shiftedto y=0 ticks numbered from -2 to -1 by 1
      from 1 to 3 by 1 /
```

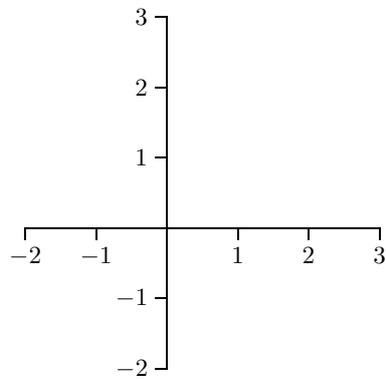
The tick-marks associated with an axis can be prevented by using a * after the axes-code (e.g. axis(LBT*R*) gives four axes but generates tick-marks only on the Left and Bottom axes). Note that any combination of axes-codes can be used. For example, the options ...axes(LRTBX*Y*), ticks(10,10) will generate a rectangular axes frame (with ticks) containing the XY axes (without ticks). The line-thickness of axes and tick-marks is controlled by the P_{CT}E_X \linethickness command.

Once the figure is finished, then the frame or axes can be easily adjusted or even removed. The figure can also be scaled in size simply by altering the units parameters. For example, the option units(3cm,1cm) will generate an X-axis having three times the scale as the Y-axis (see Figure 7.11). If complicated or more demanding axis configurations are required, then the P_{CT}E_X Manual (see Section E.6) will need to be consulted. See also Chapter C for details on positioning figures within L^AT_EX documents.

⁴Note that the order of the units(), xrange(), yrange(), ... etc in the paper() command is critical, and changing this order will generate an error message. This command will be made more flexible in future versions.



a. Using `...axes(LRTB)`



b. Using `...axes(XY)`

Figure 3.1:

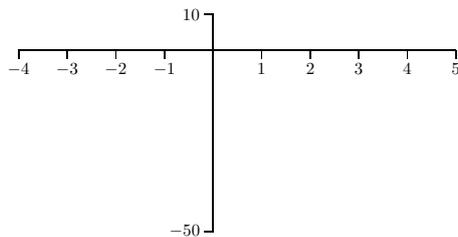
Example

Sometimes it is useful to use axes with a different scale in order to accommodate, say, an equation, as shown in the following example, where we have to specify the two Y-values and their positions on the axis.

```

\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
\setsolid
%% compress Y scale to fit equation onto paper
var u=1, v=0.075
paper{units(u cm, v cm), xrange(-4,5), yrange(-50,10), axes(XY), ticks(1,0)}
%% place values at ends of the Y-axis
\axis left shiftedto x=0
  \ ticks withvalues $-50$ $10$ /
  \ at -50 10 / /
\endpicture
\end{document}

```

**3.5.2 Second y-axis**

Sometimes a second (different) y-axis is needed (on the right). An example of how this can be achieved is as follows.

```

paper{units(1cm), xrange(0,6), yrange(72,77), axes(LBT*), ticks(1,1)}
\axis right
\ label {\lines {weight}\cr (lbs)\cr {\ } \cr {\ } }}
\ ticks withvalues 0 1 2 3 4 5 /
\ at 72 73 74 75 76 77 / /

```

Table 3.1: Conversion factors for the units used by L^AT_EX.
From: Beccari 1991 (with permission).

	mm	cm	pt	bp	pc	in	dd	cc	sp
1 mm	1,000	0,100	2,845	2,835	0,2371	0,03937	2,659	0,2216	186 467,98
1 cm	10,00	1,000	28,45	28,35	2,371	0,3937	26,59	2,216	1 864 679,8
1 pt	0,3515	0,03515	1,000	0,9963	0,08333	0,01384	0,9346	0,07788	65 536
1 bp	0,3528	0,03528	1,004	1,000	0,08365	0,01389	0,9381	0,07817	65 781,76
1 pc	4,218	0,4218	12,00	11,96	1,000	0,1660	11,21	0,9346	786 432
1 in	25,40	2,540	72,27	72,00	6,023	1,000	67,54	5,628	4 736 286,7
1 dd	0,3760	0,03760	1,070	1,066	0,08917	0,01481	1,000	0,08333	70 124,086
1 cc	4,513	0,4513	12,84	12,79	1,070	0,1777	12,00	1,000	841 489,04

3.5.3 Units

In addition to the usual units mm, cm, and pt, P_{CT}E_X accommodates all the other units used by T_EX (see Knuth (1990), Chapter 10), and also uses the same two-letter codes, namely pc (pica), in (inch), bp (big point), dd (didot), cc (cicero), sp (scaled point). The available units thus embrace the Metric system (mm, cm), the Didot system (didot, cicero), and the UK system (point, big point, pica, inch)—see Table 3.1.

Note that if only *one* unit is indicated in the `units` option, then mathspic uses the *same* unit for both the x and y axes. Thus the option `units(1mm)` in the `paper` command is translated by mathspic into the following P_{CT}E_X command in the output-file. Note that it is very important to include a numeric value with the units, or alternatively a variable with the units.

```
\setcoordinatesystem units <1mm,1mm>
```

If different scales are required (most commonly when drawing curves and equations) then both need to be specified in the mathspic `units` option. For example, if units of 1 cm and 2 mm are required for the x and y axes respectively, then this will be implemented by the mathspic command `units(1cm,2mm)`. However, when the x and y scales *are* different strange effects can occasionally occur, particularly if drawing ellipses or circular arcs. In view of this mathspic writes a warning note to the output-file and log-file when different units are being used. The drawing of complete circles will only be affected if the x -units is changed, since the mathspic starts the arc at a location having the same y -coordinate as that of the center. In general users are therefore recommended to avoid using different x and y units in the `paper` command if at all possible.

Note that variables can also be used to control the x and y units, as shown in the following example, where the radius (r) and the distance (s) between the label A and its point-location are fixed irrespective of scaling (i.e. with changes in the value of u) by dividing the relevant variables by the scaling value u .

```
var u = 1.5    %% units
paper{units(u mm), xrange(0,100), yrange(0,100)}
...
var r = 2/u, s = 4/u
point(A){30,20}[symbol=circle(r)]
text($A$){A, shift(-s,s)}
```

3.5.4 Tick-marks

It is recommended that integers are used with the `ticks()` option; the numeric format for the `ticks()` arguments must agree with that used by the `xrange()` and `yrange()` arguments. For example, if the `paper()` command uses `xrange(-5.5,5.5)` then the `ticks()` option should also use decimals, perhaps `ticks(1.0, 1.0)`. If this rule is not adhered to then \TeX may sometimes give unpredictable results when executing the `paper{}` command. In general \TeX gives more pleasing axes if integers are used throughout the `paper` command.

3.6 Points

Each point is associated with a point-name which is defined using the `point` command. For example, the following command allocates the point-name *A* to the coordinates (5,7).

```
point(A){5,7}
```

Once defined, points can be referred to by name. Consequently, points can be defined in relation to other points or lines simply by using point-names, as shown by the following commands.

```
point(C){midpoint(AB)}
point(E){intersection(AB,CD)}
point(J){Q, rotate(P, 25 deg)}  %% J = Q rotated about P by 25 deg
```

Points are interpreted according to their grouping and context. Thus two points represent either a line or its Pythagorean length. For example, the command `drawCircle(P, AB)` means draw a circle, center *P* with radius equal to the length of the line *AB*. A group of three points represents either a triangle, an angle, or two contiguous line-segments, depending on the circumstances.

3.6.1 Point-name

A point-name *must* begin with a *single* letter, and may have up to a *maximum* of three following digits. The following are valid point-names: *A, B, C3, d185*. Since `mathsPIC` is not case sensitive the points *d45* and *D45* are regarded as being the same point.

Sometimes it is necessary to re-allocate new coordinates to an existing point-name, in which case the `point*` command is used. This is often used during recursive operations whereby the `mathsPIC` file inputs another file (using the `inputfile` command) containing commands which alter the value of pre-existing points. For example, the following command increments the *x*-coordinate of point *A* by 5 units.

```
point*(A){xcoord(A)+5, ycoord(A)}
point(P){Q}  %% make P the same as Q
```

3.6.2 Point-symbol

The default point-symbol is \bullet (`\bullet`). However, `mathsPIC` allows the optional use of any \TeX character or string of characters to represent a particular point, by defining it in a following square bracket. For example, the point *A*(5,10) can be represented by the \triangle symbol by defining it as follows.

```
point(A){5,10}[symbol=$\triangle$]
```

Other examples using the `circle()` and `square()` options are:

```
point(B){A, shift(2,6)}[symbol=circle(2)]
point(C){A, polar(3,26)}[symbol=square(3)]
```

The argument for the circle is the radius, while the argument for the square is the side length.

The default point-symbol can also be changed to a circle, square, or any \TeX character or string of characters by using the `mathsPIC PointSymbol` command. Note that the `PointSymbol` command only influences subsequent `point` commands. For example, the character \odot (`\odot`) can be made the new global point-symbol by using the command `PointSymbol(\odot)`. The original default point-symbol (\bullet) can be re-instituted (reset) using the command `PointSymbol(default)`. The point-symbol is drawn at the point-location using the `drawPoint` command; for example, `drawPoint(A)`, or `drawPoint(ABCD)`.

Since most \TeX characters and symbols are typeset asymmetrically in relation to the baseline, they will not, in general, be positioned symmetrically over a point-location. Most characters are therefore not ideal for use as point-symbols, as they generally require some slight vertical adjustment in order to position them symmetrically. In view of this Table A.2 lists those \TeX characters which *are* particularly suitable, since they are automatically positioned by \TeX symmetrically with respect to a point-location (for example the \odot character `\odot`), and are therefore ideal for use in this setting.

3.6.3 Line-free zone

When lines are drawn to a point, the line will (unless otherwise instructed) extend to the point-location. However, this can be prevented by allocating an optional circular line-free zone to a point by specifying the line-free radius in a following square bracket. For example, lines to a \triangle symbol at point *A* can be prevented from being drawn through the triangle to its center by allocating a 5 unit line-free zone to the point, as follows.

```
point(A){3,10}[symbol=$\triangle$,radius=5]
```

If only the line-free radius is to be specified for the default point-symbol then we can use the command `pointsymbol(default,5)`. To change the line-free radius of an *existing* point then use the following command.

```
point*(A){A}[radius=10]
```

which is equivalent to

```
point*(A){xcoord(A), ycoord(A)}[radius=10]
```

Table 3.2 gives a list of useful point-symbols which \TeX places symmetrically over a point-location (note that `\Box` and `\Diamond` are not placed symmetrically over a point location, but `\square` and `\lozenge` are). Other useful symbols are available from the `textcomp` fonts⁵ (see also the ‘symbol list’ compiled by Pakin which shows virtually all the available symbols⁶).

⁵See Harold Harders’ useful file (`textcomp.tex`) which shows the characters of the `textcomp` font together with their names. It can be found at CTAN: `/tex-archive/info/textcomp-info/`.

⁶CTAN: `/tex-archive/info/symbols/`

For example, the following commands will draw lines between points ABC, such that the lines just touch the edge of the \odot point-symbol (line-free radius 1.2 mm; 10pt font).

```
pointSymbol( $\odot$ , 1.2)
point(A){1,1}
point(B){2,2}
point(C){1,3}
drawline(ABC)
```

Table 3.2: Useful point-symbols and their radii for 10–12pt fonts.

symbol		radius mm	symbol package
		10pt / 11pt / 12pt	
\circ	\circ	0.70 / 0.75 / 0.80	standard
\odot	\odot	1.20 / 1.35 / 1.50	standard
\oplus	\oplus	1.20 / 1.35 / 1.50	standard
\ominus	\ominus	1.20 / 1.35 / 1.50	standard
\oslash	\oslash	1.20 / 1.35 / 1.50	standard
\otimes	\otimes	1.20 / 1.35 / 1.50	standard
\bigcirc	\bigcirc	1.70 / 1.85 / 2.05	standard
\bigodot	\bigodot	1.70 / 1.85 / 2.05	standard
\bigoplus	\bigoplus	1.70 / 1.85 / 2.05	standard
\bigotimes	\bigotimes	1.70 / 1.85 / 2.05	standard
\star	\star	—	standard
\triangle	\triangle	—	standard
\square	\square	—	amssymb.sty
\blacksquare	\blacksquare	—	amssymb.sty
\lozenge	\lozenge	—	amssymb.sty
\blacklozenge	\blacklozenge	—	amssymb.sty
\bigstar	\bigstar	—	amssymb.sty
\boxdot	\boxdot	—	amssymb.sty
\boxtimes	\boxtimes	—	amssymb.sty
\boxminus	\boxminus	—	amssymb.sty
\boxplus	\boxplus	—	amssymb.sty
\divideontimes	\divideontimes	—	amssymb.sty

It is often useful to adjust the line-free radius associated with a particular point before drawing lines or arrows to it, in order to optimise the distance between an object centered at the point and the line or arrow. For example, one can use the `point*` command to set a line-free radius of 2 units for a pre-existing point (P), as follows.

```
point*(P){P}[radius=2]
```

By way of illustration, this command is used in drawing Figure 3.2 where arrows are being drawn from various directions (B , S) to a text box centered on point P \odot (the code is shown below as `mpicpm03-2.m`). By setting the line-free radius (dashed circles) associated with point P before drawing each particular arrow, one can easily adjust and optimise the distance between the arrowhead and the text box. The arrowshape used here is the default shape which is defined as follows (see Section 7.3 for details).

```
arrowshape(2mm,30,40)
```

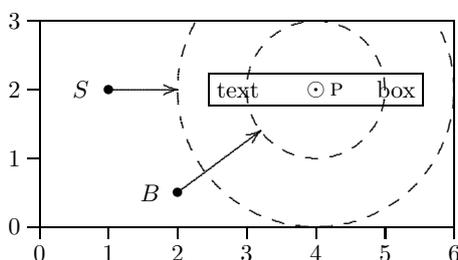


Figure 3.2:

```
%% mpicpm03-2.m (Figure 3.2)
\usepackage{mathspic}
%-----
\beginpicture
paper{units(1cm),xrange(0,6),yrange(0,3),axes(LBT*R*),ticks(1,1)}
point(P){4,2}[symbol={\odot}]
point(B){2,0.5}
point(S){1,2}
drawPoint(PBS)
\setdashes
\inboundscheckon      %% restrict circles to drawing area
drawcircle(P,1)
drawcircle(P,2)
\setsolid
%% change line-free radius of P to 1cm
point*(P){P}[radius=1]
drawArrow(BP)          %% draw arrow from B (below)
%% change line-free radius of P to 2cm
point*(P){P}[radius=2]
drawArrow(SP)          %% draw arrow from S (side)
text(\\$S$){S, shift(-0.4,0)}
text(\\$B$){B, shift(-0.4,0)}
text(\\textsc{p}){P, shift(0.3,0)}
```

```
\newcommand{\textbox}{\fbox{text\hspace{17mm}box}}%
text(\textbox){P}
\endpicture
```

Of course, sometimes it is convenient just to draw the arrows a certain length from one point towards another point. For example, in order to draw an arrow 1 unit long from point A towards point B we could use the following two commands

```
point(Z){pointonline(AB,1)} % Z is point 1 unit from A towards B
drawArrow(AZ)
```

3.6.4 Order of points

The order of points in mathsPIC commands is sometimes significant. For example, the command `point(D){PointOnLine(AB,23)}` defines the point D as being 23 units from A in the direction of B .

3.7 Lines

mathsPIC draws lines using its `drawLine` and `drawThickline` commands. For example, a line from P_1 to P_2 is drawn with the command `drawLine(P1P2)`. If a line is to be drawn through several points (say, J_1, J_2, J_3, J_4, J_5) and can be drawn without ‘lifting the pen’, then this can be achieved using the single mathsPIC command `drawLine(J1J2J3J4J5)`. Several unconnected lines can also be drawn using one command by separating each line sequence with a comma; for example, `drawLine(J1J2, J3J4J5, J1J3)`.

A line can also be drawn a specified distance from one point towards (or away from) another point, using the following two-step approach. For example, the following commands draw a line a distance d units from point A towards point B .

```
point(Z){pointonline(AB,d)}
drawline(AZ)
```

Note that the order of the points AB and the sign of the distance d are important. For example, the following commands will draw a line a distance d units from point B away from point A .

```
point(Z){pointonline(BA,-d)}
drawline(BZ)
```

Since the $\text{P}\text{T}\text{E}\text{X}$ `\putrule` command for drawing horizontal or vertical lines is much more memory efficient than the `\plot` command, mathsPIC automatically invokes the `\putrule` command for horizontal and vertical lines.

3.7.1 Line thickness

mathsPIC

mathsPIC uses the `linethickness()` command. For example, to switch to a linethickness of 2pt we would use the mathsPIC command

```
linethickness(2pt)
```

The default value is 0.4pt, and resetting to this value is achieved by the following command

```
linethickness(default)
```

Sometimes when drawing thick lines it is useful to be able to manipulate the line ends (e.g. when drawing shapes with horizontal and vertical lines). Consequently it is useful to be able to access the numeric value of the current `linethickness` (in the units defined by the `paper` command), and this can be done using the `var` command as follows.

```
var t = _linethickness_
```

The `mathsPIC drawLine()` command uses the current dot size. However, the `mathsPIC drawThickline()` command uses the `\large` dot size, but then resets the dot size to the default `\tiny`. For example, the commands

```
point(A){5,5}
point(B){10,10}
drawThickline(AB)
```

will result in the following code in the output-file.

```
%% point(A){5,5}      (5,5)
%% point(B){10,10}   (10, 10)
%% drawThickline(AB)
\setplotsymbol ({\usefont{OT1}{cmr}{m}{n}\large .})%
{\setbox1=\hbox{\usefont{OT1}{cmr}{m}{n}\large .}}%
  \global\linethickness=0.31\wd1}%
\plot 5.00000 5.00000 10.00000 10.00000 / %% PQ
\setlength{\linethickness}{0.4pt}%
\setplotsymbol ({\usefont{OT1}{cmr}{m}{n}\tiny .})%
```

P_{CT}E_X⁷

P_{CT}E_X draws lines using two different methods depending on whether the lines are (a) horizontal or vertical, (b) any other orientation. Furthermore these two groups use different commands for controlling line-thickness, as follows.

Horizontal and vertical lines (rules): Horizontal and vertical lines are drawn using the P_{CT}E_X `\putrule` command⁸ and consequently the thickness of such lines is controlled by the P_{CT}E_X

`\linethickness` command (the default line-thickness is 0.4pt). For example, the following P_{CT}E_X command changes the thickness to 1pt.

```
\linethickness=1pt
```

Note also that the P_{CT}E_X `\linethickness` command can also be reset to its default value (0.4pt) by the P_{CT}E_X `\normalgraphs` command (see chapter on P_{CT}E_X commands), which resets all P_{CT}E_X graph-drawing parameters to their default values, including `\linethickness`.

Since the graph axes are drawn using horizontal and vertical lines P_{CT}E_X draws them using the `\putrule` command, i.e. using the `\linethickness` command. For example, the following commands can be used to draw thick axes.

⁷See also Chapter E on P_{CT}E_X.

⁸Note that the P_{CT}E_X `\putrule` command employs the T_EX and L_AT_EX `\rule` command, and so is only used for horizontal and vertical lines.

```

\linethickness=2pt
paper{units(1mm),xrange(0,50),yrange(0,50),axes(XY)}
\linethickness=0.4pt    %% reset to default

```

Other lines and curves: P₁CT_EX draws all other lines (non-horizontal non-vertical) and curves are drawn using the P₁CT_EX `\plot` command which draws a continuous line of dots. Consequently the thickness of these lines is controlled by the size of the dot, which is defined using the P₁CT_EX `\setplotsymbol` command, the default size of dot being `{\tiny .}`. Larger dots therefore generate thicker lines. For example, the following P₁CT_EX command sets the dot to a larger size.

```
\setplotsymbol({\Large .})
```

3.7.2 Recommendations

In general it is recommended that the `mathsPIC linethickness()` command is used as this automatically sets both the P₁CT_EX `\putrule` and `\setplotsymbol()` commands⁹. However, under certain circumstances it may be convenient to set the P₁CT_EX commands directly, as described below.

If you do use P₁CT_EX commands for drawing lines you need to remember that since P₁CT_EX uses two groups of commands for controlling the thickness of lines (i.e. `\linethickness` and `\setplotsymbol`) it is important to use pairs of equivalent commands for ‘rules’ (horizontal and vertical lines) and dots (all other lines) when changing line-thickness. These are shown in Table 3.3 for a 10-point font (note that the default sizes are 0.4-point and `\tiny`).

Table 3.3: Equivalent P₁CT_EX commands for a 10-point font

rules (horizontal/vertical)	all other lines
<code>\linethickness=1.35pt</code>	<code>\setplotsymbol({\Large .})</code>
<code>\linethickness=1.1pt</code>	<code>\setplotsymbol({\large .})</code>
<code>\linethickness=0.9pt</code>	<code>\setplotsymbol({\normalsize .})</code>
<code>\linethickness=0.4pt</code>	<code>\setplotsymbol({\tiny .})</code>

If macros are required, then this can be done easily with a T_EX macro using the P₁CT_EX commands directly. For example, the following code draws a medium-thick line *AB* by invoking the command `\mediumthickline`.

```

\newcommand{\mediumthickline}{%
    \linethickness=1.1pt%
    \setplotsymbol({\large .})}%
...
\mediumthickline%
drawline(AB)

```

⁹See Section 3.7

3.8 Text

Text is typeset using the `text` command and, by default, is centered both horizontally and vertically at a defined point. For example, the words ‘point Z’ would be placed at the point Z using the command `text(point Z){Z}`.

Text can be located relative to a point-location using the `shift(dx,dy)` or `polar(r,θ)` commands. For example, points $P_1P_2P_3$ could have their labels located 4 units from each point as follows.

```
var d = 4
text($P_1$){P1,shift(-d,0)}
text($P_2$){P2,polar(d,10 deg)}
text($P_3$){P3,polar(d,0.29088 rad)}
```

Optionally, text can be positioned relative to a given point using appropriate combinations of the *case sensitive* P_{CT}E_X options `l t r B b` to align the **l**eft edge, **r**ight edge, **t**op edge, **B**aseline, **b**ottom edge of the text respectively, as described in the P_{CT}E_X manual. For example in the diagram below (Figure 3.3) the text box `a nice box` is aligned such that the right edge of the text box is centered vertically at the point P using the `[r]` option as follows.

```
point(P){25,5}
text(\fbox{a nice box}){P}[r]
```

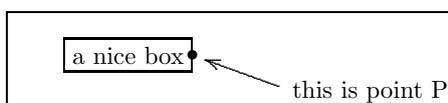


Figure 3.3:

The `mathsPIC` code for Figure 3.3 is as follows.

```
%% mpicpm03-3.m (Figure 3.3)
\usepackage{mathspic}
\framebox{\vbox{
%-----
\beginpicture
paper{units(1mm),xrange(0,28),yrange(0,10)}
point(P){25,5}[symbol=$\bullet$,radius=2]
text(\fbox{a nice box}){P}[r]
drawpoint(P)
point(J){P,polar(15,-20deg)}[radius=2]
text(this is point P){J}[1]
drawarrow(JP)
\endpicture
%-----
\ } } %% end of framebox
```

Text can also be placed at a point-location (using a `DrawPoint` command), if the text is defined as the optional point-symbol (in square brackets) associated with a point

command. Although this is useful in certain circumstances, this method is somewhat less flexible than the `text` command, since the `drawPoint` command centers the point-symbol vertically and horizontally over the point-location.

3.9 Variables and constants

3.9.1 Scalar variables

Numeric scalar variables are defined using the `var name = value` command as for example

```
var r=6
```

The name requirement is the same as that for points; an initial (single) letter optionally followed by a maximum of 3 digits. Sometimes the `var` command too restrictive as regards the variable-name, in which case a more intuitive variable-name can be allocated by using a `mathsPIC` macro (see Section 3.4). For example, we can allocate the name `WeightInKg` to some value, say 22.6 *Kg*, using the following macro definition (see also Section 4.1 for other examples).

```
%def WeightInKg()22.6
```

However, you have to then remember to use the macro with the `&` prefix (see Section 3.4).

The *value* can be either a number (e.g. 4.32 or 63), an existing variable name (e.g. `r3`), a pair of point names e.g. `AB` (i.e. representing the Pythagorean distance between the two points), or a numerical expression (e.g. `3*k/2`). Thus the command `var r3=20` allocates 20 to the variable name `r3`, which could then be used, for example, as the radius in the circle command `drawcircle(C3,r3)`.

New values can be re-allocated to existing variable-names using the same `var` command.

If it is necessary to use the same letter for a point and a variable (or constant), then a convenient strategy is to consider using upper case for points and lower case for variables and constants.

Note that several variables can be allocated in a single statement, as follows:

```
var r=6, j22=r*6/5, d=180
```

3.9.2 Scalar constants

Constants can be allocated using the `const` command as follows. A constant can be any numerical expression.

```
const j26=23.653
```

If you subsequently try and change the value of a constant, then `mathsPIC` will issue an appropriate error message.

3.9.3 Mathematics

All the usual mathematical operations can be used with variables (see Section 4.1), both when defining a variable, and in places where variables can be used as parameters. When using ‘scientific’ notation `mathsPIC` allows either `e` or `E`; for example, `var j25=7E-2` and `point(P){3, 2.34e2}`. The constants π and e are available as `_Pi_` and `_e_`. The following are examples of valid commands.

