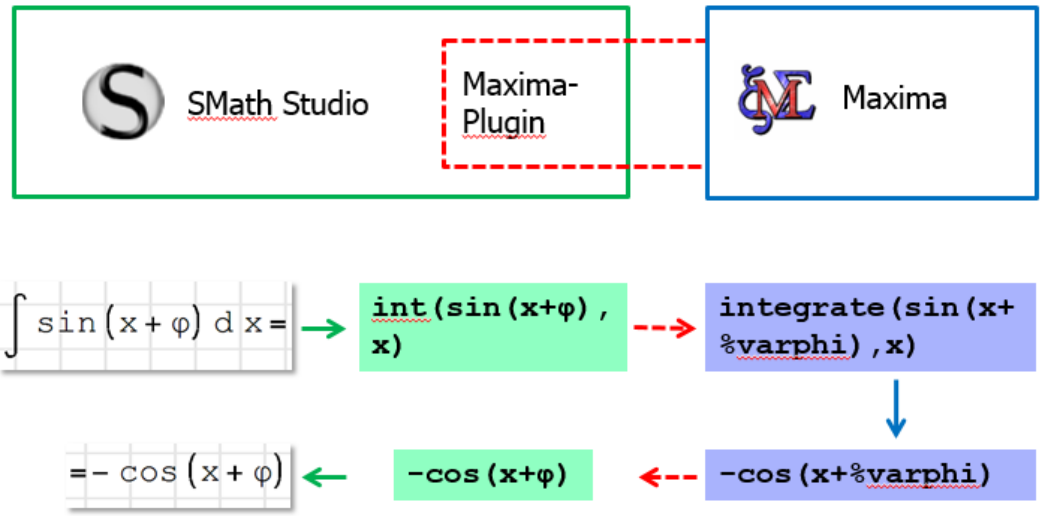
# SMath with Maxima

## Getting Started with the Maxima Plugin

Prof. Dr.-Ing. Martin Kraska
kraska@th-brandenburg.de




September 29, 2023

Martin Kraska
SMath with Maxima.
Technische Hochschule Brandenburg, 2023
DOI: 10.25933/opus4-2949

# Contents

# 1 Introduction

## Contents



SMath is a program for performing and documenting engineering calculations. An SMath document consists of formulas, texts, diagrams, and images freely arranged on the page. Formulas can be assembled from calculator-like palettes or from the keyboard without using the mouse (much faster for experienced users). There is excellent support for units of measurement. The formulas serve as both computational instructions and documentation. Formulas appear in natural mathematical notation (not like program code), allowing even those unfamiliar with the program to understand them.

SMath originated as a hobby project by Andrey Ivashov, a software developer from Saint Petersburg. It is now developed and supported by LLC SMath (ООО Эсмат), of which Andrey Ivashov is the managing director. The program gained significance within the context of the sanction relief efforts for the Russian economy and science, as it can replace the conceptually similar MathCAD by PTC (USA).

SMath is closed source commercial software. There is a free personal license plan (requiring registration). This plan has no restrictions of the mathematical features but does not include commercial use.

SMath features an integrated extension manager. This provides access to an online gallery where the community can publish program extensions (plug-ins), application examples, documentation, and code snippets. The present manual is also available there.

The author of this manual has extensive experience in using SMath Studio for research and teaching at the University of Applied sciences Brandenburg (THB) in Brandenburg an der Havel, Germany. He is active on the user forum at smath.info. Major contributions to the community are a comprehensive German handbook (Kraska 2020) and the Maxima-plugin, an interface to the free and open-source computer algebra program Maxima. This interface provides features such as symbolic integration and equation solving, as well as

three-dimensional diagrams. This is what could be Germany's concept of success: Take the best from East and West and combine it for the benefit of everyone.

This manual was written in parallel to a major rework of the Maxima Plugin and wants to encourage SMath users to actually try it.

We assume that you are already familiar with SMath and have a working installation at hand. Knowledge of Maxima would be helpful but is not mandatory.

Some chapters have file attachments (below the chapter table of contents). You can find them in the attachment view of your PDF viewer.

## 1.1 Acknowledgements

The initial version of the Maxima Plugin has been implemented in 2013 by Kay Graubmann, student at University of Applied Sciences Brandenburg (THB).

Major rework is done by Muizzuddin Fauzi, student at THB in 2023 as part of his master thesis.

