mathsPIC 2.1

Richard W. D. Nickalls,
Department of Anaesthesia,
City Hospital, Nottingham, UK.
dicknickalls@compuserve.com
Tel: +44-(0)115-9691169
Fax: +44-(0)115-9627713

November 5th, 2000

Contents

1 Introduction

MathsPIC[1] is a small filter program for use with the excellent free PICTEX drawing engine[2] (Nickalls, 1999a, 1999b; Syropoulos and Nickalls, 2000). MathsPIC parses a plain text input-file (the mathsPIC file), and generates a plain text output-file containing PICTEX and TEX commands which can then be TEXed (or LATEXed) in the usual way. Spaces and the comment % symbol are used in the same way as TEX, although unlike TEX, mathsPIC commands are *not* case-sensitive. PICTEX, TEX and LATEX commands can all be freely used in the mathsPIC 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 motivation for mathsPIC stems from the fact that, while PICTEX is an extremely versatile system for drawing figures, and offers the convenience and advantages of having the graphics code within the TEX document itself (e.g. printer-independence), it does require you to specify the coordinates of most points. This can make PICTEX quite awkward to use with 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, and the PICTEX commands for drawing it are as follows (the units are in cm).

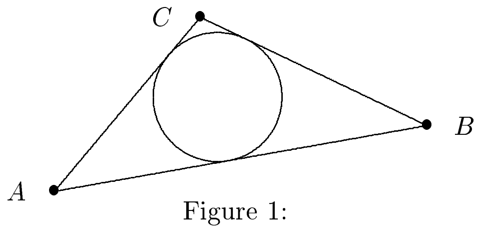


Figure 1:

```
\put {$\bullet$} at  0  0                                    % draw point A
\put {$\bullet$} at  4.924039   0.8682409                    % draw point B
\put {$\bullet$} at  1.928363   2.298133                     % draw point C
\plot 0  0   4.924039  0.8682409    1.928363  2.298133   0  0/   % draw triangle
\circulararc 360 degrees from 3.008607  1.245238 center at 2.156815  1.245238
\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 it is then necessary to resort to geometry and a calculator to determine points B and C, since AB, AC, and the included angle are 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, PICTEX 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 scalar variables, which has the effect of making even complicated mathematical diagrams very easy to create. For example, the equivalent mathsPIC commands for drawing Figure 1 are as follows (the units are in cm as before).

```
point(A){0,0}                 % point A at origin
point(B){A,polar(5,10 deg)}   % point B 5 cm from A; AB slope 10 deg
```

[1] mathsPIC (`CTAN:/tex-archive/graphics/pictex/mathspic/`) was first presented at the EuroTEX'99 conference in Heidelberg, Germany. It is a free MS-DOS program (150 KB) compatible with MS-Windows 9x, 2000, NT.

[2] `CTAN:/tex-archive/graphics/pictex/addon/`. The files input using `pictexwd.sty` total only 138 KB. See Section 9 for details.

```
point(C){A,polar(3,50 deg)}   % point C 3 cm from A; BAC = 40 deg
drawPoint(ABC)                % put $\bullet$ at points A B C
drawTriangle(ABC)
drawIncircle(ABC)
variable(d){0.5}              % d = 0.5 cm
text($A$){A, shift(-d,0)}     % label for A
text($B$){B, shift(d,0)}      % label for B
text($C$){C, shift(-d,0)}     % label for C
```

MathsPIC facilitates the drawing of PICTEX 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 (see Section 4.5). 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). Files can also be input recursively (see Section 7.6). A significant feature of mathsPIC is that it allows TeX macros to be easily used within the mathsPIC file.

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

A PERL version of mathsPIC is currently being developed (Syropoulos and Nickalls, 2000), and should be available in 2001 (see Section 11).

2 Installing mathsPIC

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

mathsPIC can be downloaded from the above directory at CTAN (Comprehensive TEX Archive Network). See the file `readme.txt`. When installing PICTEX (see Section 9) I would recommend using the new PICTEX files by Andreas Schrell (1994), namely `pictexwd.sty` (for LATEX users) and `pictexwd.tex` (for plain TEX users).

2.1 The program

```
mpic21.exe    (rename to mathspic.exe)
```

The executable program `mpic21.exe` was compiled on an MS-DOS 6.2 platform. It is compatible with WINDOWS 9x, 2000, NT. For the purposes of this manual it is suggested that `mpic21.exe` be renamed to `mathspic.exe`. The executable program must be either in the same directory as the mathsPIC input-file, or in the directory path.

2.2 Optional utilities

```
show.com    (a DOS textviewer)
clean.exe   (a DOS tool for deleting comment lines)
```

These two files should be placed in the same directory as `mathspic.exe`. `clean.exe` is a program for removing all the commented lines from the output-file (see Section 3.2 for details). `show.com` is a simple textviewer which can be invoked using the /s switch (see Section 3.1.2 for details).

2.3 The manual

```
mpicm21.tex
mpicm21.dvi
mpicm21.ps
```

```
mpicm21.pdf
mpicm01.pic - mpicm16.pic
```

The 16 figure files `mpicm01.pic`—`mpicm16.pic` are input by `mpicm21.tex`. Note that all the mathsPIC files for these figures (`mpicm01.m`—`mpicm16.m`) are also included in the package—users are free to experiment with them as they wish.

2.4 Examples

The file `examples` is a collection of example figures and corresponding mathsPIC code.

3 Running mathsPIC

3.1 Command-line syntax

Assuming the `.exe` file has been renamed to `mathspic.exe`, then the minimum command-line instruction requires only the name of an input-file as follows.

```
mathspic  inputfile
```

This will result in `mathspic` generating an output-file (in the same directory) with the same filename but with a `.mt` filename extension (see below for options), and a log-file (`.mlg`).

While mathsPIC is running, its progress is documented on the screen by a series of dots (one dot per program line). If an error is encounted then this is shown as an E instead of a dot, as follows.

```
.................E....EE..........
```

3.1.1 Files

MathsPIC actions only the first two filenames (any additional filenames are ignored). Each filename must be isolated by at least one space. The order is important; the first filename is actioned as the `inputfile` (the mathsPIC file); the second filename (if present) is actioned as the `outputfile`.

```
mathspic  inputfile  outputfile
```

If the `outputfile` is not specified then mathsPIC will create an output-file having the same filename as the `inputfile` 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  myoutfile.xyz
```

In practice, I find it convenient to use the filename extension `.m` for mathsPIC files (input-files), since I can then easily distinguish between the various files. Thus `.m` files are mathsPIC files (input-files), while `.mt` files are output TEX files containing PICTEX commands ready for TEXing, and `.mlg` files are mathsPIC log-files.

3.1.2 Switches

Five switches are currently used (`/? /h /help /b /s`). Switches are *not* case-sensitive; they can be run together without intervening spaces; and they can be positioned anywhere, provided that they are separated by at least one space from any filenames. The switches are as follows.

`/?`	Help—shows all commands (also `/h  /help`).
`/b`	Error-beep. A beep is sounded if mathsPIC detects an error.
`/s`	Screen view. Automatically shows the output-file using the free text-viewer `show.com`.

For example, the following command invokes both the error-beep and automatic viewing of the output-file by the default file-viewer `show.com`.

```
mathspic /b /s  inputfile  outputfile
```

3.2 Removing comment lines from the output-file

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 included CLEAN.EXE utility, which will strip out all lines starting with the double comment symbol %%. The /s switch can also be used with CLEAN.EXE. Examples of the syntax are as follows.

```
clean.exe /?
clean.exe  inputfilename
clean.exe  /s  inputfilename
clean.exe  /s  inputfilename outputfilename
```

If no output filename is supplied, then CLEAN.EXE will use the same filename but add the filename extension `.mtc`. For example, the following command will clean the output-file `myfile.mt`, generating the file `myfile.mtc` as the cleaned file, and then view the 'cleaned' output-file using `show.com`.

```
clean.exe  /s  myfile.mt
```

3.3 Batch processing

In addition to running the program from the command-line, mathsPIC can also be run from a batch-file (some familiarity with the MS-DOS batch-file language is assumed; note that the batch-files need to be tailored to the particular command-line sequence.) For example, the following command line with five parameters (two switches and three filenames)

```
mathspic  /b /s  testpic.m  testpic.mt  testpic.dvi
```

will be executed by the following MS-DOS batch-file (written for the EmTEX[3] implementation of LaTeX for PCs using MS-DOS), which first runs mathsPIC on the input-file `testpic.m`; then calls LaTeX 2_ε to process the output-file; and finally calls the EmTEX screen viewer `dviscrs.exe` (via `vs.bat`) to view the file `testpic.dvi` (%5).

```
REM  mathsPIC.bat (MS-DOS 6.0 and greater)
@ECHO OFF
mathspic.exe %1 %2 %3 %4 %5
CALL latex2e %4
CALL vs.bat /s3 %5
```

Note that care needs to be taken to make sure that the parameters used in the batch-file correspond to the command-line parameters. For example, if the two switches are run together without a space (e.g. `/b/s`) they will be perceived by MS-DOS as a single entity, in which case the `.dvi` file would be represented by %4 in the batch-file instead of %5. For example, the following four-parameter command-line

```
mathspic  /b/s  testpic.m  testpic.mt  testpic.dvi
```

would require the following MS-DOS batch-file.

```
@ECHO OFF
mathspic.exe  %1 %2 %3 %4
CALL latex2e.bat %3
CALL vs.bat  /s3 %4
```

[3] The EmTEX screen viewer is `dviscrs.exe` and is usually called via the batch-file `vs.bat`.

As a final example (again based on the EmTEX implementation), suppose you wish to use your own text-viewer (say, `myTextViewer.exe`) to view the output-file, with the `/b` switch (enables error-beep), and have conditional branching—i.e. have the option to either (a) quit (select the 'N' option) and return to the text-editor after viewing the output-file, or (b) proceed (select the 'Y' option) and process the output-file `testpic.mt` (%3) and then view the `.dvi` file (%4). In this example, the following four-parameter command-line

```
mathspic  /b  testpic.m  testpic.mt  testpic.dvi
```

could be run using the following batch-file.

```
REM  mPIC.bat (MS-DOS 6.0 and greater)
@ECHO OFF
mathspic.exe %1 %2 %3 %4
CALL myTextViewer.exe %3
CHOICE /C:YN  View figure?:
IF ERRORLEVEL 1 IF NOT ERRORLEVEL 2 GOTO yes
GOTO  no
:yes
CALL latex2e %3
CALL vs.bat /s3 %4
:no
```

4 The mathsPIC file and script

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, PICTEX commands, as well as lines to be copied verbatim) so that the output-file can be immediately TEXed 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 and PICTEX 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(a1a22a34a4)` can be made easier to read by writing it as `drawpoint(a1 a22 a34 a4)`.
- 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 be written as `drawPoint` or `DrawPoint` etc.
- Delimiters have a hierarchical structure as follows:
  Curved brackets contain the primary argument; e.g. `showPoint(A)`
  Braces contain required supporting arguments; e.g. `point(A){5,6}`
  Square brackets contain optional arguments; e.g. `point(p3){a,shift(1,2)}[circle,5]`
- Logically distinct groups within brackets must be separated by commas.
  e.g. `point(B2){A,polar(3,40deg)}[circle,5]`
- 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 PICTEX 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 (see Sections 4.1 and 4.2).
- 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 PICTEX commands and/or coordinate data; for example, a list of data points as part of a PICTEX `\plot` command)—see example in Section 7.6.1

4.1 PICTEX commands

Note that PICTEX commands can *only* be used within the `\beginpicture` ... `\endpicture` environment, as described in the PICTEX Manual[4] (Wichura, 1992).

