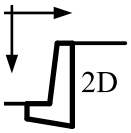
Vector graphics with *Drawj2d*



Program documentation
&
function reference

Adrian Vontobel

6th January 2017

Contents

1 The program Drawj2D

1.1 Purpose

Drawj2d creates technical line drawings using a descriptive language. It is implemented in java, thus requires the Java Runtime Environment JRE (1.7 or above).

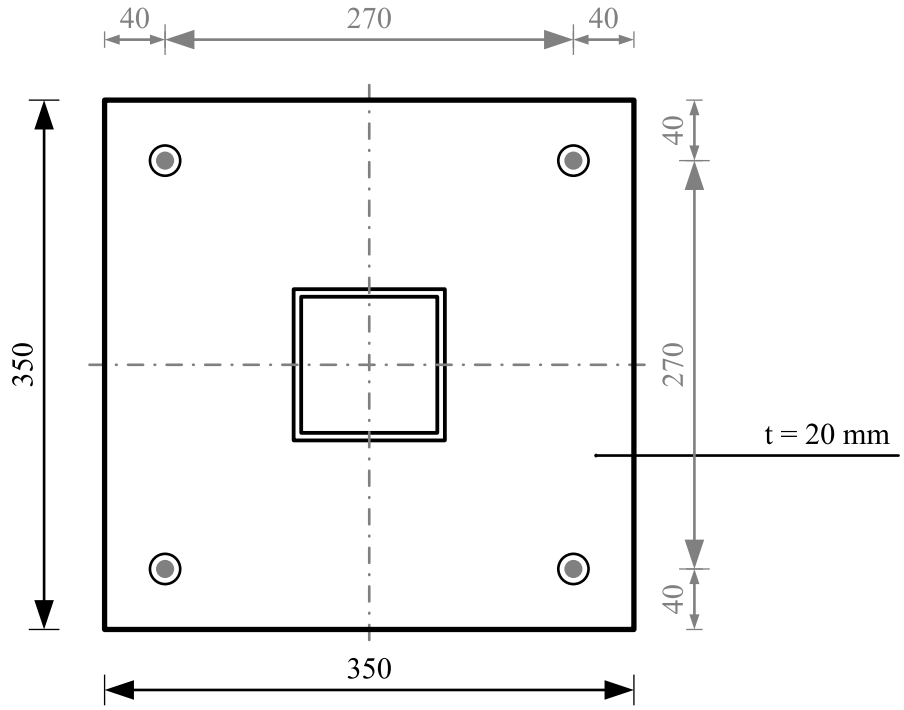


Figure 1: Drawing

The example in Figure 1 shows a drawing created by Drawj2d. The corresponding input file is printed in section 6.

```
drawj2d --type pdf --width 150 --height 120 --center drawing.hcl
```

1.2 Features

- easy to learn vector graphics language
  Descriptive vector graphics language, easy (tcl-like) syntax.
- reusable drawings
  Drawings can be parametrised using variables.
- draw to scale
  1:50, 1:100, 1:20

- different output formats
  Vector formats: pdf, svg, eps, emf. Bitmap formats png, bmp. Intermediate format: tikz.
- platform independent
  Drawj2d runs on every platform which runs *Java*: Linux, Windows, Mac OS X, Unix
- programming language
  Built in tcl-like scripting language allows advanced features and extensibility.
- viewer
  Built in viewer for the Drawj2d input files (*.hcl).

Additional functionality

- Yacas plot data
  Drawj2d can draw the 2D plot data generated by the computer algebra system Yacas (see chapter 5).
- Spread sheet csv data
  Drawj2d can draw points of a csv file (see chapter 4).
- Fachwerk background drawings
  Drawj2d can read the simple text based *bgd* format Fachwerk uses (see chapter 4).

1.3 Difference to other programs

Drawj2d is inspired by Asymptote, but it does 2D line drawing only. Both provide a programming syntax: Asymptote is C++-like, Drawj2d is tcl-like. Drawj2d benefits from its limits though: less dependencies, no installation required. Drawj2d is easier, programming experience is not necessary.

Drawj2d is not a CAD program, it provides no graphical user interface!

1.4 Licence

Drawj2d is subject to the *GNU General Public License Version 2+*. The licence disclaims all liability of the author.

The program uses several libraries. They come with their own compatible open source licences. The text of the licences is distributed together with the program source code.

- Hecl - Apache License 2
  The drawj2d vector graphics language uses and extends the Hecl scripting language.
  http://www.hecl.org
- java-getopt - LGPL 2
  Command line option parser.
  http://www.urbanophile.com/arenn/hacking/getopt
- EvalEx - MIT License (X11 License)
  Java expression evaluator used for the `expr` command.
  http://udojava.com/category/open-source/expression-evaluator

- FreeHEP Graphics2D - LGPL 2.1 or Apache License 2
  Graphical back-end for pdf, svg, emf. Fall-back mode for eps.
  freehep.github.io/freehep-vectorgraphics
- EpsGraphics - GPL 2+
  Graphical back-end for eps.
  http://sourceforge.net/projects/epsgraphics
- JTikZ - LGPL 2.1
  Graphical back-end for tikz (TikZ/PGF).
  http://sourceforge.net/projects/jtikz
- JLaTeXMath - GPL 2+
  LaTeX rendering for command `texlabel`.
  http://forge.scilab.org/index.php/p/jlatexmath
- G-library - LGPL 2.1+
  Geometry functions for some commands `geom.*`.
  http://geosoft.no/graphics
- Yacas Grapher - LGPL 2.1+
  Plotter front-end for yacas plot data.
  http://sourceforge.net/projects/yacas

Licence of this program documentation

1.5 Program development

This manual is based on version: Drawj2d 0.83 (January 2017)

Changes in syntax and functionality are to be expected. It is recommended to check the website http://drawj2d.sourceforge.net for a new version from time to time.

2 How to use the program

DRAWJ2D is a command line program. Start it from a shell (terminal, konsole, DOS-window).

2.1 Installation

No installation needed.

It is recommended though to add the drawj2d program directory to the path.

Alternatively create a batch file which calls "java -jar drawj2d.jar" and put it in a folder which is in the path. Windows: `echo %PATH%`, Linux & Mac: `echo $PATH`.

- Windows "drawj2d.bat":
  `java -jar C:\PROGRAMDIRECTORY\drawj2d.jar %*`
- Linux & Mac "drawj2d":
  `java -jar /PROGRAMDIRECTORY/drawj2d.jar $@`

2.2 Basic usage

```
drawj2d drawing.hcl

drawj2d --type pdf drawing.hcl

drawj2d --type pdf --width 150 --height 100 --center drawing.hcl

drawj2d -T pdf -W 150 -H 100 -c drawing.hcl

drawj2d -T pdf -W 150 -H 100 drawing.hcl

drawj2d -T pdf -W 150 -H 100 -X 50 -Y 50 drawing.hcl

drawj2d -T svg -W 150 -H 100 -c drawing.hcl

drawj2d -T pdf -o drawing.pdf drawing.hcl
```

2.3 Command line options

For help type

```
java -jar drawj2d.jar --help
```

The program will print the command line parameters.

```
            Welcome to Drawj2d
            Copyright (c) A. Vontobel, 2014-2017
            Version 0.83

Usage:
java -jar drawj2d.jar  [-TWHcXYrfvqhV]
                       [-o OutputFile]
                       [InputFile]

-T, --type                Output file type: pdf, svg, eps, emf, png
                                            screen (displays the drawing)
-W, --width               Graphics width (default 210mm)
-H, --height              Graphics height (default 297mm)
-c, --center              Set origin to center of sheet, instead of top left
-X, --originx             Offset origin right (default 0mm)
-Y, --originy             Offset origin down (default 0mm)
-r, --resolution          Resolution of images (default 200dpi), svg (96dpi)
-f, --fallback            Fallback mode
-F, --frontend            Input file type: hcl (default), bgd, ypd
-v, --verbose             Verbose output
-q, --quiet               No messages to stdout
-h, --help                Usage information; this help screen
-V, --version             Display version
-o  --outfile OutputFile  OutputFile name (default: out-filename)
InputFile                  InputFile. If omitted reads from stdin
```

2.4 Viewer

Preview the drawing using the screen type argument.

```
drawj2d -T screen drawing.hcl
```

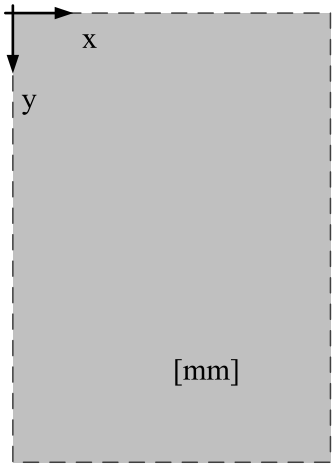
```
drawj2d -T screen -W 297 -H 210 drawing.hcl
```

To redraw press F5. This will parse the input file again. Thus the viewer can stay open, while you edit the input file in an editor. Pressing F5 will refresh the drawing. To close the window type ESC.

3 Drawj2D Function Reference

3.1 Utility commands: coordinates and units

The Drawj2d coordinate system is shown below. The y-axis points downwards!



The sheet unit is *mm*. The drawing unit is not predefined, the scale can be set. Initially the drawing unit is assumed to be *mm* (scale 1:1). The scale for arrows representing forces can be set separately.

3.1.1 unitlength (unitsize)

Set the scale.

SYNOPSIS
unitlength *scale unit*
unitlength *mm*
unitlength

scale	scaling factor, e.g. `[/ 1. 100.]`
unit	`mm` | `cm` | `meter`, `m` | `km` | `µm` | `nm` | `inch`, `in` | `foot`, `ft` | `yard`, `yd` | `mile` | `point`, `pt` | number in mm. Default: mm
mm	the length (in mm) on the sheet for one drawing unit

EXAMPLE

```
unitlength [/ 1. 50.] m; # Scale 1:50, assuming the input units are in m
puts [unitlength];       # Prints 20.0 (1000mm/50 = 20.0mm)
```

3.1.2 forceunitlength

Force unit length.

SYNOPSIS
forceunitlength *mm decdigits*
forceunitlength *mm*
forceunitlength

mm	the length (in mm) on the sheet for one force unit (usually kN)
decdigits	Number of decimal places. If omitted: 1

EXAMPLE

```
forceunitlength 5.0 1; # 1kN is drawn 5.0mm, assuming the input unit
                       # is kN. Force values are written with one
                       # decimal after the point, e.g.
                       # 10.333kN will be written 10.3.
forceunitlength;       # Equivalent to: forceunitlength 1.0 1
```

3.1.3 offset

Offset the origin (0/0) coordinate. Offset is relative to the previous origin position.

SYNOPSIS
offset *dX dY*

dX	offset to the right, measured in drawing units
dY	offset downwards, measured in drawing units

EXAMPLE

```
offset 1 1
offset [mm 1 1]
```

3.1.4 here (r)

Get the current position. Or a position relative to the current one.

SYNOPSIS
here *dx dy*
here

dx dy	Relative coordinates from the current position. "0 0" if omitted.

EXAMPLE

```
set pos [here]
puts $pos;     # Prints the coordinates of the current position.
```

3.1.5 mm

mm to unit length conversion.

SYNOPSIS
mm *length1 length2 ...*
mm *length*

length length measured in mm

EXAMPLE

```
moverel [mm 0 10]; # Moves the cursor 10mm downwards.
```

3.1.6 fu (kN)

Force unit to unit length conversion.

SYNOPSIS
fu *forcevalue1 forcevalue2 ...*
fu *forcevalue*

forcevalue force value measured in force unit, usually kN

EXAMPLE

```
arrowrel [kN 0 10]; # Draws an arrow 10kN downwards.
```

3.1.7 X, Y

Get the X (or Y) coordinate of a position {x y}.
X is equivalent to: [lindex $pos 0]. Y is equivalent to: [lindex $pos 1].

SYNOPSIS
X *pos*
Y *pos*

pos coordinate pair

EXAMPLE

```
set pos {120 300}
puts [X $pos]; # Prints 120
puts [Y $pos]; # Prints 300
```

3.1.8 FX, FY

Get the FX (or FY) component of a force {x y Fx Fy}.
FX is equivalent to: [lindex $F 2]. FY is equivalent to: [lindex $F 3].

SYNOPSIS
FX *F*
FY *F*

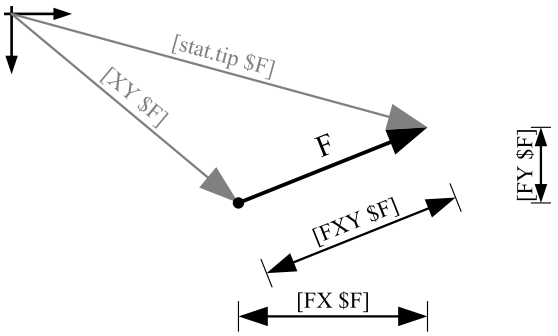
F force "x y Fx Fy"

EXAMPLE

```
set F {10 0  20 100}
puts [FX $F]; # Prints 20
puts [FY $F]; # Prints 100
```

3.1.9 XY, FXY

Get the application point "x y" or the components "Fx Fy" of a force {x y Fx Fy}.



XY is equivalent to: "[lindex $F 0] [lindex $F 1]".
FXY is equivalent to: "[lindex $F 2] [lindex $F 3]".

SYNOPSIS
XY *F*
FXY *F*

F force "x y Fx Fy"

EXAMPLE

```
set F {10 0  20 100}
puts [XY $F];  # Prints 10 0
puts [FXY $F]; # Prints 20 100
```

3.2 Utility commands: pen, font and mathematical expressions

3.2.1 pen

The pen defines the current color, line stroke width and line type.