The code was inspired by the Maple Wrapper plugin and the Image Region plugin by Viacheslav Mezentsev and by the Statistical Tools plugin by Davide Carpi.

Andrey Ivashov provided the base for all this and helped with publishing the Maxima plugin in the online gallery.

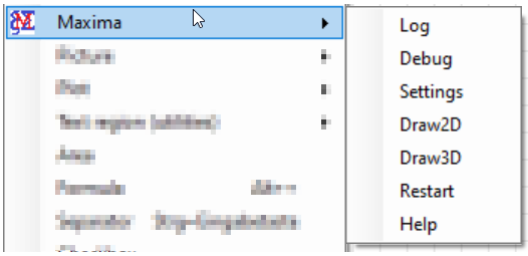
Davide Carpi and Andrey Ivashov did maintenance work (adjustment to SMath API changes) and helped getting started with c# and Visual Studio.

Davide Carpi contributed the glyph for the `Maxima()` and the `Cross()` functions as well as the Maxima palette in the side bar.

The Maxima team and the Maxima discussion list are always helpful.

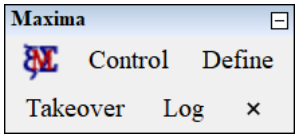
## 1.2 What You Get

**Menu entry** Main menu>Insert>Maxima




This entry provides access to the Log and Debug Dialogs, the Settings form, Maxima plot regions, called Draw2D and Draw3D regions and to the Maxima help page.

**Maxima palette** in the side panel for insertion of Maxima functions to the canvas.



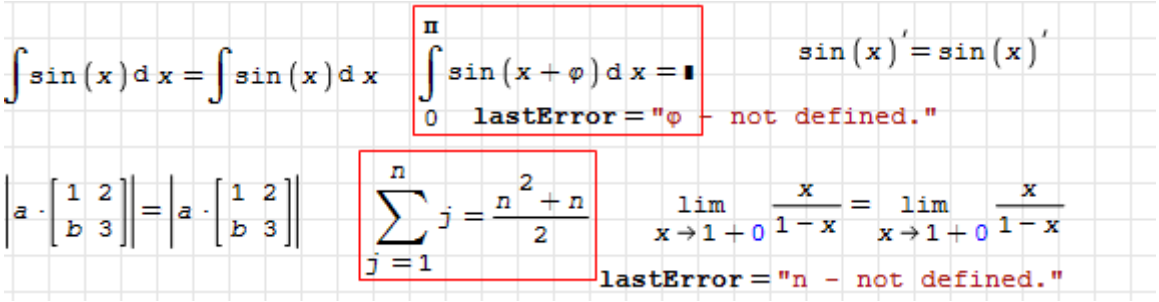

**Function** `Maxima()`, which hands over expressions to Maxima for evaluation and returns the result to SMath. The currently implenented translation capabilities already cover a significant part of Maxima's features. There is full support for units of measurement, text indices and UTF8 encoding.

**Symbolic alternatives to SMath Functions**

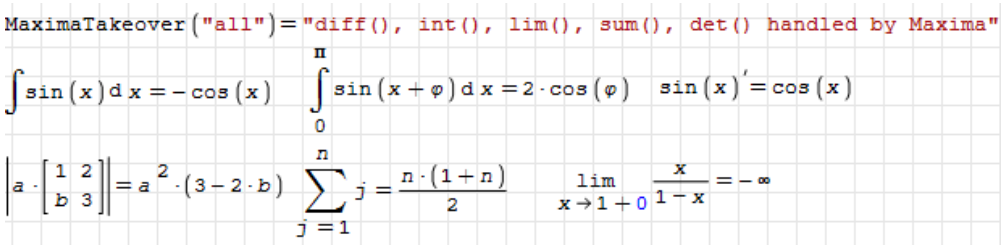
- `Int()` for definite and indefinite integration
- `Lim()` for limits, also one-sided limits are supported by adding `+'0` or `-'0` (a zero marked as unit) to the point.
- `Diff()` for derivatives
- `Det()` for determinants
- `Sum()` for sums
- `Cross()` for cross products. It has a permanent operator form  (smaller than the original SMath cross product symbol).