Most PICTEX 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 PICTEX `\setdashes` command, and this would be expressed in the mathsPIC file as follows.

```
\setdashes
```

However, some PICTEX commands are very long. It is therefore sometimes necessary to have the initial part of a PICTEX 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 PICTEX code for plotting data-points for a curve could be spread across several lines in the mathsPIC file as follows.

```
\setquadratic
\plot  1.15    -0.67
\      1.25     0.02
\      1.35     1.24
\      1.45     3.13 /
\setlinear
```

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

```
\setquadratic
\plot  1.15    -0.67
       1.25     0.02
       1.35     1.24
       1.45     3.13 /
\setlinear
```

[4]See Section 9.5 for information regarding the PICTEX manual.

4.2 TEX and LATEX commands

4.2.1 Headers and footers

It is particularly useful to include in the mathsPIC file any TEX or LATEX headers and footers which would otherwise have to be added manually to the output-file before LATEXing the file. For example, when using LATEX a typical format for a mathsPIC file might be as follows (note that in this example Andreas Schrell's style option `pictexwd.sty` is used—see Section 9.2).

```
\documentclass[a4paper]{article}
\usepackage{pictexwd}
\begin{document}
\beginpicture
...
...
\endpicture
\end{document}
```

For users of plain TEX a typical format might be as follows (see Section 9).

```
\input latexpic.tex
\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 simply convenient) to extend a TEX or LATEX command across several lines, then each additional line must be protected within the mathsPIC file using a leading `\␣...` sequence (see Section 4.1) unless a line actually starts with a TEX command. A typical example is the following macro (used in Figure 9) which defines a 'display' maths formula. The macro is split across several lines, as follows.

```
\newcommand{\formula}{%
\     $\displaystyle \sum_{p\ge0} \Delta_{jp} z^{(p+1)}$%
\     }%
text(\formula){B1}
```

Note that when using TEX or LATEX commands *within* the PICTEX picture environment, it is very important to include the comment `%` symbol at the end of such lines, to prevent PICTEX accumulating additional space characters from the ends of non-PICTEX commands, since otherwise PICTEX 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.

4.2.2 Other TEX commands

A very useful command is the `\typeout{}` command, which will make TEX 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 is 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 TEX 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 PICTEX picture environment.

4.3 The plotting area

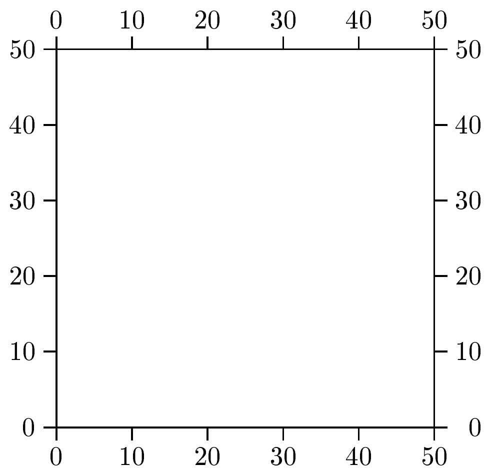
4.3.1 Axes

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` parameters[5]. The axes-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 x 5 cm with a ruled frame on four sides as shown in Figure 2a.

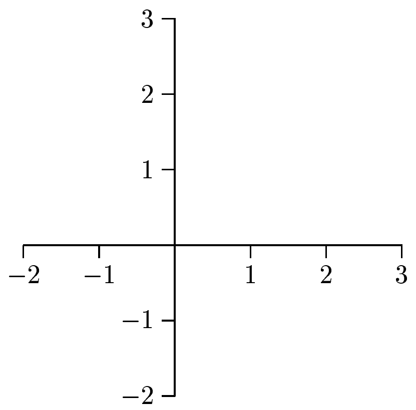
```
paper{units(mm),xrange(0,50),yrange(0,50),axes(LRTB),ticks(10,10)}
```

This particular `paper` command is converted by mathsPIC into the following PICTEX 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 /
```



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



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

Figure 2:

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 2b was generated using the following `paper` command,

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

[5]Since PICTEX uses the name 'axis', mathsPIC recognises both spellings ('axis' and 'axes').

which is converted by mathsPIC into the following PICTEX 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 PICTEX `\linethickness` command (see Section 4.6.1, and also Figure 14).

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 14). If complicated or more demanding axis configurations are required, then the PICTEX Manual (see Section 9.5) will need to be consulted. See also Section 8 for details on positioning figures within LATEX documents.

4.3.2 Units

In addition to the usual units `mm`, `cm`, and `pt`, PICTEX accomodates the other six units used by TEX[6], 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 enbrace the Metric system (mm, cm), the Didot system (didot, cicero), and the UK system (point, big point, pica, inch).

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(mm)` in the `paper` command is translated by mathsPIC into the following PICTEX command in the output-file.

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

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`.

```
variable(u){1.5}
paper{units(u mm), xrange(0,100), yrange(0,100)}
...
variable(r){2}
variable*(r){r, divide(u)}
point(A){30,20}[circle,r]
variable(s){4}
variable*(s){s, divide(u)
text($A$){A, shift(-s,s)}
```

If different scales are required (most commonly when drawing curves and equations—see Figure 14) 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

[6] These are described in Chapter 10 of the *The TEXbook* by D. Knuth.

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.

4.3.3 Tick-marks

It is recommended that integers are used with the `ticks` option, since PICTEX sometimes gives unpredictable results if decimals are used with the `xrange` and `yrange` options in conjunction with the `ticks` option. In general PICTEX gives more pleasing axes if integers are used throughout the `paper` command.

4.4 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 the length of line AB. A group of three points represents either a triangle, an angle, or a line, depending on the circumstances.

4.4.1 Point-name

A point-name *must* begin with a *single* letter, and may have up to a *maximum* of two following digits. The following are valid point-names: $A, B, C3, d45$. Since mathsPIC is not case sensitive the points $d45$ and $D45$ are regarded as being the same point. Point-names can be either separated by spaces, or simply run together.

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){A, shift(5,0)}
```

4.4.2 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 including 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}[$\triangle$]
```

The default point-symbol can also be changed to any TEX character or string of characters by using the mathsPIC `PointSymbol` command, and putting this command *before* any relevant `point` commands, since 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 reinstituted (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 be positioned symmetrically over a point-location. Most characters are therefore not ideal for use as point-symbols, as they generally require some slight adjustment in order to position them symmetrically. In view of this, Table 1 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 with PICTEX.

4.4.3 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}[$\triangle$,5]
```

If only the line-free radius is to be specified then a preceding comma must be used (e.g. `[,10]`). For example the following mathsPIC command would change the line-free radius associated with point A to 10 units.

```
point*(A){A}[,10]
```

Table 1 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` fonts[7]. 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).

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

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}[,2]
```

By way of illustration, this command is used in drawing Figure 3 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 `mpicm03a.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.

```
%% mpicm03a.m (Figure 3)
\beginpicture
paper{units(cm),xrange(0,6),yrange(0,3),axes(LBT*R*),ticks(1,1)}
point(P){4,2}[$\odot$]
point(B){2,0.5}
point(S){1,2}
drawPoint(PBS)
\setdashes
```

[7]See Harald 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/texcomp-info/`.

Table 1: Useful point-symbols and their radii for 10–12pt fonts.

symbol		radius mm 10pt / 11pt / 12pt	symbol package
`$\circ$`	$\circ$	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

```
\inboundscheckon         %% restrict circles to drawing area
drawcircle(P,1)
drawcircle(P,2)
\setsolid
point*(P){P}[,1]         %% set line-free radius of P to 1 cm
drawArrow(BP)            %% draw arrow from B (below)
point*(P){P}[,2]         %% change line-free radius of P to 2 cm
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 Figure 3, the two `point*` commands and the two `drawArrow` commands

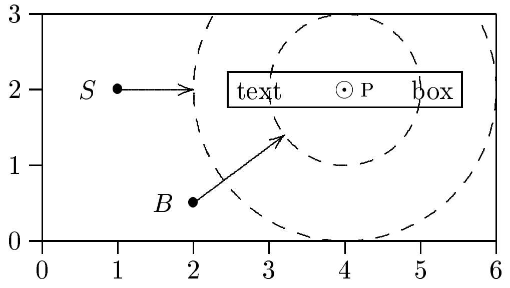


Figure 3:

could all be replaced by the following two commands using the [ ] option to indicate the length of the arrow.

```
drawArrow(SP)[1]        %% draw arrow 1 unit from point A towards P
drawArrow(BP)[1.5]      %% draw arrow 1.5 units from point B towards P
```

4.4.4 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*.

4.4.5 Number of points

The maximum combined number of named points and variables which can be used is currently set by default at 50. The total array space for points and variables can be increased by using the mathsPIC `PointNumber()` command. For example, array space for a total of 75 points/variables would be allocated by the command `PointNumber(75)`.

If the combined point/variable allocation is exceeded, then a so-called 'fatal error' arises and the program terminates. The following example (from an output-file) documents a fatal error which occured while the program was trying to define a new point Z_3.

```
%% *** point(Z3){P,shift(2,5)}
%% ... FATAL ERROR: points used exceed allocation
%% ... use the PointNumber() command to increase allocation
```

4.5 Scalar variables

Numeric scalar variables are defined using the `variable(`*name*`){`*value*`}` command. The *value* in the braces can be either a numeric (e.g. 4.32; 2.6E-2), a variable name (e.g. r3), or two points (e.g. AB—meaning the Pythagorean distance between the two points A, B). Thus the command `variable(r3){20}` allocates 20 to the variable `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 `variable*` command.

However, one must be careful not to mistake variables for points, and vice versa, since they both have the same name structure. When both are necessary, using upper case for points and lower case for variables is a convenient strategy. If you inadvertently use a point instead of a variable mathsPIC will generate an appropriate error message (see the last example in Section 5.1).

4.5.1 Arithmetic

Variables can be manipulated using the commands `advance`, `multiply`, `divide` and `mod`. The format is similar to that of the `point` commands. Thus the command `variable(a1){a,advance(-r3)}` makes the variable `a1` assume the value `a-r3` (i.e. $a_1 = a - r_3$). Note that although mathsPIC allows the use of the negative sign with variables (e.g. `-5.34`, `-r3`), the positive sign is *not* allowed. Examples of valid `variable` commands are given in Section 6.

4.5.2 Scientific notation

When numbers are either very small or very big they are generally written using the usual computer 'E' format of so-called 'scientific notation', for example, 1.243E–9 etc. mathsPIC allows the use of scientific notation in commands wherever decimals are normally allowed, as follows.

```
variable(p2){3.78E3}
point(Z3){J,shift(1.3E-1,6)}
```

Unfortunately, TEX does not process numeric values in scientific notation, and will generate an error message whenever it finds the letter `E` as part of a number. Consequently mathsPIC converts all numbers which appear within PICTEX commands in the output-file into conventional decimal format (place-value notation). If the value is less than 1.0E–7 (0.0000001) then mathsPIC simply converts the value to zero.

However, since this convertion requires additional checking, mathsPIC does not bother to do this when it outputs data to the end of a commented line. For example, in the following extract from an output-file the variables s_4 and s_5 appear in scientific notation when in commented lines, but in decimal notation in the PICTEX `\put...` commands (see the line for point J_{45}). Note also that in this example the y-coordinate of J_{45} appears as zero in the `\put...` command (since `s5` is less than 0.0000001), but as `2.3E-8` in the `%%variable(s5){...` line.