SYNOPSIS
pen [*color*] [*linewidth*] [*linetype*]
pen *color*
pen *linewidth*
pen *linetype*
pen

color	`black`, `k` | `blue`, `b` | `cyan`, `c` | `darkgray`, `d` | `gray`, `a` | `green`, `g` | `lightgray`, `l` | `magenta`, `m` | `orange`, `o` | `pink`, `p` | `red`, `r` | `white`, `w` | `yellow`, `y` | `brown` | `darkorange` | 0xrrggbb (hexadecimal values for red, green, blue)
linewidth	Line stroke width in mm. Default: 0.5 mm
linetype	`dashed` | `dotted` | `dashdotted` | `solid`
(none)	equal to: pen black 0.5 solid

EXAMPLE

```
pen red dashdotted
line 0 0 100 0
pen 0x87431D;      # color hex values. r:87, g:43, b:1D (brown)
```

3.2.2 font

Set the font.

SYNOPSIS
font [*fontname*] [*style*] [*size*]
font *fontname*
font *style*
font *size*
font

fontname	`Serif` | `Sans_Serif` | `Monospaced` | `Tex` or font name
style	`plain`, `up` | `bold`, `bf` | `italic`, `it`
size	font height in mm
(none)	equivalent to: font Serif plain 4

3.2.3 opacity

Set the opacity. Output modes supporting transparency are screen and png.

SYNOPSIS
opacity *value*
opacity

value float value from 0 to 1. 1.0 is opaque. If omitted the value is set to 1.0 (opaque).

3.2.4 nf

Returns a formatted number.

SYNOPSIS
nf *number decdigits*
nf *number*

number the number to be formatted
decdigits Amount of digits beyond point. If omitted the value set by dimline (default 3) is used.

EXAMPLE

```
set L 4.6666667
puts [nf $L 2]; # Prints 4.67
puts [nf $L];   # Prints 4.667, unless default value has been changed.
```

3.2.5 expr

Calculates the value of a mathematical expression.

SYNOPSIS
expr *mathexpr*

mathexpr Mathematical expression. Angles are expected in degrees (this differs from tcl and from the hecl trigonometric functions!).

EXAMPLE

```
set D 4.4
puts [expr $D /2 * sin(30)];  # Prints 1.1.
                              # The space after variable $D is required!
# Equivalent in pure hecl
puts [* [/ $D 2.] [sin [/ $pi 6.]]]

# The expression parser is somewhat basic.
# [expr -sqrt(3)] will cause an error ("Too many operators ...").
puts [expr 0 - sqrt(3)];      # Valid expression. Prints -1.732051
puts [expr -1*sqrt(3)];       # Valid expression. Prints -1.732051
```

3.2.6 exprinput

Returns the mathematical expression. Useful to check what the expr would get as input.

SYNOPSIS
exprinput *mathexpr*

mathexpr mathematical expression

EXAMPLE

```
set D 4.4
puts [exprinput $D /2 * sin(30)]; # Prints 4.4 /2 * sin(30)
```

3.2.7 assert

Verifies that an assertion is fulfilled. Actually a mathematical expression is evaluated. If the return value is other than 1 (== true), an exception is raised.

SYNOPSIS
assert *condition message*
assert *condition*

condition A condition that is expected to be true. For comparisons use <, >, <=, >=, = or ~=. The operator ~= tests whether both sides are approximately equal (up to 11 significant digits). "$val ~= 0" is assumed to be true for $|\$val| < 10^{-11}$.
message optional message that helps to identify the assertion in case of failure

EXAMPLE

```
set h 250
# The program will abort if an assertion turns out to be wrong
assert "$h > 200" InputCheck
```

3.3 Drawing commands: points and lines

3.3.1 moveto (m)

Move the cursor to a new position. The new position is given by its coordinates.

SYNOPSIS
moveto *x1 y1*
moveto *pos1*

x1 y1 new position
pos1 new position

EXAMPLE

```
set pos {50 0}
moveto $pos
```

3.3.2 movetox (mx), movetoy (my)

Move the cursor to a new x-coordinate while keeping the y-coordinate. Or move the cursor to a new y-coordinate while keeping the x-coordinate.

SYNOPSIS
movetox *x1*
movetox *posP*

x1 new x-coordinate
posP Position "x1 yP". The x-coordinate is used only.

EXAMPLE

```
moveto 50 0
movetox {60 45}
puts [here];  # Prints 60 0.
movetoy 30
puts [here];  # Prints 60 30.
```

3.3.3 moverel (mr)

Move the cursor relative to the actual position.

SYNOPSIS
moverel *dx dy*
moverel *vector*

dx dy coordinate increment
vector vector of movement

EXAMPLE

```
moveto 50 0
moverel 10 30
puts [here];  # Prints 60 30.
```

3.3.4 movepolar (mp)

Move the cursor in polar direction relative to the actual position.

SYNOPSIS
movepolar *dL α*

dL distance to current position
α azimuth (angle to x-axis, in degrees, clock-wise)

EXAMPLE

```
moveto 5.0 0.0
movepolar 2.0 60
puts [here];  # Prints 6.0 1.732
```

3.3.5 point (pt), dot

Mark a point at the given coordinate. Moves the cursor there. The point command draws a small circle (diameter = 1.4 × linewidth), the dot command a larger one (diameter 1.5 mm).

SYNOPSIS
point *x1 y1*
point *pos1*
point

x1 y1 point coordinates
pos1 point position

EXAMPLE

```
dot 50 0;             # Draws a dot.
point 70 0; label P; # Marks another point and names it P.
```

3.3.6 line (l), lineto (l)

Draw a line from the first coordinate to the second one. Or draw a line from the actual cursor position to the given coordinate. Sets the cursor to the end of the line (second coordinate x1 y1).

SYNOPSIS
line *x0 y0 x1 y1 ...*
line *pos0 pos1 ...*
lineto *x1 y1*
lineto *pos1*

x0 y0 Beginning position of the line. Current position if omitted.
x1 y1 ending position of the line
pos0 Beginning position of the line. Current position if omitted.
pos1 ending position of the line

EXAMPLE

```
line 5 5 5 30
lineto 30 30
set TR {30 5}
lineto $TR
```

3.3.7 linetox (lx), linetoy (ly)

Draw a horizontal line to a new x-coordinate. Or draw a vertical line to a new y-coordinate. Equivalent to commands "lineto [geom.tox $posP]".

SYNOPSIS
linetox *x1*
linetox *posP*

x1 new x-coordinate
posP Position "x1 yP". The x-coordinate is used only.

3.3.8 linerel (lr)

Draw a line from the actual cursor position to the given relative position. Sets the cursor to the end of the line.

SYNOPSIS
linerel *dx dy*
linerel *vector*

dx dy coordinate increment
vector vector of movement

EXAMPLE

```
moveto 5 5
linerel 0 25
linerel {25 0}
```

3.3.9 linepolar (lp)

Draw a line given its length and direction relative to the actual position. Sets the cursor to the end of the line.

SYNOPSIS
linepolar *dL α*

dL distance to current position
α azimuth (angle to x-axis, in degrees, clock-wise)

EXAMPLE

```
moveto 5.0 0.0
linepolar 2.0 60
puts [here];  # Prints 6.0 1.732
```

3.3.10 linemid (lm)

Draw a line given its length and direction. The middle of the line is set at the cursor position. The cursor does not move.

SYNOPSIS
linemid *L* α
linemid *L*

L line length
α Orientation of the line. Horizontal ($\alpha = 0$) if omitted.

EXAMPLE

```
moveto 17.5 5
linemid 25
```

3.3.11 arc

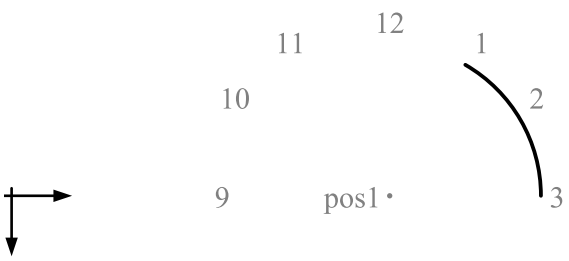
Draw an arc. Zero angle is where the x-axis points to. Clock-wise, consistent with the coordinate system. Sets the cursor to the center.

SYNOPSIS
arc *x1 y1 radius startangle endangle*
arc *pos1 radius startangle endangle*
arc *radius startangle endangle*

x1 y1	center
pos1	Center. Current cursor position if omitted.
radius	radius
startangle	in degrees, clock-wise
endangle	in degrees, clock-wise

EXAMPLE

```
set hour1 -60; set hour3 0
# Draw an arc from one-o-clock to three-o-clock
arc 50 0 20 $hour1 $hour3
```



3.3.12 quadcurve (parabola)

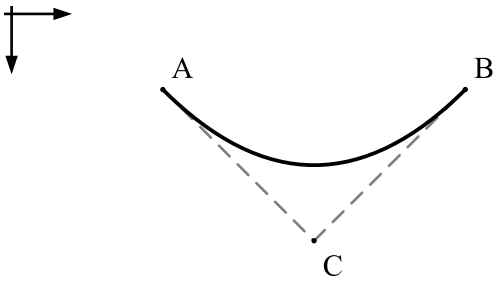
Draw a parabola. It is defined by the starting point, a control point and the end point. The control point is the intersection point of the tangents at starting and end points. Sets the cursor to the last point.

SYNOPSIS
quadcurve *x0 y0 xCtrl yCtrl x1 y1*
quadcurve *xCtrl yCtrl x1 y1*
quadcurve *pos0 posCtrl pos1*
quadcurve *posCtrl pos1*

x0 y0	starting point
pos0	Starting point. Current cursor position if omitted.
xCtrl yCtrl	control point
posCtrl	Control point. Intersection of tangents.
x1 y1	end point
pos1	end point

EXAMPLE

```
set A {20 10}; point $A; label A
set B {60 10}; point $B; label B
set C {40 30}
pen dashed 0.35 gray
line $A $C $B;                    # Tangents
pen
point $C; label C SE
quadcurve $A $C $B;               # Parabola
```



3.3.13 cubiccurve

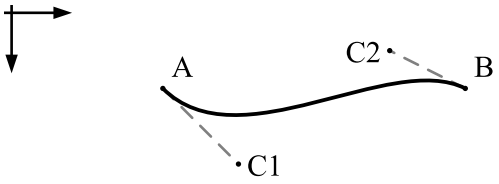
Draw a cubic parabola. It is defined by the starting point, two control points and the end point. The control points are on the tangents at the starting and end points. Sets the cursor to the last point.

SYNOPSIS
cubiccurve *x0 y0 x0Ctrl y0Ctrl x1Ctrl y1Ctrl x1 y1*
cubiccurve *x0Ctrl y0Ctrl x1Ctrl y1Ctrl x1 y1*
cubiccurve *pos0 pos0Ctrl pos1Ctrl pos1*
cubiccurve *pos0Ctrl pos1Ctrl pos1*

x0 y0	starting point
pos0	Starting point. Current cursor position if omitted.
x0Ctrl y0Ctrl	first control point
pos0Ctrl	first control point
x1Ctrl y1Ctrl	second control point
pos1Ctrl	second control point
x1 y1	end point
pos1	end point.

EXAMPLE

```
set A {20 10}; point $A; label A
set B {60 10}; point $B; label B
set C1 {30 20}
set C2 {50  5}
pen dashed 0.35 gray
line $A $C1;              # tangents
line $B $C2
pen
point $C1; label C1 E
point $C2; label C2 W
cubiccurve $A $C1 $C2 $B; # cubic parabola
```



3.4 Drawing commands: shapes and fills

3.4.1 circle, fillcircle

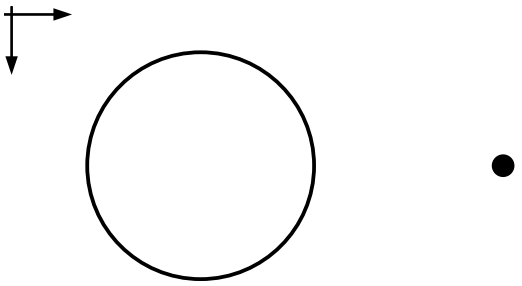
Draw or fill a circle. Sets the cursor to the center.

SYNOPSIS
circle *x1 y1 radius*
circle *pos1 radius*
circle *radius*

x1 y1	center
pos1	center
radius	radius

EXAMPLE

```
unitlength 10
circle 2.5 2.0 1.5;          # Radius 1.5 drawing units,
                             # thus equivalent to 15 mm
fillcircle 6.5 2.0 [mm 1.5]; # Radius 1.5 mm
```



3.4.2 ellipse, fillellipse

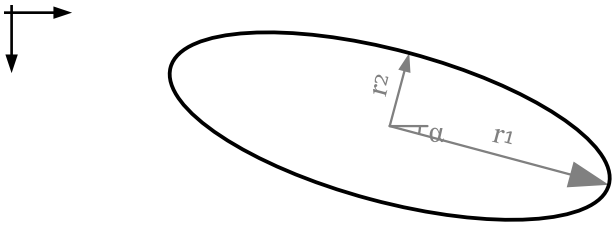
Draw or fill an ellipse. The cursor position is the center.

SYNOPSIS
ellipse *radius1 radius2 angle1*
ellipse *radius1 radius2*

radius1	radius in first direction
radius2	radius in second direction
angle1	Angle in degrees between x-axis and first direction, clock-wise. 0° if omitted.