```

var r = 6, j = r*tan(0.34)/27, d3=AB
point(C){5,5}
drawCircle(C,r/3)
var e=_e_, p1=_Pi_
var j25=7E-2
var t = _linethickness_
text({P$}{P, shift(-5.564e-1,0)}

```

3.9.4 Scientific notation

While mathsPIC *does* allow the use of the ‘E’ or ‘e’ format of so-called ‘scientific’ notation (see above) it is important to remember that \TeX does not, and consequently this influences how mathsPIC displays small numbers when they appear in the output file.

One of the curious anomalies of \TeX is that it cannot manipulate numeric values in scientific notation, and will generate an error message whenever it finds the letter E or e as part of a number. Consequently mathsPIC automatically converts all numbers destined to appear within a \PCTeX command in the output-file into ‘true’ decimal format (i.e. not E notation). mathsPIC also reduces such numbers to only five decimal places, and consequently quantities with an absolute value less than 0.00001 are therefore effectively reduced to zero (0.00000).

This is demonstrated in the following example, where the coordinates s_i appear in the output file in scientific notation when in commented lines, but in decimal notation in the \PCTeX $\text{\put} \dots$ commands. Note also that in this example the y -coordinate of J_{45} appears as zero in the $\text{\put} \dots$ command but as $2.32857142857143e-06$ when shown as the value s_5 .

```

var r = 163/7, s1=r/1E3, s2=r/1E4, s3=r/1E5, s4=r/1E6, s5=r/1E7
point(J12){s1,s2}
point(J23){s2,s3}
point(J34){s3,s4}
point(J45){s4,s5}
drawpoint(J12 J23 J34 J45)

```

appears in the output file as

```

%% var r=163/7, s1=r/1E3,s2=r/1E4,s3=r/1E5,s4=r/1E6,s5=r/1E7
%% r = 23.2857142857143
%% s1 = 0.0232857142857143
%% s2 = 0.00232857142857143
%% s3 = 0.000232857142857143
%% s4 = 2.32857142857143e-05
%% s5 = 2.32857142857143e-06
%% point(J12){s1,s2} J12 = (0.02329, 0.00233)
%% point(J23){s2,s3} J23 = (0.00233, 0.00023)
%% point(J34){s3,s4} J34 = (0.00023, 0.00002)
%% point(J45){s4,s5} J45 = (0.00002, 0.00000)
%% drawpoint(J12 J23 J34 J45)
\put {$\bullet$} at 0.02329 0.00233 %% J12
\put {$\bullet$} at 0.00233 0.00023 %% J23
\put {$\bullet$} at 0.00023 0.00002 %% J34

```

```
\put {$\bullet$} at 0.00002 0.00000 %% J45
```

Note that in practice one could write a Perl subroutine to check and filter out occurrences of unwanted E characters. For example, as when out-putting graph data in the form of (x,y) data-pairs to a file.

3.10 The LOOP environment

This operates as a simple ‘DO...LOOP’ allowing a chunk of code to be input multiple times. It takes a single argument, namely the loop number. The commands for this environment are as follows.

```
beginloop <expr>
...
endloop
```

When using a ‘DO-LOOP’ one usually has to initialise some parameters, and then increment the parameters with each passage of the loop. In particular, it is often convenient to use a loop counter so that one can see in the output file which particular loop is being processed at any stage. It is also useful to include an obvious marker at the beginning and end of the repeated section. All these are included in the following code which produces Figure 3.4 by inputting a section of the code 40 times.

```
%% mpicpm03-4.m (Figure 3.4)
\usepackage{mathspic}
%-----
\beginpicture
paper{units(1mm), xrange(0,60), yrange(0,60) axes(LB), ticks(10,10)}
point(S){7,55} % Start position
drawpoint(S)
%% initialise parameters
var n=0 % initialise mathspic counter
var a=-180 % start angle degrees
var d=50 % start length
beginLOOP 40 % loop 40 times
  var n=n+1, a = a+90, d = d-1 %% increment counter, angle, length
  point*(P){S,polar(d,a deg)} % generate new point P
  drawline(SP) % draw line from OLD S to NEW P
  point*(S){P} %% reallocate S <-- P
endLOOP
drawpoint(P)
\endpicture
```

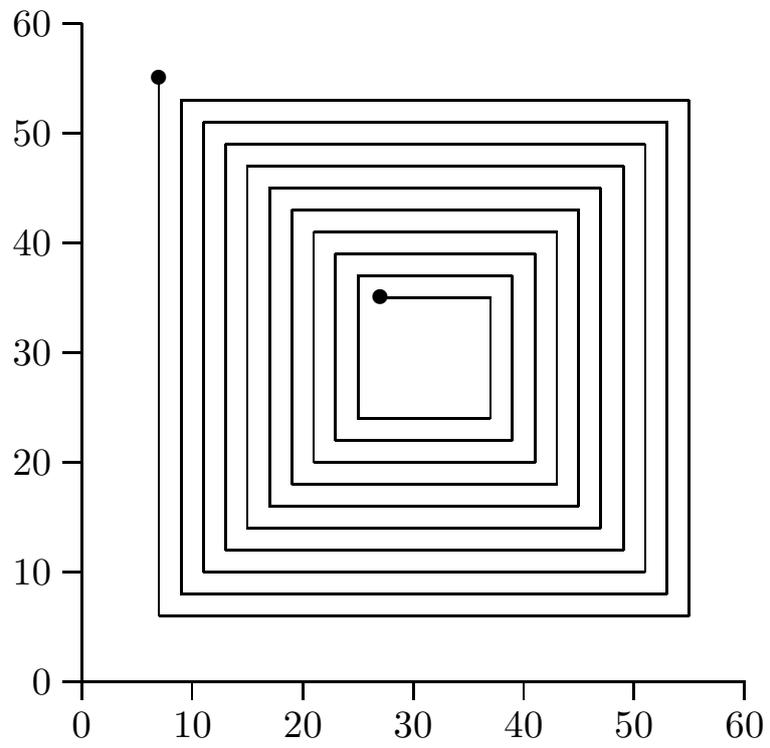


Figure 3.4:

Chapter 4

mathsPIC commands

All mathsPIC commands (except macros) are case-*insensitive*. This is a design feature which allows the user to customise the commands and make them easier to read. mathsPIC macros, however, *are* case-sensitive, but since macro-names are created by the user they are customised from the outset by definition.

The arguments of mathsPIC commands are either strings (any legitimate T_EX or L^AT_EX commands or characters which can be put into an `\hbox`), point-names (e.g. A, B2, C345), or numerical expressions. Where appropriate, mathsPIC allows scalar quantities in commands to be represented by either a numeric value (e.g. 0.432), a variable name (e.g. r2), two point names representing the Pythagorean distance between two points (e.g. AB), or even a mathematical expression¹. For example, the structure of the command `DrawCircle(<centre>,<radius>)` is quite flexible, as follows.

```
point(C){5,5}
drawCircle(C,4.32)
drawCircle(C,r2)
drawcircle(C,AB)
drawCircle(C, r3*tan(0.6)/4)
```

4.1 Mathematics

A leading zero must always be used for decimals whose absolute value is < 1 . The argument of trigonometric functions is in radians. Inverse trigonometric functions return a value in radians.

- **Constants:**
 - for π (3.14159...) use `_Pi_` or `_pi_`;
 - for 'e' (2.718281...) use `_E_` or `_e_`;
 - for '180/Pi' use `_R2D_`;
 - for 'Pi/180' use `_D2R_`;
- **Trigonometric functions:** (arguments must be in radians)
`sin()`, `cos()`, `tan()`, `asin()`, `acos()`, `atan()`
- **Remainder:** `rem()`; e.g. `var r = 12 rem(5) → 2`

¹See the definition of 'numerical expression' in Section 4.2.

- **Square root:** `sqrt()`; e.g. `var s = sqrt(14)`
- **Exponentiation:** `**`; e.g. `var j = r**2`
- **Integer:** `int()`; e.g. `var s = int(3.867) → 3`
- **Sign:** `sgn()` (returns -1, 0, or +1); e.g. `var s = sgn(-2.987) → -1`
- **Line thickness:** `_linethickness_` (returns current value of the linethickness in current units); e.g. `var t = _linethickness_`
- **Area of triangle:** `area(ABC)` e.g. `var t = area(ABC)`

4.1.1 Macros

mathsPIC allows the definition of one-line macros with or without arguments, which evaluate to a ‘numerical expression’ (see Section 4.2) (i.e. not strings). Macro definition has the following syntax:

```
%def <macro-name> ( [ <parameters> ] ) <macro-code>
```

where *<parameters>* is a list of comma separated strings (e.g. x,y,z). Always place a % symbol at the end of the *<macro-code>* to limit additional white space. The *<macro-code>* can be optionally delimited using round brackets if necessary to help make it more readable. Examples of valid macros are as follows:

```
%def two()2%    % a macro called two
%def two(2)%
```

Once a macro is defined it can be used or it can be undefined. A macro is removed (undefined) using the `%undef` command as follows.

```
%undef <macro-name>
```

Macros (see Section 3.4) are very useful in mathematics. Remember (a) that if a macro does not use parameters then the `()` are not used when part of the command when it is used as a variable, (b) that the macro-name must have the `&` prefix when it is used in a mathsPIC command, and (c) it is very important to place a % symbol at the end of the macro command in order to stop P_TE_X from collecting any following white space and distorting the diagram.

Macros without a parameter can be very useful since they can be used to allocate a meaningful variable-name (or constant-name). For example, we can allocate the variable name `SpeedOfLight` to the value 2.9979×10^8 metres per second² and then manipulate it as follows.

```
%def SpeedOfLight()2.9979e8% metres/sec
var m=1000, E = m*(&SpeedOfLight**2)
```

Note the use of the `&` symbol in the `var` command above. After running the above commands through mathsPIC the output file would show the allocation of the variable `E` as follows.

²Lloyd S (2000). Ultimate physical limits to computation. *Nature*; 406, 1047–1054 (August 31, 2000).

```
%def SpeedOfLight()2.9979e8% metres/sec
%% var m=1000, E = m*(2.9979e8**2)
%% m = 1000
%% E = 8.98740441e+19
```

By way of another example, it is often useful to have meaningful names for the factors used for converting degrees to radians and *vice versa* (e.g. *d2r*, *r2d*), as provided by the following two macros (without parameters).

```
%def d2r()_pi_/180%    % degrees to radians
%def r2d()180/_pi_%    % radians to degrees
```

For example, a variable of 30 degrees (say, *d30*) could be converted to radians (say, *r30*) in a *var* command using the new *d2r* command as follows (remembering to include the *&* prefix for the macro),

```
var d30=30, r30=d30*&d2r
```

which is processed via *mathsPIC* to

```
%% var d30=30, r30=d30*_pi_/180
%% d30 = 30
%% r30 = 0.523598775598299
```

Macros which take a parameter can be useful in a slightly different way. For example, since Perl does not have separate commands for $\log_{10}()$ and $\log_e()$ —it only has one *log* command, namely *log()* for $\log_e()$ —you may wish to define separate commands to make it easy to distinguish between the two. This is easily done with macros taking a single parameter as follows.

```
%def loge(a) log(a)%
%def log10(a) log(a)/log(10)%
```

Now if you type the command *&loge(5)* in the *mathsPIC* file you will generate the value $\log_e(5)$. Similarly, the command *&log10(3)* $\rightarrow \log_{10}(3)$.

4.1.2 Making a macro library

If you regularly use particular macros then these can be easily stored in an ASCII file as a library (say, *mathspic.lib*, as described in Section 3.4.1) as follows.

```
%%--- mathspic.lib---
%def log10(a)log(a)/log(10)%
%def loge(a)log(a)%
%def mod(a)rem(a)%
%def d2r()_pi_/180%
%def r2d() 180/_pi_%
%%---end of library---
```

This library file can then be input at the beginning of the *mathsPIC* script, as follows.

```
inputfile{mathspic.lib}
```

4.2 Command definitions

Names of points, constants and variables

The names of points, constants and variables all conform to the same name convention, as follows: The name *must* begin with a *single* letter (either upper or lower case), and may have up to a *maximum* of four³ following digits.

While constants and variables should not have the same name, it is quite possible for points and variables (and constants) to have the same name. Consequently, it is a useful rule to arrange for points to have uppercase letters and for variables and constants have lowercase letters, as follows:

```
point (P25) {0,0}
var a=2, b1360=4
const e=2.17
```

Numerical expressions

When dealing with commands we will refer frequently to the term ‘numerical expression’ by which is meant either (a) a number (integer or decimal), (b) a numeric variable or constant (defined using the `var` or `const` command), (c) any mathsPIC function, macro, or mathematical expression which evaluates to a number, or (d) a pair of point names (e.g. AB) representing the Pythagorean distance between the two points. A leading zero must be used with decimal fractions having an absolute value less than one. The syntax of the numerical expression, which we will refer to as $\langle expr \rangle$ is therefore as follows:

$$\langle expr \rangle ::= \langle two-points \rangle \mid \langle number \rangle \mid \langle variable \rangle \mid \langle maths \rangle$$

Units

When dealing with commands we will refer frequently to the term ‘unit’ by which is meant one of the valid \TeX units (see Knuth (1986) p. 57).

$$\langle unit \rangle ::= \langle mm \rangle \mid \langle cm \rangle \mid \langle pt \rangle \mid \langle pc \rangle \mid \langle in \rangle \mid \langle dd \rangle \mid \langle cc \rangle \mid \langle sp \rangle$$

A unit requires a scalar; for example, in the `units()` option in the `paper{}` command we would write something like `paper{units(1cm, 3cm), ...}` to define the scale for the (x, y) axes etc.

Line thickness

Commands and parameters which control line-thickness are described in Section 3.7.1.

4.2.1 Backslash

- $\boxed{\backslash \dots}$ $\boxed{\backslash \square \dots}$

- **Notes**

A line having a *leading* backslash is processed (copied verbatim) slightly differently depending on whether the character following the backslash is a space or not.

³From mathsPIC version 1.11 onwards.

A leading backslash *followed by a non-space character* tells mathsPIC to copy the whole line verbatim (*including* the backslash) through to the output-file (thus a line leading with the L^AT_EX command `\begin{document}` will be copied unchanged).

However, a leading backslash *followed by one or more spaces*, (e.g. `_ _ _ . . .`) tells mathsPIC to copy the rest of the line verbatim through to the output-file, but *without* the leading backslash.

4.2.2 ArrowShape

This command defines the shape of an arrowhead, and allows arrowheads to be customised (see Section 7.3 for details).

- **Syntax**

```
ArrowShape(⟨expr⟩ [⟨units⟩], ⟨expr⟩, ⟨expr⟩)
ArrowShape(default)
```

- **Notes**

The first parameter is the length of the arrow head itself. If *⟨units⟩* are not given then the current default units will be applied. The last two parameters are angles (default is degrees) which define the shape of the arrow head (see Section 7.3 for details).

The default arrow shape is equivalent to the command `Arrowshape(2mm, 30, 40)`. This default arrowhead shape can be reset using `Arrowshape(default)` command.

- **Examples**

```
Arrowshape(4mm, 30, 60)
var h = 3
Arrowshape((3*h)mm, 30, 60)
drawArrow(AB)
Arrowshape(default)
drawArrow(PQ)
```

4.2.3 beginLoop ... endLoop environment

—see Loop

4.2.4 beginSkip ... endSkip environment

—see Skip

4.2.5 Const

The `const` command is used to define scalar constants.

- **Syntax**

```
const ⟨name⟩ = ⟨expr⟩ [, ⟨name⟩ = ⟨expr⟩ ] ...
```

- **Notes**

The constant name follows the same naming convention as points and variables (see Section 4.2). The scalar argument can be any numeric expression. There is no terminal comma. If a new value is allocated to an existing constant name then mathsPIC issues an error message.

- **Example**

```
const h=5
const r = 20, r4 = r3*tan(0.3)
```

4.2.6 DashArray

The dasharray command takes an arbitrary number of paired arguments that are used to specify a dash-gap-dash... pattern.

- **Syntax**

```
dasharray ( DASH , GAP [ DASH , GAP ] ... )
DASH ::=  $\langle expr \rangle \langle unit \rangle$ 
GAP ::=  $\langle expr \rangle \langle unit \rangle$ 
```

- **Notes**

There must be an even number of arguments. If a variable or expression is used then it should be separated from the unit either by a \square or with round brackets () as shown below.

Macros are useful for allocating names to frequently used dashArray commands.

- **Example**

```
dasharray(6pt, 2pt, 1pt, 2pt)
var d=2
dasharray(6pt, 2pt, 1pt, d pt)
dasharray(6pt, 2pt, 1pt, (d)pt)
dasharray(6pt, 2pt, 1pt, (3*d)pt)
%def fancydashes()dasharray(6pt, 2pt, 1pt, 2pt)%
```

4.2.7 DrawAngleArc

This command draws an arc in the specified angle, a distance *radius* from the angle.

- **Syntax**

```
drawAngleArc{ ANGLE , RADIUS , LOCATION , DIRECTION }
ANGLE ::= angle(  $\langle three-points \rangle$  )
RADIUS ::= radius(  $\langle expr \rangle$  )
LOCATION ::= internal | external
DIRECTION ::= clockwise | anticlockwise
```

- **Notes**

The angle location is either *internal* ($\leq 180^\circ$) or *external* ($\geq 180^\circ$). The direction of the arc is either *clockwise* or *anticlockwise*, and this direction must correspond with the letter sequence specified for the angle. Strange and unexpected results will be produced if the four parameters are not internally consistent. The parameter order [angle/radius/internal/clockwise](#) etc is important.

- **Examples**

```
DrawAngleArc{angle(ABC), radius(3), external, clockwise}
var r = 2
DrawAngleArc{angle(PQR), radius(r), internal, anticlockwise}
```

4.2.8 DrawAngleArrow

This command draws an arrow in the specified angle, a distance *radius* from the angle.

- **Syntax**

```
drawAngleArrow{ ANGLE , RADIUS , LOCATION , DIRECTION }
ANGLE ::= angle( three-points )
RADIUS ::= radius( expr )
LOCATION ::= internal | external
DIRECTION ::= clockwise | anticlockwise
```

- **Notes**

The angle location is either *internal* ($\leq 180^\circ$) or *external* ($\geq 180^\circ$). The direction of the arrow is either *clockwise* or *anticlockwise*, and this direction must correspond with the letter sequence specified for the angle. Strange and unexpected results will be produced if the four parameters are not internally consistent. The parameter order [angle/radius/internal/clockwise](#) etc is important.

The radius can be any numerical expression.

- **Examples**

```
DrawAngleArrow{angle(ABC), radius(3), external, clockwise}
var r = 2
DrawAngleArrow{angle(PQR), radius(r), internal, anticlockwise}
```

4.2.9 DrawArrow

This command draws an arrow(s) joining two points.

- **Syntax**

```
drawArrow ( two-points [ , two-points ] ... )
```

- **Notes**

The direction of the arrow is in the point order specified. The shape of the arrow-head is controlled by the `ArrowShape` command (see Section 7.3). Commands and parameters which control line-thickness are described in Section 3.7.1.

- **Examples**

```
drawArrow(AB)
drawArrow(FG, HJ)
```

4.2.10 DrawCircle

Draws a circle with its centre at a given point and with a given radius.

- **Syntax**

```
DrawCircle ( CENTRE , RADIUS )
```

```
CENTRE ::=  $\langle point \rangle$ 
```

```
RADIUS ::=  $\langle expr \rangle$ 
```

- **Notes**

Note that P_JCT_EX draws circles with the `\circulararc` command, using a radius equivalent to the distance from the centre to the point at which it starts drawing the arc. Consequently, if the units of the x and y axes are different, circles may be drawn strangely. MathsPIC therefore generates a message to this effect in the output-file if different units are selected for the two axes in the `units` command.

- **Examples**

```
drawCircle(C2,5)
var r2=3
drawCircle(C2,r2)
drawCircle(C2,(r2/tan(1.2)) )
drawCircle(C2,AB)
```

4.2.11 DrawCircumcircle

Draws the circumcircle of a triangle defined by three points

- **Syntax**

```
DrawCircumcircle(  $\langle three-points \rangle$  )
```

- **Example**

```
drawCircumcircle(ABC)
```

4.2.12 DrawCurve

Draws a smooth quadratic curve through three points in the order specified.

- **Syntax**

```
DrawCurve (  $\langle three-points \rangle$  )
```

- **Notes**

This command will be upgraded in the next version to apply to more than three points.

Note that curves drawn using this command do *not* break to avoid line-free zones associated with the points (the `drawLine` command for straight lines *does* acknowledge line-free zones).

- **Example**

```
drawCurve(ABC)
```

4.2.13 DrawExcircle

Draws the excircle touching a given side of a triangle.

- **Syntax**

```
DrawExcircle( TRIANGLE , SIDE )
```

```
TRIANGLE ::= <three-points>
```

```
SIDE ::= <two-points>
```

- **Example**

```
drawExcircle(ABC,BC)
```

4.2.14 DrawIncircle

Draws the incircle of a triangle defined by three points

- **Syntax**

```
DrawIncircle( <three-points> )
```

- **Example**

```
drawIncircle(ABC)
```

4.2.15 DrawLine

This command draws a line(s) between sets of two or more points.

- **Syntax**

```
drawLine ( LINE [ , LINE ] ... )
```

```
LINE ::= <two-points> [ <point> ] ...
```

- **Notes**

Lines are drawn in the point order specified. Commands and parameters which control line-thickness are described in Section 3.7.1.

Note that the `drawline` command uses the \PCTEX `\putrule` command for horizontal and vertical lines, and the `\plot` command for lines of all other orientations.

- **Examples**

```
drawLine(AB)
```

```
drawLine(FG, HJ)
```

```
var d = 3
```

4.2.16 DrawPerpendicular

Draws the perpendicular from a point to a line.

- **Syntax**

```
DrawPerpendicular( <point> , LINE )
```

```
LINE ::= <two-points>
```

- **Example**

```
drawPerpendicular(P, AB)
```

4.2.17 DrawPoint

Draws a symbol at the point-location.

- **Syntax**

```
DrawPoint( <point> [ <point> ] ... )
```

- **Notes** There must be no commas in the list of points (spaces are allowed).

- **Example**

```
drawpoint(T4)
drawpoint(ABCDEF)
drawpoint(P1 P2 P3 P4)
```

4.2.18 DrawRightangle

Draws the standard right-angle symbol in the internal (acute) angle specified. If the angle is not exactly 90° then mathsPIC draws the required symbol in perspective (i.e., each arm is still drawn parallel to the respective lines).

- **Syntax**

```
DrawRightangle( <three-points> , <expr> )
```

- **Example**

```
drawRightangle(ABC,3)
```

4.2.19 DrawSquare

Draws a square at a given point with a given side-length.

- **Syntax**

```
DrawSquare( <point> , <expr> )
```

- **Example**

```
drawSquare(P,5)
drawSquare(P,s2)
drawSquare(P, s2*4/(3*j))
drawSquare(P,AB)
```

4.2.20 DrawThickArrow

This command draws a thickarrow(s) joining two points.

- **Syntax**

```
drawThickArrow ( <two-points> [ , <two-points> ] ... )
```

- **Notes**

The direction of the arrow is in the point order specified. The shape of the arrow-head is controlled by the ArrowShape command (see Section 7.3). Commands and parameters which control line-thickness are described in Section 3.7.1.

- **Examples**

```
drawThickArrow(AB)
drawThickArrow(FG, HJ)
```

4.2.21 DrawThickLine

This command draws a thick line(s) between sets of two or more points.

- **Syntax**

```
drawThickLine ( LINE [ , LINE ] ... )
LINE ::= <two-points> [ <point> ] ...
```

- **Notes**

Lines are drawn in the point order specified. Commands and parameters which control line-thickness are described in Section 3.7.1.

Note that the `drawThickLine` command uses the $\text{P}\text{T}\text{E}\text{X}$ `\putrule` command for horizontal and vertical lines, and the `\plot` command for lines of all other orientations.

- **Examples**

```
drawThickLine(AB)
drawThickLine(FG, HJ)
var d = 3
```

4.2.22 InputFile

Inputs a plain text file containing mathsPIC commands, one or more times.

- **Syntax**

```
InputFile( <file-name> ) [ LOOP ]
LOOP ::= <expr>
```

- **Notes**

Optionally, the file can be input [LOOP] times, in which case this command can be used to implement something akin to a primitive DO-LOOP if the file contains mathsPIC commands.

See also the LOOP command (a `beginLOOP...endLOOP` environment).

If LOOP is not an integer then mathsPIC will round the value down to the nearest integer. Note that the `inputfile*` command has no [LOOP] option.

- **Example**

The following command inputs the file `newfile.dat` 4 times in succession.

```
inputFile(myfile.dat) [4]
```

The `inputfile*` command is used to input a file in *verbatim*, i.e. a file with *no* mathsPIC commands. For example, a file containing only $\text{P}\text{T}\text{E}\text{X}$ commands or data-points for plotting etc. A typical example might be the following file (`curve-A.dat`) which would be input *verbatim* using the command `inputfile*(curve-A.dat)`. (Note that $\text{P}\text{T}\text{E}\text{X}$ requires an ODD number of points).

```

%% curve-A.dat
\setquadratic
\plot
  0      0
  3.56  4.87
  8.45  9.45
  5      7
  2.34  3.23 /
\setlinear

```

4.2.23 LineThickness

Sets a particular linethickness.

- **Syntax**

```

Linethickness ( <expr> <units> )
Linethickness (default)

```

- **Notes**

This command also sets the font to `cmr` and `plotsymbol` to `{\CM .}`, and also sets the rule thickness for drawing horizontal and vertical lines. For example the command `linethickness(2pt)` results in the following commands in the output `.mt` file.