```
%% variable(r){23}  ( 23 )
%% variable(s1){r,divide(100000)}  ( .00023 )
%% variable(s2){r,divide(1000000)}  ( .000023 )
%% variable(s3){r,divide(10000000)}  ( .0000023 )
%% variable(s4){r,divide(100000000)}  ( 2.3E-7 )
%% variable(s5){r,divide(1000000000)}  ( 2.3E-8 )
%% point(J12){s1,s2}
%% point(J23){s2,s3}
%% point(J34){s3,s4}
%% point(J45){s4,s5}
%% drawpoint(J12 J23 J34 J45)
\put {$\bullet$} at  .00023   .000023   %% J12
\put {$\bullet$} at  .000023   .0000023   %% J23
\put {$\bullet$} at  .0000023   .00000023   %% J34
\put {$\bullet$} at  .00000023   0   %% J45
```

4.6 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 segment 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 square-bracket option. For example, the following command draws a line a distance d units from point A *towards* point B.

```
drawline(AB)[d]
```

Note that in this particular command the order of the points AB and the sign of the distance `[d]` are important. For example, the following command will draw a line a distance d units from point B *away from* point A.

```
drawline(BA)[-d]
```

Since the PICTEX `\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 (see also `Pictex2.sty` in Section 9.3).

However, there are some occasions when it is necessary to draw using the `\plot` command even for horizontal and vertical lines (eg. when using `\setdots` with larger than usual dots), in which case the `drawline*` command (disables the use of `\putrule`) should be used.

4.6.1 Line thickness

PICTEX draws lines using two different methods depending on whether the lines are (a) horizontal or vertical, (b) any other orientation. 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 PICTEX `\putrule` command[8] and consequently the thickness of such lines is controlled by the PICTEX `\linethickness` command (the default line-thickness is 0.4pt). For example, the following PICTEX command changes the thickness to 1pt.

```
\linethickness=1pt
```

The `\linethickness` command is also useful for adjusting the thickness of graph axes, tick marks. For example, the following commands are used in the code for Figure 14 to draw thick axes.

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

Note also that the PICTEX `\linethickness` command can also be reset to its default value (0.4pt) by the PICTEX `\normalgraphs` command, which resets all PICTEX graph-drawing parameters to their default values including `\linethickness` (see Section 6.3).

Other lines and curves

All other lines (non-horizontal non-vertical) and curves are drawn using the PICTEX `\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 PICTEX `\setplotsymbol` command, the default size of dot being `{\tiny .}`. Larger dots therefore generate thicker lines. For example, the following PICTEX command sets the dot to a larger size.

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

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 `\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({\large .})
\plot 5  5   10  10 / %% AB
\setplotsymbol({\tiny .})
```

[8] Note that the PICTEX `\putrule` command employs the TEX and LATEX `\rule` command, and so is only used for horizontal and vertical lines.

Recommendations

Since PiCTeX 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 2 for a 10-point font (note that the default sizes are 0.4-point and `\tiny`).

Table 2: Equivalent PiCTeX commands for a 10-point font

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

The author's experience is that although mathsPIC does have commands for drawing thick lines and thick arrows, it is generally better to control the line-thickness separately using the two PiCTeX commands shown in Table 2, and to make separate macros where necessary. 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)
```

4.7 Text

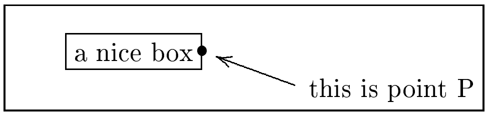
Text is typeset using the `text` command and, by default, is centered both horizontally and vertically at a defined point. For example, the triangle symbol $\triangle$ would be placed at the point Z using the command `text($\triangle$){Z}`.

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.

```
variable(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* PiCTeX 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 PiCTeX manual (see Section 9.5). For example in the diagram below, the text box $\boxed{\texttt{a nice box}}$ 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]
```



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.

5 Error-messages

A certain amount of syntax checking is performed by mathsPIC. Error-messages appear in the output-file (`.mt` file) and also in the log-file (`.mlg` file).

5.1 Output-file

A line containing an error is prefixed by `***`; the associated error-message appears on the next line and is prefixed by dots (`...`). 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 are where the mathsPIC commands are written incorrectly (e.g. missing bracket). A typical example might be in a `draw` command if a point has not been previously defined resulting in a 'point-name' error as follows.

```
%% *** drawTriangle(T1T6T3)
%% ... ?syntax error:  point T6 is not defined
```

If the maximum point/variable allocation (default is 50) is exceeded, then a so-called 'fatal error' arises and the program terminates. In the following example, a fatal error occured while the program was trying to define a new point Z_3.

```
%% *** point(Z3){P,shift(2,5)}
%% ... FATAL ERROR: points used exceed allocation
%% ... use the PointNumber() command to increase allocation
```

Some other examples of error-messages are as follows.

```
%% *** point(a2{5,6}
%% ... syntax error:  extra or missing ()
%%
%% *** point(B2){A,shift(,6)}
%% ... syntax error:  missing X coordinate in shift()
%%
%% *** point(z2){}
%% ... runtime error:  ?syntax error
%%
%% *** pointt(z4){2,3}
%% ... ?syntax error: statement not recognised
%%
%% *** drawLine(AB)
%% ... ?syntax error: points A and B are the same
%%
%% *** variable(d){j, advance(4)}
%% ... ?syntax error: j is a Point not a Variable
```

5.2 Log-file

mathsPIC outputs a log-file (`.mlg` file) which contains details of all errors, relevant line numbers and file names (e.g. <myfile.m>). The format was designed to match that of a standard TEX log-file in order to be compatible with commonly used error-checking utilities. Part of a typical log-file is as follows.

```
mathsPIC log-file    = myfile.mlg
mathsPIC input-file  = myfile.m
mathsPIC output-file = myfile.mt
```

```
! syntax error: extra or missing () bracket
l.9  point(A{10,5}
<myfile.m>

! ?syntax error:  point A is not defined
l.13  drawpoint(ABC)
<myfile.m>

! syntax error: point A is not defined
l.22  point(P){A,shift(5,0)}
<myfile.m>

! start of inputfile <newfile.dat>

! syntax error: point X is not defined
l.7  point(Q){X,shift(5,0)}
<newfile.dat>
...
...
```

6 Commands

6.1 MathsPIC commands

Although mathsPIC commands are *not* case sensitive, it is recommended (to avoid confusion) that points are represented using upper case letters, and variables are represented using lower case letters.

The parameters of mathsPIC commands are either strings (any legitimate TEX or LATEX commands or characters which are suitable for an `\hbox` are allowed), point-names (e.g. A,B,C), or scalar quantities. Where appropriate, MathsPIC allows scalar quantities in commands to be represented by either a numeric value (e.g. 0.432, or 4.32E-1), a variable name (e.g. r2), or the Pythagorean distance between two points (e.g. AB). For example, the structure of a `DrawCircle(`*`center,radius`*`)` command is quite flexible, as follows.

```
drawCircle(C,4.32)
drawCircle(C,3.6E-1)
drawCircle(C,r2)
drawcircle(C,AB)
```

The various parameters which control the thickness of lines, arrows and curves drawn by PICTEX are described in Section 4.6.1.

- `\...`   `\␣...`

  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.

  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 LATEX 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.

- `Arrowshape(`*`L, B, C`*`)`

  This command defines the shape of an arrowhead, and allows different arrowheads to be customised (see Section 7.3 for details). The default arrow shape is equivalent to the command

`Arrowshape(2,30,40)`. This default arrowhead shape can be reset using `default` as the argument, as shown in the following example.

```
Arrowshape(4,30,60)
drawArrow(AB)
ArrowShape(default)
```

- `DrawAngleArc{angle(), radius(), internal, clockwise}`

  This command draws an arc in the specified angle, a distance *radius* from the angle. The angle is either *internal* ($\leq 180°$) or *external* ($\geq 180°$). 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.

  For example, the following command draws a clockwise arc, radius 3 units, in the external angle ABC, from line AB to line BC.

  ```
  DrawAngleArc{angle(ABC), radius(3), external, clockwise}
  ```

- `DrawAngleArrow{angle(), radius(), internal, clockwise}`

  This command draws an arrow in the specified angle, a distance *radius* from the angle. The angle is either *internal* ($\leq 180°$) or *external* ($\geq 180°$). 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 arrow shape for the angleArrow is currently fixed, and cannot be changed using the `ArrowShape` command for straight arrows (the author is happy to make this more flexible if necessary).

  For example, the following command draws a clockwise arrow, radius 3 units, in the internal angle ABC from line AB to line BC.

  ```
  drawAngleArrow{angle(ABC), radius(3), internal, clockwise}
  ```

- `drawArrow(`*`AB`*`)` `drawArrow(`*`AB`*`)[d]` `drawThickarrow(`*`AB,CD,FG, ...`*`)`

  These commands draw arrows (or thick arrows) joining two points. The direction of the arrow is in the point order specified. The shape of the arrowhead is controlled by the `ArrowShape` command (see Section 7.3). Parameters which control line-thickness are described in Section 4.6.1

  ```
  drawArrow(AB)
  drawArrow(FG, HJ)
  drawThickarrow(BC)
  drawThickarrow(PQ, RS)
  ```

  An arrow can also be drawn a specified distance from one point towards another point, using a square-bracket option. For example, the following command draws an arrow a distance d units from point A towards point B.

  ```
  drawArrow(AB)[d]
  ```

- `DrawCircle(`*`center, radius`*`)`

  This command draws a circle defined by its radius and the point-name of its centre. The radius argument can be either a decimal, a variable, or the Pythagorean length of a line (e.g. AB). The following example draws a circle centre C_2, and radius 5 units.

```
drawCircle(C2,5)
```

In the following example, the radius argument is the variable `r2`.

```
drawCircle(C2,r2)
```

In the following example, the radius argument is the Pythagorean length of the line AB.

```
drawCircle(C2,AB)
```

Note that PICTEX draws circles with the `\circulararc` command, using a radius equivalent to the distance from the center 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 (see Section 4.3.2).

- `DrawCircumcircle(`ABC`)`

  This command draws the circumcircle of the triangle defined by the points A, B, C.

- `DrawCurve(`*ABC*`)`

  This command draws a smooth quadratic curve through three points A, B, C in the point order specified. At present it uses the PICTEX `\plot` command, and so it is only recommended for three points (for more than three points the resulting curve is unlikely to be 'smooth'). This command will be upgraded in the next version to give a smooth curve for 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).

- `DrawExcircle(`ABC`, `AB`)[`δr`]`

  This command draws the excircle touching side AB of the triangle ABC. The optional argument $[\delta r]$ is a fine adjustment to the radius of the excircle circle to facilitate drawing the figure in certain circumstances.

  In the following two examples, the first draws the true excircle, while the second draws the excircle with its radius reduced by 0.2 units.

```
drawExcircle(T1T2T3, T1T2)
drawExcircle(T1T2T3, T1T2)[-0.2]
```

- `DrawIncircle(`ABC`)[`δr`]`

  This command draws the incircle of a triangle. The optional argument $[\delta r]$ is a fine adjustment to the radius of the incircle circle to facilitate drawing the figure.

  In the following two examples, the first draws the true incircle, while the second draws the incircle with its radius reduced by 0.2 units.