EXAMPLE

```
moveto 50 15
ellipse 30 10  15
```



3.4.3 rectangle (rect)

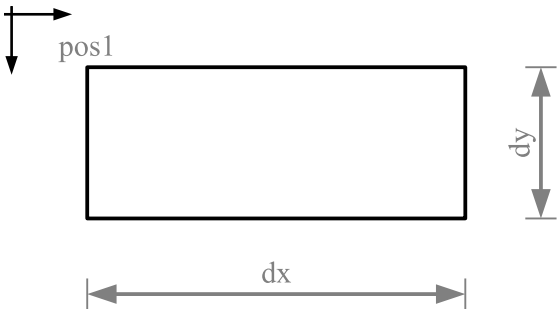
Draw a rectangle defined by its width and height. See also box and rod commands. The cursor position remains at the starting corner.

SYNOPSIS
rectangle *x1 y1 dx dy*
rectangle *pos1 dx dy*
rectangle *dx dy*

x1 y1 Coordinates of a corner. If omitted current cursor position.
pos1 Corner position (usually top left). If omitted current cursor position.
dx width
dy height

EXAMPLE

```
moveto 10 7
rectangle 50 20
```



3.4.4 fillrectangle (fillrect)

Fill a rectangle defined by its width and height. The top left corner is defined by the current cursor position. The cursor position does not move. Use the command to set the background color.

SYNOPSIS
fillrectangle *x1 y1 dx dy*
fillrectangle *pos1 dx dy*
fillrectangle *dx dy*

x1 y1 Coordinates of a corner. If omitted current cursor position.
pos1 Corner position (usually top left). If omitted current cursor position.
dx width
dy height

EXAMPLE: For png image output, fill the sheet background with white color (instead of transparent background).

```
pen white
fillrect 210 297; # Fill A4 sheet
pen;              # Reset pen to black
unitlength 10;    # Set the scale: length in mm of a drawing unit
line 2.0 0 5.0 0
```

3.4.5 box, fillbox

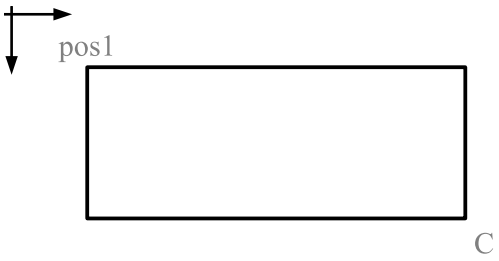
Draw or fill a rectangle defined by two diagonal corner points. The current position remains at (or moves to) the starting corner pos1.

SYNOPSIS
box *x1 y1 x2 y2*
box *pos1 posC*
box *x2 y2*
box *posC*

x1 y1	First corner of the rectangle. Cursor position moves here. If omitted current cursor position.
pos1	First corner of the rectangle. Cursor position moves here. If omitted current cursor position.
x2 y2	diagonal corner of the rectangle
posC	diagonal corner of the rectangle

EXAMPLE

```
moveto 10 7; # first corner (here top left, but not necessarily)
box 60 27;   # diagonal corner
```



3.4.6 rod, fillrod

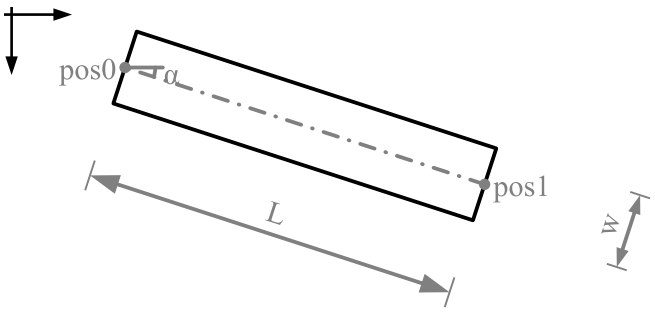
Draw or fill a rectangle defined by its axis (starting point and orientation), length and width. Sets the cursor to the end point on the axis, L from the starting point.

SYNOPSIS
rod *x0 y0 L w α*
rod *x0 y0 L w*
rod *pos0 L w α*
rod *pos0 L w*
rod *L w α*
rod *L w*

x0 y0	Coordinates on rod axis. If omitted current cursor position.
pos0	Coordinates on rod axis. If omitted current cursor position.
L	length
w	width
α	orientation, in degrees, clock-wise

EXAMPLE

```
moveto 15 7
rod 50 10 18
```



3.4.7 polygon, fillpolygon

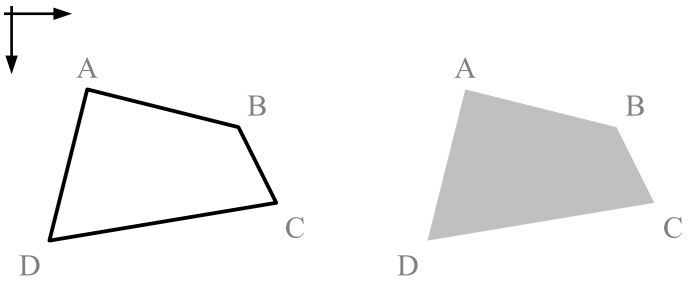
Draw a polygon defined by at least three vertices. The current position moves to the starting vertex A. The command polygon $A $B $C is equivalent to "line $A $B; lineto $C; lineto $A".

SYNOPSIS
polygon *x1 y1 x2 y2 x3 y3 .....*
polygon *posA posB posC ...*

x1 y1	First vertex of the polygon. Cursor position moves here.
posA	First vertex of the polygon. Cursor position moves here.
x2 y2 x3 y4	further vertices
posB posC	further vertices

EXAMPLE

```
set A {10 10}
set B {30 15}
set C {35 25}
set D { 5 30}
polygon $A $B $C $D
offset 50 0
pen lightgray
fillpolygon $A $B $C $D
```



3.4.8 segment, fillsegment

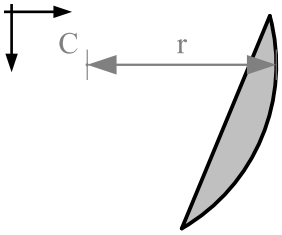
Draw or fill a segment of a circle. Zero angle is where the x-axis points to. Clock-wise, consistent with the coordinate system. Sets the cursor to the center.

SYNOPSIS
segment *x1 y1 radius startangle endangle*
segment *pos1 radius startangle endangle*
segment *radius startangle endangle*

x1 y1	center
pos1	Center. Current cursor position if omitted.
radius	radius
startangle	in degrees, clock-wise
endangle	in degrees, clock-wise

EXAMPLE

```
set C {10 7}
set r 25
set α -15
set β 60
pen lightgray
fillsegment $C $r $α $β
pen black
segment $r $α $β
```



3.4.9 sector, fillsector

Draw or fill a sector. Zero angle is where the x-axis points to. Clock-wise, consistent with the coordinate system. Sets the cursor to the center.

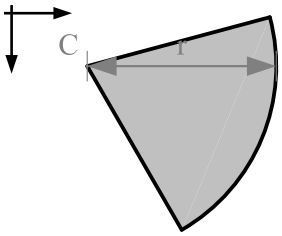
SYNOPSIS
sector *x1 y1 radius startangle endangle*
sector *pos1 radius startangle endangle*
sector *radius startangle endangle*

x1 y1	center
pos1	Center. Current cursor position if omitted.
radius	radius
startangle	in degrees, clock-wise

endangle in degrees, clock-wise

EXAMPLE

```
set C {10 7}
set r 25
set α -15
set β 60
pen lightgray
fillsector $C $r $α $β
pen black
sector $r $α $β
```



3.4.10 image

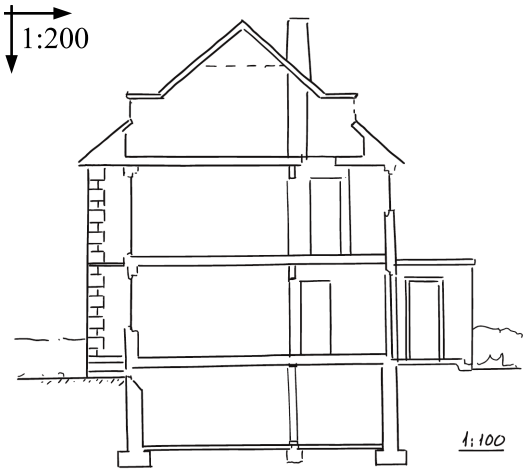
Insert an image. The cursor position does not move. Some output modes may not support well this command.

SYNOPSIS
image *filename dpi pxX pxY 1 : x*
image *filename dpi pxX pxY*
image *filename dpi*
image *filename*

filename	image file (*.png|jpeg|bmp)
dpi	image resolution [dots per inch]. Default: 200
pxX pxY	Pixel coordinates of insertion point. If omitted top left corner of the image.
1 : x	scale e.g. 100. If omitted or 0 the image will not be scaled to natural units.

EXAMPLE:

```
unitlength [expr 1/200] m
image section.png 200  0 0  100; # drawing 1:100 scanned at 200dpi
label {1:200} SE
```



3.5 Drawing commands: labels and arrows

3.5.1 label (lb)

Label the position of the cursor. Or align a label in the middle of two points.

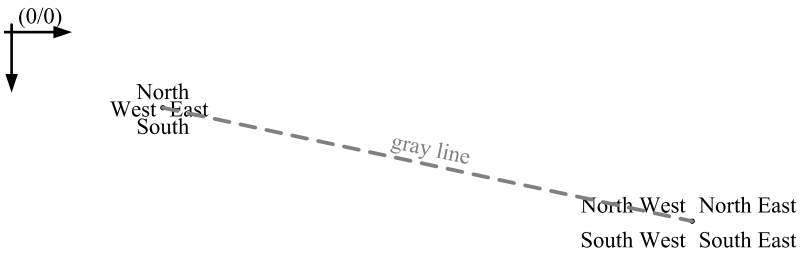
SYNOPSIS
label *Text xA yA xB yB*
label *Text posA posB*
label *Text lineAB*
label *Text placement*
label *Text*

xA yA	first point of imaginary line
xB yB	second point of imaginary line
posA	first point of imaginary line
posB	second point of imaginary line
lineAB	imaginary line "xA yA xB yB"
placement	The placement of the label: `NE` | `E` | `SE` | `S` | `SW` | `W` | `NW` | `N` | `C` | `BL`. If omitted NE.
Text	Label text. Unicode chars shall be UTF-8 formatted or escaped using *escu* and the hexadecimal index (e.g. `escu00b7` for the middle dot · unicode char U+00B7).

EXAMPLE

```
font Serif 3;               # Set font type and size
label "(0/0)"
point 20 10; set A [here]
label North N
label East E
label South S
label West W
point 90 25; set B [here]
label "North East" NE
label "South East" SE
label "South West" SW
```

```
label "North West" NW
set color gray
pen $color; pen dashed
line $A $B
# Note the label argument below:
label "$color line" $A $B; # "$color" evaluates variable,
                           # {$color} would not.
```



3.5.2 texlabel (tlb)

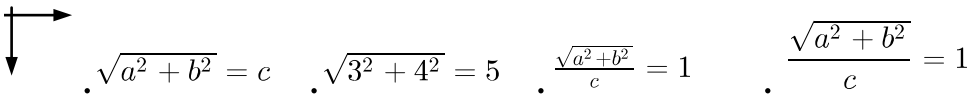
Label the position of the cursor using TEX math syntax. Or align a label in the middle of two points.

SYNOPSIS
texlabel *Text xA yA xB yB*
texlabel *Text posA posB*
texlabel *Text lineAB*
texlabel *Text placement*
texlabel *Text*

xA yA	first point of imaginary line
xB yB	second point of imaginary line
posA	first point of imaginary line
posB	second point of imaginary line
lineAB	imaginary line "xA yA xB yB"
placement	The placement of the label: `NE` | `E` | `SE` | `S` | `SW` | `W` | `NW` | `N` | `C` | `BL`. If omitted NE.
Text	Label text using TeX syntax. If there is white space, enclose the text with curly brackets. If the text contains variables, use quotation marks instead. Within quotation marks any backslash must be escaped, thus a double backslash is required.

EXAMPLE

```
point 20 10
texlabel {\sqrt{a^2+b^2} = c};  # Curly brackets
point 40 10
set a 3; set b 4
texlabel "\\sqrt{$a^2+$b^2}=5"; # Double backslash
point 70 10
texlabel {\frac{\sqrt{a^2+b^2}}{c} = 1}
point 100 10
texlabel {\displaystyle \frac{\sqrt{a^2+b^2}}{c} = 1}
```



3.5.3 text

Write some text. The words are wrapped if necessary. Sets the cursor a linefeed lower.

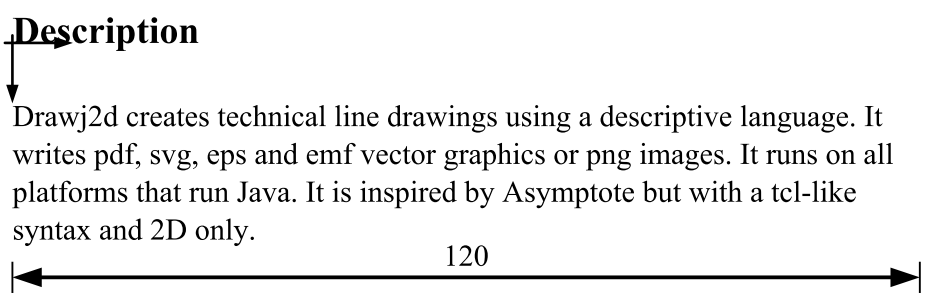
SYNOPSIS
text *Text* [*width*] [*alignment*]
text *Text*
text

width	The available width for the text in drawing units. Words are wrapped if necessary. If omitted the last used width, initially 150 mm.
alignment	`left` | `justify`. If omitted the last used alignment, initially left.
Text	Text. Unicode chars shall be UTF-8 formatted or escaped using *escu* and the hexadecimal index (e.g. `escu00b7` for the middle dot · unicode char U+00B7).