The function `MaximaTakeover()` redirects the native SMath functions to the above-mentioned alternatives. The advantage of redirection is better readabilty by use of the native operator symbols:

- SMath without Maxima




- SMath with Maxima and enabled redirection (takeover)




**Wrapper functions** for some Maxima features:

- `Solve()` symbolic solution of algebraic linear and non-linear equations and systems of equations (chapter 7)
- `ODE.2()` symbolic solution of differential equations (chapter 8)
- `Fit()` for non-linear least square fits of data to parametric functions (chapter 6)
- `MSE()` mean square error between data and model
- `Residuals()` residuals between data and model

- Functions `Draw2D()` and `Draw3D()` for creating plots (chapter 3). These can be displayed using Image regions (plugin *ImageRegion*). The functions use Maxima's *draw* package which actually uses Gnuplot as a back end.
- Draw regions (section (3.7)). They take the same commands as `Draw2D()` and `Draw3D()` but provide interactive mouse control for sizing, scaling, rotation and have a settings form for common plot options.
- Support of real and complex numbers, strings, lists and matrices.

Restrictions:

- The interface doesn't work under Linux.
- No support for names containing special characters produced using Ctrl-K.
- No units support for numerical procedures in Maxima.
- Boolean expressions are translated but have a different meaning in Maxima. Maxima has a data type `boolean`, which can assume the values `true` or `false`. In SMath this corresponds to numeric values zero for `false` and non-zero for `true`.
- There are restictions for the length of expressions due to the translation by regular expressions.
- The Maxima plugin is in beta-state of development. Not everything works as expected and things may change in the future.

Maxima on the web:

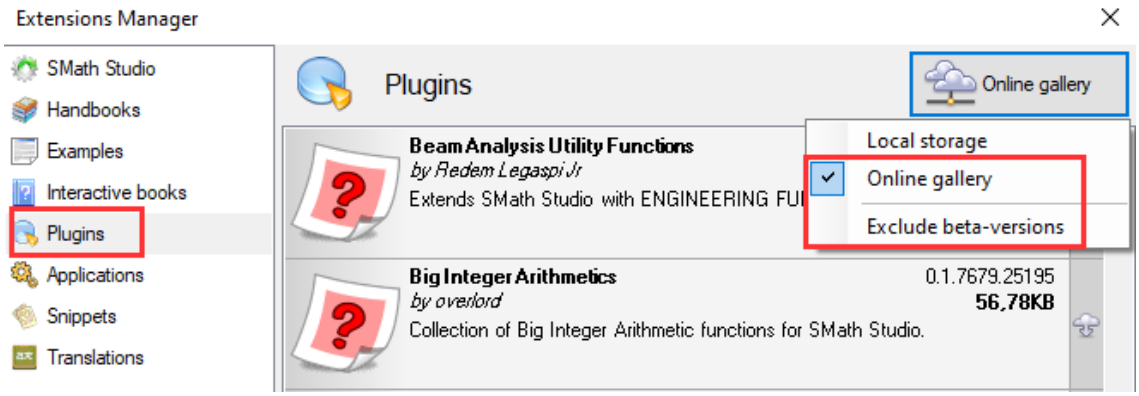
- Maxima Homepage[1] on Sourceforge,
- Maxima Overview[2] Quick overview
- Official index of handbooks and tutorials[3]

[1] maxima.sourceforge.net
[2] http://www-com.physik.hu-berlin.de/~bunk/kurs/maxima/maxima.html
[3] http://maxima.sourceforge.net/docs/manual/de/maxima_12.html#SEC79

## 1.3 Installation of the Plugin

Just like any other extension, the Maxima plugin is installed via the Extensions Manager from within an SMath session:

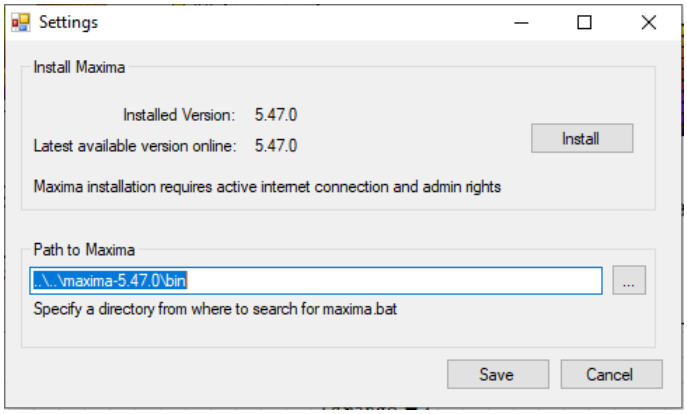
- **Tools> Plugins**
- Make sure to switch to the online gallery and unselect "exclude beta versions":




- Select the *MaximaPlugin* from the list.
- Press Install and wait for the installation of the plugin.