```
drawIncircle(T1T2T3)
drawIncircle(T1T2T3)[-0.2]
```

- `drawline(`*ABCD...*`)` `drawline(`*AB*`)[6]` `drawline(`*AB,CD,EFG, ...*`)`

  `drawline*(`*ABCD...*`)`

  This command draws a line joining two or more points. Lines are drawn in the order specified. Lines are not drawn within line-free zones of points.

  Parameters which control line-thickness are described in Section 4.6.1.

```
drawline(AB)
drawline(BCDE)
drawline(FG, HJK, PQRST)
```

A line can also be drawn a specified distance from one point towards another point, using a square-bracket option. For example, the following command draws a line a distance d units from point A towards point B.

```
drawline(AB)[d]
```

Note that some restrictions are associated with the square bracket option: (a) it can only be used when drawing a *single* line; (b) a line-free zone associated with the second point will be ignored, (c) the order of the points AB is important—the line is drawn from the first point *towards* the second point (see Figure 11).

Note that the `drawline` command uses the PICTEX `\putrule` command for horizontal and vertical lines, and the `\plot` command for all other lines. However, the use of the `\putrule` command is disabled when using the mathsPIC `drawline*` command, which uses only the `\plot` command for all lines. Since the `\plot` command should be used when using larger than normal dots (`\setdots`) for horizontal or vertical lines, then this is achieved with the `drawline*` command.

- `DrawPerpendicular(`*point,line*`)[foot]`

  This command draws the perpendicular from *point* to the *line*. If the option [foot] is specified, then this command also draws the current `pointsymbol()` character (default point-symbol is `$\bullet$`) at the point where the perpendicular joins the line. The following example draws the perpendicular from the point P to the line AB.

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

- `drawpoint(`*name*`)` `drawpoint(`*name1 name2 ...*` )`

  This command draws the point-symbol at the point-location. Commas must not be used to separate points. The default point-symbol is `$\bullet$`, unless an optional point-symbol (or string of characters) was specified in the associated `point` command. If the `[circle]` option was used with the `point` command *without* a radius parameter, then the `$\circ$` symbol will be used (see the `point` command).

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

- `DrawRightangle(`*angle,symbolsize*`)`

  This command draws the standard right-angle symbol in the internal angle specified. The following example draws a right-angle symbol (side 3 units) in the internal angle ABC.

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

- `drawThickline(`*ABCD...*`)` `drawThickline(`*AB*`)[d]` `drawThickline(`*AB,CD,EFG, ...*`)`

  This command draws a thick line joining two or more points in the point-order specified. Lines are not draw within line-free zones associated with points.

  Parameters which control line-thickness are described in Section 4.6.1.

```
drawThickline(AB)
drawThickline(BCDE)
drawThickline(FG, HIJ)
```

An arrow can also be drawn a specified distance from one point towards another point, using a square-bracket option. For example, the following command draws an arrow a distance d units from point A towards point$B$.

```
drawThickArrow(AB)[d]
```

- `DrawTriangle(`*ABC*`)`

  This command draws the triangle defined by the points A, B, C. Note that `drawTriangle(ABC)` is equivalent to `drawLine(ABCA)`. The `drawTriangle` command checks that the area is not zero, and generates an error-message if it is (note that this area check is *not* made when using the equivalent `drawLine(ABCA)` command to draw a triangle).

- `inputFile(`*mathspic.doc*`)[`*loop*`]`   `inputFile*(`*pictex.doc*`)`

  The `inputFile` command inputs a plain text file containing mathsPIC commands. Optionally, the file can be input [*loop*] times, in which case this command functions like a DO–LOOP. For example, the following command inputs the file `newfile.dat` 4 times in succession.

```
inputFile(newfile.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 PICTEX 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)`.

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

Note that the `inputfile*` command has no [*loop*] option.

- `paper{units( ),xrange( ),yrange( ),axes( ),ticks( )}`

  `units(`*unit*`)` or `units(`*xunit,yunit*`)`

  `xrange(`*xlow,xhigh*`)`

  `yrange(`*ylow,yhigh*`)`

  `axes(`*XYLRTB*`)`

  `ticks(`*x,y*`)`

  The `paper{..}` command defines a plot area with optional axes and tick-marks (see Section 4.3.1).

  For example, the following statement sets up a rectangular drawing area 50mm x 50mm with axes on the `Left` (y-axis) and `Bottom` (x-axis), and tick marks at 10mm intervals.

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

All combinations of the axis-codes (XYLRTB) are allowed, and a * following an axis-code (e.g. L*) stops ticks being drawn 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)`, or `units(cm,2mm)`. If there is no digit associated with a unit (e.g. mm), then mathsPIC actions the command as `1mm`. If only a single unit is specified (e.g. `units(cm)`) then mathsPIC automatically makes this the unit scale for *both* axes. If the `unit` option is omitted PICTEX will use the last defined units (within the same picture environment), the default units being *xunit*=1pt and *yunit*=1pt (see the PICTEX Manual, page 3; Wichura, 1992).

- `point(`*name*`){...}[`*symbol, linefree radius*`]`

  `point*(`*name*`){...}[`*symbol, linefree radius*`]`

The `point` command is used to define points. Note that a point-name *must* begin with a *single* letter (either upper or lower case), and may have up to a *maximum* of two following digits.

The default point-symbol is the `$\bullet$`. An optional alternative point-symbol (or string of characters) can be specified within square brackets e.g. `[$\triangle$]` (see Section 4.4).

By default lines are drawn to the point location. However, the radius of an optional line-free zone can be specified within an optional square bracket e.g. `[$\triangle$,4]` (see Section 4.4). If a line-free radius is specified without a point-symbol, then the preceding comma must still be included; for example `[,5]`.

An optional '`circle`' can also be used as the *symbol*. If a radius value is also included, e.g. `[circle, 5]` then a circle will be drawn having the specified radius (the radius value will also be assumed to be the line-free radius). If no radius value is specified, e.g. `[circle]`, then the small circle symbol (`$\circ$`) is used instead (and line-free radius set to zero).

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, etc.

The `point*` command re-allocates new parameters (e.g. coordinates, point-symbol, or line-free radius) to a previously defined point-name.

Some examples of valid point commands are as follows.

```
point(A){5,5}
point(B2){22,46}[circle]
point(B2){22,46}[circle,5]
point(C32){3,8}[$\Box$]
point(D1){20,2}[$\Box$,4]
point(D10){20,5}[,4]
point(D2){B2, shift(5,5)}
point(D3){D2, polar(6,32 deg)}
point(D4){D2, polar(6,1.2 rad)}
point(D2){intersection(AB,CD)}
point(F){PointOnLine(AB,5.3)}
point(G){perpendicular(P,AB)}
point(G2){Q, rotate(P, 23 deg)}
point(H){circumcircleCenter(ABC)}
point(J){incircleCenter(ABC)}
point(K){excircleCenter(ABC,BC)}
point*(A){6,3}
point*(B){B, shift(5,0)}
point*(C){C}[,6]
```

- PointNumber(n)

This command allows the array space used for holding the total number of points and variables (combined) to be increased (or decreased). This statement must therefore be placed before any point statements, and so is best used right at the beginning of the mathsPIC file. The default number is set at 50 points.

```
PointNumber(55)
```

If the total number of points and variables combined exceeds the allocated number then a 'fatal error' is triggered, and the program terminates (see Section 4.4.5).

- PointSymbol(*symbol, linefree radius*)

This command allows the default point-symbol (`$\bullet$`, line-free radius zero) to be changed. The syntax of this command mirrors exactly that of the square-bracket option associated with the `point` command. The PointSymbol command is particularly useful where a set of points uses the same point-symbol (Figure 16), and also when drawing graphs (Figure 15).

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

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

Note that the `PointSymbol` command only influences subsequent `point` commands. For example, the following commands will make points A and B be drawn as circles having radius of 5 units.

```
variable(r){5}
PointSymbol(circle,r)
point(A){5,5}
point(B){10,12}
```

Note that the optional square bracket of the `point` command overrides the PointSymbol command. The point-symbol can be reset to the default $\bullet$ (`$\bullet$`) using the command `PointSymbol(default)`.

- show ...

This command forces mathsPIC to return the value of a calculation or specified parameter; for example, the value of a particular angle, or the length of a line. The result is shown in the output-file as a commented line. This allows mathsPIC commands to be adjusted in the light of calculations.

```
show(B)          %% for both Points and Variables
show(r)          %% for both Points and Variables
showLength(AB)
showIntersection(AB,CD)
showAngle(ABC)
showArea(ABC)
```

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

```
%% show(B)             Point (22.12432, 18.96723)
%% show(r)             Variable (8.25367)
%% showLength(AB)             25.35746
%% showIntersection(AB,CD)   (15.13245, 22.74723)
%% showAngle(ABC)             39.35746 degrees (0.68691 radians)
%% showArea(ABC)              54.54267
```

- `text(`*string*`){`*x,y*`}[`*position*`]`

  `text(`*string*`){`*pointname, adjustment*`}[`*position*`]`

  where the *adjustment* option is either `shift(`dx`,`dy`)` or `polar(`r`,`θ`)`.

  This command puts the given text-string either at the named point, or with a displacement specified by the optional `shift(`dx`,`dy`)` or `polar(`r`,`θ`)` parameters. By default the text is centered vertically and horizontally at the specified point.

  Optionally, text can be placed relative to a point using appropriate combinations of the PICTEX *position* options `l t r B b` to align the **l**eft edge, **r**ight edge, **t**op edge, **B**aseline, **b**ottom edge respectively of the text box with the point-location (see the PICTEX 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 centered vertically at the point P, using `text(This is point P){P}[r]`.

  The `text` command can be used to label points, or to simply position text in a diagram, as shown in the following examples.

```
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, polar(5, 1.34 rad)}
text(\framebox{$Z5$}){Z3}
```

- `variable(`*name*`){`*value*`}`

  `variable*(`*name*`){`*value*`}`

  The `variable` command is used to define scalar variables (see Section 4.5). Note that a variable-name *must* begin with a *single* letter (either upper or lower case), and may have up to a *maximum* of two following digits. Variables and points have the same name structure, and a variable cannot have the same name as a point. The scalar argument in the braces can be either a numeric value, another variable, or two points (implying the distance between the two points).

  New values can be re-allocated to existing variable-names using the equivalent `variable*` command; thus the variable `r3` can be re-allocated the value 15 using the command `variable*(r3){15}` (if the variable-name has not been used before then mathsPIC will generate an error message).

  Variables can be manipulated using the commands `advance`, `multiply`, `divide` and `mod`.