EXAMPLE

```
font bold 5
text {Description} [mm 120]
font plain 4
text;              # Empty line
text {Drawj2d creates technical line drawings using a descriptive
   language. It writes pdf, svg, eps and emf vector graphics or png
   images. It runs on all platforms that run Java. It is inspired by
   Asymptote but with a tcl-like syntax and 2D only.}

dimlinerel [mm 120 0]
```



3.5.4 arrow, arrowto

Draw an arrow from the first coordinate to the second one. Or draw an arrow from the actual cursor position to the given coordinate. Sets the cursor to the arrow head.

SYNOPSIS
arrow *x0 y0 x1 y1*
arrow *pos0 pos1*
arrowto *x1 y1*
arrowto *pos1*

x0 y0 Beginning position of the arrow. Current position if omitted.
x1 y1 ending position of the arrow (arrow head)
pos0 Beginning position of the arrow. Current position if omitted.
pos1 ending position of the arrow (arrow head)

EXAMPLE

```
arrow 5 5 5 30
arrowto 30 30
set TR {30 5}
arrowto $TR
```

3.5.5 arrows, arrowsto

Draw a double headed arrow from the first coordinate to the second one. Or draw a double headed arrow from the actual cursor position to the given coordinate. Sets the cursor to the new position.

SYNOPSIS
arrows *x0 y0 x1 y1*
arrows *pos0 pos1*
arrowsto *x1 y1*
arrowsto *pos1*

x0 y0 Beginning position of the double headed arrow. Current position if omitted.
x1 y1 ending position of the double headed arrow
pos0 Beginning position of the double headed arrow. Current position if omitted.
pos1 ending position of the double headed arrow

EXAMPLE

```
arrows 5 5 5 30
arrowsto 30 30
set TR {30 5}
arrowsto $TR
```

3.5.6 arrowrel

Draw an arrow from the actual cursor position to the given relative position. Sets the cursor to the arrow head.

SYNOPSIS
arrowrel *dx dy*
arrowrel *vector*

dx dy coordinate increment
vector vector of movement

EXAMPLE

```
moveto 5 5
arrowrel 0 25
arrowrel {25 0}
```

3.5.7 arrowsrel

Draw a double headed arrow from the actual cursor position to the given relative position. Sets the cursor to the new position.

SYNOPSIS
arrowsrel *dx dy*
arrowsrel *vector*

dx dy coordinate increment
vector vector of movement

EXAMPLE

```
moveto 5 5
arrowsrel 0 25
arrowsrel {25 0}
```

3.5.8 force

analogy: **texforce**

Draw a force arrow (pointing away from the cursor). The label is either its absolute force value or any text. The command sets the cursor to the arrow head. The command does not do anything if the absolute force value is zero. See forceunitlength.

SYNOPSIS
force *x0 y0 Fx Fy Label*
force *x0 y0 Fx Fy*
force *Fx Fy Label*
force *Fx Fy*

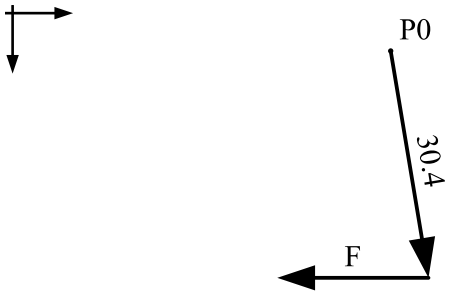
x0 y0 force application point

Fx Fy force components

Label Label. If omitted the absolute force value is written. Use {""} or {{}} to suppress any labelling.

EXAMPLE

```
point 50 5; label P0
force 5 30;            # Writes 30.4. The cursor moves to the tip.
force -20 0 F;         # Writes F
```



3.5.9 force2

analogy: **texforce2**

Draw a force arrow (pointing to the cursor). The label is either its absolute force value or any text. The command sets the cursor to the arrow head. The command does not do anything if the absolute force value is zero. See forceunitlength.

SYNOPSIS
force2 *x1 y1 Fx Fy Label*
force2 *x1 y1 Fx Fy*
force2 *Fx Fy Label*
force2 *Fx Fy*

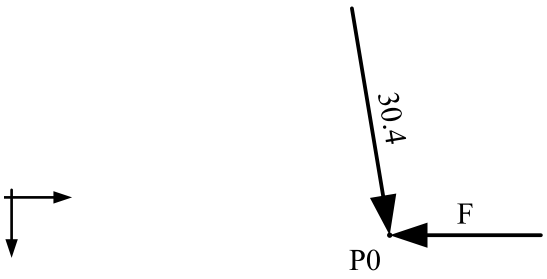
x1 y1 force application point

Fx Fy force components

Label Text. If omitted the absolute force value is written. Use {""} to suppress any labelling.

EXAMPLE

```
point 50 5; label P0 SW
force2 5 30;           # Writes 30.4
force2 -20 0 F;        # Writes F
```



3.5.10 dimline, dimlineto

analogy: **texdimline, texdimlineto**

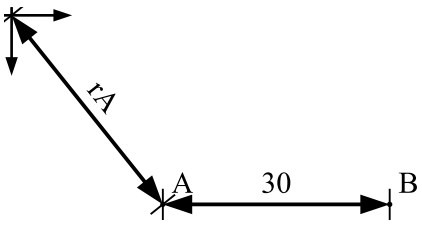
Draw a dimension line (double headed arrow). The label is either its length (in drawing units) or any text. Sets the cursor to the new position (last coordinate x1/y1).
The command followed by just a natural number sets the number of decimal digits (see dimlinerel for an example).

SYNOPSIS
dimline *x0 y0 x1 y1 Label*
dimline *x0 y0 x1 y1*
dimline *pos0 pos1 Label*
dimline *pos0 pos1*
dimline *decdigits*
dimlineto *x1 y1 Label*
dimlineto *x1 y1*
dimlineto *pos1 Label*
dimlineto *pos1*

x0 y0	Beginning position of the dimension line. Current position if omitted.
x1 y1	ending position of the dimension line
pos0	Beginning position of the dimension line. Current position if omitted.
pos1	ending position of the dimension line
Label	Text. If omitted the length measured in drawing units is written. The label must not contain any white space, use double quotation marks to overcome this restriction: "\"$F kN\"". Consider using separate arrows and label/texlabel commands.
decdigits	Maximal number of decimal places to be written. This setting is used by any following dimension line command.

EXAMPLE

```
set A {20 25}; point $A; label A
set B {50 25}; point $B; label B
dimline {0 0} $A rA;              # Writes rA
dimlineto $B;                     # Writes 30
```



3.5.11 dimlinerel

analogy: **texdimlinerel**

Draw a dimension line (double headed arrow) from the cursor position to a relative position. The label is either its length (in drawing units) or any text. Sets the cursor to the new position.

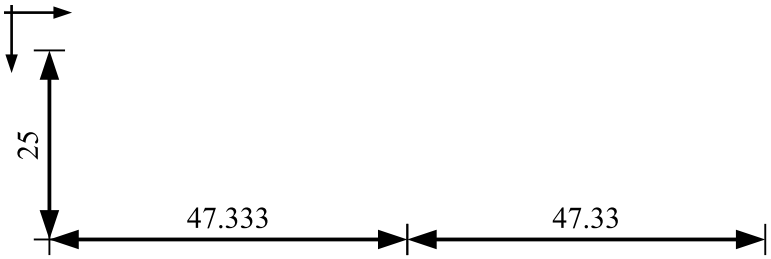
SYNOPSIS
dimlinerel *dx dy Label*
dimlinerel *dx dy*
dimlinerel *vector Label*
dimlinerel *vector*

dx dy	coordinate increment
vector	vector of movement
Label	Text. If omitted the length measured in drawing units is written. The label must not contain any white space, use double quotation marks to overcome this restriction: "\"$F kN\"". Consider using separate arrows and label/texlabel commands.

EXAMPLE

```
moveto 5 5
dimlinerel 0 25.0;         # Writes 25
dimlinerel {47.333333 0}; # Writes 47.333

dimline 2;                 # Sets the number of decimal digits to 2.
dimlinerel {47.333333 0}; # Writes 47.33
```



3.5.12 dimangle

analogy: **texdimangle**

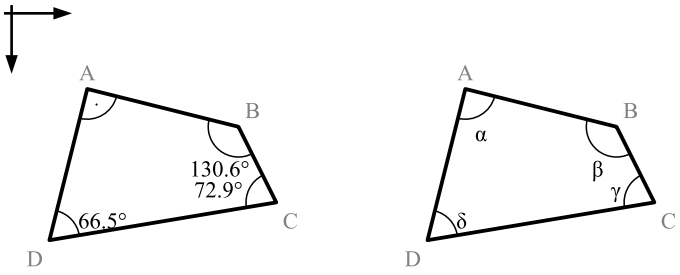
Draw a dimension arc at a vertex. The label is either its angle in degrees or any text. Clock-wise. The texdimangle command uses TEX typesetting is used. Sets the cursor to the vertex (pos0).

SYNOPSIS
dimangle *x0 y0 xP yP xQ yQ Label*
dimangle *pos0 posP posQ Label*
dimangle *x0 y0 xP yP xQ yQ*
dimangle *pos0 posP posQ*
dimangle *xP yP xQ yQ Label*
dimangle *posP posQ Label*
dimangle *xP yP xQ yQ*
dimangle *posP posQ*
dimangle *dirP dirQ Label*
dimangle *dirP dirQ*
dimangle *decdigits*

x0 y0	Vertex. If omitted the current cursor position is assumed.
pos0	Vertex. If omitted the current cursor position is assumed.
xP yP	adjacent vertex
posP	adjacent vertex
xQ yQ	adjacent vertex
posQ	adjacent vertex
dirP	azimuth to vertex
dirQ	azimuth to vertex
decdigits	Maximal number of decimal places to be written. This setting is used by any following dimension arc (dimangle/texdimangle) command.
Label	Text. If omitted the angle measured in degrees is written. The label must not contain any white space. A "." will draw the right angle sign (whether the angle is 90°or not).

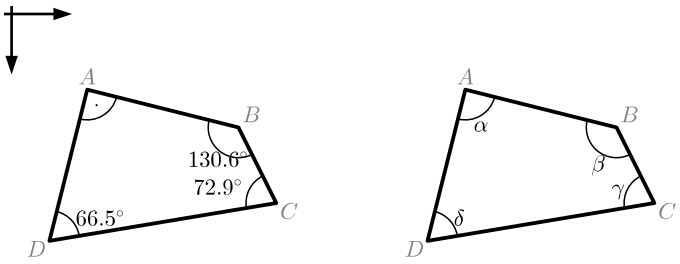
EXAMPLE

```
polygon $A $B $C $D
pen 0.2; font 3
dimangle $A $B $D
dimangle $B $C $A
dimangle $C $D $B
dimangle $D $A $C
offset 50 0; pen;   # Drawing to the right
polygon $A $B $C $D
pen 0.2
dimangle $A $B $D α
dimangle $B $C $A β
dimangle $C $D $B γ
dimangle $D $A $C δ
```



EXAMPLE using TEX typesetting

```
polygon $A $B $C $D
pen 0.2; font 3
texdimangle $A $B $D
texdimangle $B $C $A
texdimangle $C $D $B
texdimangle $D $A $C
offset 50 0; pen;              # Drawing to the right
polygon $A $B $C $D
pen 0.2
texdimangle $A $B $D {\alpha}
texdimangle $B $C $A {\beta}
texdimangle $C $D $B γ
texdimangle $D $A $C δ
```



3.6 Utility commands: blocks

A block is an invisible frame in the drawing that has its own coordinate system. It is useful to insert part drawings from a library. A block can be rotated or mirrored, without influencing the main coordinate system.

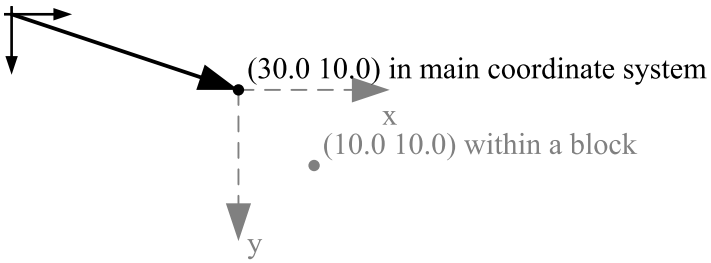
3.6.1 block, endblock

The block command starts its own coordinate system. Its origin is at the current location. The endblock command puts back the previous coordinate system and the cursor is placed where it was when the block command was called. Blocks may be nested.

SYNOPSIS
block
endblock

EXAMPLE

```
arrowto 30 10
pen dashed 0.25 gray
block
        arrow 0 0  20 0; label x S
        arrow 0 0  0 20; label y E
        dot 10 10
        label "([here]) within a block"
endblock
pen
dot
label "([here]) in main coordinate system"
```



The block coordinate system can be manipulated using the commands block.rotate, block.flip, block.scale or offset. In contrast to these commands that act on the block coordinate system, changes to unitlength or forceunitlength act globally.

3.6.2 block.rotate

Rotate the block's coordinate system. The cursor stays at its absolute position.

SYNOPSIS
block.rotate α

α azimuth (angle to global x-axis, in degrees, clock-wise)

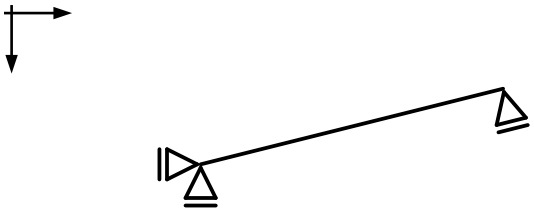
Blocks are useful for inserting drawing blocks, e.g from a separate file (block; source externaldrawing.hcl; endblock). In the example below a typical part is wrapped in a procedure (see chapter 3.9).