Usually, there is no need to perform a restart of the SMath application.

When the installation is finished, the "Settings" dialog pops up. Otherwise you can call the dialog via **Insert> Maxima> Settings** from the main menu.




Note: the plugin is just the interface between SMath and Maxima. Maxima is not included in the plugin. You can point to an existing installation of Maxima or you can use the Install button. These options are described in the next section.

## 1.4 Installation of Maxima

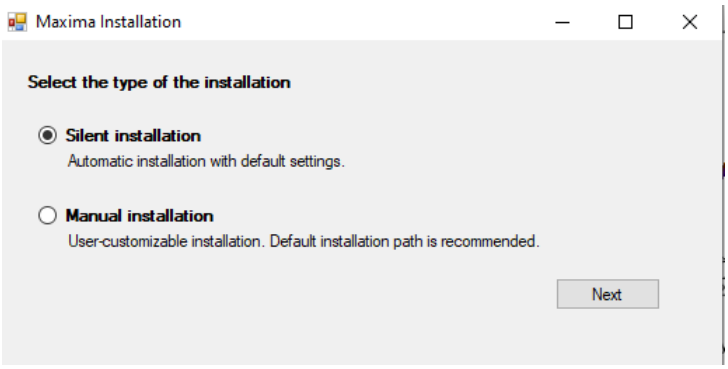
Open the Settings dialog: **Insert**> **Maxima**> **Settings**

**You already have a working Maxima installation on your computer?**

1. Use the file selector [...] to indicate the path to your installation. It is sufficient to point to the top level directory, e.g. *C:\maxima-5.47.0*.
2. Press [Save]. SMath starts searching for *maxima.bat* and tries to launch Maxima. If that works, just confirm the message.

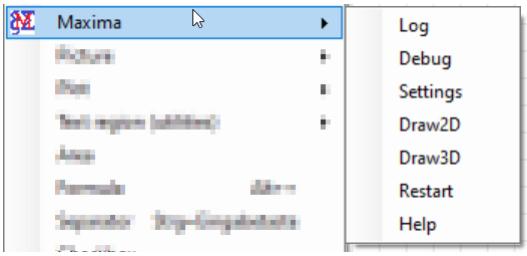
**You don't have Maxima on your computer?** For a standard installation you need Internet connection and admin privileges. The installation will take slightly more than 700 MB of disk space.

Press [Install] and select one of the install options:

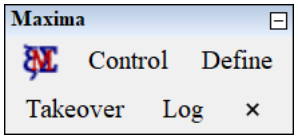



It should be safe to use the silent installation. In any case, the installation is a two-step process: download and the actual installation. The software is downloaded using the official links. SMath does not modify the installation in any way.

Upon finishing, the Settings window is closed and a Maxima session is launched in the background. Check the **Insert** menu for a Maxima submenu




In the side panel you should find the Maxima palette with quick access buttons.



## 1.5 Other Useful Plugins

***Custom Glyphs*** provides nice operator forms for SMath functions.

***Custom Functions*** provides the substitution function `at()`, which is very comfortable to use with results of any kind of algorithms (solver, curve fit).

***Development Tools*** provides convenient access to the installation directory of the *Maxima* plugin. In this directory Maxima stores temporary graphics files and the configuration file *maxima.xml*. Select **Insert> Development Tools> Plugin's folder> MaximaPugin.**

***ImageRegion*** plugin provides a region for display of image files generated by Maxima's `Draw2D()` and `Draw3D()` functions. This is the most transparent way to use Maxima plotting, while using the Maxima Draw regions is more convenient.

## 1.6 The Maxima Session

As soon as any of the Maxima-based features is used in an SMath document, a Maxima session is established via TCP/IP connection. You find this process as *sbcl.exe* (Steel Bank Common Lisp) in the Windows task manager.

- Each SMath instance has it's own Maxima process, which communicates with the currently open SMath document.
- Once started, the Maxima session is kept open.
- The session can be reset using the function `MaximaControl()`. The argument `"restart"` closes the session and starts a new one. The Argument `"cleanup"` just let's Maxima forget defined variables and loaded packages.

  ```
  MaximaControl("restart")="Restart complete."

  MaximaControl("cleanup")="Cleanup complete."
  ```

  Alternatively you can issue a restart using **Insert> Maxima> Settings: Save**.
- Switching between open document in a single instance of SMath can have side effects, because all documents of a given SMath instance share the same Maxima process.