  The following are examples of valid `variable` commands.

```
variable(r3){20}
variable(z3){3.46E-2}
variable(r4){r1}
variable(r5){AB}
variable*(r5){6}
variable(x1){w, advance(-6)}
variable(x2){w2, divide(-p)}
variable(x3){AB, multiply(2)}
variable*(x1){x1, mod(3)}
variable(g){angle(ABC)}
variable(g2){area(ABC)}
variable(x3){Xpoint(P3)}   % gets the x-coordinate of a point
variable(y3){Ypoint(P3)}   % gets the y-coordinate of a point
variable(p1){3.14159}      % use p1 to represent Pi
variable(e){2.71828}
```

6.2 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 are represented using lower case letters. Note that a leading \␣ is used for copying TEX or PICTEX lines verbatim, for example with macros or coordinates of data points etc.

```
\   4.5    6.3
\setdashes
\typeout{drawing circle C2 now}%
arrowShape(4,30,40)
arrowShape(default)
drawAngleArc{angle(ABC), radius(3), internal, clockwise}
drawAngleArrow{angle(ABC), radius(3), internal, clockwise}
drawArrow(AB)
drawArrow(AB)[d]           %% draw arrow distance d from A towards B
drawArrow(AB,CD)
drawCircle(J,12)
drawCircle(C,r2)
drawCircle(C,AB)           %% the radius is length of line AB
drawCircumcircle(ABC)
drawCurve(ABC)             %% best for only three points
drawExcircle(ABC)[-0.2]
drawIncircle(ABC)[-0.3]
drawLine(AB)
drawline*(AB, CDEF)     %% the * option disables use of \putrule
drawLine(AB)[d]         %% draw line distance d from A towards B
drawLine(ABCDE)
drawPerpendicular(P,AB)
drawPerpendicular(P,AB)[foot]     %% draws pointSymbol at the foot
drawPoint(B)
drawPoint(ABP1P2)
drawRightAngle(ABC, 4)
drawThickArrow(AB)
drawThickArrow(AB)[d]    %% draw thickarrow distance d from A towards B
drawThickArrow(AB,CD)
drawThickLine(PQ)
drawThickLine(RS,TU)
drawTriangle(ABC)
inputFile(mathspic.dat)
inputFile(mathspic.dat)[4]
inputfile*(fig2.dat)     %% disables mathsPIC processing of file
paper{units(1mm,3mm),xrange(-5,50),yrange(-5,50),axes(LR)}
paper{units(cm),xrange(0,5),yrange(0,9),axes(X*Y),ticks(1,1)}
point(D1){20,2}
point(D4){6.3,8.9534E-1}
point(P){x1,y1}
point(P){AB,PQ}
point(D2){20,2}[circle]
point(D3){20,3}[circle,5]
point(D4){20,4}[$\Box$,4]
point(D5){20,7}[,4]     %% set the line-free radius
point(E1){D1, shift(5,6)}[$\Box$]
point(E5){D1, shift(r2,6)}
point(E2){D2, polar(8,45)}     %% deg is the default
```

```
point(E2){D2, polar(8,45 deg)}
point(E6){D2, polar(r4,45 deg)}
point(E5){D2, polar(AB,45 deg)}[,5]
point(D32){midpoint(AB)}
point(D2){intersection(AB,CD)}
point(F){PointOnLine(AB,5.3)}
point(G){perpendicular(P,AB)}
point(G){perpendicular(P,AB)}
point(G2){Q, rotate(P, 23 deg)}  %% rotate Q about P by 23 deg
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}[,6]             %% change the line-free radius
pointNumber(67)          %% default is 50 points
pointSymbol($\odot$)
pointSymbol($\odot$,3)
pointSymbol(,4)
pointSymbol(default)
show(P3)
show(m)
showAngle(ABC)
showArea(ABC)
showIntersection(AB,CD)
showLength(AB)
text(P){5,7}
text($A$){A}
text($K$){K}[r]
text($B$){B, shift(5,5)}
text(\framebox{$C$}){C, polar(5,62 deg)}[Br]
variable(r1){15}
variable(z3){4.875E-2}
variable(d){AB}
variable(d2){d1}
variable(x1){w, advance(6)}
variable(x1){w, advance(-23.55)}
variable(x2){w2, divide(-p)}
variable(x3){AB, multiply(2)}
variable(g){angle(ABC)}
variable(g2){area(ABC)}
variable(x3{Xpoint(P3)}   % gets the x-coordinate of a point
variable(y3){Ypoint(P3)}  % gets the y-coordinate of a point
variable(p1){3.14159}     % use p1 to represent Pi
variable(e){2.71828}
variable*(r1){20}
variable*(x1){x1, mod(3)}
```

Note that the old `latex{...}` `pictex{...}` and `tex{...}` ‘wrapper’ commands (for copying their argument unchanged into the output file) have been superseded by the more convenient `\` and `\␣` commands, but are still available to allow downwards compatibility with earlier versions of mathsPIC.

6.3 Useful PICTEX commands

The following is a list of PICTEX 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. Note that there is a section on PICTEX in Alan Hoenig's recent book *TEX Unbound*[9] which also includes a brief list of commands. Note that where small angle brackets are shown (e.g. <) then these must be used exactly as shown.

```
\grid {cols} {rows}  %% eg \grid {5} {10}
\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)
\plotheading{..}        %% eg \plotheading{This is a quadratic curve}
\headingtoplotskip=1cm  %% separation between plotheading and 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  30 5   %% corners at Top-left Bottom-right
\inboundscheckon        %% restricts plotting to plotting area
\inboundscheckoff       %%
\normalgraphs           %% restores default values for graph parameters
\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, ie  a:b)
```

Note that PICTEX also has an excellent rotation facility. PICTEX 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 the 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
```

A PICTEX error-bar facility is also available by loading the file `errorbar.tex` (see Section 9.4). Note that PICTEX 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 PICTEX Manual (see Section 9.5), and are currently outside the scope of mathsPIC.

[9] Hoenig A (1998). *TEX Unbound: LATEX and TEX 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.

7 Examples

This section describes some practical examples of figures drawn using mathsPIC, together with the associated code. A collection of more complicated examples can be found in the the companion document `examples.tex` which is included in the mathsPIC package.

When drawing a new figure or diagram, the author finds it best to start with a graduated coordinate frame (see Figure 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. Finally (if the frame is not required), the frame should be moved close to the edge of the figure by adjusting the `xrange` and `yrange` parameters, in order to remove unnecessary surrounding white space, after which the the `axes` and `ticks` options of the `paper` command can be removed, ready for inserting into the document. The code can then be either pasted into the document directly, or kept as a separate file and then `\input` as required.

As regards scales and units, the author finds it most convenient to use the mm units and to keep the x and y scales the same whenever possible (i.e. use `paper{units(mm)...}`), since this allows easy scaling up and down after the figure has been finished (see Section 4.3.2 for details regarding the use of variables to control the units).

The appropriate LaTeX commands to `\input` the file and position the graphic on the page are described in Section 8.

7.1 Input- and output-files

The following example mathsPIC file (input-file) illustrates how some of these commands are used to draw Figure 4. Note that the dashed line AD is drawn after the PiCTeX `\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 TeX `\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 TeX symbols has to be determined by trial and error—see Table 1 (Section 4.4.3).

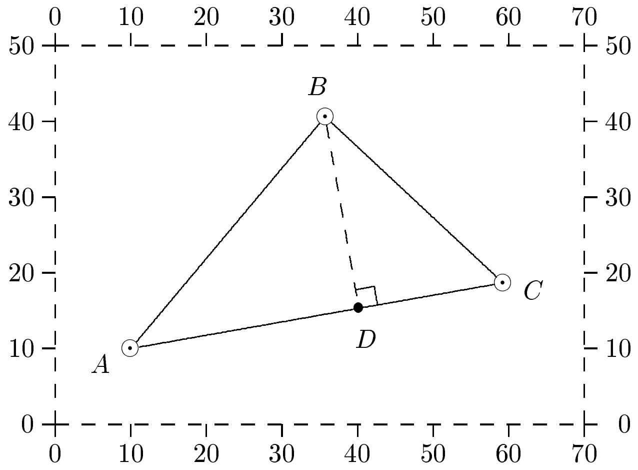


Figure 4:

```
% mPICm04.m  (Figure 4)
\documentclass[a4paper]{article}
\usepackage{pictexwd}
\begin{document}
```

```
%-------------------
\beginpicture
\setdashes
paper{units(mm),xrange(0,70),yrange(0,50),axes(LRTB),ticks(10,10)}
\setsolid
point(A){10,10}[$\odot$,1.2]     %% anchor point
point(B){A, polar(40, 50 deg)}[$\odot$,1.2]
point(C){A, polar(50, 10 deg)}[$\odot$,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)
\endpicture
%------------------
\end{document}
```

When the above file is processed by mathsPIC the output-file (`.mt` file) is as follows. Note how the PICTEX commands are preceded by their mathsPIC commands (commented out), some of which have additional information added (e.g. the coordinates of a derived point—see the line `%% point(D)...`). Being able to compare the mathsPIC and resulting PICTEX commands in the output-file is particularly useful when debugging. Note also how the `show` commands return the lengths AD, BC, and the area ABC.

```
% mPICm04.m  (Figure 4)
\documentclass[a4paper]{article}
\usepackage{pictexwd}
\begin{document}
%-------------------
\beginpicture
%% paper{units(mm),xrange(0,70),yrange(0,50),axes(LB),ticks(10,10)}
\setcoordinatesystem units <1mm, 1mm>
\setplotarea x from  0 to  70, y from  0 to  50
\axis left ticks numbered from  0 to  50 by  10  /
\axis bottom ticks numbered from  0 to  70 by  10  /
%% point(A){10,10}[$\odot$,1.2]  ( 10 , 10 )  %% anchor point
%% point(B){A,polar(40,50deg)}[$\odot$,1.2]  ( 35.7115 , 40.64178 )
%% point(C){A,polar(50,10deg)}[$\odot$,1.2]  ( 59.24039 , 18.68241 )
%% point(D){perpendicular(B,AC)}  ( 40.17626 , 15.32089 )
%% drawPoint(ABCD)
\put {$\odot$} at  10  10  %% A
\put {$\odot$} at  35.7115  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     34.94016  39.72252 /  %% AB
```

```
\plot   36.58879  39.82301     58.3631  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  34.7115  44.64178
%% text($A$){A,shift(-4,-2)}
\put {$A$} at  6  8
%% text($C$){C,shift(4,-1)}
\put {$C$} at  63.24039  17.68241
%% text($D$){D,shift(1,-4)}
\put {$D$} at  41.17626  11.32089
%% showLength(BD)    25.7115
%% showLength(AC)    50
%% showArea(ABC)    642.7875
\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 removing all the `%%` comment lines (using the CLEAN.EXE utility—see Section 3.2), as well as some of the header and footer lines which are not now required, as follows.

```
% mPICm04.pic (Figure 4)
%------------
\beginpicture
\setcoordinatesystem units < 1mm, 1mm>
\setplotarea x from  0 to  70, y from  0 to  50
\axis left ticks numbered from  0 to  50 by  10  /
\axis bottom ticks numbered from  0 to  70 by  10  /
\put {$\odot$} at  10  10  %% A
\put {$\odot$} at  35.7115  40.64178  %% B
\put {$\odot$} at  59.24039  18.68241  %% C
\put {$\bullet$} at  40.17626  15.32089  %% D
\plot   10.77135  10.91925     34.94016  39.72252 /  %% AB
\plot   36.58879  39.82301     58.3631  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  34.7115  44.64178
\put {$A$} at  6  8
\put {$C$} at  63.24039  17.68241
\put {$D$} at  41.17626  11.32089
\endpicture
%----------
```

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 CLEAN.EXE utility only removes lines having a leading `%%` pair of characters, so comment lines prefixed with only a single `%` character will remain.

7.2 Line modes

When drawing figures with both solid and dashed lines it is necessary to switch between the PICTEX commands `\setdashes` and `\setsolid` (see Section 6.3), as in the following code for drawing the rectangular box shown in Figure 5.

The default `\setdashes` gives alternating lines and spaces, each of width 5pt. More fancy dash-patterns (see Figure 5) can be easily generated using the PICTEX `\setdashpattern` command which defines the pattern cycle, and hence takes an even number of parameters. If a particular dash-pattern is to be used several times in a figure, then a defining macro may be useful, as follows.

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

Note that in Figure 5 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.