```

\linethickness=2.00000pt\Linethickness{2.00000pt}%
\font\CM=cmr10 at 9.94451pt%
\setplotsymbol ({\CM .})%

```

The command `linethickness(default)` restores the working linethickness to the default value of 0.4pt.

The current value of `linethickness` (in the current units as defined in the paper command) can be accessed as the value `_linethickness_` as follows.

```
var t = _linethickness_
```

See also—the chapter on `PTEX` commands as there is a similar `PTEX` command with the same name (but with a different syntax).

- **Example**

```

linethickness(2pt)
linethickness(default)
var t = _linethickness_

```

4.2.24 Loop environment

This environment cycles a block of code a number of times.

- **Syntax**

```
beginLoop <expr>...endLoop
```

- **Notes** The block of code which lies within the environment is input $\langle expr \rangle$ times.
- **Example**

```
beginLoop 5
...
endLoop
```

4.2.25 Paper

Defines the plot area and scale, and optional axes and tick marks.

- **Syntax**

```
Paper{ UNITS , XRANGE , YRANGE , AXES , TICKS }
UNITS ::= units(  $\langle expr \rangle$   $\langle units \rangle$  [ ,  $\langle y-expr \rangle$   $\langle y-units \rangle$  ] )
XRANGE ::= xrange(  $\langle low-expr \rangle$  ,  $\langle high-expr \rangle$  )
YRANGE ::= yrange(  $\langle low-expr \rangle$  ,  $\langle high-expr \rangle$  )
AXES ::= axes( [ $\langle L \rangle$ [*]] [ $\langle R \rangle$ [*]] [ $\langle T \rangle$ [*]] [ $\langle B \rangle$ [*]] [ $\langle X \rangle$ [*]] [ $\langle Y \rangle$ [*]] )
TICKS ::= ticks(  $\langle x-expr \rangle$  ,  $\langle y-expr \rangle$  )
```

- **Notes**

The following statement sets up a rectangular drawing area 5 cm \times 5 cm with axes on the Left (y-axis) and Bottom (x-axis), and tick marks at 1 cm intervals. Note that the order of the options `units()` . . . `ticks()` MUST adhere to the order shown here (this will be made flexible in a later version).

```
paper{units(1cm),xrange(0,5),yrange(0,5),axes(LB),ticks(1,1)}
```

All combinations of the axis-codes (XYLRBTB) are allowed, and a * following an axis-code (e.g. L*) disables the drawing of ticks on the specified axis.

If it is necessary to have *different* unit scales for the x and y axes, say 1 cm and 2 mm respectively, then this is implemented by `units(1cm,2mm)`. If only a single unit is specified (e.g. `units(3cm)`) then `mathsPIC` automatically makes this the unit scale for *both* axes. If the `unit` option is omitted `P1CTEX` will use the last defined units (within the same picture environment), the default units being $xunit=1pt$ and $yunit=1pt$ (see the `P1CTEX` Manual, page 3; Wichura, 1992).

If different units or different sizes of the same unit are used for the x and y scales then `mathsPIC` issues a warning message to this effect (since strange effects arise, e.g., with circles, ellipses & angles, when the x and y axes are scaled unequally). Note that the modern rectangular thin computer screens may also ‘stretch’ circles sideways, depending on the setup, in a way not seen when the image is printed.

- **Example**

```
paper{units(1cm),xrange(0,10),yrange(0,10),axes(XY)}
var j=5
paper{units(1cm),xrange(0,j),yrange(0,5),axes(LBT*R*),ticks(1,1)}
var u = 3
paper{units((u)mm),xrange(0,50),yrange(0,50),axes(LBTB)}
```

4.2.26 Point

Allocates coordinates to a point-name. A point name *must* begin with a *single* letter (either upper or lower case), and may have up to a *maximum* of four⁴ following digits, e.g., *P1326*.

- **Syntax**

```
Point [*] ( <point> ) { <point> } [ OPTION [, OPTION] ]
Point [*] ( <point> ) { LOCATION } [ OPTION [, OPTION] ]
LOCATION ::= <x-expr> , <y-expr>
LOCATION ::= xcoord( <point> ) , ycoord( <point> )
LOCATION ::= midpoint( <two-points> )
LOCATION ::= intersection( <two-points> , <two-points> )
LOCATION ::= PointOnLine( <point> , <expr> )
LOCATION ::= perpendicular( <point> , <two-points> )
LOCATION ::= Circumcirclecenter( <three-points> )
LOCATION ::= Incirclecenter( <three-points> )
LOCATION ::= Excirclecenter( <three-points> , <two-points> )
LOCATION ::= <point> , POINT-FUNCTION
POINT-FUNCTION ::= shift( <x-expr> , <y-expr> )
POINT-FUNCTION ::= polar( <r-expr> , <θ-expr> )
POINT-FUNCTION ::= rotate( <point> , <θ-expr> )
POINT-FUNCTION ::= vector( <two-points> )
OPTION ::= symbol = CHARS | radius = <expr>
CHARS ::= <string> | square( <expr> ) | circle( <expr> )
```

- **Notes**

The default point-symbol is the \bullet . An optional alternative point-symbol (or string of characters) can be specified within square brackets, as for example: `[symbol= \triangle]`

By default lines drawn to a ‘point’ are drawn to the point location. However, the radius of an optional line-free zone (line-free radius) can be specified within an optional square bracket e.g. `[symbol= \triangle ,radius=2]`. A line-free radius option can be specified by itself, for example `[radius=2]`.

The `polar(r,theta)` defaults to radians for the angle `theta`. If degrees are required then need to append `<deg>`; e.g. `polar(r,theta deg)`. Note that `direction()` or `directiondeg()` can be used instead of `theta`, as for example: `point(D4){D2, polar(6,direction(AB))}` (see also the *Examples* at the end on the following page).

An optional `symbol=circle()` or `symbol=square()` can also be used as the *symbol*. In these cases a numeric expression is required as the argument, and is taken to be the side length (for `square()`) or circle-radius (for `circle()`). If

⁴From `mathsPIC` version 1.11 onwards.

a radius $\langle expr \rangle$ is also included then the square will have associated with it a line-free radius of the specified value, e.g., `[symbol=square(2), radius=5]`; and similarly if the symbol is a circle.

Note (1) there must be a comma between the options, and (2) the options are not order specific. If no line-free radius value is specified, the line-free radius used is the current (default) value).

Points can also be defined relative to *previously defined* points. For example, as the intersection of two existing lines, or as a +ve or -ve extension (in the direction indicated by the letters) between two previously defined points, (e.g. the `PointOnLine()` construction).

If the command `point()` is used to reallocate new parameters to an existing point-name then `mathsPIC` will generate a warning message indicating that an existing point-name is being reallocated. For example, the following code

```
point(S){6,6}
point(S){7,7}
```

will generate the following warning message in the output file, but still reallocate the point-name.

```
%% point(S){6,6} S = (6, 6)
%% *** Line 15: point(S
%% ***           ){7,6}
%% ... Warning: Point S has been used already
%% point(S){7,6} S = (7, 6)
```

The `point*` command allocates parameters to a point-name irrespective of whether that point-name has been previously defined or not. Thus in the above example, the reallocation proceeds without any warning message at all, as follows

```
%% point(S){6,6} S = (6, 6)
%% point*(S){7,6} S = (7, 6)
```

• Examples

```
point(A){5,5}
point(B2){22,46}[symbol=$\odot$]
point(B2){22,46}[symbol=circle(2),radius=5]
point(B2){22,46}[symbol=square(3),radius=5]
point(B2){22,46}[radius=5]
point(D2){B2, shift(5,5)}
point(D3){D2, polar(6,32 deg)}
point(D4){D2, polar(6,1.2 rad)}
point(D4){D2, polar(6,direction(AB))}
point(D4){D2, polar(6,directiondeg(AB) deg)}
point(G2){Q, rotate(P, 23 deg)}
point(J4){B2, vector(AB)}
point(D2){midpoint(AB)}
point(D2){intersection(AB,CD)}
point(F){PointOnLine(AB,5.3)}
```

```

point(G){perpendicular(P,AB)}
point(H){circumcirclecenter(ABC)}
point(J){incirclecenter(ABC)}
point(K){excirclecenter(ABC,BC)}
point*(A){6,3}
point*(B){B, shift(5,0)}
point*(P){J} %% make J have the same coords as P
point*(P){xcoord(J),ycoord(J)} %% same as Point*(P){J}

```

4.2.27 PointSymbol

Sets a particular point-symbol and line-free radius.

- **Syntax**

```

pointSymbol ( <chars> , <expr> )
pointSymbol (square(<expr> ) , <expr> )
pointSymbol (circle(<expr> ) , <expr> )

```

- **Notes**

This command allows the default point-symbol (\bullet , line-free radius zero) to be changed. The `square` option takes the side-length as argument. The `circle` option takes the radius as argument.

The `PointSymbol` command is particularly useful where a set of points uses the same point-symbol, and also when drawing graphs.

For example, the following command changes the point-symbol to \odot having a line-free radius of 0.7 units.

```
PointSymbol( $\odot$  , 0.7)
```

Note that the `PointSymbol` command only influences subsequent `point` commands, i.e., it should be placed before the defining `point` command. The optional square bracket of the `point` command overrides the `PointSymbol` command. The point-symbol can be reset to the default (\bullet , zero line-free radius) using the command `PointSymbol(default)`.

- **Example**

```

PointSymbol( $\odot$  , 0.7)
PointSymbol(square(4) , 5)
PointSymbol(circle(3) , 3)
var r = 1.2
PointSymbol( $\triangle$  , 0.7)
PointSymbol(circle(r) , r)
pointSymbol(default)

```

4.2.28 Skip environment

This is an environment within which commands are not actioned.

- **Syntax**

```
beginSKIP...endSKIP
```

- **Notes**

It is useful in development for by-passing commands (i.e. its function is equivalent to the `\iffalse... \fi` commands in a \TeX document).

4.2.29 Show...

This command makes mathspic return the value of a calculation or specified parameter.

- **Syntax**

ShowLength (*<two-points>*)

ShowAngle (*<three-points>*)

ShowArea (*<three-points>*)

ShowPoints

ShowVariables

- **Notes**

The `showarea()` command is only for triangles. The results of the Show... commands are shown in the output-file as a commented line.

- **Example**

showLength(AB)

showAngle(ABC)

showArea(ABC)

showPoints

showVariables

When the above examples are processed, the results appear as commented lines in the output-file as follows.

```
%% Length(AB) = 25.35746
%% Angle(ABC) = 39.35746 deg (0.68691 rad)
%% Area(ABC) = 54.54267
%%-----
%%                L I S T   O F   P O I N T S
%%-----
%% q = (8.000, 9.000), LF-radius = 5.000, symbol = $\bigodot$
%% s = (6.000, 6.000), LF-radius = 0.000, symbol = $\bullet$
%% p = (5.000, 5.000), LF-radius = 3.000, symbol = $\circ$
%% t = (5.000, 5.000), LF-radius = 7.000, symbol = $\square$
%%-----
%%                E N D   O F   L I S T   O F   P O I N T S
%%-----
%%-----
%%                L I S T   O F   V A R I A B L E S
%%-----
%% a = 4
%% b = 12
%%-----
%%                E N D   O F   L I S T   O F   V A R I A B L E S
%%-----
```

4.2.30 System

This command allows the user to access the command line and execute system commands. Allows programs to be run from within mathsPIC.

- **Syntax**

```
system ( " command " )
```

- **Notes**

The whole command must be delimited by inverted commas. For example, the following command will temporarily stop mathsPIC processing, access the command-line and run \LaTeX on the file `myfile.tex`, and then seamlessly return and continue mathsPIC processing.

```
system("latex2e myfile.tex")
```

Alternatively, the `system()` command may be used to create a small batch file on-the-fly in order to facilitate some procedure. In mathsPIC a common use is for running a Perl program to write mathsPIC commands to a temporary file which is then input by mathsPIC (see the examples at the end of the Examples chapter).

- **Example**

```
system("dir > mydir-listing.txt")
system("perl drawbox.pl 5 5 temp.txt")
inputFile(temp.txt)
```

4.2.31 Text

Places text at a given location.

- **Syntax**

```
Text ( string ) { LOCATION } [ POSSN [, POSSN] ]
LOCATION ::= point
LOCATION ::= x-expr , y-expr
LOCATION ::= point , POINT-FUNCTION
POINT-FUNCTION ::= shift( x-expr , y-expr )
POINT-FUNCTION ::= polar( r-expr , theta-expr )
POSSN ::= l | t | r | R | b
```

- **Notes**

This command puts the given text-string either at the named point, or with a displacement specified by the optional `shift()` or `polar()` or `rotate()` functions. By default the text is centred vertically and horizontally at the specified location.

Optionally, text can be placed relative to the specified location using appropriate combinations of the \PCTEX *position* options `l t r B b` to align the left edge, right edge, top edge, Baseline, bottom edge respectively of the text box with the point-location (see the \PCTEX Manual, page 5; Wichura, 1992).

For example, the text box This is point P would be aligned such that the right edge of the text box would be centred vertically at the point P , using `text(This is point P){P}[r]` (see figure 3.3).

Note that \TeX and \LaTeX macros are very useful for defining blocks of text in this setting, since the macro name can be used in the `text()` command for convenience, as shown in the examples below.

- **Example**

```
text(A){5,6}
text($A_1$){A1, shift(2, 2)}
text($Z2$){Z2, shift(5, -5)}[tr]
text($Z3$){Z2, polar(5, 20 deg)}[Br]
text($Z4$){Z2, rotate(P, 45)}
text(\framebox{$Z5$}){Z3}
\newcommand{\mybox}{\framebox{$Z5$}}
text(\mybox){P421}
```

4.2.32 Var

The `var` command is used to define scalar variables. The variable name *must* begin with a *single* letter (either upper or lower case), and may have up to a *maximum* of four⁵ following digits.

- **Syntax**

```
var <name> = <expr> [, <name> = <expr>] ...
```

- **Notes**

The variable name follows the same naming convention as points and variables (see Section 4.2). The scalar argument can be any numeric expression.

New values can be re-allocated to existing variable-names; however, when this occurs then `mathsPIC` does not issue a warning message to highlight this fact. If it is important to be warned if a potential variable is accidentally reallocated then one should consider using the `const` command instead (since `mathsPIC` does generate an error message if a constant is reallocated).

Note that in addition to all the mathematical functions, the following functions can also be used.

```
angle(<three-points>)
angledeg(<three-points>)
direction(<two-points>)
directiondeg(<two-points>)
area(<three-points>)
xcoord(<point>)
ycoord(<point>)
```

- **Example**

⁵From `mathsPIC` version 1.11 onwards.

```

var r = 20, r4 = r3*tan(0.3), j = (r*2e3)**2, r5 = AB
var e = _e_, p1 = _Pi_
var t = _linethickness_
var L25 = AB          %% gives length of line AB
var g137 = angle(ABC)  %% (default: returns in radians)
var g200 = angleddeg(ABC)  % returns in degrees
var d = direction(PQ)  % angular direction of PQ in radians
var d = directiondeg(PQ) % angular direction of PQ in degrees
var h = area(ABC)
var r = incircradius(ABC)
var x2000 = xcoord(A), y2000 = ycoord(A)
var m5 = 12 rem 3     %% remainder after dividing by 3
var s1 = sgn(h)       % returns the sign (+1, -1, 0) of h
var i = int(k)        % returns the integer value of k

```

4.3 Summary of mathsPIC commands

The following list shows the format and typical usage of all mathsPIC commands. Although mathsPIC commands are *not* case sensitive, in this summary of commands points are represented using upper case letters, and variables and constants are represented using lower case letters. Note that a leading `_` is used for copying T_EX or P_TE_X lines verbatim, for example with macros or coordinates of data points etc.

```

\_ 4.5 6.3
%def loge(n)log(n)%
%def fancydashes() dasharray(1pt,2pt,3pt,4pt)%
%def plaindashes() dasharray(1pt,1pt)%
%def d2r()_pi_/180%
%def r2d()180/_pi_%
ArrowShape(4mm,30,50)
ArrowShape(default) %% generates arrowshape(2mm,30,40)
beginLoop ... endLoop 5
beginLoop ... endLoop n
beginNoOutput ... endNoOutput
beginSKIP ... endSKIP
const r40=40
drawAngleArc{angle(ABC), radius(3), internal, clockwise}
drawAngleArrow{angle(ABC), radius(3), internal, clockwise}
drawArrow(AB)          %from A to B
drawArrow(AB,CD)
drawCircle(J,12)
drawCircle(C,r2)
drawCircle(C,AB)      %% the radius is length of line AB
drawCircumcircle(ABC)
drawCurve(ABC)        %% three points only
drawExcircle(ABC)
drawIncircle(ABC)
drawLine(AB)
drawLine(ABCDE)

```

```

drawPerpendicular(P,AB)
drawPoint(B)
drawPoint(ABP1P2)
drawRightAngle(ABC, 4)
drawSquare(P,2) %% side = 2
drawThickArrow(AB)
drawThickArrow(AB,CD)
drawThickLine(RS,TU)
inputFile(mathspic.dat)
inputFile(mathspic.dat)[4]
inputfile*(fig2.dat) %% * disables mathsPIC processing of file
linethickness(2pt)
paper{units(1mm,3mm),xrange(-5,50),yrange(-5,50),axes(LR)}
paper{units(2cm),xrange(0,5),yrange(0,9),axes(X*Y),ticks(1,1)}
paper{units(k mm),xrange(0,5),yrange(0,9),axes(X*Y),ticks(1,1)}
point(D1){20,2}
point(D4){6.3,8.9}
point(P){x1,y1}
point(P){Q} %% make P have same coords as Q
point(P){xcoord(Q),ycoord(Q)} %% same as point(P){Q}
point(P){AB,PQ}
point(P){3,4}[symbol=$\odot$,radius=2]
point(P){3,4}[radius=2]
point(D2){20,2}[symbol=square(2),radius=3]
point(D3){20,3}[symbol=circle(r),radius=5]
point(D4){20,4}[symbol=$\Box$]
point(E1){D1, shift(5,6)}
point(E5){D1, shift(r2,6)}
point(E2){D2, polar(8, 1.34)} %% radians is the default
point(E2){D2, polar(8, 1.34 rad)}
point(E2){D2, polar(8,45 deg)}
point(E6){D2, polar(r4,45 deg)}
point(E2){D2, polar(8, direction(AB))} %% radians is the default
point(E2){D2, polar(8, directiondeg(AB) deg)}
point(E5){D2, polar(AB,45 deg)}
point(G2){Q, rotate(P, 23 deg)} %% rotate Q about P by 23 deg
point(P){J, vector(AB)} %% from J with length and direction AB
point(D32){midpoint(AB)}
point(D2){intersection(AB,CD)}
point(F){PointOnLine(AB,5)}
point(F){PointOnLine(AB,-d)}
point(G){perpendicular(P,AB)}
point(H){circumcirclecenter(ABC)}
point(J){incirclecenter(ABC)}
point(K){excirclecenter(ABC,BC)}
point*(D1){20,3}
point*(E1){D1, shift(3,0)}
point*(E1){E1}[radius=3] %% change the line-free radius
PointSymbol(circle(r))
PointSymbol(square(1),2)

```

```

PointSymbol( $\odot$ ,2)
PointSymbol(default)
PointSymbol(default,0.5)
showAngle()
showArea()
showLength()
showVariables
showPoints
system("dir > mydir-listing.txt")
system("perl myprogram.pl")
text(P){5,7}
text( $A$ ){A}
text( $K$ ){K}[r]
text( $B$ ){B, shift(5,5)}
text( $Z_2$ ){Z2, polar(5, 20 deg)}[Br]
text(\framebox{ $Z_5$ }){Z3}
\newcommand{\mybox}{\framebox{ $Z_5$ }}
text(\mybox){P421}
text(\framebox{ $C$ }){C, polar(5,62 deg)}[Br]
var r3 = 20, r4 = r3 * tan(0.4), r5 = AB, e=_e_, p1 = _Pi_
var g = angle(ABC)% (returns in radians)
var g = angledeg(ABC)% (returns in degrees)
var a = area(ABC)
var d = direction(PQ) % angular direction of PQ in radians
var d = directiondeg(PQ) % angular direction of PQ in degrees
var x2 = xcoord(A), y2 = ycoord(A)
var m5 = 12 rem 3 %% remainder after dividing by 3
var t = _linethickness_

```

Chapter 5

P_ICT_EX commands

In this section we present a list of P_ICT_EX commands for use with mathsPIC which is adequate for most purposes. For a more extensive listing see the excellent 8-page summary by Duggan (1990). A 3-page German summary is available from Siart (2008). Alternatively, there is a nice section on P_ICT_EX in Hoenig (1998), which includes a brief list of commands.

The complete list of P_ICT_EX commands is long and detailed, and is, unfortunately, only available currently by purchasing the small *The P_ICT_EX Manual* by Wichura (1992)¹. See Chapter E for full details on P_ICT_EX itself and Wichura's manual (Section E.6). Wichura's manual is well worth obtaining if you intend to be adventurous with P_ICT_EX.

P_ICT_EX commands can *only* be used within the `\beginpicture ... \endpicture` environment. Most P_ICT_EX commands are short one-line commands starting with a leading backslash. Such commands can be used in the normal way as mathsPIC will automatically copy lines starting with a backslash command unchanged through into the output-file (.mt file). For example, drawing with dashed lines is enabled using the following P_ICT_EX command

```
\setdashes
```

However, some P_ICT_EX commands are very long. It is therefore sometimes necessary to have the initial part of a P_ICT_EX command on one line, with the rest of the command continuing onto the next line, such that the second and subsequent lines may well not have a leading backslash command. In order to protect such subsequent lines from being processed as mathsPIC commands, they must be protected by a leading backslash *followed by one or more spaces* (e.g. `_ . . .`) as this instruction tells mathsPIC to copy the rest of the line unchanged (*without* the leading backslash) through into the output-file. For example, the P_ICT_EX code for plotting data-points for a curve could be spread across several lines in the mathsPIC file as follows (an important point to note is that the P_ICT_EX algorithm requires that the number of such points is odd—see Section 7.7).

```
\setquadratic
\plot 1.15   -0.67
\_   1.25   0.02
\_   1.35   1.24 /
\setlinear
```

¹Available from Personal T_EX Inc. (texsales@pctex.com, <http://www.pctex.com/books.html>)

When these commands are processed by mathsPIC, they will appear in the output T_EX file (.mt file) as follows.

```
\setquadratic
\plot 1.15 -0.67
      1.25 0.02
      1.35 1.24 /
\setlinear
```

5.1 Useful P_IC_TE_X commands

The following is a small list of P_IC_TE_X commands which are particularly useful for including in the mathsPIC file, mainly for controlling the thickness of lines and axes, customising dash patterns and symbol spacing, and for plotting and rotation.

Some care needs to be taken with the syntax of P_IC_TE_X commands, which uses spaces, angle brackets and commas. For example, where small angle brackets are shown (e.g. `<...>`) then these must be used exactly as shown below, including spaces (shown here as `< >`). Thus, if you wish to use a following comment (`% ...`) on the same line as a P_IC_TE_X command, then you must ensure there is a trailing space (`< >`) after the command. For example, you would implement a dash pattern of dash (4pt) alternating with a gap of 2pt using the following command.

```
\setdashpattern<4pt,<2pt>% this command sets the pattern
```

The more extensive summary of P_IC_TE_X commands (`pictexsum.pdf`) by Duggan (1990) mentioned above will certainly suffice for most people's use.

```
\grid {cols} {rows} %% eg. \grid {5} {10}
\nogridlines
\plot
\setdashpattern <4pt, 2pt>
\setdashes <7pt> %% equivalent to \setdashpattern <7pt, 7pt>
\setdashes %% default is \setdashes <5pt>
\setdots <4pt>
\setdots %% default is \setdots <5pt>
\setsolid %% sets solid-line mode (ie not dashes/dots)
\setlinear %% sets straight-line plotting mode
%% (ie not quadratic)
\setquadratic %% sets curved plotting for graphs etc
\setplotsymbol({\large .})
\setplotsymbol({\tiny .}) %% default size for curves
%% (used by \plot)
\plotsymbolspacing=2pt %% sets spacing between plot symbols
%% (used by \plot)
\plotheadings{..} %% eg \plotheadings{This is a quadratic}
\plot x1 y1 x2 y2 ... / %% the <space>/ is the list terminator
\headingtoplotskip=1cm %% separation between plotheadings & graph
\linethickness=2pt %% for horiz & vert lines: default 0.4pt
\frame <sep> {text} %% eg \frame <5pt> {Hello}
\rectangle <width> <height> %% eg \rectangle <2cm> <1cm>
\putrectangle corners at 5 10 and 30 5 %% corners at Top-left
```

```

%% and Bottom-right
\inboundscheckon      %% restricts plotting to plotting area
\inboundscheckoff    %%
\normalgraphs         %% restores default values for
%% graph parameters (see Section 4.6.1)

\ticksin
\ticksout
\visibleaxes
\invisibleaxes
\circulararc 30 degrees from 3.5 4.5 center at 5 5
\ellipticalarc axes ratio 2:1 360 degrees from 3 3 center at 5 5
%%(axes ratio is major-axis:minor-axis, i.e. a:b)

```

Note that P_ICT_EX also has an excellent rotation facility (but only in conjunction with P_ICT_EX function commands, e.g., `\ellipticalarc` as shown below, and so will not rotate points input by `mathsPIC` via a data-file). P_ICT_EX will rotate about a given point, by a given angle, all picture elements (but not text) which are detailed between its `\startrotation...` and `\stoprotation` commands. However the decimal value of the Sine and Cosine angle must be supplied (see below). If a rotation point is not specified then rotation is performed about the origin. The format is as follows.