EXAMPLE

```
# The procedure could be put in a file library.hcl
# and then loaded by the command: source library.hcl
proc bearing {rot} {
        block
        block.rotate $rot
        set bl [mm 4]; # baselength
        set ofs [+ [mm 0.25] [* 0.05 $bl]]; # offset
        set ofl [+ [mm 0.25] [* 1.3 $bl]]; # line offset
        moveto 0 $ofs
        linerel [/ $bl -2.] $bl
        linerel $bl 0
        lineto 0 $ofs
```

```
        moveto [/ $bl -2.] $ofl
        linerel $bl 0
        endblock
}

set A {25 20}
set B {65 10}
set β [geom.azimuth $A $B]
moveto $A
bearing 0
bearing 90
lineto $B
bearing $β
```



3.6.3 block.flip

Flip the block's coordinate system. Thus the local y-axis points upwards instead of downwards. Be aware that labels will be mirrored too. The cursor stays at its absolute position.

SYNOPSIS
block.flip

3.6.4 block.scale

Scale the local axes. These scale factors are applied to the previous scale. Be aware that everything will be scaled, e.g. line widths and labels too.

SYNOPSIS
block.scale *sx sy*
block.scale *s*

sx scale factor for local x-axis
sy scale factor for local y-axis
s scale factor for both axes

3.7 Geometry commands

The geometry commands geom.* are utility commands for calculations with vectors "x y". The commands do not draw.

3.7.1 geom.vector (geom.v)

Constructs the vector specified by two points. Equivalent to [-- $posB $posA].

SYNOPSIS
geom.vector *x1 y1 x2 y2*
geom.vector *posA posB*
geom.vector *x2 y2*
geom.vector *posB*

x1 y1 First point. If omitted the current cursor position is assumed.
x2 y2 second point
posA First point. If omitted the current cursor position is assumed.
posB second point

EXAMPLE

```
puts [geom.vector 10 10  37 20]; # Writes 27.0 10.0
```

3.7.2 geom.azimuth (geom.azi)

Computes the azimuth (in degrees) from point A to point B.

SYNOPSIS
geom.azimuth *x1 y1 x2 y2*
geom.azimuth *posA posB*
geom.azimuth *x2 y2*
geom.azimuth *posB*

x1 y1 First point. If omitted the current cursor position is assumed.
x2 y2 second point
posA First point. If omitted the current cursor position is assumed.
posB second point

EXAMPLE

```
puts [geom.azimuth 10 10  30 30]; # Writes 45.0
```

3.7.3 geom.add (++)

Computes the vector sum.

SYNOPSIS
geom.add *v1 v2 ...*
++ *v1 v2 ...*

v1 first vector or point
v2 second vector

EXAMPLE

```
puts [++ {3.0 0.5} {2 2}]; # Writes 5.0 2.5
```

3.7.4 geom.subtract (--)

Computes the vector subtraction.

SYNOPSIS
geom.substract *v1 v2 ...*
-- v1 v2 ...

v1 first vector or point
v2 second vector

EXAMPLE

```
puts [-- {3.0 0.5} {2 2}];       # Writes 1.0 -1.5
puts [-- {12 37} {3 12} {2 4}]; # Writes 7.0 21.0
```

3.7.5 geom.multiply (**)

Computes a vector scaled by multiplication.

SYNOPSIS
geom.multiply *factor vector*
** *factor vector*

factor scaling factor
vector vector "dx dy"

EXAMPLE

```
puts [** 1.5 {2 3}]; # Writes 3.0 4.5
```

3.7.6 geom.divide (//)

Computes a vector scaled by division.

SYNOPSIS
geom.divide *vector quotient*
// *vector quotient*

vector vector "dx dy"
quotient scaling quotient

EXAMPLE

```
puts [// {3.0 4.5} 1.5]; # Writes 2.0 3.0
```

3.7.7 geom.tox (tx), geom.toy (ty)

Returns the position with a given x coordinate that is horizontally aligned with the cursor position. Or returns the position with a given y coordinate that is vertically aligned with the cursor position.

SYNOPSIS
geom.tox *posP*
geom.tox *x1*

posP position "x1 yP"
x1 x-coordinate

EXAMPLE

```
moveto {5.0 1.2}
set P {9.0 3.5}
puts [tx 8.5]; # Writes 8.5 1.2
puts [tx $P];  # Writes 9.0 1.2
puts [ty 3.0]; # Writes 5.0 3.0
puts [ty $P];  # Writes 5.0 3.5
```

3.7.8 geom.intersect

Returns the intersection point of the extensions of two lines $\overline{AB}$ and $\overline{CD}$. If the lines are parallel the command raises an exception.

SYNOPSIS
geom.intersect *xA yA xB yB xC yC xD yD*
geom.intersect *posA posB posC posD*

xA yA first point of line AB
xB yB second point of line AB
xC yC first point of line CD
xD yD second point of line CD
posA first point of line AB
posB second point of line AB
posC first point of line CD
posD second point of line CD

EXAMPLE
see below

3.7.9 geom.area

Computes the area within a polygon. The command always returns a positive value.

SYNOPSIS
geom.area *xA yA xB yB xC yC ...*
geom.area *posA posB posC ...*

xA yA	first point of polygon
xB yB	second point of polygon
xC yC	third point of polygon
posA	first point of polygon
posB	second point of polygon
posC	third point of polygon

EXAMPLE
see below

3.7.10 geom.centroid

Returns the center of gravity of an area (surrounded by a polygon).

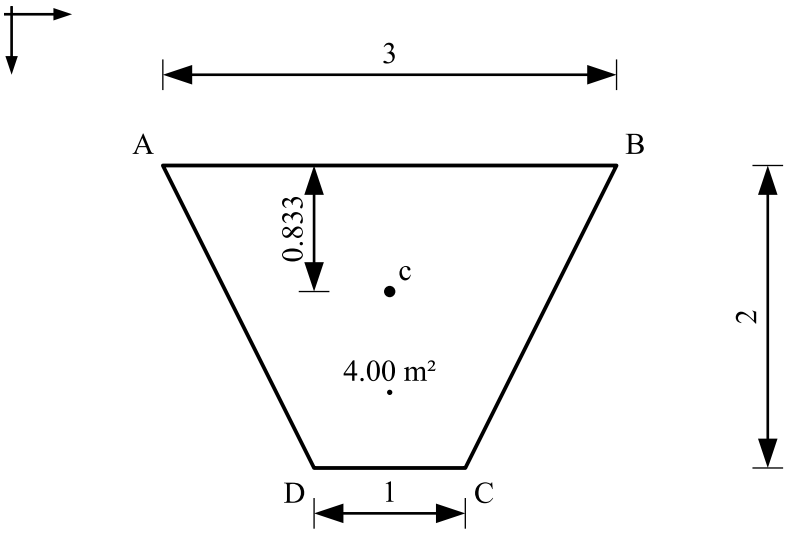
SYNOPSIS
geom.centroid *xA yA xB yB xC yC ...*
geom.centroid *posA posB posC ...*

xA yA	first point of polygon
xB yB	second point of polygon
xC yC	third point of polygon
posA	first point of polygon
posB	second point of polygon
posC	third point of polygon

EXAMPLE

```
unitlength 20
set A {1 1}; m $A; lb A NW
set B {4 1}; m $B; lb B NE
set C {3 3}; m $C; lb C SE
set D {2 3}; m $D; lb D SW
polygon $A $B $C $D
set area [geom.area $A $B $C $D]
pt [geom.intersect "$A $C" "$B $D"]
label "[nf $area 2] m²" N
set c [geom.centroid $A $B $C $D]; # centroid: 2.5 1.8333333333333333
dot $c; lb c
pen 0.35; dimline 5 [Y $A] 5 [Y $D]
dimline 2 [Y $A] 2 [Y $c]
```

```
dimline [X $A] 0.4 [X $B] 0.4
dimline [X $C] 3.3 [X $D] 3.3
```



3.7.11 geom.intersectlinepath

Returns the first intersection point of the extensions of a line $\overline{AB}$ with a path $\overline{P_1 P_2 \ldots}$. If the extension of the line does not intersect the path, an exception is raised.

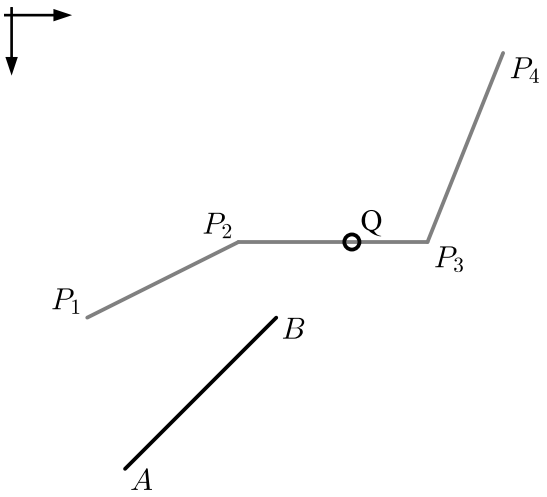
SYNOPSIS
geom.intersectlinepath *xA yA xB yB x1 y1 x2 y2 ...*
geom.intersectlinepath *posA posB posP1 posP2 ...*

xA yA	first point of line AB
xB yB	second point of line AB
x1 y1	first point of path
x2 y2	second point of path
posA	first point of line AB
posB	second point of line AB
posP1	first point of path
posP2	second point of path

EXAMPLE

```
unitlength 10
set A {1.5 6.0}; m $A; tlb A SE
set B {3.5 4.0}; m $B; tlb B SE
set P1 {1.0 4.0}; m $P1; tlb P_1 NW
set P2 {3.0 3.0}; m $P2; tlb P_2 NW
set P3 {5.5 3.0}; m $P3; tlb P_3 SE
set P4 {6.5 0.5}; m $P4; tlb P_4 SE
pen gray
l $P1 $P2 $P3 $P4
pen black
```

```
l $A $B
circle [geom.intersectlinepath $A $B  $P1 $P2 $P3 $P4] [mm 1]
label Q
```



3.7.12 geom.online

Returns the point on the line defined by "x0 y0 x1 y1" a given fraction from x0,y0.
E.g. [geom.online $A $B 0.5] returns the position in the middle of the two points, while [geom.online $A $B 0] returns the position of $A and [geom.online $A $B 1] the position of $B.

SYNOPSIS
geom.online *x1 y1 x2 y2 fraction*
geom.online *posA posB fraction*

x1 y1	first point
x2 y2	second point
posA	first point
posB	second point
fraction	fraction of the distance AB

EXAMPLE

```
puts [geom.online 10 10  24 10  0.6]; # Writes 18.4 10.0
```

3.7.13 geom.angle

Computes the angle (in degrees) between three points. A is the center point. If the argument is a vector its direction (azimuth) is computed.

SYNOPSIS
geom.angle *xA yA xP yP xQ yQ*
geom.angle *posA posP posQ*

geom.angle *xP yP xQ yQ*
geom.angle *posP posQ*
geom.angle *vector*

xA yA	Center point. If omitted the current cursor position is assumed.
xP yP	first distant point
xQ yQ	second distant point
posA	Center point. If omitted the current cursor position is assumed.
posP	first distant point
posQ	second distant point
vector	vector

3.7.14 geom.anglerad

Computes the angle (in radians) between three points. A is the center point. If the argument is a vector its direction (azimuth) is computed.

SYNOPSIS
geom.anglerad *xA yA xP yP xQ yQ*
geom.anglerad *posA posP posQ*
geom.anglerad *xP yP xQ yQ*
geom.anglerad *posP posQ*
geom.anglerad *vector*

xA yA	Center point. If omitted the current cursor position is assumed.
xP yP	first distant point
xQ yQ	second distant point
posA	Center point. If omitted the current cursor position is assumed.
posP	first distant point
posQ	second distant point
vector	vector

3.7.15 geom.crossproduct

Computes the cross product of two vectors. The command returns the scalar z component of the resulting vector.

SYNOPSIS
geom.crossproduct *dx1 dy1 dx2 dy2*
geom.crossproduct *v1 v2*

dx1 dy1	first vector
dx2 dy2	second vector
v1	first vector
v2	second vector

EXAMPLE

```
puts [geom.crossproduct  3 0.5  2 2]; # Writes 5.0
```

3.7.16 geom.dotproduct

Computes the (scalar) dot product of two vectors.

SYNOPSIS
geom.dotproduct *dx1 dy1 dx2 dy2*
geom.dotproduct *v1 v2*

dx1 dy1	first vector
dx2 dy2	second vector
v1	first vector
v2	second vector

EXAMPLE

```
puts [geom.dotproduct 3 0.5  2 2]; # Writes 7.0
```

3.7.17 geom.rotate

Returns a vector that is rotated by an angle (in degrees) compared to the input vector.

SYNOPSIS
geom.rotate *dx dy θ*
geom.rotate *v θ*
geom.rotate *v*

dx dy	vector
v	vector
θ	Angle in degrees. If omitted 90°

EXAMPLE

```
puts [geom.rotate {1 0} 30]; # Writes 0.866 5.0
puts [geom.rotate 1 0];      # Writes 0.0 1.0
```

3.7.18 geom.polar

Returns a vector that is rotated by an angle (in degrees) from the x-axis. If the length is omitted the unit vector {1 0} is assumed to be rotated, otherwise {dL 0}.

SYNOPSIS
geom.polar *dL α*
geom.polar *α*

dL length
α azimuth (angle to x-axis, in degrees, clock-wise)

EXAMPLE

```
puts [geom.polar 2.0 60];  # Prints 1.000 1.732
puts [geom.polar 60];      # Prints 0.500 0.866
```

3.7.19 geom.length (geom.hypot, geom.abs)

Computes the length of a vector. Thus returns the hypotenuse of two catheti in a right-angled triangle.