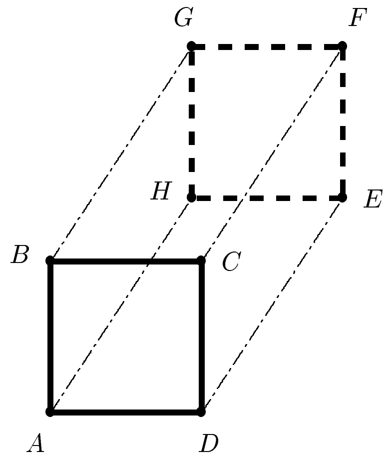


Figure 5:

```
%% mPICm05.m (Figure 5)
\beginpicture
paper{units(mm), xrange(0,50), yrange(0,62)}
variable(s){20}  % Sides front & back
variable(L){34}  % Length
variable(a2){56.6}  % Angle degrees
\def\fancydashes{\setdashpattern <6pt, 2pt, 1pt, 2pt>}
point(A){5,7}
point(B){A, polar(s,90)}
point(C){B, polar(s,0)}
point(D){A, polar(s,0 )}
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(ABCDA)
\setdashes
drawline(HGFEH)
\linethickness=0.4pt  \fancydashes
drawline(AH, BG, CF, DE)
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
```

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 standard PICTEX commands, the mathsPIC Arrowshape command makes this easier by allowing you to define the angle parameters (B and C) of the arrow-head directly (see box). The default arrowshape is equivalent to the following command

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

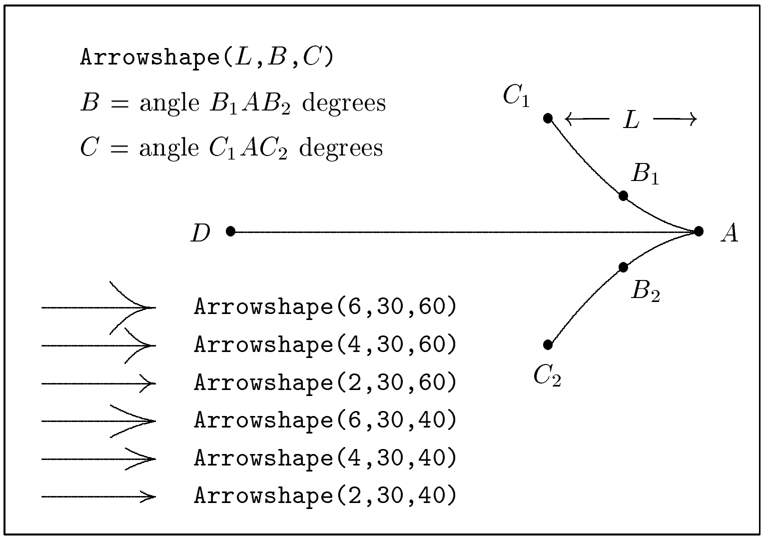


Figure 6:

If the arrowshape has been altered, it can be reset using the command ArrowShape(default). 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 and 8).

Arrows can also be used to link elements in a diagram, as shown in Figure 8. The right-hand diagram uses the drawArrow command; the small gap between the arrows and the letters P, Q, R, T

```
% mPICm07.m (Figure 7)
\beginpicture
paper{units(mm),xrange(5,45),yrange(5,45)}
point(A){30,30}
point(P){10,10}
point(B){30,10}
drawPoint(APB)
drawLine(APBA)
text($A$){A,shift(1,5)}
text($B$){B,shift(5,0)}
text($P$){P,shift(-5,0)}
drawAngleArrow{angle(BPA),radius(11),anticlockwise,internal}
text($\psi$){P,polar(7,22.5 deg)}
drawRightangle(ABP,2.5)
\endpicture
```

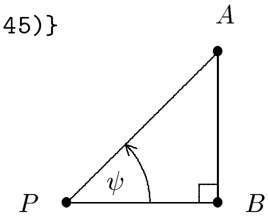
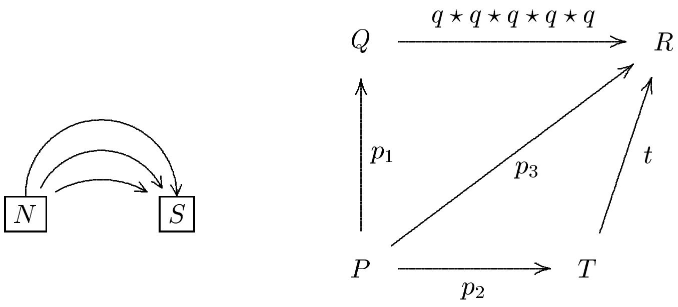


Figure 7:

being due to the 5 unit line-free radius associated with these points (see `mpicm08b.m`). The arrows are easily ‘stretched’ to accommodate their labels simply by adjusting the separation of the nodes using the `polar(`r`,`θ`)` commands (cf. Feruglio, 1994).



a. Circular arrows

b. Straight arrows

Figure 8:

```
%% mpicm08a.m (Figure 8a)
\beginpicture
paper{units(mm),xrange(10,40),yrange(0,45)}
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),clockwise,internal}
point(N1){N,shift(2,1)}
point(S1){S,shift(-2,1)}
point(Z1){Z,shift(0,-3)}
drawAngleArrow{angle(N1Z1S1),radius(N1Z1),clockwise,internal}
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),clockwise,internal}
\endpicture
```

```
%% mpicm08b.m (Figure 8b)
\beginpicture
paper{units(mm),xrange(0,45),yrange(0,45)}
point(P){5,10}[$P$,5]
point(Q){P,polar(30,90 deg)}[$Q$,5]
point(R){Q,polar(40,0 deg)}[$R$,5]
point(T){P,polar(30,0 deg)}[$T$,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 macro for the label
\newcommand{\q}{$q \star q \star q \star q \star q$}
text(\q){Q1,shift(-1,3)}
\endpicture
```

7.4 Circles

MathsPIC allows the point-symbol to be designated as a circle using the `[circle,r]` option 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 9. This construction greatly simplifies the drawing of directed graphs, trees and equivalent structures.

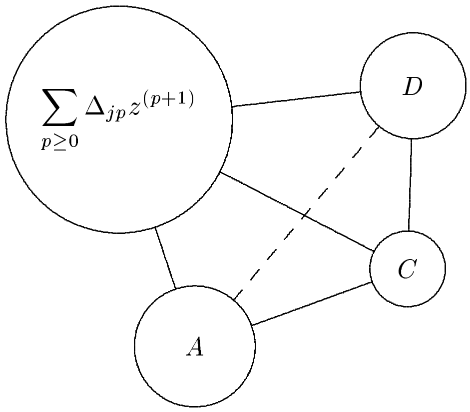


Figure 9:

```
%% mPICm09.m  (Figure 9)
\beginpicture
paper{units(mm),xrange(0,70),yrange(0,60)}
point(A){30,11}[circle,8]
point(B){A,shift(-10,30)}[circle,15]     %% big circle
point(C){A,polar(30,20 deg)}[circle,5]
point(D){A,polar(45,50 deg)}[circle,7]
drawPoint(ABCD)
drawLine(AB,AC,BC,BD,CD)
\setdashes
drawLine(AD)
text($A$){A}
%% use a macro for the formula
\newcommand{\formula}{%
\     $\displaystyle \sum_{p\ge0} \Delta_{jp} z^{(p+1)}$%
\     }%
text(\formula){B}
text($C$){C}
text($D$){D}
\endpicture
```

Note that in this particular case it is necessary to define the maths formula using `\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 10. Notice the use of the variable `r` for the radius of the circle (allocated using the command `variable(r){20}`), which then allows the use of `r` to define the radius in the `polar` commands for the points P, Q, S.

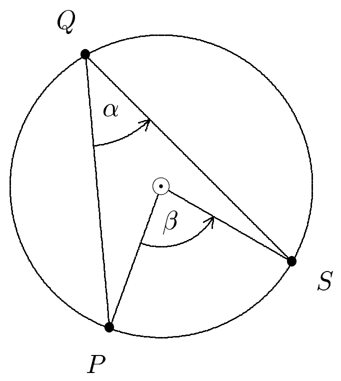


Figure 10:

```
%% mPICm10.m (Figure 10)
\beginpicture
paper{units(mm), xrange(5,55), yrange(5,55)}
point(C){30,30}[$\odot$,1.2]   %% center
variable(r){20}                %% radius
drawcircle(C,r)
point(P){C, polar(r,250 deg)}
point(Q){C, polar(r,120 deg)}
```

```
point(S){C, polar(r,-30 deg)}
drawpoint(CPQS)
drawline(PCSQP)
text($P$){P, polar(5,250 deg)}
text($Q$){Q, polar(5,120 deg)}
text($S$){S, polar(5,-30 deg)}
drawAngleArrow{angle(PCS), radius(8) anticlockwise internal}
text($\beta$){C, polar(5,285 deg)}
drawAngleArrow{angle(PQS), radius(12) anticlockwise internal}
text($\alpha$){Q, polar(8,-65 deg)}
show angle(PQS)     % alpha
show angle(PCS)     % beta
\endpicture
```

Note that the returned values in the output-file from the `showAngle` commands (see below) for Figure 10 indicate that β is twice α, as one would expect.

```
%% showAngle(PQS)  40 degrees / radians 0.6981316       % alpha
%% showAngle(PCS)  79.99999 degrees / radians 1.396263  % beta
```

MathsPIC offers a range of other circle commands (`drawIncircle`, `drawExcircle`, `drawCircumcircle`), specifically for geometry diagrams (cf. Cameron, 1992), as shown in Figure 11.

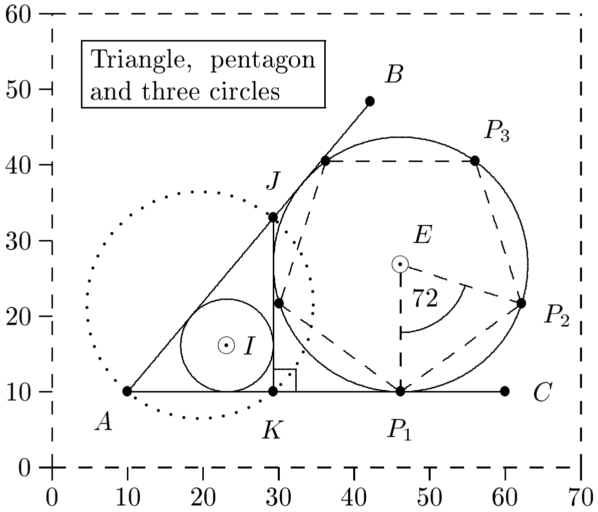


Figure 11:

```
%% mpicm11.m  (Figure 11)
\beginpicture
\setdashes
paper{units(mm),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)
point(I){IncircleCenter(AJK)}[$\odot$]
point(E){ExcircleCenter(AJK,JK)}[$\odot$,1.2]
point(P1){perpendicular(E,AC)}
variable(r){EP1}  %% radius of excircle
variable(d){72}   %% angle of pentagon (deg)
variable(a1){-90}
variable(a2){a1, advance(d)}
variable(a3){a2, advance(d)}
variable(a4){a3, advance(d)}
variable(a5){a4, advance(d)}
point(P2){E, polar(r,a2)}
point(P3){E, polar(r,a3)}
point(P4){E, polar(r,a4)}
point(P5){E, polar(r,a5)}
drawPoint(ABCJKIEP1P2P3P4P5)
\setplotsymbol({\tiny .})
\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}
variable(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)}
text($P_2$){P2,polar(s, a2)}
text($P_3$){P3,polar(s, a3)}
\endpicture
```