```

\startrotation by cos(t) sin(t) [about x y]
...
\stoprotation

```

This command is particularly useful for rotating curves. For example, to rotate an ellipse by 30 degrees about the point (5,5) one would write

```

\startrotation by 0.86602 0.5 about 5 5
\ellipticalarc.....
\stoprotation

```

Note that the P_ICT_EX `\startrotation` command would be easily amenable to being rewritten as a `mathsPIC` macro (to avoid calculating the radian values etc).

A P_ICT_EX error-bar facility is also available by loading the file `errorbars.tex` (see Section E.4). Note that P_ICT_EX does allow more sophisticated graph axes and tick-marks to be setup, as well as shading of enclosed areas. However, these are complicated and require access to the P_ICT_EX Manual (see Section E.6), and are currently outside the scope of `mathsPIC`.

5.2 Using the \$ symbol with P_ICT_EX

Because P_ICT_EX was originally written for Plain T_EX there is a difficulty when using the \$ symbol in the labels for axes. The ‘work-around’ fix for this is to re-encode it as, say, `\dollar`, and express \$2 as `{\dollar}2`, as shown in the following example which has dollars on the y-axis (note the % at the end of the first line to stop unwanted horizontal space being grabbed by P_ICT_EX and pushing the Y-axis slightly to the right).

```

\newcommand{\dollar}{\char'44}%
\setcoordinatesystem units <1mm, 1mm>
\setplotarea x from 0 to 50, y from 0 to 50

```

```
\axis bottom ticks numbered from 0 to 50 by 10 /  
\axis left ticks withvalues {\dollar}1 {\dollar}2 / at 10 20 / /  
\plotheadng{A nice picture costing \$25}
```

Chapter 6

TEX and L^ATEX commands

A mathsPIC file (script) can contain any appropriate TEX and L^ATEX commands, and most of these commands are for use in the preamble (e.g., headers and footers).

6.1 Headers and footers

It is particularly useful to include in the mathsPIC file any TEX or L^ATEX headers and footers which would otherwise have to be added manually to the output file before L^ATEXing the file. Note that the L^ATEX command `\thispagestyle{empty}` is included for convenience in `mathspic.sty` in order to stop a page number appearing in a figure (see Section 2.7 for details).

For example, when using L^ATEX a typical format for a mathsPIC file might be as follows (for use with plain TEX see Section 3.2).

```
\documentclass[a4paper]{article}
\usepackage{mathspic, color}
\begin{document}
%-----
\beginpicture
...
...
\endpicture
%-----
\end{document}
```

Note that when using TEX or L^ATEX commands *within* the P_{IC}TEX picture environment, it is very important to include the comment `%` symbol at the end of such lines, to prevent P_{IC}TEX accumulating additional `<space>` characters from the ends of non-P_{IC}TEX commands, since otherwise P_{IC}TEX incorporates such spaces into the horizontal distance used for representing x -coordinates, with the effect that all subsequent picture elements will be displaced slightly to the right.

If it is necessary (or simply convenient) to extent a TEX or L^ATEX command across several lines, then each additional line must be protected by using a leading `_` (a backslash followed by one or more spaces) unless the line actually starts with a TEX or L^ATEX command. A typical example is in the following macro (used in Figure 7.6)

which defines a ‘display’ maths formula. The macro is split across several lines as follows:

```
\newcommand{\formula}{%
\    $\displaystyle \sum_{p\ge 0} \Delta_{jp} z^{(p+1)}$%
\    }%
text(\formula){B}
```

6.2 The `\thispagestyle{empty}` command

This is an important command to include in the preamble, since this disables page numbering on the same page as the diagram, and for this reason this command is included in `mathspic.sty` (see Section 2.7).

If you intend to crop the image automatically using the `-E` option with `DVIPS` command to make an EPS and PDF image, for example,

```
dvips -E myfigure.dvi -o myfigure.eps
epstopdf myfigure.eps
```

then if a page number has been typeset then the rest of the page including the page number will be included in the cropping.

Of course, if you intend to use `mathsPIC` commands in the main `.tex` document (as opposed to simply using the EPS or PDF image) then this command can be edited out of the file `mathspic.sty` as described in Section C.3.

6.3 The `\typeout{}` command

This is a very useful command which makes T_EX print comments to the screen while the output-file is being processed. For example, the following commands in the `mathsPIC` file will print a message to the screen just before processing a data-file for a curve.

```
\typeout{processing the data-file now}%
inputfile*(curve.dat)
```

This command is also useful when a file is input several times in a loop using the `mathsPIC inputFile{. . }[]` command. In such cases it is quite useful to include the line `\typeout{. . }%` at the beginning of the file being input, as this results in T_EX printing `. . .` to the screen each time the file is input.

Note the importance of including the comment `%` symbol at the end of the line when the `\typeout{}` command is used **within** the P_CT_EX picture environment—this applies mainly to lines ending with the right brace `}` of a T_EX or L^AT_EX command (see Section 3.2).

6.4 The Color package

The standard L^AT_EX Color package can be used to draw parts of a diagram in colour, in much the same way one would when dealing with text (note that the color package must be installed *after* `mathspic`). One simply inserts the the usual L^AT_EX ‘color’ commands between `mathsPIC` commands (preferably on a separate line).

Since the `\color{}` command is a L^AT_EX command, it is important to include the comment `%` symbol at the end of the command when it is used **within** the P_CT_EX picture environment. For example, the following code will draw the line *PQ* in blue.

```
\usepackage{mathspic, color}
%-----
\beginpicture
....
....
\color{blue}%
\drawline(PQ)
\color{black}%
....
....
\endpicture
```

Note however, that while all PS and PDF viewers show colours, there are some DVI-viewers which do not currently show colour.

Colour spilling (old T_EX systems only): If you find that colour is spilling into unexpected zones (say, into ordinary text following a figure), then this can be fixed by using the L^AT_EX command `\normalcolor%` immediately before the `\endpicture` command.

6.5 Other useful L^AT_EX commands

For more details of these useful commands see Knuth (1990) p 272–279; Eijkhout (1992); Salomon (1995). See also Chapter 8 for details of using L^AT_EX commands for accessing parameter values. Note, finally, that it is important to put a `%` on the end of these L^AT_EX commands in order to prevent P_CT_EX from adding spaces to the end.

```
\typeout{...}
\thispagestyle{empty}
\scrollmode %(forces TeX to display errors but not to stop)
\nonstopmode
\batchmode
\newlength{}
\newcommand{}
\color{..}
\normalcolor
\settowidth{}
\settolength{}
\settodepth{}
\write
\newwrite
\immediate\write18{...}
\openout
\closeout
\the
\number
\showthe
\jobname
```

Chapter 7

Examples

This section describes some practical examples of figures drawn using `mathsPIC`, together with the associated code.

When drawing a new figure or diagram, it seems best to start with a graduated coordinate frame (see Figure 7.1) using the `axes` and `ticks` options of the `paper` command. The next step is to define an *anchor point* from which other points can be derived—this has the advantage that the whole figure can then be moved by simply changing the coordinates of the anchor point. If necessary, different parts of a complicated figure can be made having their own separate anchor points, allowing the various parts to be easily adjusted relative to each other.

Remember that the command `\thispagestyle{empty}` is, for convenience, implemented by `mathspic.sty` (see Section 2.7), in order to prevent page numbers appearing in a figure.

As regards scales and units, it seems most convenient to use the 1mm units and to keep the x and y scales the same whenever possible (i.e. use `paper{units(1mm) . . .}`), since this allows easy scaling up and down after the figure has been finished, as shown in Section 7.9.

In practice, the whole process of editing and viewing can be conveniently automated using a small script, to run `mathsPIC`, trim the image to the bounding box (BBox) and generate the required file formats, as described in Section C.1.

7.1 Input- and output-files

The following example `mathsPIC` file (input-file) illustrates how some of these commands are used to draw Figure 7.1. Note that the dashed line BD is drawn after the $\text{\P\TeX}\ \backslash\text{setdashes}$ command is invoked; following this `\setsolid` is used before drawing the right-angle symbol. Also, the points A, B, C are defined using the $\text{\TeX}\ \backslash\text{odot}$ symbol \odot , in conjunction with a line-free zone of 1.2 mm in order to make the lines go to the edge of the symbol—the value of the radius of such $\text{\TeX}\ \backslash\text{odot}$ symbols has to be determined by trial and error—see Table 1.

```
% mpicpm07-1.m (Figure 7.1)
\documentclass[a4paper]{article}
\usepackage{mathspic}
```

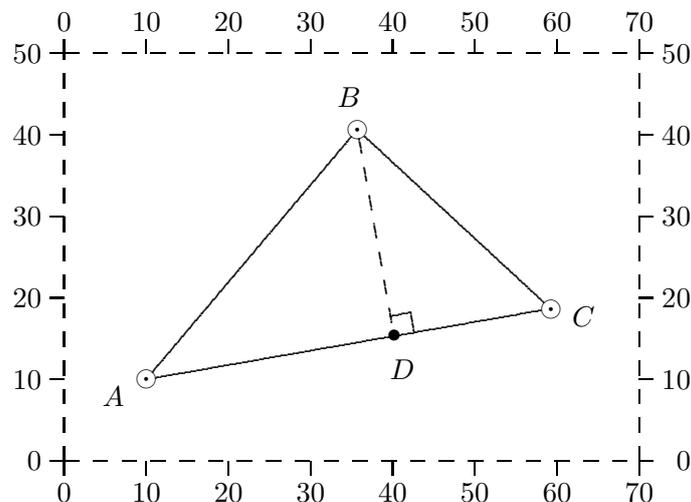


Figure 7.1:

```

\begin{document}
%-----
\beginpicture
\setdashes
paper{units(1mm),xrange(0,70),yrange(0,50),axes(LRTB),ticks(10,10)}
\setsolid
point(A){10,10}[symbol=$\odot$, radius=1.2]      %% anchor point
point(B){A, polar(40, 50 deg)}[symbol=$\odot$, radius=1.2]
point(C){A, polar(50, 10 deg)}[symbol=$\odot$, radius=1.2]
point(D){perpendicular(B,AC)}
drawPoint(ABCD)
drawLine(ABCA)
\setdashes
drawLine(BD)
\setsolid
drawRightangle(BDC,2.5)
text($B$){B, shift(-1,4)}
text($A$){A, shift(-4,-2)}
text($C$){C, shift(4,-1)}
text($D$){D, shift(1,-4)}
showLength(BD)
showLength(AC)
showArea(ABC)
showAngle(ABC)
\endpicture
\end{document}

```

When the above mathsPIC file is processed by mathsPIC the output-file (.mt file) is as follows. Note how the \LaTeX commands are preceded by their mathsPIC commands (commented out), some of which have additional information (e.g. the coordinates of a derived point—see the line `%% point(D) . . .`). Being able to compare the mathsPIC

commands and the resulting \LaTeX commands in the output-file is particularly useful when debugging. Note also how the `show...` commands at the end of the file return the lengths AD, BC, the area ABC, and the angle ABC.

```

%* -----
%* mathsPIC (Perl version 0.99.22 Dec 29, 2004)
%* A filter program for use with PiCTeX
%* Copyright (c) 2004 A Syropoulos & RWD Nickalls
%* Command line: mpic09922.pl -b mpicpm07-1.m
%* Input filename : mpicpm07-1.m
%* Output filename: mpicpm07-1.mt
%* Date & time: 2005/01/05 10:00:40
%* -----
%% mpicpm07-1.m (figure 7.1)
%% mathsPIC Perl triangle
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
\setdashes
% paper{units(1mm),xrange(0,70),yrange(0,50),axes(LRTB),ticks(10,10)}
\setcoordinatesystem units <1mm,1mm>
\setplotarea x from 0.00000 to 70.00000, y from 0.00000 to 50.00000
\axis left ticks numbered from 0 to 50 by 10 /
\axis right ticks numbered from 0 to 50 by 10 /
\axis top ticks numbered from 0 to 70 by 10 /
\axis bottom ticks numbered from 0 to 70 by 10 /
\setsolid
%% point(A){10,10}[symbol=$\odot$, radius=1.2]
%% anchor point A = (10.00000, 10.00000)
%% point(B){A, polar(40, 50 deg)}[symbol=$\odot$, radius=1.2]
%% B = (35.71150, 40.64178)
%% point(C){A, polar(50, 10 deg)}[symbol=$\odot$, radius=1.2]
%% C = (59.24039, 18.68241)
%% point(D){perpendicular(B,AC)} D = (40.17626, 15.32089)
%% drawPoint(ABCD)
\put {$\odot$} at 10.00000 10.00000 %% A
\put {$\odot$} at 35.71150 40.64178 %% B
\put {$\odot$} at 59.24039 18.68241 %% C
\put {$\bullet$} at 40.17626 15.32089 %% D
%% drawLine(ABCA)
\plot 10.77135 10.91925 34.94015 39.72253 / %% AB
\plot 36.58878 39.82302 58.36311 19.50117 / %% BC
\plot 58.05862 18.47403 11.18177 10.20838 / %% CA
\setdashes
%% drawLine(BD)
\plot 35.91988 39.46001 40.17626 15.32089 / %% BD
\setsolid
%% drawRightangle(BDC,2.5)

```

```

\plot 42.63828 15.75501 42.20416 18.21703 /
\plot 39.74214 17.78291 42.20416 18.21703 /
%% text($B$){B, shift(-1,4)}
\put {$B$} at 34.711500 44.641780
%% text($A$){A, shift(-4,-2)}
\put {$A$} at 6.000000 8.000000
%% text($C$){C, shift(4,-1)}
\put {$C$} at 63.240390 17.682410
%% text($D$){D, shift(1,-4)}
\put {$D$} at 41.176260 11.320890
%% length(bd) = 25.7115062228898
%% length(ac) = 50.0000025076019
%% area(abc) = 642.7877063896
%% angle(abc) = 86.97613 deg ( 1.51802 rad)
\endpicture
\end{document}

```

If the above output-file (.mt file) is to be included or `\input` into a document (say, into a figure environment), it is sometimes useful to reduce the size of the file by disabling the writing of `%%` comment lines (using the command-line `-c` switch), as well as some of the header and footer lines which are not now required, as follows.

```

%* -----
%* mathsPIC (Perl version 0.99.22 Dec 29, 2004)
%* A filter program for use with PiCTeX
%* Copyright (c) 2004 A Syropoulos & RWD Nickalls
%* Command line: mpic09922.pl -c mpicpm07-1.m
%* Input filename : mpicpm07-1.m
%* Output filename: mpicpm07-1.mt
%* Date & time: 2005/01/05 11:50:14
%* -----
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%-----
\beginpicture
\setdashes
\setcoordinatesystem units <1mm,1mm>
\setplotarea x from 0.00000 to 70.00000, y from 0.00000 to 50.00000
\axis left ticks numbered from 0 to 50 by 10 /
\axis right ticks numbered from 0 to 50 by 10 /
\axis top ticks numbered from 0 to 70 by 10 /
\axis bottom ticks numbered from 0 to 70 by 10 /
\setsolid
\put {$\odot$} at 10.00000 10.00000 %% A
\put {$\odot$} at 35.71150 40.64178 %% B
\put {$\odot$} at 59.24039 18.68241 %% C
\put {$\bullet$} at 40.17626 15.32089 %% D
\plot 10.77135 10.91925 34.94015 39.72253 / %% AB
\plot 36.58878 39.82302 58.36311 19.50117 / %% BC

```

```

\plot 58.05862 18.47403      11.18177 10.20838 / %% CA
\setdashes
\plot 35.91988 39.46001      40.17626 15.32089 / %% BD
\setsolid
\plot 42.63828 15.75501      42.20416 18.21703 /
\plot 39.74214 17.78291      42.20416 18.21703 /
\put {$B$} at 34.711500 44.641780
\put {$A$} at 6.000000 8.000000
\put {$C$} at 63.240390 17.682410
\put {$D$} at 41.176260 11.320890
%% length(bd) = 25.7115062228898
%% length(ac) = 50.0000025076019
%% area(abc) = 642.7877063896
%% angle(abc) = 86.97613 deg ( 1.51802 rad)
\endpicture
\end{document}

```

Note that the remaining comments at the end of the `\plot` and `\put` lines are generally sufficient to understand what each line relates to. Note also that the `-c` switch only removes comment lines having a leading `%%` pair of characters (but not the results of the `show()` commands); comment lines prefixed with only a single `%` character remain.

7.2 Line modes

When drawing figures with both solid and dashed lines it is necessary to switch between the $\text{P}\text{T}\text{E}\text{X}$ commands `\setdashes` and `\setsolid`, as in the following code for drawing the rectangular box shown in Figure 7.2. The default `\setdashes` gives alternating lines and spaces, each of width 5pt.

More fancy dash-patterns (see Figure 7.2) can be easily generated using the `mathsPIC` `dasharray()` command which defines the pattern cycle, and hence takes an even number of parameters. For example the command

```
dasharray(6pt,2pt,1pt,2pt)
```

generates the dash pattern used for lines *AH* and *DE* in Figure 7.2.

If several different dash patterns are required then it may be useful to define them as separate `mathsPIC` macros, as follows for example.

```
%def fancydashes() dasharray(6pt,2pt, 1pt, 2pt)
%def simpledashes() dasharray(4pt,4pt)
```

The separate patterns can then be called as required as follows (see Figure 7.2).

```

...
fancydashes
drawline(AH, DE)
simpledashes
drawline(BG, CF)
...

```

The equivalent $\text{P}\text{T}\text{E}\text{X}$ command is the `\setdashpattern` command which also takes an even number of parameters but has a slightly different syntax as follows.

```
\setdashpattern <6pt,2pt,1pt,2pt>
```

The equivalent L^AT_EX macro would be as follows.

```
\def\fancydashes{\setdashpattern <6pt, 2pt, 1pt, 2pt>}%
...
\fancydashes
```

Note that in Figure 7.2 all the points are defined, directly or indirectly, relative to point *A*, with the effect that the position of the whole figure can be adjusted simply by altering the coordinates of point *A*. This can be useful when drawing diagrams having several components since the relative position of each component can then be easily adjusted.

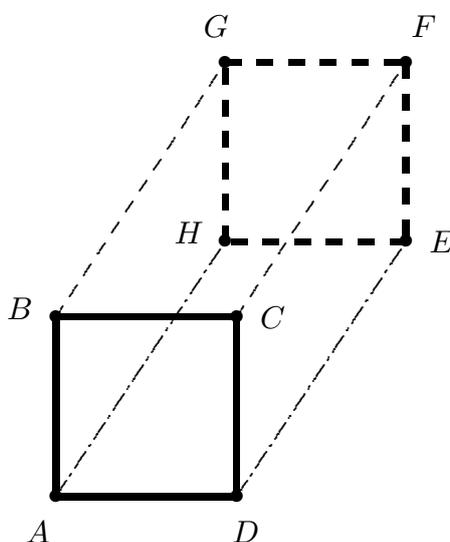


Figure 7.2:

```
% mpicpm07-2.m (Figure 7.2)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
paper{units(1mm), xrange(0,50), yrange(0,62)}
var s= 20 % Sides front & back
var L = 34 % Length
var a2 = 56.6 % angle degrees
%def fancydashes() dasharray(6pt,2pt, 1pt, 2pt)%
%def simpledashes() dasharray(4pt,4pt)%
point(A){5,7}
point(B){A, polar(s,90 deg)}
point(C){B, polar(s,0 deg)}
point(D){A, polar(s,0 deg)}
point(H){A, polar(L,a2 deg)}
```

```
point(G){B, polar(L,a2 deg)}
point(F){C, polar(L,a2 deg)}
point(E){D, polar(L,a2 deg)}
drawpoint(ABCDEFGH)
linethickness(2pt)
  \setsolid
drawline(ABCD)
  \setdashes
drawline(HGFEH)
linethickness(default)
&fancydashes
drawline(AH, DE)
&simplifiedashes
drawline(BG, CF)
text($A$){A, shift(-2,-4)}
text($B$){B, shift(-4,1)}
text($C$){C, shift(4,0)}
text($D$){D, shift(1,-4)}
text($E$){E, shift(4,0)}
text($F$){F, shift(2,4)}
text($G$){G, shift(-1,4)}
text($H$){H, shift(-4,1)}
\endpicture
\end{document}
```

7.3 Arrows

Arrows can be drawn in all possible orientations, will *stretch* between points, and arrow-heads are readily customised using the mathsPIC Arrowshape command (see also Salomon, 1992).

Although arrow shape can of course be controlled using the standard \LaTeX `\arrow` command, the mathsPIC Arrowshape command makes this easier by allowing you to define (in degrees) the angle parameters (B and C) of the arrow-head directly (see box). The default arrowshape is equivalent to the following command

```
Arrowshape(2mm,30,40)
```

and can be invoked using the command

```
ArrowShape(default)
```

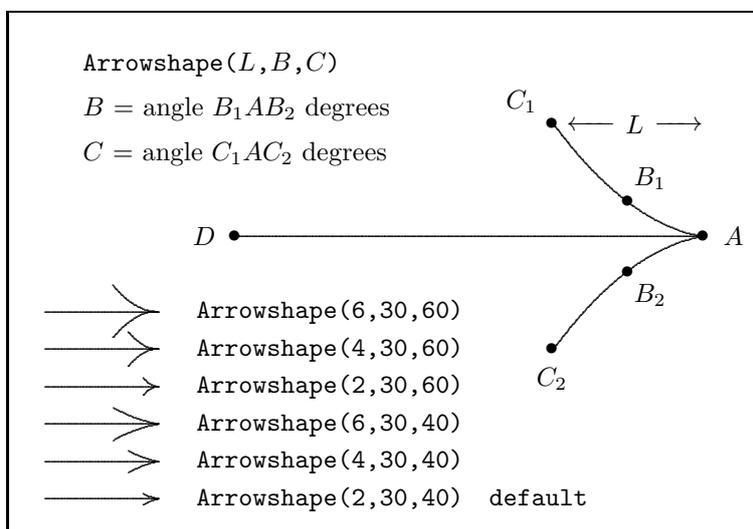


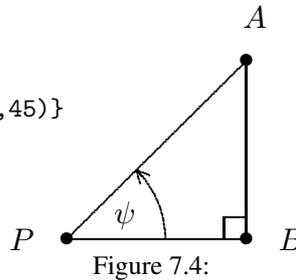
Figure 7.3: The mathsPIC code for this figure is given in the appendix

If the arrowshape has been altered, it can be reset using the `ArrowShape(default)` command. Curved arrows (circular arcs) are drawn using the `drawAngleArrow` command, which takes parameters for the angle, radius of arc, direction, and whether the angle is internal or external (see Figures 7.4 and 7.5).

```

%% mpicpm07-4.m (Figure 7-4)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
paper{units(1mm),xrange(5,45),yrange(5,45)}
point(A){30,30}
point(P){10,10}
point(B){30,10}
drawPoint(APB)
drawLine(APBA)
var d = 5
text($A$){A,shift(1,d)}
text($B$){B,shift(d,0)}
text($P$){P,shift(-d,0)}
drawAngleArrow{angle(BPA),radius(11),internal, anticlockwise}
text($\psi$){P,polar(7,22.5 deg)}
drawRightangle(ABP,2.5)
\endpicture
\end{document}

```



Arrows can also be used to link elements in a diagram, as shown in Figure 7.5. The right-hand diagram uses the `drawArrow` command; the small gap between the arrows and the letters P, Q, R, T being due to the 5 unit line-free radius associated with these points (see Figure 7.5b). The arrows are easily ‘stretched’ to accommodate their labels simply by adjusting the separation of the nodes using the `polar` (r, θ) commands (see Feruglio, 1994).

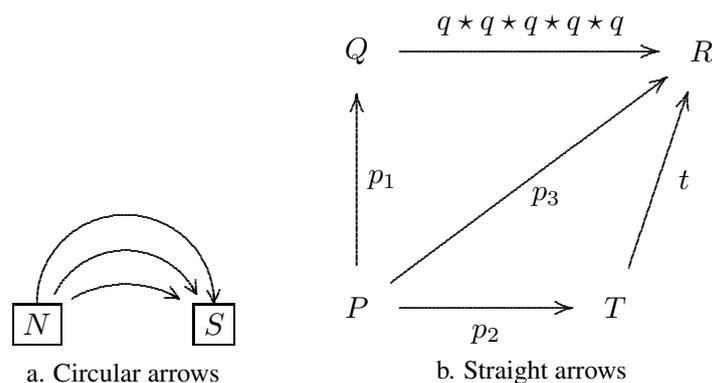


Figure 7.5:

```

%% mpicpm07-5a.m (Figure 5a)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}

```

```

%%-----
\beginpicture
paper{units(1mm),xrange(10,40),yrange(0,45)}%, axes(LB), ticks(10,10)}
point(N){15,20}
point(S){N,shift(20,0)}
text(\framebox{N$}){N,shift(0,-2.5)}
text(\framebox{S$}){S,shift(0,-2.5)}
point(Z){midpoint(NS)}
drawAngleArrow{angle(NZS),radius(NZ),internal,clockwise}
point(N1){N,shift(2,1)}
point(S1){S,shift(-2,1)}
point(Z1){Z,shift(0,-3)}
drawAngleArrow{angle(N1Z1S1),radius(N1Z1),internal,clockwise}
point(N2){N1,shift(2,-0.5)}
point(S2){S1,shift(-2,-0.5)}
point(Z2){Z,shift(0,-10)}
drawAngleArrow{angle(N2Z2S2),radius(N2Z2),internal,clockwise}
\endpicture
\end{document}

```

```

%% mpicpm07-5b.m (Figure 7.5b)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
paper{units(1mm),xrange(0,45),yrange(0,45)}%, axes(LB), ticks(10,10)}
point(P){5,10}[symbol=$P$,radius=5]
point(Q){P,polar(30,90 deg)}[symbol=$Q$,radius=5]
point(R){Q,polar(40,0 deg)}[symbol=$R$,radius=5]
point(T){P,polar(30,0 deg)}[symbol=$T$,radius=5]
drawPoint(PQRT)
drawArrow(PQ,QR,PT,TR,PR)
point(P1){midpoint(PQ)}
text($p_1$){P1,shift(3,0)}
point(P2){midpoint(PT)}
text($p_2$){P2,shift(0,-3)}
point(P3){midpoint(PR)}
text($p_3$){P3,shift(2,-2)}
point(T1){midpoint(TR)}
text($t$){T1,shift(3,0)}
point(Q1){midpoint(QR)}
%% use a LaTeX macro for the label
\newcommand{\q}{\q \star q \star q \star q \star q}
text(\q){Q1,shift(-1,3)}
\endpicture
\end{document}

```

7.4 Circles & colour

MathsPIC allows the point-symbol to be designated as a circle using the option `[symbol=circle(r),radius=z]` to the `point()` command, which not only gives the circles an internal line-free zone, but also arranges that they are drawn by the `drawPoint` command, as shown in Figure 7.6. This construction greatly simplifies the drawing of directed graphs, trees and equivalent structures. In Figure 7.6 we have also made use of the L^AT_EX Color package to make the circles red, the lines blue, and the text and labels black, as can be seen from the code in the `mathsPIC` file. Some DVI-viewers may not show colours.