SYNOPSIS
geom.length *dx dy*
geom.length *v*

dx dy vector
v vector

EXAMPLE

```
puts [geom.length 30 40]; # Writes 50.0
```

3.7.20 geom.norm

Returns a unit length vector with the same direction as the input vector.

SYNOPSIS
geom.norm *dx dy*
geom.norm *v*

dx dy vector
v vector

EXAMPLE

```
puts [geom.norm 30 40]; # Writes 0.6 0.8
```

3.7.21 geom.parallel

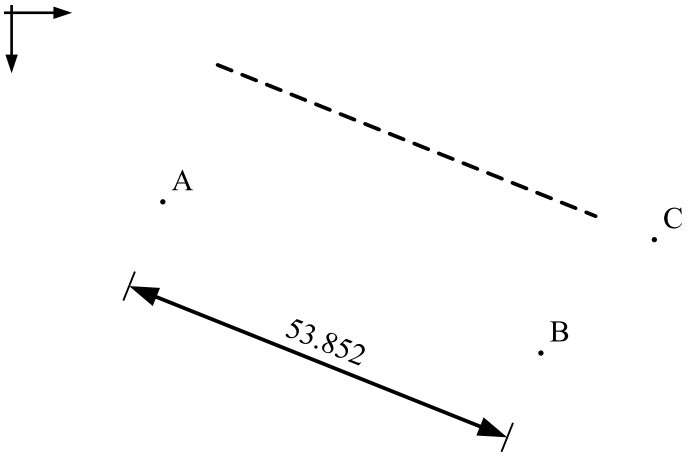
Returns a parallel line defined by two points.

SYNOPSIS
geom.parallel *x1 y1 x2 y2 distance*
geom.parallel *pos1 pos2 distance*
geom.parallel *x1 y1 x2 y2*
geom.parallel *pos1 pos2*

x1 y1	first point of existing line
x2 y2	second point of existing line
pos1	first point of existing line
pos2	second point of existing line
distance	Distance. If omitted new parallel line goes through the current cursor position.

EXAMPLE

```
set A {20 25}; point $A; label A
set B {70 45}; point $B; label B
set C {85 30}; point $C; label C
# Draw a dimension line at 12mm distance to AB
dimline [geom.parallel $A $B [mm 12]]
pen dashed
# Draw a parallel line through C (current cursor position)
moveto $C
line [geom.parallel $A $B]
```



3.7.22 geom.extend

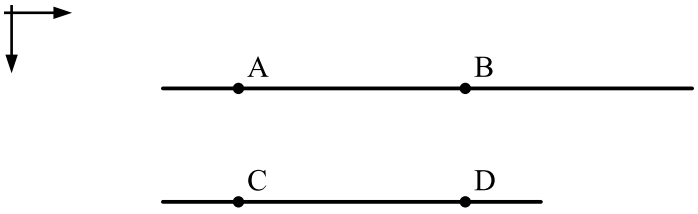
Returns an extended line defined by two points.

SYNOPSIS
geom.extend *x1 y1 x2 y2 dLA dLB*
geom.extend *pos1 pos2 dLA dLB*
geom.extend *x1 y1 x2 y2 dL*
geom.extend *pos1 pos2 dL*

x1 y1	first point of existing line
x2 y2	second point of existing line
pos1	first point of existing line
pos2	second point of existing line
dLA dLB	extension values
dL	extension value at both sides

EXAMPLE

```
set A {30 10}; dot $A; label A
set B {60 10}; dot $B; label B
line [geom.extend $A $B 10 30]
set C {30 25}; dot $C; label C
set D {60 25}; dot $D; label D
line [geom.extend $C $D 10]
```



3.7.23 geom.distance (geom.dist)

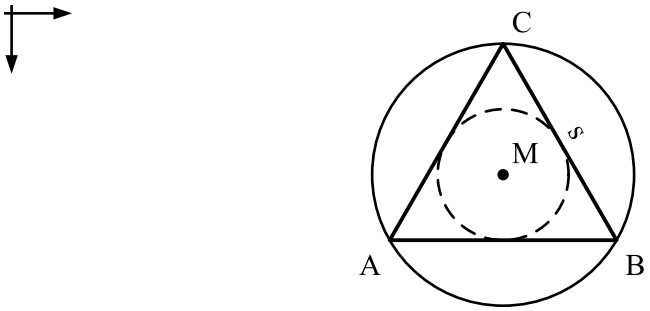
Computes the distance of a line (or position) to the current position.

SYNOPSIS
geom.distance *x1 y1 x2 y2*
geom.distance *pos1 pos2*
geom.distance *x1 y1*
geom.distance *pos1*

x1 y1	first point of existing line or just point
x2 y2	second point of existing line
pos1	first point of existing line or just point
pos2	second point of existing line

EXAMPLE

```
set s 30
set A {50 30}
set B [++ $A "$s 0"]
set C [++ $A "[expr $s /2] [expr 0-sqrt(3)*$s /2]"]
m $A; label A SW
m $B; label B SE
m $C; label C
label s $B $C
polygon $A $B $C; # cursor now is at $A
moverel [expr $s /2] [expr 0-sqrt(3)*$s /6]
dot; label M
pen 0.35
circle [geom.distance $A]
pen dashed
circle [geom.distance $A $B]
```



3.8 Statics commands

The statics commands stat.* are utility commands, that do calculations with forces "x y Fx Fy". The commands do not draw.

3.8.1 stat.add (+++)

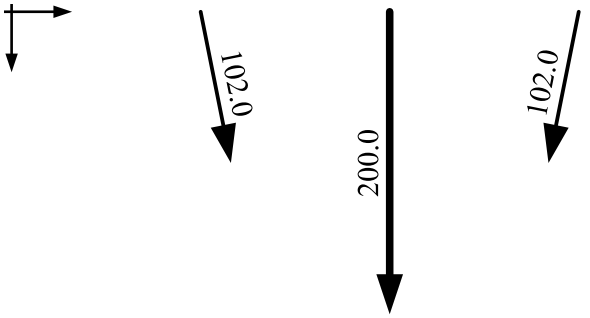
Computes the force vector sum and a valid application point.

SYNOPSIS
stat.add *F1 F2 ...*
+++ *F1 F2 ...*

F1 first force
F2 second force

EXAMPLE

```
set F1 {5.0 0.0  20 100}
set F2 {15.0 0.0  -20 100}
force $F1
force $F2
pen 1.0
set R  [stat.add $F1 $F2]
force $R
puts $R;                    # Writes 10.0 0.0 0.0 200.0
```



3.8.2 stat.subtract (---)

Computes the force vector subtraction and a valid application point.

SYNOPSIS
stat.subtract *F1 F2 ...*
--- F1 F2 ...

F1 first force (sum)
F2 second force (to be subtracted from sum)

EXAMPLE

```
set F1 { 5.0 0.0    0 100}
set F2 {15.0 0.0  -20 100}
set R  [stat.add $F1 $F2]
puts [stat.subtract $R $F2]; # Writes 5.0 0.0 0.0 100.0
```

3.8.3 stat.multiply (***)

Computes a force scaled by multiplication. The application point does not change.

SYNOPSIS
stat.multiply *factor force ...*
*** *factor force ...*

factor scaling factor
force force "x0 y0 Fx Fy"

EXAMPLE

```
puts [*** 1.5 {5.00 0.00  20 30}]; # Writes 5.0 0.0 30.0 45.0
```

3.8.4 stat.move

Returns an equivalent force to the input force. The application point is moved along the action line of the force, until it intersects the extension of a line $\overline{AB}$.

SYNOPSIS
stat.move *x0 y0 Fx Fy xA yA xB yB*
stat.move *F posA posB*

x0 y0 application point of input force
Fx Fy components of input force
xA yA first point of line AB
xB yB second point of line AB

F	input force "x0 y0 Fx Fy"
posA	first point of line AB
posB	second point of line AB

EXAMPLE
see below

3.8.5 stat.move2

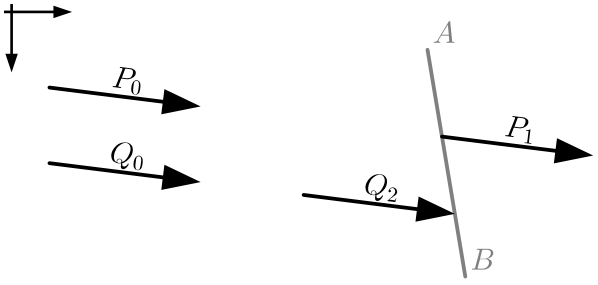
Returns an equivalent force to the input force. The application point is moved along the action line of the force, until the arrow head touches the extension of a line $\overline{AB}$.

SYNOPSIS
stat.move2 *x0 y0 Fx Fy xA yA xB yB*
stat.move2 *F posA posB*

x0 y0	application point of input force
Fx Fy	components of input force
xA yA	first point of line AB
xB yB	second point of line AB
F	input force "x0 y0 Fx Fy"
posA	first point of line AB
posB	second point of line AB

EXAMPLE

```
unitlength 10; forceunitlength 0.5
pen gray
set A {5.5 0.5}; m $A; tlb A
set B {6.0 3.5}; m $B; tlb B
l $A $B
pen
set P0 {0.5 1.0  40 5}; texforce $P0 P_0
set Q0 {0.5 2.0  40 5}; texforce $Q0 Q_0
texforce [stat.move $P0 $A $B] P_1
texforce [stat.move2 $Q0 $A $B] Q_2
```



3.8.6 stat.actionline

Returns two points on the action line of a force.

SYNOPSIS
stat.actionline *x0 y0 Fx Fy fA fB*
stat.actionline *F fA fB*
stat.actionline *x0 y0 Fx Fy f*
stat.actionline *F f*
stat.actionline *x0 y0 Fx Fy*
stat.actionline *F*

x0 y0	application point of input force
Fx Fy	components of input force
F	input force "x0 y0 Fx Fy"
fA fB	extension factors (left, right), analogue to geom.online
f	extension factor

EXAMPLE
see below

3.8.7 stat.tip

Returns the tip position of the force arrow.

SYNOPSIS
stat.tip *x0 y0 Fx Fy*
stat.tip *F*

x0 y0	application point of input force
Fx Fy	components of input force
F	input force "x0 y0 Fx Fy"

3.8.8 stat.abs

Computes the absolute value of a force: $\sqrt{F_x^2 + F_y^2}$.

SYNOPSIS
stat.abs *x0 y0 Fx Fy*
stat.abs *F*

Fx Fy	force components
x0 y0	Application point. Does not influence the result.
F	force "x0 y0 Fx Fy"

EXAMPLE
see below

3.8.9 stat.distance (stat.dist)

Computes the distance of a force to the current position.

SYNOPSIS
stat.distance *x0 y0 Fx Fy*
stat.distance *F*

x0 y0	application point of force
Fx Fy	components of force
F	force "x0 y0 Fx Fy"

EXAMPLE
see below

3.8.10 stat.moment

Computes the moment of a force relative to the current position.

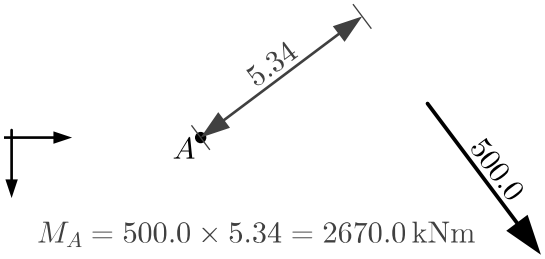
SYNOPSIS
stat.moment *x0 y0 Fx Fy*
stat.moment *F*

x0 y0	application point of force
Fx Fy	components of force
F	force "x0 y0 Fx Fy"

EXAMPLE

```
unitlength [/ 1. 200.] m
forceunitlength [/ 5. 100.]
set A { 5.0 0}
set F {11.0 -0.90  300 400}
puts [stat.abs $F];                 # Writes 500.0
moveto $A
set r [stat.distance $F]; puts $r;  # Writes 5.34
set M [stat.moment $F]; puts $M;    # Writes 2670.0

# Draw the point A, the force F and distance A-F
dot $A; tlb A SW
texforce $F
pen 0.35 darkgray
m $A; texdimlinerel [geom.rotate [** $r [geom.norm [FXY $F]]] -90]
# Write the moment M_A of the force F
moveto 0.5 3.0
texlabel "M_A = [stat.abs $F] \\times $r = $M \\,\\text{kNm}"
```



3.8.11 stat.mequi

Scales an input force until moment equilibrium is fulfilled. Moment equilibrium is done around the current cursor position.

SYNOPSIS
stat.mequi *xB yB FxB FyB x0 y0 Fx Fy*
stat.mequi *FB F*
stat.mequi *FB M*

xB yB	position of sliding bearing
FxB FyB	force components indicating direction of support (absolute value has no influence)
FB	support force, absolute value has no influence
F	force "x0 y0 Fx Fy" to be equilibrated
M	moment to be equilibrated

EXAMPLE
see below

3.8.12 stat.equi

Returns the force (components and application point) that is required to complete equilibrium. Equivalent to [*** -1 [+++ $F1 $F2]].