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. This is demonstrated in Figure 12, where 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)}`

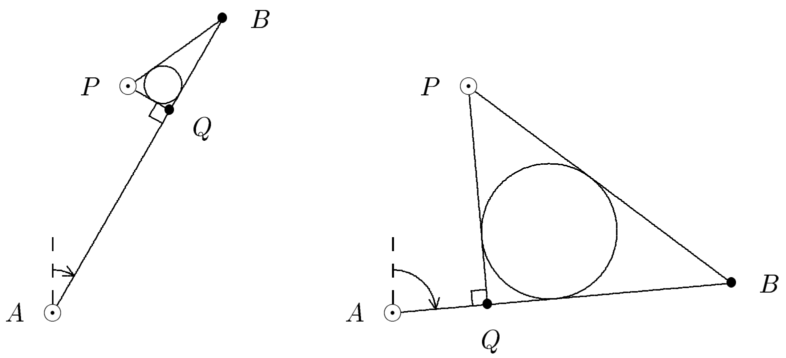


Figure 12: The mathsPIC code for the two diagrams differs *only* in the angle of the line AB as defined in the `point(B){...}` command (see `mPICm12.m`).

```
%% mPICm12.m (Figure 12)
\beginpicture
paper{units(mm),xrange(5,120),yrange(0,45)}
point(A){15,5}[$\odots$,1.2]
point(P){A,shift(10,30)}[$\odots$,1.2]
point(B){A,polar(45,60 deg)}
point(Q){perpendicular(P,AB)}   % from P to line 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}
%% now draw vertical line and angle
point(N){A,shift(0,12)}
\setdashes
drawLine(AN)
\setsolid
drawAngleArrow{angle(NAB),radius(7),clockwise,internal}
\endpicture
```

Note that in Figure 12 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 line AB in-line with the points PQ, using the command `point(S){pointOnLine(QP,-5)}`.

7.6 Inputting files and recursion

mathsPIC allows the recursive input of blocks of mathsPIC commands as files, using the `inputfile` command. In practice, this functions as a 'DO–LOOP' in a program. For example, the file `myfile.dat` would be input six times sequentially using the command `inputfile(myfile.dat)[6]`.

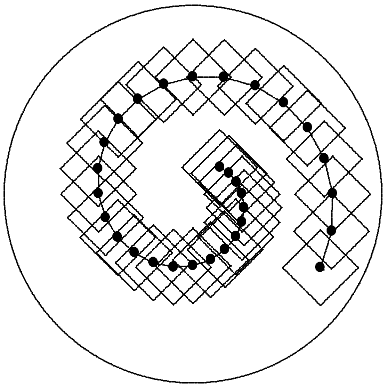


Figure 13:

Figure 13 was produced by the following code which inputs a small file of mathsPIC code (`mpicm13.dat`) recursively 30 times using the command `inputfile(mpicm13.dat)[30]`. Note the use of the `variable*` and `point*` commands and the `advance` operator in order to re-allocate variables and points recursively.

```
%% mpicm13.m (Figure 13)
%% spiral
\beginpicture
paper{units(mm), xrange(0,60), yrange(0,60)}% axes(LB), ticks(10,10)}
point(C){30,30}   % circle center
drawcircle(C,25)
variable(a){315}  % angle deg
variable(r){20}   % radius of spiral
variable(s){5}    % square, semi-diagonal
point(T){C,polar(r,330 deg)}
inputfile(mpicm13.dat)[30]
\endpicture
```

```
    %% mpicm13.dat  (Figure 13)
    %% spiral routine
    variable*(r){r,advance(-0.5)}   % let r = r - 0.5
    variable*(a){a,advance(15)}     % let a = a + 15 deg
    point*(P){C,polar(r,a deg)}
    drawpoint(P)
    drawline(TP)
    point*(T){P}      % let T = P
    %% make a square centered on P
    point*(Q1){P,polar(s,0)}
    point*(Q2){P,polar(s,90)}
    point*(Q3){P,polar(s,180)}
    point*(Q4){P,polar(s,270)}
    drawline(Q1Q2Q3Q4Q1)
```

7.6.1 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 PICTEX commands and/or points for plotting curves (see Section 6). For example, the following mathsPIC code (`mpicm14.m`) draws the quartic curve shown in Figure 14, by inputting in *verbatim* a datafile (`mpicm14.dat`) containing some PICTEX 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.

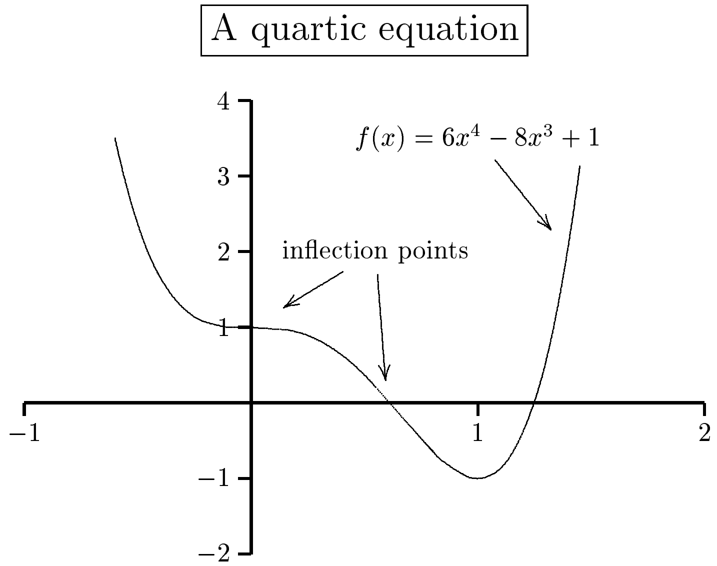


Figure 14:

```
%% mpicm14.m  (Figure 14)
\beginpicture
\linethickness=1pt       %% use a thick line for the axes
paper{units(3cm,cm),xrange(-1,2),yrange(-2,4),axes(XY),ticks(1,1)}
\linethickness=0.4pt     %% reset to default value
\headingtoplotskip=8mm
\plotheading{\fbox{\Large A quartic equation}}
inputfile*(mpicm14.dat)  %% input file containing data points for curve
variable(r){0.3}         %% define a line-free radius r = 0.3
point(E1){1,3.5}[,r]
text($f(x)=6x^4 - 8x^3 + 1$){E1}  % center the equation at E1
point(E2){1.4,2}[,r]
drawArrow(E1E2)
point(J1){0.55,2}[,r]
text(inflection points){J1}  %% center inflection text at J1
point(J2){0,1}[,r]
point(J3){0.6,0}[,r]
drawArrow(J1J2,J1J3)
\endpicture
```

The datafile for the curve is as follows. Note that PICTEX requires an *odd* number of pairs of data points to satisfy its curve-drawing algorithm.[10]

```
%% mpicm14.dat  (Figure 14)
%% quartic curve data (use an odd number of data points)
```

[10]See the PICTEX manual (Wichura, 1992).

```
\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
```

However, when there are only a few data points, it is somtimes more convenient just to plot the points separately and then draw connecting lines, as shown in Figure 15.

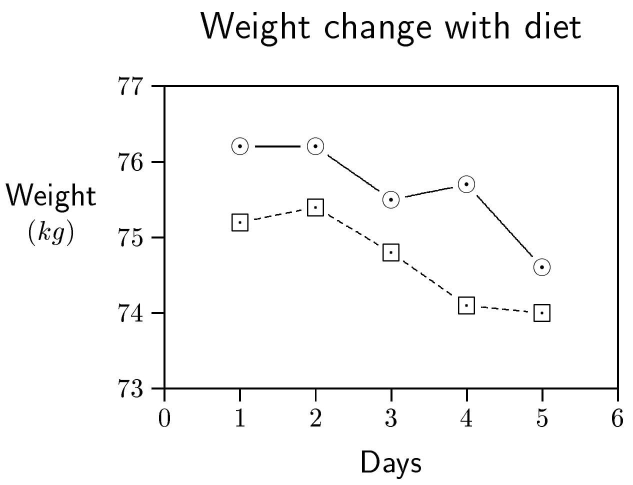


Figure 15:

```
%% mpicm15.m  (Figure 15)
\beginpicture
paper{units(cm),xrange(0,6),yrange(73,77),axes(LBT*R*),ticks(1,1)}
variable(r){0.2}  %% line-free radius
pointsymbol($\odot$,r)
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)
%
pointsymbol($\boxdot$,r)
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)
%
\plotheading{\textsf{\Large Weight change with diet}}
text(\shortstack{\textsf{\large Weight}\\($kg$)}){-1.5,75.3}
text(\textsf{\large Days}){3,72}
\endpicture
```

The `drawcurve` command can also be used for drawing smooth curves linking a number of points or touching lines. For example, Figure 16 shows a smooth curve touching a piecewise linear closed line[11], some of the points being constructed using a Bezier technique[12]. The smooth curve is drawn using the `drawcurve` command for successive three-point sequences.

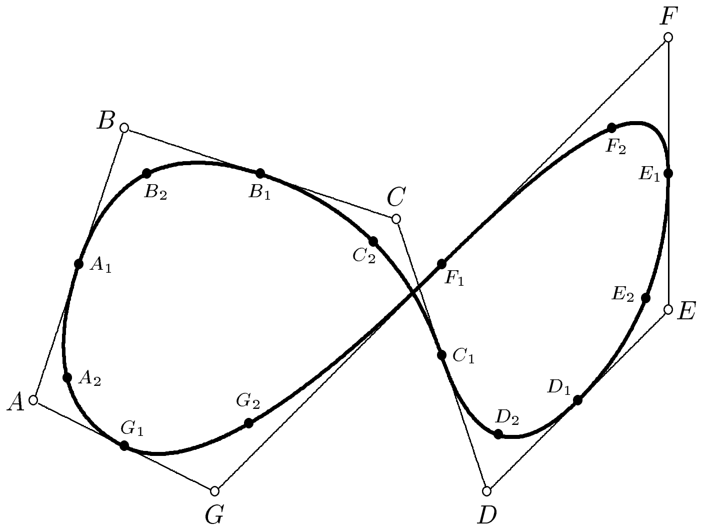


Figure 16: A smooth curve inscribed in the intersecting closed line $ABCDEFGA$

```
%% mpicm16.m  (Figure 16)
\beginpicture
variable(u){12}             %%  units = 12 mm
paper{units(u mm),xRange(0,9),yRange(-1,6)}
variable(r){0.7}            %% line-free radius of \circ = 0.7mm
variable*(r){r, divide(u)}  %% scale line-free radius 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}
```

[11]Figure 16 was constructed and drawn by Frantisek Chvala.
[12]see *The Metafont book* by DE Knuth, chapter 3 for details regarding Bezier curves.

```
pointsymbol(default)  %% restore $\bullet$ symbol for points
%
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)}
%
DrawPoints(ABCDEFG)
DrawPoints(A1B1C1D1E1F1G1)
DrawPoints(A2B2C2D2E2F2G2)
%
DrawLine(ABCDEFGA)
\setplotsymbol ({\Large.})
DrawCurve(A1B2B1)
DrawCurve(B1C2C1)
DrawCurve(C1D2D1)
DrawCurve(D1E2E1)
DrawCurve(E1F2F1)
DrawCurve(F1G2G1)
DrawCurve(G1A2A1)
%
Text($A$){A,shift(-.2,0)}
Text($B$){B,shift(-.2,.1)}
Text($C$){C,shift(0,.25)}
Text($D$){D,shift(0,-.25)}
Text($E$){E,shift(.2,0)}
Text($F$){F,shift(0,.25)}
Text($G$){G,shift(0,-.25)}
\scriptsize
Text($A_1$){A1,shift(.25,0)}
Text($B_1$){B1,shift(0,-.2)}
Text($C_1$){C1,shift(.25,0)}
Text($D_1$){D1,shift(-.2,.15)}
Text($E_1$){E1,shift(-.2,0)}
Text($F_1$){F1,shift(.15,-.15)}
Text($G_1$){G1,shift(.1,.2)}
```

```
Text($A_2$){A2,shift(.25,0)}
Text($B_2$){B2,shift(.1,-.2)}
Text($C_2$){C2,shift(-.1,-.15)}
Text($D_2$){D2,shift(.1,.2)}
Text($E_2$){E2,shift(-.25,.05)}
Text($F_2$){F2,shift(.05,-.2)}
Text($G_2$){G2,shift(0,.25)}
\endpicture
```

Note the technique used in the code for Figure 16, 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 (see also Section 4.3.2).