```
%% mpicpm07-6.m (figure 7.6)
%% 4 circles figure
\documentclass[a4paper]{article}
\usepackage{mathspic,color}
\begin{document}
%%-----
\beginpicture
paperfunits(1mm),xrange(0,70),yrange(0,60)}%
point(A){30,11}[symbol=circle(8),radius=8]
point(B){A,shift(-10,30)}[symbol=circle(15),radius=15]
point(C){A,polar(30,20 deg)}[symbol=circle(5),radius=5]
point(D){A,polar(45,50 deg)}[symbol=circle(7),radius=7]
\color{red}%
drawPoint(ABCD)
\color{blue}%
drawLine(AB,AC,BC,BD,CD)
\setdashes
drawLine(AD)
\color{black}%
text({A$}){A}
%% use a macro for the formula
\newcommand{\formula}{%
\

$$\sum_{p \geq 0} \Delta_{jp} z^{(p+1)}$$

}%
text(\formula){B}
text({C$}){C}
text({D$}){D}
\endpicture
\end{document}
```

Note that in this particular case it is necessary to define the maths formula using the L^AT_EX `\displaystyle`, in order to avoid the embedded `\textwidth` white space associated with using `\vbox{\formula}` in the `\text()` command, and hence allow centering of the figure. Note also how the `mathsPIC _ . . .` commands make it easy to include multi-line macros in the `mathsPIC` file.

Points on circles (and their labels) are most easily defined and positioned using the `polar(r,θ)` option, as shown in the `mathsPIC` file for Figure 7.7. Notice the use of the variable `r` for the radius of the circle (allocated using the command `var r = 20`),

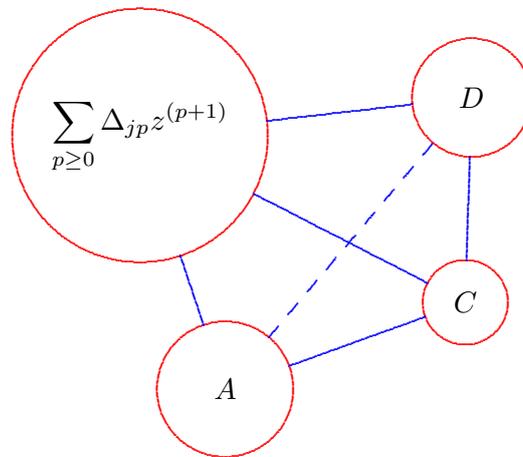


Figure 7.6:

which then allows the use of `r` to define the radius in the polar commands for the points P, Q, S .

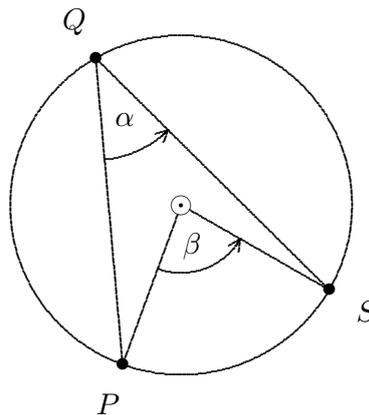


Figure 7.7:

```
%% mpicpm07-7.m (Figure 7.7)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\begin{picture}
paper{units(1mm), xrange(5,55), yrange(5,55)}
point(C){30,30}[symbol=$\odot$,radius=1.2] %% center
var r = 20 %% radius
drawcircle(C,r)
point(P){C, polar(r,250 deg)}
point(Q){C, polar(r,120 deg)}
```



```

\beginpicture
\setdashes
paper{units(1mm),xrange(0,70),yrange(0,60),axes(LBT*R*),ticks(10,10)}
\setsolid
point(A){10,10} %% anchor point
point(B){A, polar(50,50 deg)}
point(C){A,polar(50,0 deg)}
point(J){pointonline(AB,30)}
point(K){perpendicular(J,AC)}
drawRightangle(JKC,3)
drawLine(AB,AC,JK)
drawIncircle(AJK)
drawExcircle(AJK,JK)
\setplotsymbol({\large .}) \setdots
drawCircumcircle(AJK)
\setplotsymbol({\tiny .})
point(I){IncircleCenter(AJK)}[symbol=$\odot$, radius=1.2]
point(E){ExcircleCenter(AJK,JK)}[symbol=$\odot$, radius=1.2]
point(P1){perpendicular(E,AC)}
var r = EP1 %% radius of excircle
var d = 72 %% angle of pentagon (deg)
var a1 = -90, a2=a1+d, a3=a2+d, a4=a3+d, a5=a4+d
point(P2){E, polar(r,a2 deg)}
point(P3){E, polar(r,a3 deg)}
point(P4){E, polar(r,a4 deg)}
point(P5){E, polar(r,a5 deg)}
drawPoint(ABCJKIEP1P2P3P4P5)
\setdashes
drawLine(P1P2P3P4P5P1,EP1,EP2)
\setsolid
drawAngleArc{angle(P2EP1),radius(9),internal,clockwise}
\newcommand{\figtitle}{%
  \fbox{%
    \begin{minipage}{30mm}%
      \ Triangle, pentagon and three circles%
    \end{minipage}%
  \ }}%
text(\figtitle){20,52}
var s = 5
text($A$){A,polar(s,230 deg)}
text($B$){B,polar(s,50 deg)}
text($C$){C,polar(s,0 deg)}
text($J$){J,polar(s,90 deg)}
text($K$){K,polar(s,270 deg)}
text($E$){E,polar(s,a3 deg)}
text($72$){E,polar(5.5,-54 deg)}
text($I$){I,shift(3, 0)}
text($P_1$){P1,polar(s, a1 deg)}
text($P_2$){P2,polar(s, a2 deg)}
text($P_3$){P3,polar(s, a3 deg)}

```

```
\endpicture
\end{document}
```

7.5 Functionally connected diagrams

When constructing diagrams it is often useful to write the mathsPIC file in such a way that the position of each new point is related to that of earlier points, since then the structure of the diagram is maintained even when points are moved as shown in Figure 7.9. Although the two diagrams appear to be quite different the mathsPIC code for the two diagrams differs *only* in the angle of the line AB (left diagram, 60 degrees; right diagram, 5 degrees) as defined in the `point(B){...}` command as follows.

- Left-hand diagram: `point(B){A,polar(45,60 deg)}`
- Right-hand diagram: `point(B){A,polar(45,5 deg)}`

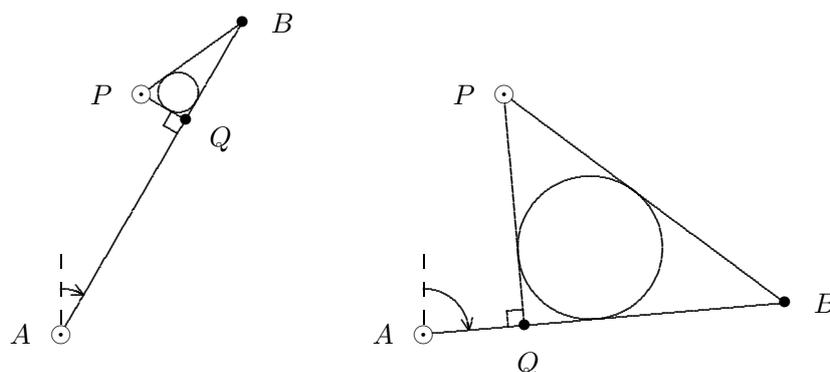


Figure 7.9: The mathsPIC code for the two diagrams differs *only* in the angle of the line AB as defined in the `point(B){...}` command (see `mpicpm07-9.m`).

```
%% mpicpm07-9.m (Figure 7.9)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
paper{units(1mm), xrange(5,110), yrange(0,45)}
point(A){15,5}[symbol=$\odot$, radius=1.2]
point(P){A, shift(10,30)}[symbol=$\odot$, radius=1.2]
point(B){A, polar(45,60 deg)}
point(Q){perpendicular(P,AB)}
drawRightangle(PQA,2)
drawPoint(ABPQ)
drawLine(ABPQ)
drawIncircle(PQB)
var d = 5
```

```

text($A$){A,shift(-d, 0)}
text($B$){B,shift(d, 0)}
text($P$){P,shift(-d, 0)}
point(S){pointOnLine(QP,-5)}
text($Q$){S}
point(N){A,shift(0,10)}
\setdashes
drawline(AN)
\setsolid
drawAngleArrow{angle(NAB), radius(5.7), internal, clockwise}
%%----- second figure -----
point*(A){60,5}[symbol=$\odot$, radius=1.2]
point*(P){A, shift(10,30)}[symbol=$\odot$, radius=1.2]
point*(B){A, polar(45,5 deg)} %%5 60 deg
point*(Q){perpendicular(P,AB)}
drawRightangle(PQA,2)
drawPoint(ABPQ)
drawLine(ABPQ)
drawIncircle(PQB)
text($A$){A,shift(-5, 0)}
text($B$){B,shift(5, 0)}
text($P$){P,shift(-5, 0)}
point*(S){pointOnLine(QP,-5)}
text($Q$){S}
point*(N){A,shift(0,10)}
\setdashes
drawline(AN)
\setsolid
drawAngleArrow{angle(NAB), radius(5.7), internal, clockwise}
\endpicture
\end{document}

```

Note that in Figure 7.9 the location of the label ‘Q’ is made to lie outside the figure by placing the label at point S, which is defined as being 5 mm to the right of the line AB and in-line with the points PQ, using the command `point(S){pointOnLine(QP,-5)}`.

7.6 Inputting the same data-file repeatedly

There are two methods:

- use the `beginloop [n] ... endloop` environment
The file `myfile.dat` is input six times sequentially using the commands

```

beginLOOP 6
  inputfile(myfile.dat)
endLOOP

```

- use the `inputfile [n]` command

The file `myfile.dat` is input six times sequentially using the single command `inputfile(myfile.dat) [6]`.

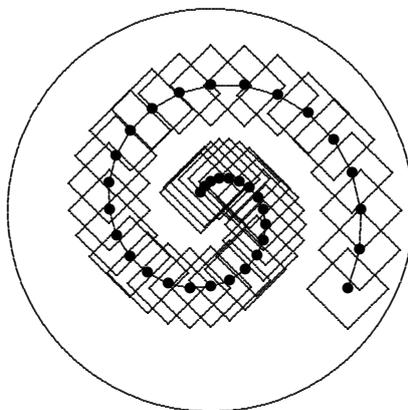


Figure 7.10:

Figure 7.10 was produced by the following code which inputs a small file of mathSPIC code (`mpicpm07-10.dat`) repeatedly 35 times using the command `inputfile(mpicpm07-10.dat) [35]`. Note the use of the `point*` commands in the data-file to re-allocate points.

```
%% mpicpm07-10.m (figure 7-10)
%% mathsPIC small spiral by recursion
%% (requires datafile mpicpm07-10.dat)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
paper{units(1mm), xrange(0,60), yrange(0,60)}% axes(LB), ticks(10,10)}
point(C){30,30} % circle center
drawcircle(C,25)
%% initialise the reusable points and variables
var a=315 % angle degrees
var r=20 % start radius
var s=5 % square semi-diagonal (see datafile)
point(T){C,polar(r,330 deg)}
%% cycle datafile 35 times
inputfile(mpicpm07-10.dat)[35]
\endpicture
\end{document}

%% mpicpm07-10.dat -----first line-----
%% mathsPIC (spiral data) input by file mpicpm07-10.m
var a = a+15, r = r-0.5 %% increment angle and radius
point*(P){C,polar(r,a deg)} % increment point P
drawpoint(P)
```

```

drawline(TP)          %% draw line from OLD T to NEW P
point*(T){P}         %% reallocate T <-- P
%% make a rotated square centered on P with side/2=s
point*(Q1){P,polar(s,0 deg)}
point*(Q2){P,polar(s,90 deg)}
point*(Q3){P,polar(s,180 deg)}
point*(Q4){P,polar(s,270 deg)}
drawline(Q1Q2Q3Q4Q1)
%% eof =====

```

When the `inputfile` command is used `mathsPIC` writes the iteration number to the output file, prefixed using three `%` symbols (to make sure they are not deleted when using the `-c` switch, as follows.

```

%% Iteration number: 26

```

For example, the following extract shows the resulting output `.mt` file showing how `mathsPIC` writes the iteration number of each data-file input to the output file just before it inputs the data-file. This example includes the whole of the 26th cycle of data-file input. It also demonstrates the value of labelling the beginning and the end of the data-file differently in order to make it easy to see the cycling of the input file. This is very helpful as it allows you to check whether the data-file is working correctly.

```

...
...
\plot 31.49519 26.25000          36.49519 21.25000 / %% Q3Q4
\plot 36.49519 21.25000          41.49519 26.25000 / %% Q4Q1
%% eof =====
%% Iteration number: 26
%% mpicpm07-10.dat -----first line-----
%% mathsPIC (spiral data) input by file mpicpm07-10.m
%% var a = a+15, r = r-0.5 %% increment angle and radius
%% a = 705
%% r = 7
%% point*(P){C,polar(r,a deg)} %increment P, P=(36.76148, 28.18827)
%% drawpoint(P)
\put {$\bullet$} at 36.76148 28.18827 %% P
%% drawline(TP)          %% draw line from OLD T to NEW P
\plot 36.49519 26.25000          36.76148 28.18827 / %% TP
%% point*(T){P} %% reallocate T <-- P          T = (36.76148, 28.18827)
%% make a square on P with side/2=s (s is defined in mpicpm07-10.m)
%% point*(Q1){P,polar(s,0 deg)}          Q1 = (41.76148, 28.18827)
%% point*(Q2){P,polar(s,90 deg)}          Q2 = (36.76148, 33.18827)
%% point*(Q3){P,polar(s,180 deg)}          Q3 = (31.76148, 28.18827)
%% point*(Q4){P,polar(s,270 deg)}          Q4 = (36.76148, 23.18827)
%% drawline(Q1Q2Q3Q4Q1)
\plot 41.76148 28.18827          36.76148 33.18827 / %% Q1Q2
\plot 36.76148 33.18827          31.76148 28.18827 / %% Q2Q3
\plot 31.76148 28.18827          36.76148 23.18827 / %% Q3Q4
\plot 36.76148 23.18827          41.76148 28.18827 / %% Q4Q1
%% eof =====

```

```

%%% Iteration number: 27
%% mpicpm07-10.dat -----first line-----
...
...

```

Because mathsPIC prefixes the iteration number comment with %%% these comments remain even when the `-c` switch is used, as the following extract of the same data shows.

```

...
...
\plot 31.49519 26.25000          36.49519 21.25000 / %% Q3Q4
\plot 36.49519 21.25000          41.49519 26.25000 / %% Q4Q1
%%% Iteration number: 26
\put {$\bullet$} at 36.76148 28.18827 %% P
\plot 36.49519 26.25000          36.76148 28.18827 / %% TP
\plot 41.76148 28.18827          36.76148 33.18827 / %% Q1Q2
\plot 36.76148 33.18827          31.76148 28.18827 / %% Q2Q3
\plot 31.76148 28.18827          36.76148 23.18827 / %% Q3Q4
\plot 36.76148 23.18827          41.76148 28.18827 / %% Q4Q1
%%% Iteration number: 27
\put {$\bullet$} at 36.50000 30.00000 %% P
...
...

```

7.6.1 Using the `beginloop...endloop` environment

The script for the previous spiral figure of squares can be written more simply (i.e. without repeatedly inputting a separate file) using the `beginloop...endloop` environment as shown in the following example. Some care needs to be taken in initialising (before the loop) all those quantities which get changed or incremented with each cycle of the loop. Note also that (a) since we are not using the `inputfile` command it is useful to create a loop counter (n) so we can see which loop is which when we read the output `.mt` file, (b) the squares will not be rotated in this particular example as we are using the `[symbol=square(s)]` option with the `point` command in order to generate the square, and (c) since all mathsPIC commands are case-insensitive we have chosen to capitalise the ‘loop’ of the commands for clarity.

```

\beginpicture
paper{units(1mm), xrange(0,60), yrange(0,60) axes(LB), ticks(10,10)}
point(C){30,30} % circle center
drawcircle(C,25)
%% initialise loop counter (n), angle (a), radius (r), side (s)
var n=0, a=315, r=20, s=7
point(T){C,polar(r,330 deg)}
beginLOOP 35 % loop 35 times
  %% increment loop counter (n), angle (a) and radius (r)
  var n=n+1, a = a+15, r = r-0.5
  %% increment position of new square with side s
  point*(P){C,polar(r,a deg)}[symbol=square(s)]
  drawpoint(P) % draw new square
  text($\bullet$){P} % draw bullet in center of new square
endLOOP
\endpicture

```

```

        drawline(TP)          % draw line from OLD T to NEW P
        point*(T){P}         % reallocate T <-- P
    endLOOP
\endpicture

```

7.6.2 Using L^AT_EX to cycle a loopcounter

A slightly irritating problem associated with processing diagrams which involve repeated loops is that while the .mt file is being processed by L^AT_EX there is no screen activity to show how things are proceeding.

In this example program we therefore illustrate a simple way of using some L^AT_EX register commands (Knuth (1990) p 118–121; 272–273; Eijkhout (1992) p 242–244) in order to make L^AT_EX indicate the loop status on the screen while it is processing the .mt file. This can be useful for debugging especially if the loop is cycled many times.

We first allocate a suitable name to an unused T_EX integer register (a so-called count register) using the T_EX `\newcount` command (here we have used the name `\loopcounter`), and then we initialise it to zero, as follows (all in the main calling program).

```

\newcount\loopcounter
\loopcounter=0

```

Then in the loop (or in the data file itself) we increment the counter and also print the result to the screen using the `\typeout{...}%` command, remembering to include a % symbol at the end.

```

\advance\loopcounter by 1
\typeout{loop = \the\loopcounter}%

```

A typical use might therefore be as follows:-

```

\beginpicture
\newcount\loopcounter %% allocate a TeX register
\loopcounter=0        %% initialise the register
%% create a convenient macro
\newcommand{\showloopnumber}{%
    \advance\loopcounter by 1          %increment TeX loop counter
    \typeout{loop = \the\loopcounter}% %print loop no to screen
    \ }%
...
beginLOOP 26
    \showloopnumber%
    ....
endLOOP
\endpicture

```

The screen output during the L^AT_EX processing (with respect to the command `latex mpicpm07-10.mt`) then appears as the following (see Figure 7.10).

```

This is TeX, Version 3.14159 (Web2C 7.4.5)
(./mpicpm07-10.mt
LaTeX2e <2001/06/01>
Babel <v3.7h> and hyphenation patterns for american, french, german,

```

```

basque, italian, portuges, russian, spanish, nohyphenation, loaded.
(/usr/share/texmf/tex/latex/base/article.cls
Document Class: article 2001/04/21 v1.4e Standard LaTeX document class
(/usr/share/texmf/tex/latex/base/size10.clo))
(/usr/share/texmf/tex/latex/base/mathspic.sty
Loading mathsPIC package (c) RWD Nickalls & A Syropoulos 08/08/2004
(/usr/share/texmf/tex/generic/pictex/repictex.tex)
(/usr/share/texmf/tex/generic/pictex/pictexwd.tex)
(/usr/share/texmf/tex/generic/pictex/postpictex.tex)) (./mpicpm07-10.aux)
loop = 1
loop = 2
loop = 3
loop = 4
...
...
loop = 33
loop = 34
loop = 35
[1] (./mpicpm07-10.aux) )
Output written on mpicpm07-10.dvi (1 page, 106560 bytes).
Transcript written on mpicpm07-10.log.

```

7.7 Plotting graphs

Data-files which do *not* contain mathsPIC commands can be input using the `inputfile*` command. This command inputs files *verbatim*, and so can be used for inputting files containing, for example, only P_TEX commands and/or points for plotting curves. For example, the following mathsPIC code (`mpicpm07-11.m`) draws the quartic curve shown in Figure 7.11, by inputting in *verbatim* a datafile containing some P_TEX commands and a set of data points for plotting. Note that in this example the x -axis is stretched by using `units(3cm,cm)` in the `paper{}` command.

```

%% mpicpm07-11.m (Figure 7-11)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
\linethickness=1pt %% make a thick line for the axes
paper{units(3cm,1cm),xrange(-1,2),yrange(-2,4),axes(XY),ticks(1,1)}
\linethickness=0.4pt %% reset to default value
\headingtoplotskip=8mm
\plotheadings{\fbox{\Large A quartic equation}}
%% now load file containing data points for curve
inputfile*(mpicpm07-11.dat)
pointssymbol(default,0.3) % define line-free-radius
point(E1){1,3.5}
text($f(x)=6x^4 - 8x^3 + 1$){E1} % center the equation at E1

```

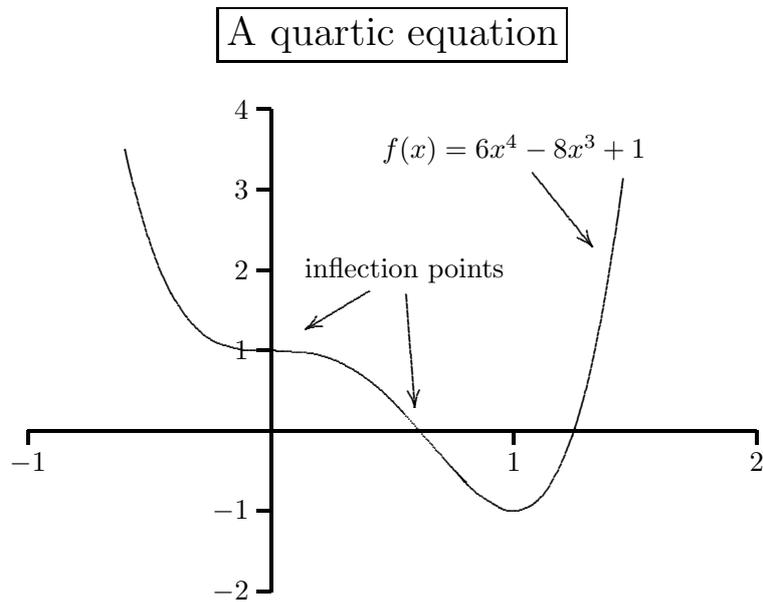


Figure 7.11:

```

point(E2){1.4,2}
drawarrow(E1E2)
point(J1){0.55,2}
text(inflection points){J1} % center inflection text at J1
point(J2){0,1}
point(J3){0.6,0}
drawarrow(J1J2,J1J3)
\endpicture
\end{document}

```

The datafile for the curve is as follows—note the leading `_` on the data-point lines to force `mathsPIC` to read these lines as data (i.e., to copy the lines unchanged into the `.mt` file) and not to process them as `mathsPIC` commands. Remember that `PTCTEX` requires an *odd* number of pairs of data points to satisfy its curve-drawing algorithm.¹ Note also that the final data-point in the sequence must be ‘terminated’ using a `_/` pair, as shown below.