SYNOPSIS
stat.equi *F1 F2 ...*

F1	first force
F2	second force

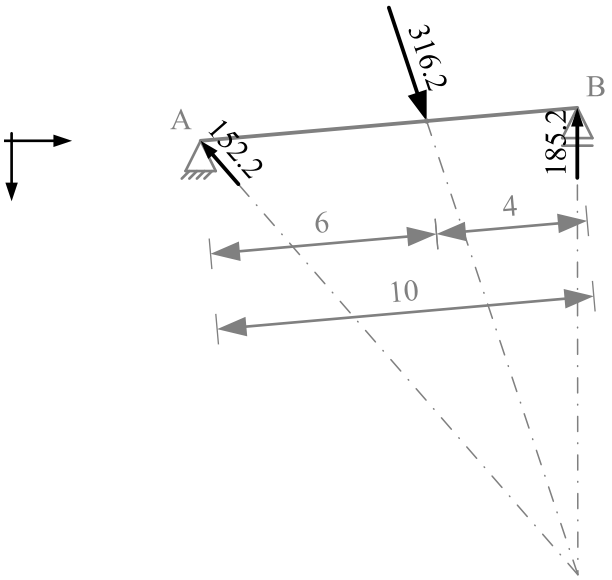
EXAMPLE

```
unitlength [/ 1. 200.] m
forceunitlength [/ 5. 100.]
set A {5 0}
set B "[++ $A [geom.rotate {10 0} -5]]"
set C [geom.online $A $B 0.6]
set F "$C 100 300"
```

```
force2 $F

# Draw beam, points, bearings (not shown), dimension lines
pen gray; moveto $A; label A NW
lineto $B; label B; point $C
dimline [geom.parallel $A $C [mm 15]]
dimlinerel [-- "$B" "$C"]
dimline [geom.parallel $A $B [mm 25]]
# graphical statics
pen gray dashdotted 0.2
line [stat.actionline $F 1 4]
line $B [here]
lineto $A

# Calculate reaction forces
pen solid red 0.5
moveto $A;  # moment equilibrium at pos. A in order to get force FB
set fB "$B 0 1"
set FB [stat.mequi $fB $F]
force2 $FB
set FA [stat.equi $F $FB];  # equilibrium to get force FA
set FA [stat.move $FA $A $B]
force2 $FA
```



3.8.13 stat.fequi

Equilibrates a known force F by two other forces A and B, of which the azimuths are known. The command returns the force A. Equilibrium is defined F + A + B = 0. Then B is [stat.equi $F $A].

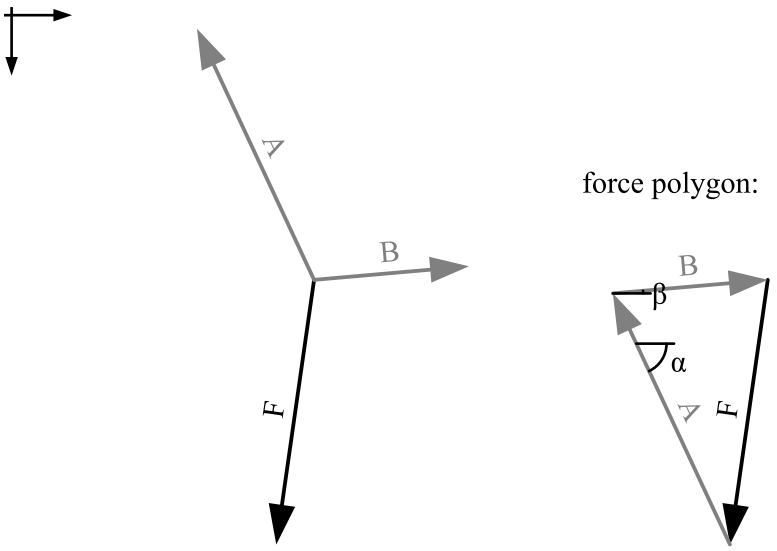
SYNOPSIS
stat.fequi *F α β*

F1 known force
α azimuth (in degrees) of searched force A

β azimuth (in degrees) of force B

EXAMPLE

```
forceunitlength [/ 1. 10.]
set F {40 35  -50 350}
force $F F
set A [stat.fequi $F 65 -5]
set B [stat.equi $F $A]
pen gray
force $A A
force $B B
pen black;  # Draw the force polygon
m 100 25; lb {force polygon:} NW
moveto 100 35
force [FXY $F] F
pen gray
force [FXY $A] A
force [FXY $B] B
assert "[X [here]] == 100 && [Y [here]] == 35" "Force polygon not closed"
```



3.9 Programming commands

For a programming introduction check the hecl tutorial http://hecl.org/docs/tutorial.html.

For a complete reference, check the hecl command reference http://hecl.org/docs/commands.html

3.9.1 Variables: set, $

set — Set a variable.

SYNOPSIS
set *varname* [*value*]

Description
The set command sets the *value* of a variable *varname* to value *value*. If *value* is not provided, returns the value of *varname*.

EXAMPLE

```
set foo "bar"
set bee bop
puts "foo is $foo and bee is $bee"
```

Produces:
1
foo is bar and bee is bop

3.9.2 Math commands

Floating point math commands. See also **expr**.

abs acos acos asin asin atan atan cbrt ceil cos cosh cosh exp expm1 floor hypot log log10 log1p pow random round signum sin sinh sqrt tan tanh tanh

SYNOPSIS
command number [*number*]

3.9.3 External script: source

The source command evaluates the Hecl script located in file *filename*.

SYNOPSIS
source *filename.hcl*

EXAMPLE

```
# Variable foo is defined as "Hello world" in foo.hcl
source foo.hcl
puts $foo
```

Produces:
Hello world

3.9.4 Conditions: if

if — Conditionally execute code.

SYNOPSIS
if *test code* [elseif | *test* | *code* ...] [else | *code*]

Description
The if command executes Hecl code conditionally. In its most basic form, it executes a test. If the results are not 0, then it executes code. If not, no further actions take place. if may take any number of elseif

clauses, which have their own test and code. Finally, if none of the conditions has matched, it is also possible to supply an else clause that will be executed if the results of the if and elseif tests were all false.

EXAMPLE

```
if { > 0 1 } {
    puts "true"
} else {
    puts "false"
}
```

Produces:
false

3.9.5 Loops: for, foreach

for — For loop.

SYNOPSIS
for *initialization test step body*

Description
The **for** command is like in many other languages like C and Java. As arguments, it takes an *initialization* option, which is often used to set a variable to some initial value, a *test* to determine whether to continue running, a *step* script option which is run at each iteration of the body (to increment a variable, for example), and the body itself.

EXAMPLE

```
set out {}
for {set i 0} {< $i 10} {incr $i} {
    append $out $i
}
puts $out
```

Produces: 0123456789

foreach — Iterate over elements in a list.

SYNOPSIS
foreach *varname list body*
foreach *varlist list body*

Description
The foreach command iterates over a list. For each element of the list, *varname* is set to a new element of the *list*, and then *body* is run.

EXAMPLE

```
set lst {a b c d e}
set res {}
foreach el $lst {
    append $res $el
}
puts $res
```

Produces: abcde

3.9.6 Procedures: proc, rename

The **proc** command creates new procedures, which are virtually indistinguishable from built-in Hecl commands. By default, Hecl variables are always local. Global variables are not visible from within procedures. The **global** command makes global variable *varname* visible within a procedure.

SYNOPSIS
proc *[name] arglist body*
global *varname [varname...]*

EXAMPLE

```
set debug false
proc ignore {command} {
        global debug
        foreach cmd $command {
                proc $cmd {args} {global debug; if {$debug} {puts "command
                    ignored"}}
                if {$debug} {puts "Ignore command: $cmd"}
        }
}
# ignore list
ignore {model geomTransf uniaxialMaterial}
ignore {eigen timeSeries pattern load loadConst rayleigh}
ignore {constraints numberer system test}
ignore {algorithm integrator analysis analyze}
ignore {open close recorder wipeAnalysis wipe}
```

rename — Rename a command

rename *cmdname newcmdname*

EXAMPLE

```
rename expr exprhecl
proc expr {args} {return $args}
```

3.9.7 Hash tables

hash — Create and manipulate hash tables.

SYNOPSIS
hash *list*
hget *hash key*
hset *hash key value*
hcontains *hash key*
hclear *hash*
hkeys *hash*
hremove *hash key*

Description
The **hash** command takes an even-numbered list and creates a hash table from it, using the even elements as keys, and odd elements as values. A new hash table is returned. The **hget** and **hset** commands operate on hash tables. Both take a hash table as their first argument. hget also takes a key, and returns the corresponding value, or an error if no key by that name exists. To determine whether a given key exists, use the **hcontains** command, which returns true or false depending on whether the key exists in the hash table.

The **hkeys** command returns the keys of the hash table, as a list.

The **hclear** command clears an entire hash table, whereas **hremove** removes the value associated with a given key.

EXAMPLE

```
set foo [hash {a b c d}]
hset $foo a 42
puts [hget $foo a]
```

Produces:
42

EXAMPLE

```
set nd [hash {}]; # Hash list nd saves node coordinates.
proc node {number x y} {
        global nd
        set Y [* -1 $y]; # Opensees y coordinate points upwards, drawj2d
            downwards.
        hset $nd $number "$x $Y"
        circle $x $Y [mm 0.4]
        if {>= $x 0} {label $number NE} else {label $number NW}
}
```

4 Support for spread sheet csv data and Fachwerk background drawings bgd

Use the Drawj2d command line parameter *--frontend bgd* or *-F bgd*.

csv (spread sheet)

A list of point coordinates can be created in a spread sheet application. The list has to be saved as comma separated file with the ending *csv*. Drawj2d displays the points. The scale is automatically chosen (1:5, 1:10, 1:20, 1:50, ...).

```
drawj2d -T pdf -F bgd --width 150 --height 100 points.csv
```

bgd (text file)

The program Fachwerk for structural engineers (fachwerk.sourceforge.net) supports a simple text based drawing format called bgd for background drawings. These can be exported by Drawj2d to pdf, svg or other vector formats.

```
drawj2d -T pdf -F bgd --width 150 --height 80 doku.bgd
```

Bgd drawings are created using a text editor, similar to the Drawj2d native *hcl* format. The file has to be saved with the suffix *bgd*. Figure 2 shows an example of such a text file and the drawing as it would be displayed by Fachwerk.

The numbers in a row can be separated by space, tabulator or comma. The order of the commands (Point, Line, Circle, Arc) does not matter. For example the file may start with commands for lines, continue with points and then list a line command again. Only the first character of a command is required (Circle: *C* or *K* for Kreis (German), Arc: *A* or *B* for Bogen). Rows starting with # or // or with the word *rem* indicate comments. Fachwerk accepts coordinates in both *x,z* and *x,y,z* formats. In Fachwerk2D mode and Drawj2D the y-value is not used.

The bgd extension is an example for programmers how to access the Drawj2d java API.

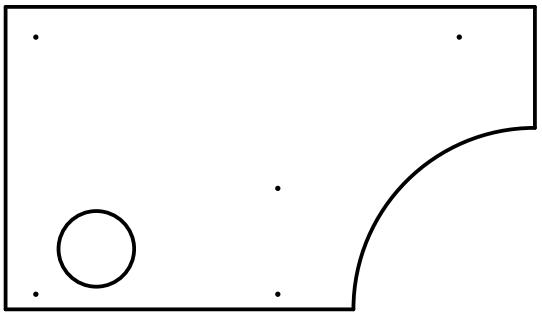
```
Points
0.2   0.2          ← x, z - coordinates 1. point
0.2   1.9          ← x, z - coordinates 2. point
1.8   1.2
1.8   1.9
3.0   0.2

Line
3.5   0            ← start
3.5   0.8          ← end

Line
2.3   2.0          ← start
0     2.0
0     0
3.5   0            ← end

Circle
0.6   1.6          ← centre
0.25               ← radius

Arc
3.5   2.0          ← centre
1.2, 90, 180       ← radius, start angle, end angle (anti-clockwise)
```



1:50.0

Figure 2: Input text file *.bgd* and resulting drawing

5 Support for Yacas plot data

The computer algebra system Yacas (yacas.org) uses Gnuplot for plotting. If gnuplot is not available, yacas can prepare plot data using the Plot2D option *output=java*. Drawj2d can read the plot data and draw rudimentary plots.

Use the Drawj2d command line parameter *--frontend ypd* or *-F ypd*.