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

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

8 Positioning figures in a document

Once a diagram or figure has been finished it can be easily placed in a document either by including the PiCTeX code directly in the main document within the `\begin{figure}...\end{figure}` environment, or by `\input`ting 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 the LaTeX 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.

```
\begin{figure}[hbt]
\begin{center}
   \strut\input{myPIC.pic}
   \caption{...}
   \label{...}
\end{center}
\end{figure}
```

It is often useful when adjusting its position on the page, to initially place the Figure inside a frame in order to see the exact extent of the Figure, in which case 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` is useful, as follows:

```
\begin{figure}[hbt]
\begin{center}
  \strut\input{myPIC.pic}
  \parbox{10cm}{
      \vspace{5mm}
      \caption{...}
      \label{...}
      }
\end{center}
\end{figure}
```

When typesetting text and a figure side-by-side the following format for using two adjacent minipages works well—this was the construction used for displaying the program code and the associated Figure 7.

```
\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.

```
\begin{figure}[hbt]
\begin{center}
    %-------
    \noindent %\framebox{
    \begin{minipage}{3cm}
       \begin{center}
       %\vspace{...}  %% controls space above picture
       \input{mPICm08a.pic}
       %\vspace{...}  %% controls space between pict and caption
       a. Circular arrows
       \end{center}
    \end{minipage}
    %} %end of framebox
```

```
    %-----------------------
    \hspace{12mm} %% controls horiz space between figs
    % \bigskip\bigskip  %% use this to place figs vertically
    %----------------------
    %\framebox{
    \begin{minipage}{5cm}
       \begin{center}
       %\vspace{...}  %% controls space between pict and caption
       \input{mPICm08b.pic}
       %\vspace{...}  %% controls space between pict and caption
       b. Straight arrows
       \end{center}
    \end{minipage}
    %} %end of framebox
    %---------
    \caption{.....\hspace{7mm}}  % add \hspace{} to adjust horiz possn
\end{center}
\end{figure}
```

9 Downloading and installing PICTEX

PICTEX is an excellent small graphics package, freely available from CTAN and the usual TEX CD-ROM discs[13]. Download all the files in the following two directories, and place them where your TEX system can find them[14].

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

If you are a LATEX user, then use `\usepackage{pictexwd}`. If you are a plain TEX user, then use `\input pictexwd.tex` (see Section 4.2.1 and Section 9.2 for details).

Unfortunately, the PICTEX documentation is not available from CTAN—the PICTEX manual has to be purchased separately (see Section 9.5). However, mathsPIC users will mostly find this unnecessary, since those PICTEX commands which are particularly useful in conjunction with mathsPIC are described in the mathsPIC manual (Section 6.3), together with examples in the code for the various Figures.

9.1 The original files (1986)

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

The original PICTEX 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 TEX users the option of using the LATEX Picture macros `\line`, `\vector`, `\circle`, `\oval`, `\thicklines` and `\thinlines` (for syntax and use with PICTEX see Wichura, 1992).

Note that only two of the files are required for using with plain TEX, while three files are required when running LATEX.

```
latexpic.tex    11243 bytes  (latexpicobjs.tex  --for plain TeX only)
prepicte.tex     1293 bytes  (prepictex.tex     --for LaTeX)
pictex.tex     133388 bytes  (pictex.tex        --for LaTeX and plain TeX)
postpict.tex     1614 bytes  (postpictex.tex    --for LaTeX)
```

[13]Available from TEX user groups. Note that a particularly good 3-disk CTAN archive is published annually by the German TEX users group DANTE (*dante@dante.de*, `http://www.dante.de`).

[14]For EmTeX, this would be the directory `c:\emtex\texinput\pictex\`

The three files required for use with LaTeX need to be loaded (input) *in the order shown above.* Note that when using PiCTeX with LaTeX 2_ε it is necessary to redefine the LaTeX 2.09 command `\fiverm` (because PiCTeX was originally written for LaTeX 2.09). This is most easily done as follows.

```
\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 LaTeX 2_ε documents).

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

A significant problem with the original PiCTeX package was that it was very memory hungry. However, in 1994 this problem was overcome by a significant rewrite by Andreas Schrell (see Section 9.2).

9.2 The new updated files (1994)

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

In 1994 Andreas Schrell uploaded a set of updated PiCTeX files into the `CTAN: .../pictex/addon/` directory. One of these additional files (`pictexwd.tex`) is a replacement for the original (`pictex.tex`), and is extremely economic in its use of TeX'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`).

```
pictexwd.sty        416 bytes
pictexwd.tex     133232 bytes
picmore.tex        2952 bytes
piccorr.sty        4608 bytes
pictex.sty          311 bytes
```

- `pictexwd.sty`

  For LaTeX users. This replaces `pictex.sty`. It inputs `prepictex.tex`, `pictexwd.tex`, and `postpictex.tex`, as well as inputting `piccorr.sty` and `picmore.tex` if these are available.

- `pictexwd.tex`

  For TeX users.

- `picmore.tex`

  An extension for drawing impulse diagrams.

- `piccorr.sty`

  A correction for the original PiCTeX `\betweenarrows` command.

- `pictex.sty`

  The LaTeX style-option for loading the *original* PiCTeX files. Note that it also inputs `piccorr.sty` and `picmore.tex` if these are available.

All the necessary files required for running PiCTeX are automatically input in the correct order by `pictexwd.sty` (LaTeX users), or by `pictexwd.tex` (TeX users). Users of LaTeX 2_ε should include the following command in the preamble.

```
\usepackage{pictexwd}
```

Users of plain TeX need to include the following.

```
\input latexpic.tex
\input pictexwd.tex
```

Note that it is still necessary to download all the original PiCTeX files even when using Andreas Schrell's new files, as some of the original files are still required.

9.3 Pictex2.sty

`CTAN/tex-archive/macros/latex/contrib/supported/pictex2.sty`
`16418 bytes 09/05/1999`

William Park[15] has written a style option (`pictex2.sty`) which adds two useful commands to standard PICTEX, 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—see Section 4.6). These two commands are as follows.

- `\putanyline` command

  This command invokes the PICTEX `\putrule` command (instead of `\plot`) in cases where lines are either vertical or horizontal.

- `\setanyline` command

  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.

9.4 Errorbar.tex

`CTAN/tex-archive/graphics/pictex/errorbar.tex`
`3041 bytes 20/04/1988`

In 1988 Dirk Grunwald implemented the following errorbar command for PICTEX which draws error bars using a modified `\plot` command as follows.

- `\plotWithErrorBars` command

  This command, which is case sensitive, has the format

```
\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 `\crossbarlength=7pt`.

  To use this command, input the file `errorbar.tex` after all the PICTEX files.

9.5 The PICTEX Manual

You may wish to purchase *The PICTEX Manual* by Michael J. Wichura (The University of Chicago, Chicago, Illinois, USA; *wichur@galton.uchicago.edu*). This excellent booklet (version 1.1, third printing, March 1992; 85 pages) used to be available from TUG as Publication No. 6 in the TUG TEXniques series. Unfortunately, TUG (`http://www.tug.org/`) has now stopped publishing the manual, and *The PICTEX Manual* is currently only available from *Personal TEX Inc.* (*texsales@pctex.com* `http://www.pctex.com/`) at approximately $59 plus postage.

When you buy a copy of the manual you also receive a floppy disk containing PICTEX files. Unfortunately these are only the original 1986 files, and do not as yet include any of Andreas Schrell's 1994 files.

[15] `parkw@better.net`

10 Bug reports

The author would be grateful for any bug reports, constructive comments, and any ideas for improving this program. The author is grateful to Apostolos Syropoulos, Bob Schumacher, Frantisek Chvala, Orlando Rodriguez, Boris Kuselj, and others for testing this program, and for their many ideas and suggestions.

11 Perl version of mathsPIC

A Perl version of mathsPIC is currently in progress (Syropoulos and Nickalls, 2000). It will have a number of enhanced features, particularly with regard to handling of variables. The authors expect it to be first released in the Autumn of 2001.

12 History

- 2.1 (November, 2000):

  Revised the documentation; fixed numerous bugs; introduced the \ and \␣... commands to facilitate the use of TEX and LATEX commands and for copying lines verbatim. The following commands have also been added/extended.

```
drawCurve(ABC)
drawline(AB)[d]                 %% added the [] option
drawline*(AB)                   %% disables use of \putrule
point(S){Q, rotate(P, 23 deg)}  %% added the rotate() option
pointsymbol(char,lineFreeRadius) %% added the line-free radius option
```

- 1.9b (May 2000):

  Fixed a bug associated with the `drawAngleArc` and `drawAngleArrow` commands.

- 1.9a (January 2000):

  Added the options X and Y to the `axes` option in the `paper` command, which allows X and Y axes to be drawn. Also added the option of following any of the `axes` options by a * which prevents ticks being drawn on that particular axis.

- 1.8f (October 1999):

  Fixed problem with scientific notation; added current loop number indicator; added the following new commands:

```
variable(){area()}
variable(){angle()}
variable(){Xpoint()}
variable(){Ypoint()}
inputfile*()
PointSymbol(default)
```

- 1.7u (September 1999): First release.

13 References

- Cameron P. J. (1992). Geometric diagrams in LATEX. *TUGboat* **13** (No. 2), 215–216.
- Feruglio G. V. (1994). Typesetting commutative diagrams. *TUGboat* **15** (No. 4), 466–484.

- Nickalls RWD (1999a). MathsPIC: a filter program for use with PICTEX. *EuroTEX'99 Proceedings*; 197–210. (Heidelberg, Germany; August 1999).
- Nickalls RWD (1999b). MathsPIC: a filter program for use with PICTEX. *Eutupon*[16] [the Greek TEX Friends' journal]; No. 3 (October, 1999), 33–49 (English). [This is an updated version of Nickalls (1999a)].
- Salomon D. (1992). Arrows for technical drawing. *TUGboat* **13** (No. 2), 146–149.
- Syropoulos A. and Nickalls R. W. D. (2000). A PERL porting of the mathsPIC graphics package. *TUG2000 Conference* (Oxford, UK); August 13–16, 2000.
- Wichura M. J. (1992). The PICTEX manual. Pub: Personal TEX Inc., 12 Madrona Avenue, Mill Valley, CA 94941, USA. *texsales@pctex.com* `http://www.pctex.com`.

———

[16] The editor is Apostolos Syropoulos (email: *apostolo@obelix.ee.duth.gr*).