```
%% mpicm07-11.dat (Figure 7.11)
%% quartic curve data (use an odd number of data points)
\setquadratic
\plot
\_ -0.6      3.50
\_ -0.5      2.37
\_ -0.4      1.66
\_ -0.35     1.43
\_ ...
\_ ...
\_ 1.15     -0.67
\_ 1.25      0.02
\_ 1.35      1.24
\_ 1.45      3.13 /
\setlinear
%%EOF
```

However, when there are only a few data points, it is sometimes more convenient just to plot the points separately and then draw connecting lines, as shown in Figure 7.12.

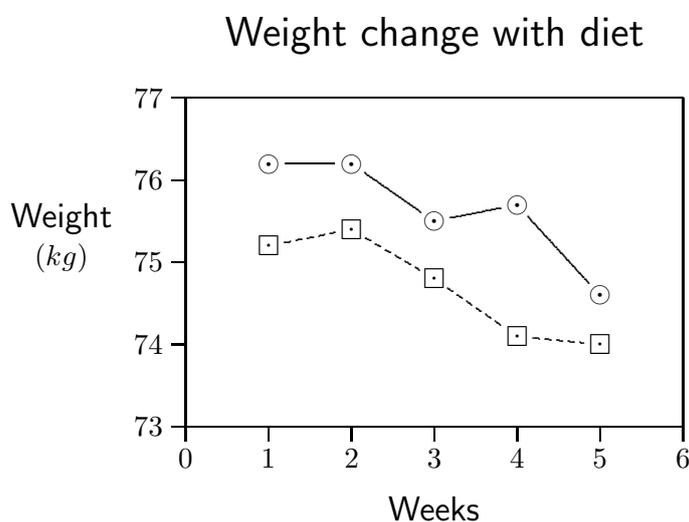


Figure 7.12:

```
%% mpicpm07-12.m (Figure 7.12)
\documentclass[a4paper]{article}
```

¹See the `PTCTEX` manual (Wichura, 1992).

```

\usepackage{mathspic}
\usepackage{amssymb}
\begin{document}
%%-----
\beginpicture
paper{units(1cm), xrange(0,6), yrange(73,77), axes(LBT*R*), ticks(1,1)}
pointssymbol( $\odot$ , 0.2)
point(d1){1, 76.2}
point(d2){2, 76.2}
point(d3){3, 75.5}
point(d4){4, 75.7}
point(d5){5, 74.6}
drawpoint(d1d2d3d4d5)
drawline(d1d2d3d4d5)
%
pointssymbol( $\boxdot$ , 0.2)
point(k1){1, 75.2}
point(k2){2, 75.4}
point(k3){3, 74.8}
point(k4){4, 74.1}
point(k5){5, 74.0}
drawpoint(k1k2k3k4k5)
\setdashpattern <2pt, 2pt>
drawline(k1k2k3k4k5)
%
\plotheadingsf{\textsf{\Large Weight change with diet}}
text{\shortstack{\textsf{\large Weight}\(\($kg$\))}{-1.5,75.3}}
text{\textsf{\large Weeks}}{3,72}
\endpicture
\end{document}

```

7.8 Drawing other curves

The `drawcurve` command can also be used for drawing smooth curves linking a number of points or touching lines. For example, Figure 7.13 shows a smooth curve touching a piecewise linear closed line², some of the points being constructed using a Bézier technique³. The smooth curve is drawn using the `drawcurve()` command for successive three-point sequences.

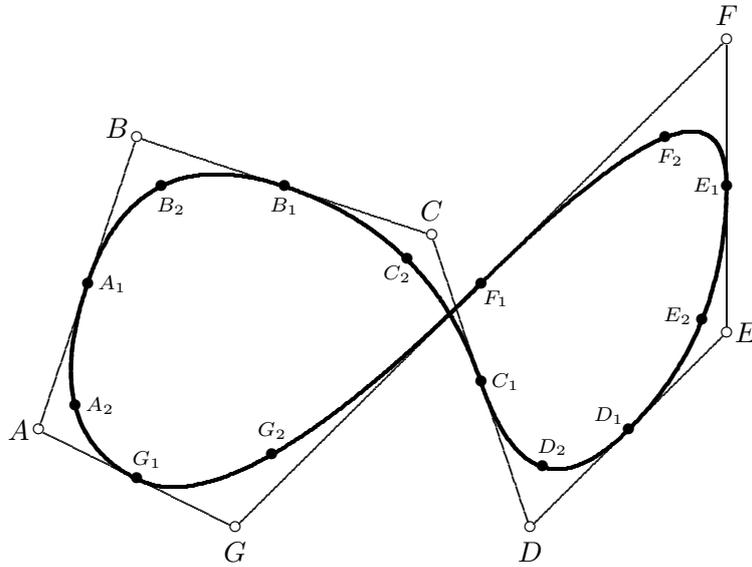
```

%% mpicpm07-13.m (Figure 7.13)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
var u = 12 %% units = 12 mm
paper{units(Umm), xrange(0.5,8.5), yRange(-0.5,5.5)}

```

²Figure 7.13 was constructed and drawn by František Chvála.

³see *The Metafont book* by DE Knuth, chapter 3 for details regarding Bézier curves.

Figure 7.13: A smooth curve inscribed in the intersecting closed line $ABCDEFGA$

```

var r = 0.7 %% line-free radius of \circ (0.7mm)
var r = r/u %% line free radius scaled for U mm
pointsymbol($\circ$,r)
Point(A){1,1}
Point(B){2,4}
Point(C){5,3}
Point(D){6,0}
Point(E){8,2}
Point(F){8,5}
Point(G){3,0}
pointsymbol(default) %% restore $\bullet$
%
Point(A1){midpoint(AB)}
Point(B1){midpoint(BC)}
Point(C1){midpoint(CD)}
Point(D1){midpoint(DE)}
Point(E1){midpoint(EF)}
Point(F1){midpoint(FG)}
Point(G1){midpoint(GA)}
%
Point(A2){midpoint(G1A1)}
Point*(A2){midpoint(AA2)}
Point(B2){midpoint(A1B1)}
Point*(B2){midpoint(BB2)}
Point(C2){midpoint(B1C1)}
Point*(C2){midpoint(CC2)}
Point(D2){midpoint(C1D1)}
Point*(D2){midpoint(DD2)}

```

```

Point(E2){midpoint(D1E1)}
Point*(E2){midpoint(EE2)}
Point(F2){midpoint(E1F1)}
Point*(F2){midpoint(FF2)}
Point(G2){midpoint(F1G1)}
Point*(G2){midpoint(GG2)}
%
DrawPoint(ABCDEFG)
DrawPoint(A1B1C1D1E1F1G1)
DrawPoint(A2B2C2D2E2F2G2)
%
DrawLine(ABCDEFGA)
\setplotsymbol ({\Large.})
DrawCurve(A1B2B1)
DrawCurve(B1C2C1)
DrawCurve(C1D2D1)
DrawCurve(D1E2E1)
DrawCurve(E1F2F1)
DrawCurve(F1G2G1)
DrawCurve(G1A2A1)
%
Text($A$){A,shift(-0.2,0)}
Text($B$){B,shift(-0.2,0.1)}
Text($C$){C,shift(0,0.25)}
Text($D$){D,shift(0,-0.25)}
Text($E$){E,shift(0.2,0)}
Text($F$){F,shift(0,0.25)}
Text($G$){G,shift(0,-0.25)}
\scriptsize
Text($A_1$){A1,shift(0.25,0)}
Text($B_1$){B1,shift(0,-0.2)}
Text($C_1$){C1,shift(0.25,0)}
Text($D_1$){D1,shift(-0.2,0.15)} %
Text($E_1$){E1,shift(-0.2,0)}
Text($F_1$){F1,shift(0.15,-0.15)}
Text($G_1$){G1,shift(0.1,0.2)}
Text($A_2$){A2,shift(0.25,0)}
Text($B_2$){B2,shift(0.1,-0.2)}
Text($C_2$){C2,shift(-0.1,-0.15)}
Text($D_2$){D2,shift(0.1,0.2)}
Text($E_2$){E2,shift(-0.25,0.05)} %
Text($F_2$){F2,shift(0.05,-0.2)} %
Text($G_2$){G2,shift(0,0.25)} %
\endpicture
\end{document}

```

7.9 Scaling

Note the technique used in the code for Figure 7.13 above, for making the *physical* line-free radius (r) invariant with respect to the scaling value (u) (i.e. does not change when the figure is enlarged or reduced by varying the value of the variable u), as follows.

```

....
var u = 12          %% units = 12 mm
paper{units(u mm),xRange(0,9),yRange(-1,6)}
var r = 0.7        %% line-free radius of \circ = 0.7mm
var r = r/u       %% scaled linefree radius for u mm
pointsymbol($\circ$, r)
point(A){1,1}
....

```

This technique can be very useful when it is necessary to scale the figure after having designed the figure in order, say, to make it fit into a particular space in a document.

7.10 Using Perl programs

Since mathsPIC allows access to the command-line via the `system()` command, users can write Perl programs which can then be used as a powerful aid for drawing complicated mathsPIC diagrams. In other words, components of a diagram (particularly those components which are frequently used) can be encoded as a small Perl program which can then be invoked via the mathsPIC `system()` command. Particularly useful is the fact that Perl programs can have parameters passed to them, which greatly increases their value and utility with regard to mathsPIC. Indeed, a mathsPIC file for drawing a particularly complicated diagram could usefully input several Perl programs, each drawing separate elements of the diagram.

In view of the interplay between the mathsPIC file and the Perl program it calls, we now describe two typical examples in some detail. In the following some familiarity with the Perl programming language is assumed.

7.10.1 Example-1

This example (see Figure 7.14) uses the small Perl program `drawcurvedarrow.pl` (see below) to generate the mathsPIC commands for drawing a curved arrow (note that there is no mathsPIC command for doing this at the moment), and place them in a temporary file which can then be accessed by the mathsPIC file.

Perl program (drawcurvedarrow.pl)

```

#!/usr/bin/perl
## drawcurvedarrow.pl
# 5 command-line parameters A B C h filename
my ($A, $B, $C, $h, $filename)=@ARGV;
open (outfile, ">$filename")|| die "ERROR can't create file $filename\n";
print (outfile "Point*(P999){pointonline($B$A, -($A$B)/3)}\n");
print (outfile "Point*(H999){pointonline($C P999,$h)}\n");
print (outfile "Drawcurve($A$B H999)\n");

```

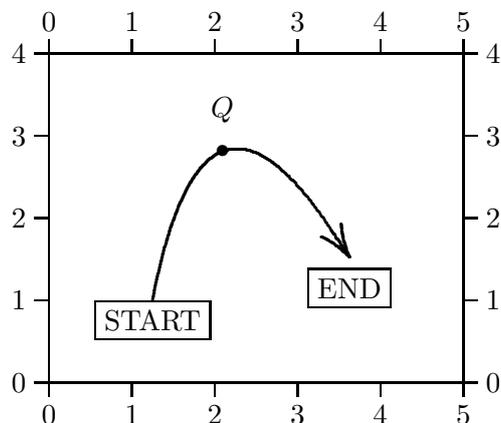


Figure 7.14: A curved arrow drawn using the Perl program `drawcurvedarrow.pl`

```
print (outfile "drawArrow(H999 $C)\n");
close (outfile);
```

The above Perl program accepts five parameters and writes the `mathsPIC` commands required for drawing a curved arrow through the three points A, B, C (from A to C). The program places an arrowhead (length h) at the end pointing at point C , and places all the necessary `mathsPIC` commands into the temporary file `filename`.

Note that in this example all new points defined in the Perl program are defined using the `point*()` command, since this allows the program to be reused without needing to know which points have been defined before. The point names `P999` and `H999` have been chosen in order to try and avoid clashing with any point names likely to be used in the `mathsPIC` file. Indeed, the authors suggest that point numbers ≥ 900 be reserved for use in Perl programs in this way. Note also that these particular point names (e.g. `H999`) need to be separated by a space from a previous string name (e.g. `AB H999`) since they are not Perl variable names (Perl variable names are prefixed with a `$` symbol).

`System()` command

`MathsPIC` ‘calls’ a Perl program (with appropriate parameters) via the `system()` command. For example, the following `mathsPIC` commands (a) call the Perl program `drawcurvedarrow.pl` to draw the curved arrow PQR with an arrowhead length 0.4 units, and places the commands in the file `temp.txt`, and (b) inputs the temporary file which then contains the commands.

```
....
system("perl drawcurvedarrow.pl P Q R 0.4 temp.txt")
inputfile(temp.txt)
....
```

It is useful to include the filename as one of the parameters, since this allows us to write the `mathsPIC` command to input the same file immediately afterwards. Note also that since we are, in effect, using the command-line here then the parameters following the program name must be separated by spaces, as required by Perl syntax.

The mathsPIC file

The following example mathsPIC file uses the above commands to draw a curved arrow from a START box (just below the first point P) to an END box (below the last point R) as shown in Figure 7.14. Note the use of the [t] option at the end of two of the `text()` commands which places the point at the [t]op of the `\fbox{}`.

```
%% mpicpm07-14.m (Figure 7.14)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
paper{units(1cm) xrange(0,5) yrange(0,4), axis(LRTB), ticks(1,1)}
point(P){1.25,1}
point(Q){P, polar(2, 65 deg)}
point(R){Q, polar(2,-40 deg)}
drawpoint(Q)
text(\fbox{START}){P}[t]
text($Q$){Q, shift(0,0.5)}
text(\raisebox{-5mm}{\fbox{END}}){R}[t]
linethickness(1pt)
arrowshape(0.4cm, 30,40)
system("perl drawcurvedarrow.pl P Q R 0.4 temp.txt")
inputfile(temp.txt)
\endpicture
\end{document}
```

The temporary file

When the above mathsPIC file is run it creates (in the same directory) the following four-line temporary ASCII text-file (named `temp.txt`) containing the mathsPIC commands which actually draw the curve, and then inputs the file `temp.txt` into the mathsPIC file for processing.

```
Point*(P999){pointonline(QP, -(PQ)/3)}
Point*(H999){pointonline(R P999,0.4)}
Drawcurve(PQ H999)
drawArrow(H999 R)
```

Output file

The output \TeX file which is generated by running the file `mpicpm07-14.m` through mathsPIC is as follows. Towards the end of the file you can see the temporary file code and the results after processing by mathsPIC located between the following two lines

```
%% ... start of file <temp.txt> loop [1]
...
%% ... end of file <temp.txt> loop [1]
```

The full output file is as follows.

```

%* -----
%* mathsPIC (Perl version 0.99.22 Dec 29, 2004)
%* A filter program for use with PiCTeX
%* Copyright (c) 2004 A Syropoulos & RWD Nickalls
%* Command line: mpic09922.pl mpicpm07-14.m
%* Input filename : mpicpm07-14.m
%* Output filename: mpicpm07-14.mt
%* Date & time: 2005/01/05    09:30:55
%* -----
%% mpicpm07-14.m (Figure 7.14)
%% curved arrow
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%%-----
\beginpicture
%% paper{units(1cm) xrange(0,5) yrange(0,4), axis(LRTB),ticks(1,1)}
\setcoordinatesystem units <1cm,1cm>
\setplotarea x from 0.00000 to 5.00000, y from 0.00000 to 4.00000
\axis left ticks numbered from 0 to 4 by 1 /
\axis right ticks numbered from 0 to 4 by 1 /
\axis top ticks numbered from 0 to 5 by 1 /
\axis bottom ticks numbered from 0 to 5 by 1 /
%% point(P){1.25,1}          P = (1.25000, 1.00000)
%% point(Q){P, polar(2, 65 deg)}      Q = (2.09524, 2.81262)
%% point(R){Q, polar(2,-40 deg)}      R = (3.62733, 1.52704)
%% drawpoint(Q)
\put {$\bullet$} at 2.09524 2.81262 %% Q
%% text(\fbox{START}){P}[t]
\put {\fbox{START}} [t] at 1.250000 1.000000
%% text($Q$){Q, shift(0,0.5)}
\put {$Q$} at 2.095240 3.312620
%% text(\raisebox{-5mm}{\fbox{END}}){R}[t]
\put {\raisebox{-5mm}{\fbox{END}}} [t] at 3.627330 1.527040
%% linethickness(1pt)
\linethickness=1.00000pt\Linethickness{1.00000pt}%
\font\CM=cmr10 at 9.97226pt%
\setplotsymbol ({\CM .})%
%% arrowshape(0.4cm, 30,40)
%% arrowLength = 0.4cm, arrowAngleB = 30 and arrowAngleC = 40
%% system("perl drawcurvedarrow.pl P Q R 0.4 temp.txt")
%% inputfile(temp.txt)
%% ... start of file <temp.txt> loop [1]
%%% Iteration number: 1
%% point*(P999){pointonline(QP, -(PQ)/3)}          P999 = (2.37699, 3.41683)
%% point*(H999){pointonline(R P999,0.4)}          H999 = (3.40661, 1.86063)
%% Drawcurve(PQ H999)
\setquadratic
\plot
1.25000 1.00000 %P

```

```

2.09524 2.81262 %Q
3.40661 1.86063 / %H999
\setlinear
%% drawArrow(H999 R)
\arrow <0.40000cm> [0.2679,0.7279] from 3.4066 1.8606 to 3.6273 1.5270
%% ... end of file <temp.txt> loop [1]
\endpicture
\end{document}

```

L^AT_EXing this file then generates the DVI file for Figure 7.14.

7.10.2 Example-2

Once the main working part of a Perl program has been debugged, then some finishing touches can be added. In the following example (`drawcube.pl`; to draw a cube) we have, therefore, made it a somewhat more sophisticated program by including (a) error messages, (b) made it delete the temporary file automatically, (c) useful macros, (d) added colour.

Note also how the one-line `system()` command in the `mathsPIC` file calling the `drawcube.pl` program is processed into the temporary file necessary for drawing the cube. This illustrates the value of being able to encode certain elements of a diagram as a separate Perl program capable of receiving parameters, and so making it possible for them to be used as and when necessary.

Delete temporary file

In Linux we can automatically delete the temporary file by writing the `mathsPIC` command

```
system("rm temp.txt")
```

(where `rm` is the Linux command for ‘remove’).

Now, this can be done by the Perl program itself (providing the temporary file is not to be input more than once), in which case it would have to output the above command. However, in order for it to do this (i.e., for Perl to write itself a Perl command), the Perl command would have to be written as follows:-

```
print (outfile qq(system("rm $filename")\n));
```

where the `qq()` command exports the argument in inverted commas, as required.

Useful macros

In addition, we have made the `mathsPIC` file more readable by creating the macros `¢er()`, `&side()` and `&filename()` to hold the three parameters, namely center point, the sidelength and the temporary filename. In this way we can pass P , s and `temp.txt` as `¢er(P)`, `&side(s)` and `&filename(temp.txt)` (note that when `mathsPIC` macros are used then they need to be prefixed with the `&` symbol).

```

%def center(j)j%      %%center point
%def side(j)j%       %%sidelength
%def filename(j)j%   %% temporary file name

```

```
point(P){5,5}[symbol=${\bigodot$}]
var s=4 %% sidelength
system("perl drawcube.pl &center(P) &side(s) &filename(temp.txt)")
inputfile(temp.txt)
```

Added colour

We have also made use of the L^AT_EX Color package and coloured the sides of the cube blue, the diagonals red, and the points and labels black. Always load the color package *after* the mathsPIC package.

MathsPIC program

All these improvements are implemented in the following example mathsPIC script & Perl program `drawcube.pl` which draws a simple cube (side s) about a central point P (see Figure 7.15).

```
%% mpicpm07-15.m (Figure 7.15)
\documentclass[a4paper]{article}
\usepackage{mathspic,color}
\begin{document}
%%-----
\beginpicture
\normalcolor%
paper{units(1cm) xrange(0,6) yrange(0,6), axis(LB), ticks(1,1)}
point(P){3,3}[symbol=${\bigodot$}] %center of the cube
%def center(j)j% %% center point
%def side(j)j% %% sidelength
%def filename(j)j% %% temp filename
var s=3 %% sidelength
system("perl drawcube.pl &center(P) &side(s) &filename(temp.txt)")
inputfile(temp.txt)
\normalcolor%
\endpicture
\end{document}
```

Note that the point-name P , side-length s , and filename `temp.txt` are all passed to the Perl program as parameters using the `system()` command. The code of the Perl program `drawcube.pl` is listed below.

Perl program (drawcube.pl)

```
#!/usr/bin/perl
## drawcube.pl
# pickup command line parameters P,s,filename
# use: system("perl drawcube.pl P s3 temp.txt")
#=====
my ($argnumber) = $#ARGV + 1;
if ($argnumber != 3){
    print "=====\n";
    print "ERROR: drawcube.pl requires 3 arguments\n";
}
```

```

    print "USE:  drawcube.pl <pointname> <sidelength> <filename> \n";
    print "=====\n";
    exit(1);
}
my ($point, $side, $filename)=@ARGV;
open (outfile, ">$filename")|| die "ERROR can't create file $filename\n";
print (outfile <<EOF);
\%\\%-----mathsPIC code-----
var r=$side*sqrt(2)/2
var a=30 %% angle degrees
Point*(P990){$point, polar(r/2, (a-180) deg)}
Point*(P991){P990, polar(r, 45 deg)}
Point*(P992){P991, rotate(P990, 90)}
Point*(P993){P991, rotate(P990, 180)}
Point*(P994){P991, rotate(P990, 270)}
Point*(P995){P991, polar(r, a deg)}
Point*(P996){P992, polar(r, a deg)}
Point*(P997){P993, polar(r, a deg)}
Point*(P998){P994, polar(r, a deg)}
%% draw the sides
\\color{blue}
drawline(P991 P992 P993 P994 P991, P994 P998 P995 P996 P992, P991 P995)
\\setdashes
drawline(P993 P997 P998, P997 P996)
%% draw the diagonals
\\color{red}
drawline(P991 P997, P992 P998, P996 P994, P995 P993)
%% draw the points and labels
\\setsolid\\color{black}
drawpoint($point P991 P992 P993 P994 P995 P996 P997 P998)
text(\\$$point\\$){$point, shift(-0.5,-0.1)}
text(\\$P991\\$){P991, shift(-0.3,0.3)}
text(\\$P995\\$){P995, shift(-0.3,0.3)}
%%-----end of mathspic code-----
EOF
# now delete the temp file
print (outfile qq(system("rm $filename")\n));
close outfile
__END__

```

Note the use of the <<EOF... EOF environment in the Perl programme to contain the chunk of mathsPIC code which is written to text file temp.txt. Note also that where you want \$ and \ characters written to the output file (temporary file—for use by mathsPIC) by the Perl program, it is important to remember that these characters need to be ‘escaped’ using a preceding backslash. In the above example the point name *P* is held in the Perl variable \$point; consequently since we need the Perl program to write the mathsPIC command text(\$P\$){P,shift(-0.5, -0.1)} to the temporary file the Perl code needs to be text(\\\$\$point\\\$){\$point, shift(-0.5,-0.1)}

The temporary file

```

%%-----mathsPIC code-----
var r=s*sqrt(2)/2
var a=30 %% angle degrees
Point*(P990){P, polar(r/2, (a-180) deg)}
Point*(P991){P990, polar(r, 45 deg)}
Point*(P992){P991, rotate(P990, 90)}
Point*(P993){P991, rotate(P990, 180)}
Point*(P994){P991, rotate(P990, 270)}
Point*(P995){P991, polar(r, a deg)}
Point*(P996){P992, polar(r, a deg)}
Point*(P997){P993, polar(r, a deg)}
Point*(P998){P994, polar(r, a deg)}
%% draw the sides
\color{blue}
drawline(P991 P992 P993 P994 P991, P994 P998 P995 P996 P992, P991 P995)
\setdashes
drawline(P993 P997 P998, P997 P996)
%% draw the diagonals
\color{red}
drawline(P991 P997, P992 P998, P996 P994, P995 P993)
%% draw the points and labels
\setsolid\color{black}
drawpoint(P P991 P992 P993 P994 P995 P996 P997 P998)
text({P, shift(-0.5,-0.1)})
text({P991, shift(-0.3,0.3)})
text({P995, shift(-0.3,0.3)})
system("rm $filename")
%%-----

```

Output file

When the mathsPIC script is run the output file is as follows.