Start Yacas (yacas-gui, yacas, java -jar yacas.jar or alternatively mavscript-yacas) and type:

```
f(x) := x^2-x-2        ← The function to plot
g(x) := x^2            ← A second function to plot
```

Write a yacas plot data (ypd) file.
`ToFile("plot-f.ypd") Plot2D(f(x),x=-5:5,output=java)`

Multiple functions can be written in one plot data file:
`ToFile("plot-fg.ypd") Plot2D({f(x),g(x)},x=-5:5,output=java)`

Call Drawj2d directly out of Yacas to produce a plot (see Figure 3):
`SystemCall("drawj2d -F ypd -W150 -H100 plot-f.ypd")`

Drawj2d can also be called from the command line.
`drawj2d -T pdf -F ypd --width 150 --height 100 plot-fg.ypd`

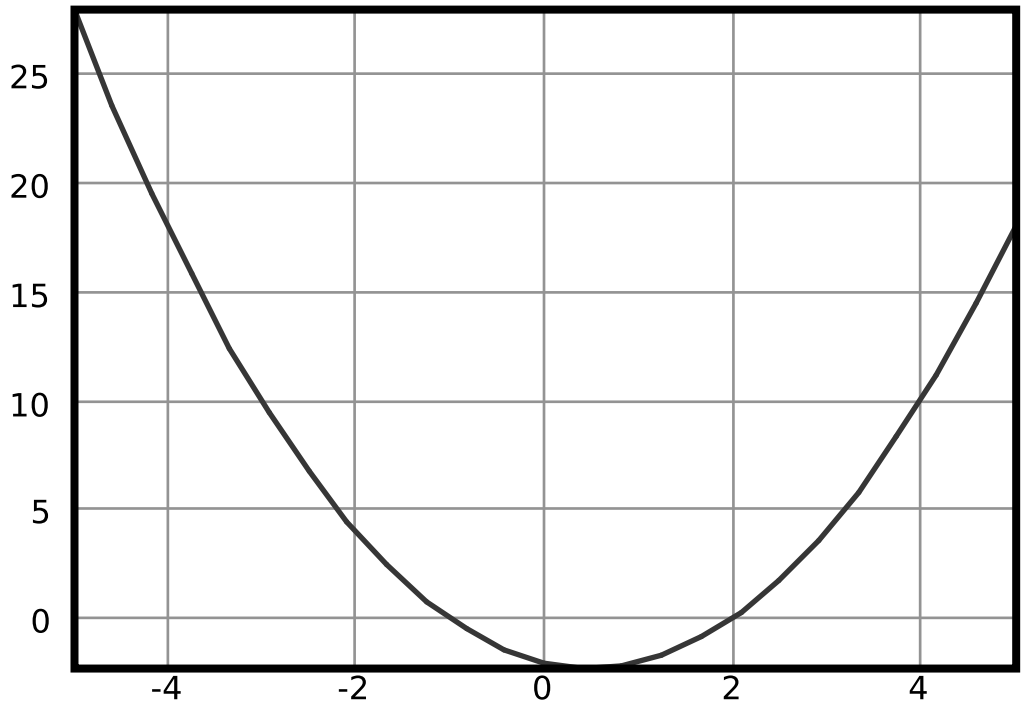


Figure 3: Draw yacas plot data: $f(x)$

6 Drawj2d Input Examples

6.1 Drawings

Anchor plate

Input of the drawing Figure 1.

drawing.hcl

```
#! drawj2d -T pdf -W 150 -H 120 -c drawing.hcl

# Nullpunkt ist Plattenmittelpunkt
# SI-Einheiten: mm, kN
unitlength 0.2 mm;    # Das heisst Massstab 1:5, bei Einheit Millimeter

# Eingaben
# --------
set b   350;    # Plattenbreite
set h   350;    # Plattenhöhe
set t    20;    # Plattenstärke
set d    12;    # zu zeichnender Dübeldurchmesser im Durchgangsloch
set df   20;    # Durchgangsloch in Ankerplatte
set c    40;    # nomineller Randabstand Dübel

# Profil zeichnen
box -50 -50  50 50
box -45 -45  45 45


# Eingabekontrollen
assert "$b > 4*$c"      {Eingabekontrolle Plattenbreite}
assert "$h > 4*$c"      {Eingabekontrolle Plattenhöhe}
assert "$t >= 8"        {Eingabekontrolle Plattenstärke}
assert "$d > 5"         {Eingabekontrolle Dübeldurchmesser}
assert "$df > 5"        {Eingabekontrolle Durchgangsloch}

# Zeichnen der Achsen
set ueberstand  10; # Achsüberstand
pen gray dashdotted 0.35
m 0 0
linemid [expr $b + 2 * $ueberstand] 0
linemid [expr $h + 2 * $ueberstand] 90

# Zeichnen der Ankerplatte
set TL [// "-$b -$h" 2]; set TR [// "$b -$h" 2]
set BL [// "-$b  $h" 2]; set BR [// "$b  $h" 2]
pen black solid 0.7;
box $TL $BR

# Zeichnen des Dübels
set DbTL [++ $TL "$c $c"];      set DbTR [++ $TR "-$c $c"]
set DbBL [++ $BL "$c -$c"];     set DbBR [-- $BR "$c $c"]
pen gray 0.5;
fillcircle $DbTL  [/ $d 2.];    fillcircle $DbTR  [/ $d 2.]
fillcircle $DbBL  [/ $d 2.];    fillcircle $DbBR  [/ $d 2.]
```

```
pen 0.35 black
circle $DbTL  [/ $df 2.];       circle $DbTR  [/ $df 2.]
circle $DbBL  [/ $df 2.];       circle $DbBR  [/ $df 2.]

# Vermassung
set xdim  [mm 8]
# Ankerplatte
pen 0.35
dimline [geom.parallel $BL $BR $xdim]
dimline [geom.parallel $TL $BL $xdim]
p 150 60; lr [mm 40 0]; label "t = $t mm" NW
# Dübel
pen gray
m $TR; mr $xdim 0
dimlinerel 0 $c
dimlineto [ty $DbBR]
dimlineto [ty $BR]
m $TL; mr 0 -$xdim
dimlineto [tx $DbTL]
dimlineto [tx $DbTR]
dimlineto [tx $TR]
```

Pythagoras

Pythagoras $a^2 + b^2 = c^2$ (Figure 4).

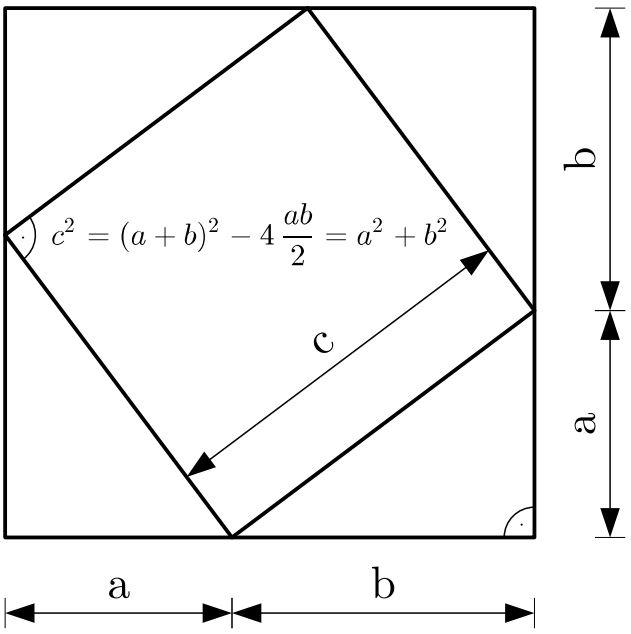


Figure 4: Pythagoras

pythagoras.hcl

```
#! drawj2d -W 100 -H 90 pythagoras.hcl
```

```
# original example from asymptote gallery
offset 80 75
unitlength 10

set a 3
set b 4

# vertices
set ML "-[+ $a $b] -$b" ; # middle left
set BM "-$b 0"          ; # bottom middle
set BL "-[+ $a $b] 0"   ; # bottom left

# draw squares
m 0 0; rectangle -[+ $a $b] -[+ $a $b];
m $BM; lr $b -$a; lr -$a -$b; lr -$b $a; l $BM;

# draw dimension lines
pen black 0.2
font tex 6; # Sets font to computer modern, 6mm
set d [mm 10]
m [++ $BL "0 $d"]
dimlinerel $a 0  a
dimlinerel $b 0  b
dimline [geom.parallel "0 -$a" $BM $d] c
m "$d 0"
dimlinerel 0 -$a a
dimlinerel 0 -$b b

# draw perpendicular sign
pen black
dimangle $ML [++ $ML "$b -$a"] $BM
m "0 0"; dimangle 180 270

# write equation
m $ML; mr [mm 4  4]; font
texlabel {\displaystyle  c^2 = (a+b)^2 - 4 \frac{a b}{2} = a^2 + b^2}
```

6.2 Statics

Drawj2d has built in support for basic statics, see 3.8. It is useful for geotechnical tasks, e.g. earth pressure and retaining walls. The examples below are about cable equilibrium.

Cable equilibrium

For a given geometry the cable forces are calculated (Figure 5).

cable.hcl

```
#! drawj2d -W 150 -H 80 -X 20 -Y 20 cable.hcl

unitlength [/ 1. 50.] m
forceunitlength 2
```

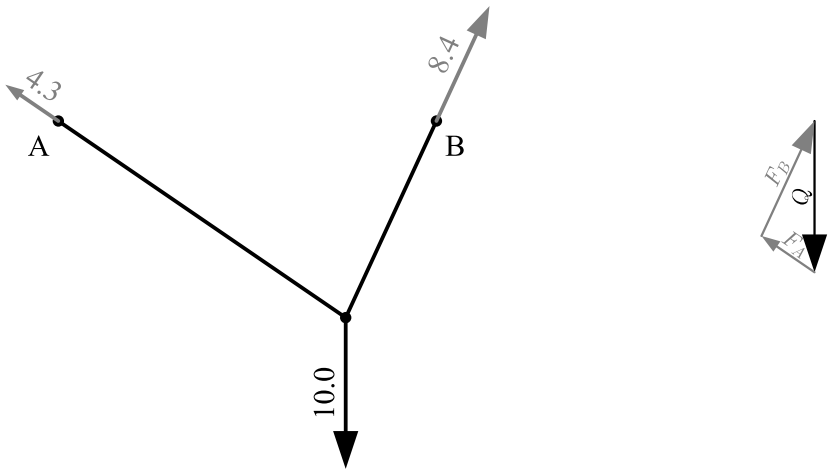


Figure 5: Cable equilibrium

```
# cable
set A {0 0}; dot $A; lb A SW
set B {2.5 0}; dot $B; lb B SE
set C {1.9 1.3}; dot $C
line $A $C $B

# weight (10 kN)
set Q "$C  0 10"
force $Q

# calculate reaction forces
pen gray
# moment equilibrium at pos. A in order to get force FB
moveto $A
set fB "$B  [geom.vector $C $B]"
set FB [stat.mequi $fB $Q]
force $FB
# equilibrium to get force FA
set FA [stat.equi $Q $FB]
set FA [stat.move $FA $A $B]; # move along action line
force $FA

# force polygon
moveto 5 0
set posPolygon [here]
pen black 0.3; font 3
texforce [FXY $Q] Q
pen gray
texforce [FXY $FA] F_A
texforce [FXY $FB] F_B
# verify the force polygon is closed
assert "[geom.distance $posPolygon] ~= 0" {equilibrium check}
```

Cable shape

In this example the cable shape is calculated for multiple external forces (Figure 6). The bearing points and the initial cable angle (at point A) are given.

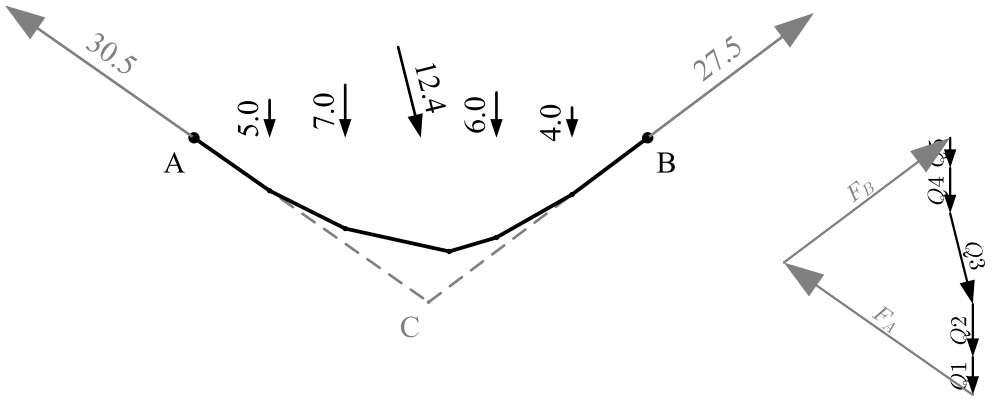


Figure 6: Cable shape

cable2.hcl

```
#! drawj2d -W 150 -H 60 -X 30 -Y 20 cable2.hcl

unitlength [/ 1. 50.] m
forceunitlength 1

# bearings
set A {0 0}; dot $A; lb A SW
set B {3.0 0}; dot $B; lb B SE
# cable angle at A
set alpha 35

# forces acting on the cable
pen 0.35
set Q1 "0.5 0  0 5";  force2 $Q1
set Q2 "1.0 0  0 7";  force2 $Q2
set Q3 "1.5 0  3 12"; force2 $Q3
set Q4 "2.0 0  0 6";  force2 $Q4
set Q5 "2.5 0  0 4";  force2 $Q5

# resultant
set R [stat.add $Q1 $Q2 $Q3 $Q4 $Q5]
set C [XY [stat.move $R $A [++ $A [geom.polar $alpha]]]]
pen dashed gray
point $C; lb C SW
line $A $C $B

# global moment equilibrium at pos. B in order to get force FA
pen solid gray
moveto $B
set fA "$A  [geom.polar $alpha]"
set FA [stat.mequi $fA $R]
force $FA
```

```
# equilibrium to get force FB
set FB [stat.equi $R $FA]
set FB [stat.move $FB $A $B]; # move along action line
force $FB

# cable geometry
pen
set F1 [stat.equi $FA $Q1]
set F2 [--- $F1 $Q2]
set F3 [--- $F2 $Q3]
set F4 [--- $F3 $Q4]
set F5 [--- $F4 $Q5]
set C1 [XY [stat.move $Q1 [XY $FA] [stat.tip $FA]]]; point $C1
set C2 [XY [stat.move $Q2 [XY $F1] [stat.tip $F1]]]; point $C2
set C3 [XY [stat.move $Q3 [XY $F2] [stat.tip $F2]]]; point $C3
set C4 [XY [stat.move $Q4 [XY $F3] [stat.tip $F3]]]; point $C4
set C5 [XY [stat.move $Q5 [XY $F4] [stat.tip $F4]]]; point $C5
line $A $C1 $C2 $C3 $C4 $C5 $B
moveto $A; # verify the forces F5 and FB are equal
assert "[stat.moment $F5] ~= [stat.moment $FB]" {moment check F5 = FB}
assert "[stat.abs $F5] ~= [stat.abs $FB]" {check F5 = FB}

# force polygon
moveto 5 0; set posPolygon [here]
pen black 0.3; font 3
texforce [FXY $Q5] Q5
texforce [FXY $Q4] Q4
texforce [FXY $Q3] Q3
texforce [FXY $Q2] Q2
texforce [FXY $Q1] Q1
pen gray
texforce [FXY $FA] F_A
texforce [FXY $FB] F_B
# verify the force polygon is closed
assert "[geom.distance $posPolygon] ~= 0" {equilibrium check}
```