```

%* -----
%* mathsPIC (Perl version 0.99.22 Dec 29, 2004)
%* A filter program for use with PiCTeX
%* Copyright (c) 2004 A Syropoulos & RWD Nickalls
%* Command line: mpic09922.pl mpicpm07-15.m
%* Input filename : mpicpm07-15.m
%* Output filename: mpicpm07-15.mt
%* Date & time: 2005/01/05 09:46:22
%* -----
%% mpicpm07-15.m (Figure 7.15)
\documentclass[a4paper]{article}
\usepackage{mathspic,color}
\begin{document}
%%-----
\beginpicture
%% paper{units(1cm) xrange(0,6) yrange(0,6), axis(LB),ticks(1,1)}

```

```

\setcoordinatesystem units <1cm,1cm>
\setplotarea x from 0.00000 to 6.00000, y from 0.00000 to 6.00000
\axis left ticks numbered from 0 to 6 by 1 /
\axis bottom ticks numbered from 0 to 6 by 1 /
%% point(P){3,3}[symbol=${\bigodot}] %center P = (3.00000, 3.00000)
%%def center(j)j% %% center point
%%def filename(j)j% %% temp filename
%%def side(j)j% %% sidelength
%% var s=3 %% sidelength
%% s = 3
%% system("perl drawcube.pl P s temp.txt")
%% ... start of file <temp.txt> loop [1]
%% Iteration number: 1
%%-----mathsPIC code-----
%% var r=s*sqrt(2)/2
%% r = 2.12132034355964
%% var a=30 %% angle degrees
%% a = 30
%% point*(P990){P, polar(r/2, (a-180) deg)} P990 = (2.08144, 2.46967)
%% point*(P991){P990, polar(r, 45 deg)} P991 = (3.58144, 3.96967)
%% point*(P992){P991, rotate(P990, 90)} P992 = (0.58144, 3.96967)
%% point*(P993){P991, rotate(P990, 180)} P993 = (0.58144, 0.96967)
%% point*(P994){P991, rotate(P990, 270)} P994 = (3.58144, 0.96967)
%% point*(P995){P991, polar(r, a deg)} P995 = (5.41856, 5.03033)
%% point*(P996){P992, polar(r, a deg)} P996 = (2.41856, 5.03033)
%% point*(P997){P993, polar(r, a deg)} P997 = (2.41856, 2.03033)
%% point*(P998){P994, polar(r, a deg)} P998 = (5.41856, 2.03033)
%% draw the sides
\color{blue}
%% drawline(P991 P992 P993 P994 P991, P994 P998 P995 P996 P992, P991 P995)
\putrule from 3.58144 3.96967 to 0.58144 3.96967 %% P991P992
\putrule from 0.58144 3.96967 to 0.58144 0.96967 %% P992P993
\putrule from 0.58144 0.96967 to 3.58144 0.96967 %% P993P994
\putrule from 3.58144 0.96967 to 3.58144 3.96967 %% P994P991
\plot 3.58144 0.96967 5.41856 2.03033 / %% P994P998
\putrule from 5.41856 2.03033 to 5.41856 5.03033 %% P998P995
\putrule from 5.41856 5.03033 to 2.41856 5.03033 %% P995P996
\plot 2.41856 5.03033 0.58144 3.96967 / %% P996P992
\plot 3.58144 3.96967 5.41856 5.03033 / %% P991P995
\setdashes
%% drawline(P993 P997 P998, P997 P996)
\plot 0.58144 0.96967 2.41856 2.03033 / %% P993P997
\putrule from 2.41856 2.03033 to 5.41856 2.03033 %% P997P998
\putrule from 2.41856 2.03033 to 2.41856 5.03033 %% P997P996
%% draw the diagonals
\color{red}
%% drawline(P991 P997, P992 P998, P996 P994, P995 P993)
\plot 3.58144 3.96967 2.41856 2.03033 / %% P991P997
\plot 0.58144 3.96967 5.41856 2.03033 / %% P992P998
\plot 2.41856 5.03033 3.58144 0.96967 / %% P996P994

```

```

\plot 5.41856 5.03033      0.58144 0.96967 / %% P995P993
%% draw the points and labels
\setsolid\color{black}
%% drawpoint(P P991 P992 P993 P994 P995 P996 P997 P998)
\put {$\bigodot$} at 3.00000 3.00000 %% P
\put {$\bullet$} at 3.58144 3.96967 %% P991
\put {$\bullet$} at 0.58144 3.96967 %% P992
\put {$\bullet$} at 0.58144 0.96967 %% P993
\put {$\bullet$} at 3.58144 0.96967 %% P994
\put {$\bullet$} at 5.41856 5.03033 %% P995
\put {$\bullet$} at 2.41856 5.03033 %% P996
\put {$\bullet$} at 2.41856 2.03033 %% P997
\put {$\bullet$} at 5.41856 2.03033 %% P998
%% text($P$){P, shift(-0.4,-0.1)}
\put {$P$} at 2.600000 2.900000
%% text($P991$){P991, shift(-0.3,0.3)}
\put {$P991$} at 3.281440 4.269670
%% text($P995$){P995, shift(-0.3,0.3)}
\put {$P995$} at 5.118560 5.330330
%%-----
%% system("rm temp.txt")
%% ... end of file <temp.txt> loop [1]
\endpicture
\end{document}

```

and the diagram is shown in Figure 7.15.

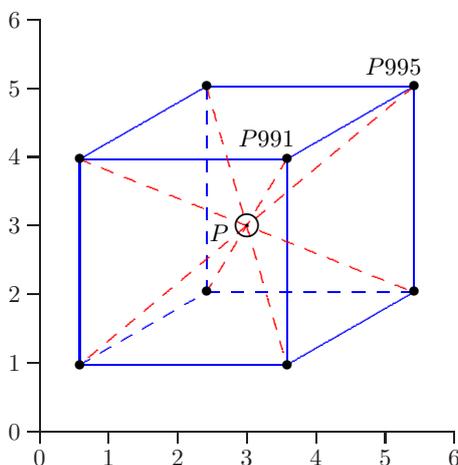


Figure 7.15:

3-D cube drawn about a central point $P(3,3)$ and with sidelength s ($s = 3$). The PDF and PostScript versions of the output show the sides coloured blue, the diagonals red, and the points and labels black. The mathsPIC command used was:-

```
system("perl drawcube.pl &center(P) &side(s) &filename(temp.txt)").
```

7.10.3 Commands for processing the files

In Linux the command-line commands used to generate and print figure 7.15 were as follows:

```
perl mathspic.pl mpicpm07-15.m
latex mpicpm07-15.mt
xdvi mpicpm07-15.dvi    %% to view dvi file
dvips -o mpicpm07-15.ps mpicpm07-15.dvi %% generate the .ps version
gv mpicpm07-15.ps      %% view the .ps version
lpr mpicpm07-15.ps     %% print the .ps version
```

Chapter 8

Accessing T_EX parameter values¹

It is sometimes useful for mathsPIC to be able to access T_EX parameter values. For example, mathsPIC could be made to automatically scale the height and width of a graph to the as yet unknown page size of the T_EX document. Knowledge of the T_EX parameters `\textheight` and `\textwidth` during processing would allow the height and width of a graph to be defined as a function of these page parameters.

One way of doing this is to include in the T_EX file some code to write the required parameter values to a temporary data file, in the form of mathsPIC `var` commands or macros. Using the mathsPIC `system()` command mathsPIC can run the T_EX file, and then access the parameter values by inputting the data-file.

8.1 Useful T_EX commands

```
\settowidth{\variablename}{text}  
\settoheight{\variablename}{text}  
\settodepth{\variablename}{text}
```

For example, the following code would put the length of the word January into the variable `\janlength`. Note that `\the` gives the value in points, while `\number` returns the value as an integer in scaled points (see Table 8.1).

```
\newlength{\janlength}  
\settowidth{\janlength}{January}  
The length of the word January in points is \the\janlength  
The length of the word January in 'scaled' points is \number\janlength
```

The output is as follows (note that `\the\janlength` returns the numeric points but includes the characters `pt` on the end).

```
The length of the word January in points is 35.16676pt  
The length of the word January in 'scaled' points is 2304689
```

¹Much of this chapter was originally presented by RWD Nickalls to the November 2001 ukTUG meeting.

T_EX has a number of parameters which are classified according to type, e.g. integers, dimensions, glue, muglue, token lists. For a full list of all the T_EX parameters which have values which can be accessed see Knuth (1990; page 272–275).

The token list `\jobname` is very useful as it holds the filename (but not the filename extension) which T_EX is currently working on, and can be used for generating derivative temporary files, e.g. `\jobname.dvi`, `\jobname.toc` etc.

8.2 Outputting data to a file

T_EX requires a file-number as a handle to identify an ‘open’ file, and has the command `\newfile` specifically for allocating an unused file-number to a variable name. We then use the commands `\openout`, `\write`, and `\closeout` to open, write to, and close the file. For example, the following code will write `...blah blah blah ..` to the file `texfiledata.dat` using the file handle `\outfile`.

```
\newwrite\outfile
\openout\outfile=texfiledata.dat
\write\outfile{...blah blah blah...}
\closeout\outfile
```

Note that the `\write` command only writes the data to the file when the `.dvi` file is created, so if these commands are in a file which may not actually create a `.dvi` file, then we may need to force output (`dvi` file creation) by including something writable like `\strut` on a line.

Alternatively, we can force T_EX to write to the file immediately (i.e. without waiting until it gets to the end of file processing) by using the `\immediate` command, remembering to include it with *all* the commands `\openout`, `\write`, and `\closeout`, as follows.

```
\newwrite\outfile
\immediate\openout\outfile=texfiledata.dat
\immediate\write\outfile{...blah blah blah...}
\immediate\closeout\outfile
```

Since we are interested in accessing the values of T_EX parameters, we need to explore some of the T_EX commands for accessing such values. For example, the commands `\the`, `\number` and `\showthe` all reveal the numeric value, but in slightly different formats and location, as follows.

```
\the\textwidth      ---> 400.0pt          includes the characters pt
\number\textwidth   ---> 26214400       scaled points (see Table)
\showthe\textwidth ---> only writes to the log file
```

For the purposes of mathsPIC accessing the numeric value as a variable or macro, it is most convenient to use the `\number` command (yields an integer value in the ‘scaled points’ used internally by T_EX)² and to incorporate it into a mathsPIC variable or macro so it is ready to be used once the temporary file it is written to has been input by mathsPIC. For example, the following code allocates the scaled point value of `\textwidth` to the mathsPIC variable `w555`.

²65536 scaled points = 1 printers point (pt).

```
\immediate\write\outfile{var w555 = \number\textwidth}
```

If the `\textwidth` was 400pt (14.058 cm), then the output of the above line would be as follows (65536 scaled points = 1 printers point pt).

```
var w555 = 26214400
```

In practice, it is useful to have this line commented to indicate which T_EX parameter the value relates to, as follows.

```
var w555 = 26214400% \textwidth (scaled points)
```

However, since what came after the % symbol would be ignored in this setting, we need to define the % as a character we can print to a file, by allocating it the catcode 12 (instead of the catcode of 14 which it normally has), and calling it by a different name (`\percentchar`), and delimiting the whole line by a curly brace (to keep it local), as follows.⁴

```
{\catcode'\%=12 \global\def\percentchar{}}%
```

So now, the following code will also include a trailing comment indicating the T_EX name of the parameter.

```
\newcommand{\comment}{\percentchar\space}
\immediate\write\outfile{var w555 = \number\textwidth
\comment\textwidth is \the\textwidth}
```

For example, the resulting line in the output file is as follows.

```
var w555 = 29368707% \textwidth is 448.1309pt
```

Table 8.1: Note that for accurate working always use scaled points (sp) and convert in a single step using the following conversion factors (modified from: Beccari C, 1991)

	sp
mm	186467.98
cm	1864679.8
pt	65536
in	4736286.70

⁴From: Abrahams, Berry and Hargreaves (1990), p 292.

8.3 The final code

So now we can put all this code together into one chunk within the target T_EX file, or more usefully, keep it as a separate file which can then just be `\input` whenever it is required. For example inputting the following file (and L^AT_EXing it) will output a data file containing appropriate mathsPIC commands with the embedded values of `\textwidth` and `\textheight`. Note that in the following example we have defined two mathsPIC macros called `textwidthcms` and `textheightcms` which will contain the relevant values in cms (see Table 8.1 for conversion factors).

```
%% grabtexdata.tex
%-----
\scrollmode % prevent LaTeX stopping if there are errors
%-----
% make a print command macro
\newcommand{\print}[1]{\immediate\write\outfile{#1}}
%-----
% make a comment % command macro
% first need to define percentchar for the write statement
% (From "TeX for the Impatient" (1990), p 292)
{\catcode'\%=12 \global\def\percentchar{}}%
\newcommand{\comment}{\percentchar\space}
%
% make a \macro command --> %def<space>
\newcommand{\mydef}{def}
\newcommand{\macro}{\percentchar\mydef\space}
%-----
% create and open a new file with filename = textfiledata.dat
\newwrite\outfile
\immediate\openout\outfile=teXfiledata.dat
%-----
%% write file header & general info
\print{\percentchar\percentchar\space file: teXfiledata.dat}
\print{\percentchar\percentchar\space accessing TeX parameter values}
%-----
%% now get \textwidth and \textheight values from the tex file
\print{var w555 = \number\textwidth\comment\textwidth=scaled points}
\print{var w556 = \number\textwidth\comment\textwidth=\the\textwidth}
\print{var w557 = \number\textwidth/1864679.8\comment (\textwidth in cms)}
\print{\comment =====}
\print{\macro textwidthcms()\number\textwidth/1864679.8\comment}
\print{\macro textheightcms()\number\textheight/1864679.8\comment}
\print{\comment =====}
%-----
% close the file
\immediate\closeout\outfile
```

In practice one would simply include the following line

```
\input{grabtexdata.tex}
```

in the T_EX file we want data from (say, `myfile.tex`), and then L^AT_EX the file to generate the output data file. Alternatively we could L^AT_EX the file from within mathsPIC using the system command, and then input the resulting data file as follows.

```
system('latex2e myfile.tex')
input(texfiledata.dat)
```

Either way, the resulting output data file `texfiledata.dat` for a file having a standard `{article}` format is as follows

```
%% texfiledata.dat
%% accessing TeX parameter values
var w555 = 22609920% \textwidth =scaled points
var w556 = 22609920% \textwidth =345.0pt
var w557 = 22609920/1864679.8% (\textwidth in cms)
% =====
%def textwidthcms()22609920/1864679.8%
%def texheightcms()39190528/1864679.8%
% =====
```

Once the above file (`texfiledata.dat`) is input into a mathsPIC file we can then use mathsPIC commands to manipulate the `textwidth` and `texheight` values, as shown in the following example mathsPIC file.

```
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
.....
system('latex2e myfile.tex')
inputfile(texfiledata.dat)
var w=&textwidthcms, w2= &textwidthcms/2
var h=&texheightcms, h2=&texheightcms/2
...
\end{document}
```

Processing the mathsPIC file gives the following output. Notice how useful it is to have the accompanying comments.

```
%% inputfile(texfiledata.dat)
%% ... start of file <texfiledata.dat> loop [1]
%% Iteration number: 1
%% texfiledata.dat
%% accessing TeX parameter values
%% var w555 = 22609920% \textwidth =scaled points
%% w555 = 22609920
%% var w556 = 22609920% \textwidth =345.0pt
%% w556 = 22609920
%% var w557 = 22609920/1864679.8% (\textwidth in cms)
%% w557 = 12.1253632929364
% =====
%def textwidthcms()22609920/1864679.8%
%def texheightcms()39190528/1864679.8%
```

```
% =====  
%% ... end of file <texfiledata.dat> loop [1]  
%% var w=22609920/1864679.8, w2= 22609920/1864679.8/2  
%% w = 12.1253632929364  
%% w2 = 6.0626816464682  
%% var h=39190528/1864679.8, h2=39190528/1864679.8/2  
%% h = 21.0172963744231  
%% h2 = 10.5086481872116
```

Chapter 9

Miscellaneous

9.1 Acknowledgements

The authors are grateful to Mikael Möller for extensive testing of `mathsPICPerl` with the ActiveState Win32 implementation of Perl (known as ActivePerl). We also thank Joachim Schneider for highlighting a significant system bug (fixed February 2007).

We are also grateful to a large number of people for testing the earlier MS-DOS version of the program, and for their many helpful ideas and suggestions, including: František Chvála, Bob Schumacher, Orlando Rodriguez, Glen Ritchie, Boris Kuselj, Raj Chandra, Ju-myong Kim, Munpyung O, Rex Shudde, Tobias Wahl.

9.2 Feedback

The authors welcome feedback regarding bugs and platform specific issues, as well as comments, ideas and collaboration with a view to improving this program.

9.3 Development history

Perl version

- version 1.13 (April 26, 2010): bug-fix release
— fixed right-angle macro bug & typos
- version 1.11 (June 2009): bug-fix release
- version 1.10 (February 2007): bug-fix release
- version 1.00 (February 2005): First Perl-version release

MS-DOS version—**not supported**

- d2.1 (November, 2000): Second MS-DOS-version release
- d1.7u (September 1999): First MS-DOS-version release.

Appendix A

Tables

Table A.1: Equivalent P_TCT_EX commands for a 10-point font

rules (horizontal/vertical)	all other lines
<code>\linethickness=1.35pt</code>	<code>\setplotsymbol({\Large .})</code>
<code>\linethickness=1.1pt</code>	<code>\setplotsymbol({\large .})</code>
<code>\linethickness=0.9pt</code>	<code>\setplotsymbol({\normalsize .})</code>
<code>\linethickness=0.4pt</code>	<code>\setplotsymbol({\tiny .})</code>

Table A.2: Useful point-symbols and their radii for 10–12pt fonts.

symbol		radius mm	symbol package
		10pt / 11pt / 12pt	
<code>\circ</code>	○	0.70 / 0.75 / 0.80	standard
<code>\odot</code>	⊙	1.20 / 1.35 / 1.50	standard
<code>\oplus</code>	⊕	1.20 / 1.35 / 1.50	standard
<code>\ominus</code>	⊖	1.20 / 1.35 / 1.50	standard
<code>\oslash</code>	⊘	1.20 / 1.35 / 1.50	standard
<code>\otimes</code>	⊗	1.20 / 1.35 / 1.50	standard
<code>\bigcirc</code>	◯	1.70 / 1.85 / 2.05	standard
<code>\bigodot</code>	⊙	1.70 / 1.85 / 2.05	standard
<code>\bigoplus</code>	⊕	1.70 / 1.85 / 2.05	standard
<code>\bigotimes</code>	⊗	1.70 / 1.85 / 2.05	standard
<code>\star</code>	*	—	standard
<code>\triangle</code>	△	—	standard
<code>\square</code>	□	—	amssymb.sty
<code>\blacksquare</code>	■	—	amssymb.sty
<code>\lozenge</code>	◇	—	amssymb.sty
<code>\blacklozenge</code>	◆	—	amssymb.sty
<code>\bigstar</code>	★	—	amssymb.sty
<code>\boxdot</code>	◻	—	amssymb.sty
<code>\boxtimes</code>	⊠	—	amssymb.sty
<code>\boxminus</code>	⊞	—	amssymb.sty
<code>\boxplus</code>	⊞	—	amssymb.sty
<code>\divideontimes</code>	⋈	—	amssymb.sty

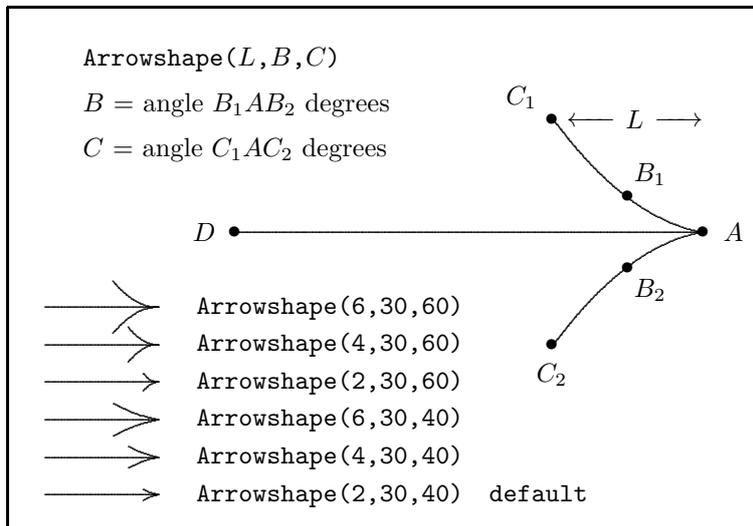
Table A.3: Conversion factors for the units used by L^AT_EX. (From Beccari (1991) with permission)

	mm	cm	pt	bp	pc	in	dd	cc	sp
1 mm	1,000	0,100	2,845	2,835	0,2371	0,03937	2,659	0,2216	186 467,98
1 cm	10,00	1,000	28,45	28,35	2,371	0,3937	26,59	2,216	1 864 679,8
1 pt	0,3515	0,03515	1,000	0,9963	0,08333	0,01384	0,9346	0,07788	65 536
1 bp	0,3528	0,03528	1,004	1,000	0,08365	0,01389	0,9381	0,07817	65 781,76
1 pc	4,218	0,4218	12,00	11,96	1,000	0,1660	11,21	0,9346	786 432
1 in	25,40	2,540	72,27	72,00	6,023	1,000	67,54	5,628	4 736 286,7
1 dd	0,3760	0,03760	1,070	1,066	0,08917	0,01481	1,000	0,08333	70 124,086
1 cc	4,513	0,4513	12,84	12,79	1,070	0,1777	12,00	1,000	841 489,04

Appendix B

Arrows

The mathsPIC code for drawing the following figure is given below.



```

%% mpicpm07-3.m (Figure 7.3)
\documentclass[a4paper]{article}
\usepackage{mathspic}
\begin{document}
%-----
\beginpicture
paper{units(1mm), xrange(0,100), yrange(0,70), axes(LBTR)}
%% first do all the small arrows (from top down)
point(J11){5,30}
  point(J12){20,30}
point(J9){5,25}
  point(J10){20,25}
point(J7){5,20}
  point(J8){20,20}

```

```

point(J5){5,15}
  point(J6){20,15}
point(J3){5,10}
  point(J4){20,10}
point(J1){5,5}
  point(J2){20,5}
%%
arrowshape(6,30,60)
  drawArrow(J11J12)
arrowshape(4,30,60)
  drawArrow(J9J10)
arrowshape(2,30,60)
  drawArrow(J7J8)
arrowshape(6,30,40)
  drawArrow(J5J6)
arrowshape(4,30,40)
  drawArrow(J3J4)
arrowshape(2,30,40)
  drawArrow(J1J2)
%%
text(\texttt{Arrowshape(2,30,40) \ default}){J2,shift(5,0)}[1]
text(\texttt{Arrowshape(4,30,40)}){J4,shift(5,0)}[1]
text(\texttt{Arrowshape(6,30,40)}){J6,shift(5,0)}[1]
text(\texttt{Arrowshape(2,30,60)}){J8,shift(5,0)}[1]
text(\texttt{Arrowshape(4,30,60)}){J10,shift(5,0)}[1]
text(\texttt{Arrowshape(6,30,60)}){J12,shift(5,0)}[1]
%%
%% now draw big arrow
point(D){30,40}
point(A){D,shift(62,0)}
arrowshape(20,50,75)
drawArrow(DA)
point(B1){82,44.75}
point(B2){82,35.25}
point(C1){72,55}
point(C2){72,25}
drawpoint(DAB1B2C1C2)
text($A$){A, shift(4,0)}
text($B_1$){B1, shift(3,3)}
text($B_2$){B2, shift(3,-3)}
text($C_1$){C1, shift(-4,3)}
text($C_2$){C2, shift(0,-4)}
text($D$){D, shift(-4,0)}
%%
point(T1){10,63}
point(T2){10,57}
point(T3){10,51}
text(\texttt{Arrowshape($L$, $B$, $C$)}){T1}[1]
text($B$ = \mbox{angle $B_1AB_2$ degrees}){T2}[1]
text($C$ = \mbox{angle $C_1AC_2$ degrees}){T3}[1]

```

```
%%  
\betweenarrows {$L$} from 74 55 to 92 55  
\endpicture  
\end{document}
```

Appendix C

Positioning the figures

C.1 Generating PS, EPS, PDF, JPEG files.

Once a diagram or figure has been finished it can be easily placed in a document using either an EPS (DVI document) or PDF (L^AT_EX document). Being able to generate a JPEG version is also useful since, unlike EPS or PDF files, this can be readily manipulated and converted to other image formats using the GIMP.¹

PS, EPS, PDF

If only a PDF version is required, then this is easily done via PDFL^AT_EX. For example, if the mathsPIC file is, say, `picture.m`, then the commands would be as follows:

```
mathspic picture.m
pdflatex picture.mt
```

However, the process of trimming excess white space can be most conveniently done via `dvips`. The process of editing and viewing can be conveniently automated using a BASH script, such as the following, which runs mathsPIC, trims the image to the bounding box (BBox) and generates the associated PS, EPS and PDF files.

```
## makefigs.sh
## use:  bash makefigs.sh filename [no filename extension]
mathspic $1.m
latex $1.mt
dvips $1.dvi -o $1.ps
dvips -E $1.dvi -o $1.eps ## determines the BBox -->eps
epstopdf $1.eps
echo ... end of run
```

For example, to process the mathsPIC file `myfile.m`, one would type the command

```
$ bash makefigs.sh myfile
```

remembering *not* to include the filename extension here (since the script needs to append different filename extensions for the different processes).

¹Peck A (2008). *Beginning GIMP*. 2nd ed. (Apress); <http://www.apress.com/>.

JPEG

In order to make a JPEG file we make use of (a) the Perl program `fitps12.pl` (originally made by S. Rahtz (1996) and modified slightly² by Apostolos Syropoulos (2006) as version 1.2), and then (b) GhostScript, as follows:

```
## makefigs.sh
## use:  bash makefigs.sh filename [no filename extension]
mathspic $1.m
latex $1.mt
dvips $1.dvi -o $1.ps
dvips -E $1.dvi -o $1.eps      ## determines the BBox -->eps
perl fitps12.pl $1.eps $1.EPS ## outputs a modified EPS file
gs -sDEVICE=jpeg -r400 -sOutputFile=$1.jpg $1.EPS
echo ... end of run
```

Note that in the above example, the GhostScript command (`gs`) inputs the modified EPS file, and generates a JPEG file with resolution 400 dpi. See the GS help file for details (type `gs --help`). Resolutions 300 or 400 are generally fine for most purposes. See the Ghostscript man page and also

</usr/share/doc/HOWTO/HTML/en/Printing/ghostscript.html>

C.2 Using the figure environment

A `mathsPIC` diagram or figure can also be placed in a document by including the `P`_{CT}`E`_{X code directly in the main document within the `\begin{figure}... \end{figure}` environment, or by `\inputting` the code as a separate file. Note, also, the various extension packages available for facilitating the placement of pictures and images (e.g., the *subfigure* package).}

For example, if the `mathsPIC` file was called `mypic.m` and this generated the output file `mypic.mt`, then one would comment-out the `L`_A`T`_E`X` headers and footers from the `.mt` file, keeping just the part of the file within the `\beginpicture... \endpicture` environment (renaming it, say, `mypic.pic`) as follows.

```
%% this is file mypic.pic
\beginpicture
...
...
\endpicture
```

The `mypic.pic` file can then be `\input` into a document as a centered Figure as follows. Always use the `\centering` command as this avoids the additional vertical space added by the `\begin{center}... \end{center}` environment.

```
\begin{figure}[hbt]
\centering
```

²The original version required that there was exactly one space between each of the BBox values. This was a problem, since if you manually adjusted the BBox values, then the spacing could easily end up being slightly different, and the program would then fail. The new version is more flexible, and tolerates any number of spaces.

```

\strut\input{mypic.pic}
\caption{...}
\label{...}
\end{figure}

```

It is often useful when adjusting the position of a Figure on the page to initially place the Figure inside a frame in order to see the exact extent of the Figure. To do this just replace the `\input` line above with the following:

```
\strut\framebox{\input{mypic.pic}}
```

Sometimes more flexibility is needed regarding positioning the `\caption`, in which case a `\parbox` or a `minipage` is useful, as follows:

```

\begin{figure}[hbt]
\centering
\strut\input{mypic.pic}
\begin{minipage}{8cm}
\caption{\label{...}}
\end{minipage}
\end{figure}

```

Note that the general convention is that Tables have the caption at the top while Figures have the caption underneath the Figure.

Getting the vertical position and horizontal width of the caption just right can be awkward. However, using a `minipage` gives a lot more flexibility as in the following example, which defines the `\captionwidth` in terms of the `\textwidth`. Note also that here the `\caption{}` is empty—it is here just to trigger the Figure counter—we place the actual caption in the `minipage`.

```

\newcommand{\captionwidth}{0.8\textwidth}
%%
\begin{figure}[hbt]
\centering
%\framebox{
\strut \input{mypic.pic}
%}
[empty line]
\vspace\baselineskip
\begin{minipage}{\captionwidth}
\caption{\label{fig:fcubic}}%
{blah blah...}
\end{minipage}
\end{figure}

```

When typesetting text and a figure side-by-side the following format for using two adjacent `minipages` works quite well.

```

\begin{figure}[hbt]
\noindent
\begin{minipage}{4cm}

```

```

%% put some text or a figure here
...
...
\end{minipage}
%%-----
\hspace{3cm} %% controls the horizontal space between Figures
%%-----
\begin{minipage}{4cm}
  \vspace{3mm}
  \strut\hspace*{...}\input{circle.pic}
  \vspace{...} % adjusts vertical position of caption
  \caption{\label{...}}
\end{minipage}
%-----
\hfill
\end{figure}

```

Finally, it is often necessary to have two figures side-by-side, each with separate sub-captions, in which case the following rather similar format is useful—this was the construction used for displaying the two figures of Figure 8. Note the commented-out `\framebox{}` commands; these are very useful for revealing the full extent of any white-space surrounding the separate figures, since such unwanted white-space is probably the most common cause of difficulty when trying to position and center figures in a document. Note that while the `\centering` command can be used inside a Figure, the `\begin{center}... \end{center}` environment has to be used within a minipage.

```

\begin{figure}[hbt]
\centering
%-----
\noindent %\framebox{
\begin{minipage}{3cm}
  \begin{center}
    %\vspace{...} %% controls space above picture
    \input{mpicm08a.pic}
    %\vspace{...} %% controls space between picture and caption
    a. Circular arrows
  \end{center}
\end{minipage}
%} %end of framebox
%-----
\hspace{12mm} %% controls horiz space between figs
% \bigskip\bigskip %% use this to adjust vertical space
%-----
%\framebox{
\begin{minipage}{5cm}
  \begin{center}
    %\vspace{...} %% controls space between picture and caption
    \input{mpicm08b.pic}
    %\vspace{...} %% controls space between pict and caption
    b. Straight arrows
  \end{center}
\end{minipage}
%} %end of framebox
\end{figure}

```

```

        \end{center}
    \end{minipage}
    %} %end of framebox
    %-----
    \caption{\label{...}}
\end{figure}

```

C.3 Using the $\text{P}\text{T}\text{E}\text{X}$ code

A mathsPIC diagram or figure can also be placed in a document by including the $\text{P}\text{T}\text{E}\text{X}$ code directly in the main document within the `\beginpicture... \endpicture` environment, or by `\inputting` the code as a separate file. For example, if the mathsPIC file was called `mypic.m` and this generated the output-file `mypic.mt`, then one would comment-out everything except the picture environment itself, and renaming it, say, `mypic.pic`, as follows.

```

%% this is file mypic.pic
\beginpicture
...
...
\endpicture

```

The `mypic.pic` file can then be `\input` into a document as a centered Figure as follows.

```

\documentclass[a4paper]{article}
\usepackage{mathspic}
...
...
\begin{figure}[hbt]
\centering
    \strut\input{mypic.pic}
    \caption{...}
    \label{...}
\end{figure}

```

Finally, it is important to note that if this approach is used, then it would be necessary to comment out from the `mathspic.sty` file the $\text{L}\text{A}\text{T}\text{E}\text{X}$ command `\thispagestyle{empty}` in order to enable page numbering on the first page of the document.

Appendix D

Installing Perl in MS-Windows

Perl is available for all MS-Windows platforms, and there are several Perl implementations to choose from. A good choice (which we describe here) would seem to be the free ActiveState implementation of Perl in view of their excellent web site, documentation and other resources.

D.1 Perl

The free ActivePerl implementation is a well regarded choice of Perl for the Win32 platform, and is easy to download and install. The current version for download at the time of writing is Perl 5.8.6. There is excellent web support (e.g. documentation etc) via their ASPN website (ActiveState Programming Network).

- <http://www.activestate.com/Products/ActivePerl/>
- <http://aspn.activestate.com/ASPN/docs/ActivePerl/>

The complete ActivePerl package consists of the following:-

- Perl (binary core Perl distribution)
- An installer package
- Perl Package Manager (PPM—a Perl extension installer and manager)
- Documentation
- Perl ISAPI (IIS plug-in that enhances the speed of standard Perl)
- PerlScript (ActiveX scripting engine)
- PerlEz (embedded Perl)

Installation

A useful installation guide can be found at <http://www.activestate.com/ASPN/docs/ActivePerl/install.html>.

In addition to the Perl system itself, the older Windows systems 2000/NT also require you to download a special ‘installer’ package (`instMsiW.exe`) to implement the installation (see below). Note that the filename and sizes indicated below are those at the time of writing.

Step 1:

First, create a temporary directory for files which need to be downloaded. Then go to the ActiveState web-site (<http://www.activestate.com/>) and click on the Language/ActivePerl link which will take you to the ActivePerl page. Follow the 'download' links until you reach the 'download' page (you can skip the 'contact-info' section by clicking the 'next' button), and then read the Windows section for the latest version of ActivePerl (version 5.8.6).

Step 2:

Download the latest ActivePerl .msi file (approx 12 MB).
For example, ActivePerl-5.8.6.811-MSWin32-x86-122208.msi

Step 3:

Installation: the exact procedure depends on your particular Windows system as detailed below. The default installation is the recommended 'complete' installation.

- Windows XP/2003:
To install simply double click on the ActivePerl .msi file and follow the instructions (these systems do *not* require a separate installer file).

Running Perl

Now that Perl is installed, we can progress to creating and running a Perl program. To create a Perl program, simply open your favorite text-editor (see below for information on some suitable text-editors) and write the program (as an ASCII text file), and finally save it with a .pl extension.

For example, open a new file and type the following into it, and then save it with the file-name `test.pl` (note that in Perl the # sign is the 'comment' symbol; the \n starts a new line; variable names start with a \\$.).

```
#!/usr/bin/perl
# test.pl
print "hello world\n";
$a=3, $b=7;
$sum=$a + $b;
print "sum = $sum\n";
```

When this is run it will show the following output

```
hello world
sum = 10
```

We can run this Perl program several ways, as follows.

- Use the command-line:
Open a DOS box, move to the required directory, and at the command-line prompt type the following command, and then press <enter>

```
perl test.pl
```

The output generated by the program will be written to the screen in the usual way. If you want the output written to a file (say. `test.txt`) then instead type `perl test.pl > test.txt`

- Click on the filename

If you just click on the filename in the directory listing (provided it has the `.pl` filename extension) then ActivePerl will automatically open a DOS box, run the program, and then close the DOS box. This is fine if the program output is directed to a file, but if the output is simply directed to the screen (the default) then you need to add the `<>` command at the end of the file, as this will keep the screen open (so you can read the screen) until you hit a key.

```
#!/usr/bin/perl
# test.pl
print "hello world\n";
$a=3, $b=7;
$sum=$a + $b;
print "sum = $sum\n";
# -----
print "press any key to exit\n";
<>
```

- Using a batch file:

Create a separate batch file with the name, say, `test.bat` containing the following lines:

```
perl test.pl %1 %2
```

This batch file now has to be run at the command-line as shown above. Note that ActivePerl can create a batch file from an ordinary perl program by using the command utility `p12bat.bat`. Simply access the command-line and type

```
p12bat test.pl
```

which will then generate the file `test.bat` which can then be run at the command-line by just typing `test`.

Perl modules

Additional modules can be downloaded from the huge Perl archives available on the internet. These Perl modules can be installed either automatically (via the internet using the PPM utility), or manually (download the module and then use the `nmake` utility). The `nmake` utility is a Microsoft port of the Unix `make` utility, and can be downloaded from the Microsoft internet archive at <ftp://ftp.microsoft.com/Softlib/MSLFILES/>. The current version is bundled in the self-extracting package `nmake15.exe`

Useful Perl links

- *Introduction to Perl on Windows.*
<http://www.wdvl.com/Authoring/Languages/Perl/Windows/>

- CPAN Perl Ports (binary distributions).
<http://www.cpan.org/ports/index.html>
- How to install Perl modules.
<http://www.cpan.org/modules/INSTALL.html>
- Perl documentation.
<http://www.perldoc.com>

D.2 Text editors

Crimson Editor (free)

This is a good straight-forward ‘all-purpose’ editor for Win-32 platforms, with good syntax highlighting, available from <http://crimson.emeraldeditor.com/>

WinEdt (shareware)

This is an excellent ‘all-purpose’ text editor designed for use with T_EX and L^AT_EX; it has syntax highlighting and can be tailored for use with Perl. It is available for download from <http://www.winedt.com/>. Note that a free licence for this editor is provided as a membership benefit by many T_EX User Groups (e.g. ukTUG).

WinEDIT

The WinEDIT PowerPack is an excellent (commercial) general purpose text editor available for download (21-day free trial) from <http://www.winedit.com/>. It is very similar to WinEdt, and has the useful facility for column copying/pasting, and also an optional vertical window holding the directory tree.

GNU Emacs (free)

This is a public domain (free) text editor. It is available from the GNU distribution page as follows <http://www.gnu.org/software/emacs/windows/>. See also Harry Halpin’s excellent web page entitled *How to install Perl and Emacs on a Windows machine* which can be found at <http://lcb.unc.edu/software/perl/perl.html>.

Appendix E

P_ICT_EX

P_ICT_EX is an excellent small graphics package, which is available on the T_EXLive DVD and from CTAN. If you have a recent T_EX installation then it is likely that P_ICT_EX is already installed¹. If P_ICT_EX is not installed, then create a P_ICT_EX subdirectory in your ‘local’ T_EX space, typically at

```
/usr/local/texlive/texmf-local/tex/latex/local/pictex/
```

Now copy into this new subdirectory all the P_ICT_EX files in the following CTAN directories:

```
CTAN:/tex-archive/graphics/pictex/  
CTAN:/tex-archive/graphics/pictex/addon/
```

Unfortunately, the *P_ICT_EX Manual* is not available from CTAN—the manual has to be purchased separately (see Section E.6). However, mathsPIC users will mostly find this unnecessary, since those P_ICT_EX commands which are particularly useful in conjunction with mathsPIC are described here in Chapter 5.

In the following, we present some background, including details about how to obtain a copy of the P_ICT_EX manual.

E.1 The original files (1986)

```
CTAN/tex-archive/graphics/pictex/
```

The original P_ICT_EX package by Michael Wichura (21/09/1987) originally consisted of the 4 files listed below. On the left is the MS-DOS name (truncated to 8 characters), and on the right is the full UNIX name).

The file `latexpic.tex` gives Plain T_EX users the option of using the L^AT_EX Picture macros `\line`, `\vector`, `\circle`, `\oval`, `\thicklines` and `\thinlines` (for syntax and use with P_ICT_EX see Wichura, 1992).

¹Check for any P_ICT_EX directories by typing the command `$ locate pictex`

Note that only two of the following files are required when using plain $\text{T}_{\text{E}}\text{X}$, while three files are required when using $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$.

<code>latexpic.tex</code>	11243 bytes	(<code>latexpicobjs.tex</code> - for plain $\text{T}_{\text{E}}\text{X}$ only)
<code>prepicte.tex</code>	1293 bytes	(<code>prepicte.tex</code> - for $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$)
<code>pictex.tex</code>	133388 bytes	(<code>pictex.tex</code> - for $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ & plain $\text{T}_{\text{E}}\text{X}$)
<code>postpict.tex</code>	1614 bytes	(<code>postpictex.tex</code> - for $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$)

The three files required for use with $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ need to be loaded (input) *in the order shown above*. Note that when using $\text{P}_{\text{I}}\text{C}_{\text{T}}\text{E}_{\text{X}}$ with $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2_{\epsilon}$ it is necessary to redefine the $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2.09$ command `\fiverm` (because $\text{P}_{\text{I}}\text{C}_{\text{T}}\text{E}_{\text{X}}$ was originally written for $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2.09$). This is most easily done as follows (and hence it is included in the `mathspic.sty` file).

```
\newcommand{\fiverm}{\rmfamily\tiny}
```

Alternatively you can use the more robust method (i.e. for *wizards*) suggested by Michael J Downes as follows (I believe Michael Downes suggested this originally on the *comp.text.tex* usenet group—see also the file `fntguide.tex` among the $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X } 2_{\epsilon}$ documents).

```
\declarefixedfont{\fiverm}{\encodingdefault}{\rmdefault}{m}{n}{5}
```

A significant problem with the original $\text{P}_{\text{I}}\text{C}_{\text{T}}\text{E}_{\text{X}}$ package was that it was very memory hungry. However, in 1994 this problem was overcome by a significant rewrite by Andreas Schrell, as described in the next section.

E.2 The new updated files (1994)

CTAN/`tex-archive/graphics/pictex/addon/`

In 1994 Andreas Schrell uploaded a set of updated $\text{P}_{\text{I}}\text{C}_{\text{T}}\text{E}_{\text{X}}$ files into the CTAN: `.../graphics/pictex/addon/` directory. One of these additional files of Schrell's (`pictexwd.tex`) is a replacement for the original (`pictex.tex`), and is extremely economic in its use of $\text{T}_{\text{E}}\text{X}$'s dimension registers, allowing significantly better memory usage with exactly the same functionality. The other files correct some errors (`piccorr.sty`), and add new functionality (`picmore.tex`).

<code>pictexwd.sty</code>	416 bytes
<code>pictexwd.tex</code>	133232 bytes
<code>picmore.tex</code>	2952 bytes
<code>piccorr.sty</code>	4608 bytes
<code>pictex.sty</code>	311 bytes

- `pictexwd.sty`

For $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ users. This replaces `pictex.sty`. It inputs `prepicte.tex`, `pictexwd.tex`, and `postpictex.tex`, as well as inputting `piccorr.sty` and `picmore.tex` if these are available.

- `pictexwd.tex`

For $\text{T}_{\text{E}}\text{X}$ users.

- `picmore.tex`
An extension for drawing impulse diagrams.
- `piccorr.sty`
A correction for the original P_{CT}E_X `\betweenarrows` command.
- `pictex.sty`
The L_AT_EX style-option for loading the *original* P_{CT}E_X files. Note that it also inputs `piccorr.sty` and `picmore.tex` if these are available.

All the necessary files required for running P_{CT}E_X are automatically input in the correct order by `pictexwd.sty` (L_AT_EX users), or by `pictexwd.tex` (T_EX users). Users of L_AT_EX 2_ε should include the following command in the preamble.

```
\usepackage{pictexwd}
```

Users of plain T_EX need to include the following.

```
\input latexpic.tex
\input pictexwd.tex
```

Note that even when using Andreas Schrell's new files, all the original P_{CT}E_X files are still required.

E.3 Pictex2.sty

CTAN/tex-archive/macros/latex/contrib/supported/pictex2.sty
16418 bytes 09/05/1999

William Park has written a style option (`pictex2.sty`) which adds two useful commands to standard P_{CT}E_X, which force the use of the `\putrule` command where lines are either horizontal or vertical, thus saving on memory (note that `mathsPIC` automatically implements the use of `\putrule` in these circumstances). These two commands are as follows.

- `\putanyline`
This command invokes the P_{CT}E_X `\putrule` command (instead of `\plot`) in cases where lines are either vertical or horizontal.
- `\setanyline`
This is a line-drawing mode (similar to `\setlinear`) which forces subsequent `\plot` commands to invoke `\putrule` whenever the line is either horizontal or vertical.

E.4 Errorbars.tex

CTAN/tex-archive/graphics/pictex/errorbars.tex
3041 bytes 20/04/1988

In 1988 Dirk Grunwald implemented a new $\text{P}\text{I}\text{C}\text{T}\text{E}\text{X}$ command for drawing error bars using a modified $\backslash\text{plot}$ command, as follows.

- $\backslash\text{plotWithErrorBars}$

This command, which is case sensitive, has the format

```
 $\backslash\text{plotWithErrorBars}$  mark M at
  x1 y1 e1
  ...
  xn yn en /
```

where M is a TEX character (e.g. \bullet), and e is the size of vertical error bars which are plotted above and below the point. The default cross-bar length is 5pt, but can be changed, say to 7pt, using the command $\backslash\text{crossbarlength}=7\text{pt}$.

To use this command, input the file `errorbars.tex` after all the $\text{P}\text{I}\text{C}\text{T}\text{E}\text{X}$ files.

E.5 DCpic

DCpic (by Pedro Quaresma) is a package of TEX macros for drawing commutative diagrams in a $\text{L}\text{A}\text{T}\text{E}\text{X}$ or $\text{C}\text{O}\text{N}\text{T}\text{E}\text{X}$ document. It uses $\text{P}\text{I}\text{C}\text{T}\text{E}\text{X}$ to implement commands which manipulate its various ‘objects’ and arrows. This package (February 2002) is available on CTAN (and $\text{T}\text{E}\text{X}\text{L}\text{i}\text{v}\text{e}$) at

CTAN: `tex-archive/macros/generic/diagrams/dcpic/`

and includes the following documentation files:

`manPT_dcpic.pdf`
`man_dcpic.pdf`

E.6 The $\text{P}\text{I}\text{C}\text{T}\text{E}\text{X}$ Manual

I would recommend purchasing the excellent *The $\text{P}\text{I}\text{C}\text{T}\text{E}\text{X}$ Manual* by Michael J. Wichura (The University of Chicago, Chicago, Illinois, USA). This small booklet (version 1.1, third printing, March 1992; 85 pages) used to be available from TUG as Publication No. 6 in the TUG TEX niques series. Unfortunately, TUG (<http://www.tug.org/>) has now stopped publishing the manual, and so *The $\text{P}\text{I}\text{C}\text{T}\text{E}\text{X}$ Manual* is currently only available from *Personal TEX Inc.* (texsales@pctex.com, <http://www.pctex.com/books.html>).

Summary of the manual

Probably the most useful alternative document is the excellent and detailed 8-page summary of $\text{P}\text{I}\text{C}\text{T}\text{E}\text{X}$ commands made by Angus Duggan (1990), available on CTAN (as *summary.zip*) at

CTAN: `../info/pictex/summary/pictexsum.pdf`
 CTAN: `../info/pictex/summary/pictexsum.tex` (Latex209 version)
 CTAN: `../info/pictex/summary/useful.sty`
 CTAN: `../info/pictex/summary/ukdate.sty`

These are also included in the CTAN section of the T_EXLive-2010 DVD. Note that both of the style files `useful.sty` and `ukdate.sty` are required to easily latex his `.tex` file.

A short 3-page German summary by Siart (2008) is available at <http://www.siart.de/typografie/pictex-referenz.pdf>

Other info

Lueck (2008) is a useful list of further information. Note some other documentation is available on the T_EXLive-2010 DVD, and is typically installed at the following locations:

```
/usr/local/texlive/2010/texmf-dist/doc/generic/FAQ-en/html/FAQ-docpictex.html  
/usr/local/texlive/2010/texmf-dist/doc/generic/FAQ-en/html/FAQ-usepictex.html  
/usr/local/texlive/2010/texmf-dist/doc/generic/doc-pictex  
/usr/local/texlive/2010/texmf-dist/doc/generic/doc-pictex/Doc-PiCTeX.txt
```

References

- Abrahams P. W., Berry K. and Hargreaves K. A. (1990). *T_EX for the impatient*. (Addison-Wesley).
- Cameron P. J. (1992). Geometric diagrams in L^AT_EX. *TUGboat* **13** (No. 2), 215–216.
- Beccari C. (1991). *LaTeX – Guida ad un sistema di editoria elettronica*. (Editore: Ulrico Hoepli, Milano.) ISBN:88-203-1931-4. (for details see: *T_EX and TUG News (1992)*; Vol 1, No 1: p 14. CTAN:[/tex-archive/digests/ttn/ttn1n1.tex](#))
- Duggan A. (1990). P_ICT_EX command summary (summary.zip) [note you need both style files to process the tex file (latex 209)]
CTAN: [../tex-archive/info/pictex/summary/pictexsum.pdf](#)
CTAN: [../tex-archive/info/pictex/summary/pictexsum.tex](#)
CTAN: [../tex-archive/info/pictex/summary/useful.sty](#)
CTAN: [../tex-archive/info/pictex/summary/ukdate.sty](#)
<http://chem.skku.ac.kr/~wpark/tutor/chem/ppchtex/pictexsum.ps>
- Eijkhout V. (1992). *T_EX by topic: a T_EXnician's reference* (Addison-Wesley). This excellent book is now out of print, but Victor Eijkhout has kindly made it available on-line at <http://www.eijkhout.net/tbt/>
- Feruglio G. V. (1994). Typesetting commutative diagrams. *TUGboat* **15** (No. 4), 466–484.
- Hoenig A (1998). *T_EX Unbound: L^AT_EX and T_EX strategies for fonts, graphics, & more*. (Oxford University Press, UK) pp. 580. ISBN: 0-19-509685-1 (hardback), 0-19-509686-X (paperback).
[see pages 377–389 for details of P_ICT_EX and a summary of commands]
- Holzner S. (1999) Perl Core Language—Little black book. (Technology Press—The Coriolis Group) ISBN 1–57610–426–5.
- Knuth D. E. (1990). *The T_EXbook*; 19th printing. (Addison-Wesley).
- Lueck U (2008). Documentation for P_ICT_EX.
[/texlive/2010/texmf-dist/doc/generic/doc-pictex/Doc-PiCTeX.txt](#)
- Nickalls R. W. D. (1999). MathsPIC: a filter program for use with P_ICT_EX. *EuroT_EX'99 Proceedings*; 197–210. (Heidelberg, Germany; August 1999). <http://www.uni-giessen.de/~g029/eurotex99/nickalls.pdf> [MS-DOS version] [reproduced in: *Eutupon*, No. 3 (October, 1999), 33–49 (the Greek T_EX Friends' journal)]

- Nickalls R. W. D. (2000). `mathsPICDOS 2.1`
<http://www.tex.ac.uk/tex-archive/graphics/mathspic/dos/>
- Peck A (2008). *Beginning GIMP*. 2nd ed. (Apress); <http://www.apress.com/>.
- Ramsey N. (1994). Literate programming simplified. *IEEE Software*; 11(5), 97–105.
- Salomon D. (1992). *The advanced T_EXbook*. (Springer)
- Salomon D. (1992). Arrows for technical drawing. *TUGboat* **13** (No. 2), 146–149.
- Siart U (date: ?)
<http://www.siart.de/typografie/pictex-referenz.pdf> [from Lueck (2008)]
- Syropoulos A. (1999). Literate programming: the other side of the coin. *RAM Magazine*; 129, 248–253 [in Greek]
- Syropoulos A. and Nickalls R. W. D. (2000). A PERL porting of the `mathsPIC` graphics package. *TUG2000 Annual Meeting*, Oxford, UK (August 13–16, 2000). *TUGboat*, 21, 292–297. <http://www.tug.org/TUGboat/articles/letters/tb21-3/tb68syro.pdf>
- Syropoulos A and Nickalls RWD (2005). `MathsPICPerl 1.0` (<http://www.tex.ac.uk/tex-archive/graphics/mathspic/perl/>)
[A Perl filter program for implementing the P_TE_X graphics package]
- Syropoulos A and Nickalls RWD (2007). `MathsPICPerl 1.1` (<http://www.tex.ac.uk/tex-archive/graphics/mathspic/perl/>)
[A new bug-fix version: February 2007]
- Syropoulos A., Tsolomitis A. and Sofroniou N. (2003). *Digital typography using L_AT_EX*. (Springer)
- Wichura M. J. (1988). P_TE_X: Macros for drawing P_ICtures *TUGboat*, 9(2), pp. 193–197.
<http://www.tug.org/TUGboat/Articles/tb09-2/tb21wichura-pictex.pdf> [from Lueck (2008)]
- Wichura M. J. (1992). The P_TE_X manual. Pub: Personal T_EX Inc., 12 Madrona Avenue, Mill Valley, CA 94941, USA.
texsales@pctex.com, <http://www.pctex.com/books.html>