  To be safe, insert `MaximaControl("restart")=` in your documents before using any Maxima feature. Opening just one sheet per SMath instance completely avoids the problem.
- Save your work frequently. Maxima oder SMath actually can crash. Usually, the plugin manages to kill crashed Maxima sessions. If not, find and kill the process *sbcl.exe* (Steel Bank Common Lisp-Interpreter) using the Windows task manager.

## 1.7 Replacement of Functions (Takeover)

The interface is under development, so the redirection (takeover) of SMath native functions by Maxima is deactivated by default. For activation and deactivation we have the function

`MaximaTakeover()`. The function accepts one or more string parameters, which are searched for the keys given in table 1.1 to redirect individual functions or "all" for all (as you might guess). If none of these are found, redirection is deactivated.

This redirection can be done at any location in the SMath sheet and is valid until the next call to `MaximaTakeover()`. It is also reset by restart or cleanup using `MaximaControl()`. Note that the takeover state can only be changed by functions on the sheet, not by menu commands like **Insert**> **Maxima**> **Settings**> **Save** or **Insert**> **Maxima**> **Restart.**

| Function | Key | SMath | Operator | Maxima |
|---|---|---|---|---|
| Derivative | "diff" | `diff(1)` | ▪′ | `Diff(1)` |
| | | `diff(2)` | $\frac{d}{d\,▪}\,▪$ | `Diff(2)` |
| | | `diff(3)` | $\frac{d}{d\,▪}\,▪$ | `Diff(3)` |
| Indefinite integral | "int" | `int(2)` | $\int ▪\,d\,▪$ | `Int(2)` |
| Definite integral | "int" | `int(4)` | $\int_{▪}^{▪} ▪\,d\,▪$ | `Int(4)` |
| Determinant | "det" | `det()` | $\lvert ▪ \rvert$ | `Det()` |
| Limit | "lim" | `lim()` | $\lim_{▪ \to ▪}$ | `Lim()` |
| Sum | "sum" | `sum(4)` | $\sum_{▪=▪}^{▪}$ | `Sum(4)` |

Table 1.1: SMath functions and corresponding Maxima functions available via `MaximaTakeover()`. The capitalized Maxima functions are always available, yet they don't have operator representations.

Symbolic results from Maxima can be displayed using Optimization> Symbolic or Optimization> None in the context menu of the formula. In case of "Symbolic" SMath tries to further simplify the result, not always to the better. It is sometimes worth a try to switch off optimization.

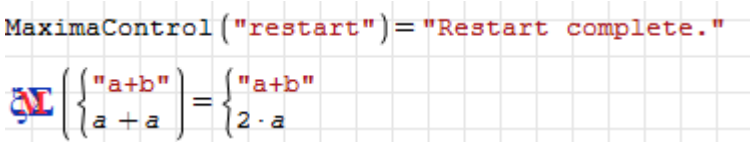
## 1.8 The `Maxima()` Function

This function

1. Takes an SMath expression (which SMath tries to handle before it is actually sent to the function).
2. Translates it to Maxima syntax.
3. Uses this as input for the current Maxima session.
4. Translates the Maxima output to SMath

This is the most general way to use Maxima from within SMath. Frequently it is just used to simplify SMath expressions.

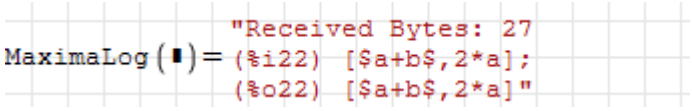
Die `Maxima()`function has an operator form (Maxima logo), which in earlier versions was provided by the *Custom Glyphs* plugin, but now is part of the Maxima plugin itself.




| Object | SMath | Maxima |
|---|---|---|
| Decimal separator | Comma or period | Period |
| Argument separator | Semicolon or comma | Comma |
| Text index | $a_b$ | `a_%_b` |
| Underscores in names | $a\_b$ | `a_b` |
| Special constants | e, i | `%e, %i` |
| Lists (systems) | $\begin{cases} \blacksquare \\ \blacksquare \end{cases}$ | `[]` |
| Matrices | $\begin{bmatrix} \blacksquare & \blacksquare \\ \blacksquare & \blacksquare \end{bmatrix}$ | `matrix()` |
| Element index | $v_j$ | `v[j]` |
| Strings | "abc" | `"abc"` |
| Units | cm | `%unitcm` |

Table 1.2: SMath objects and their representation in Maxima

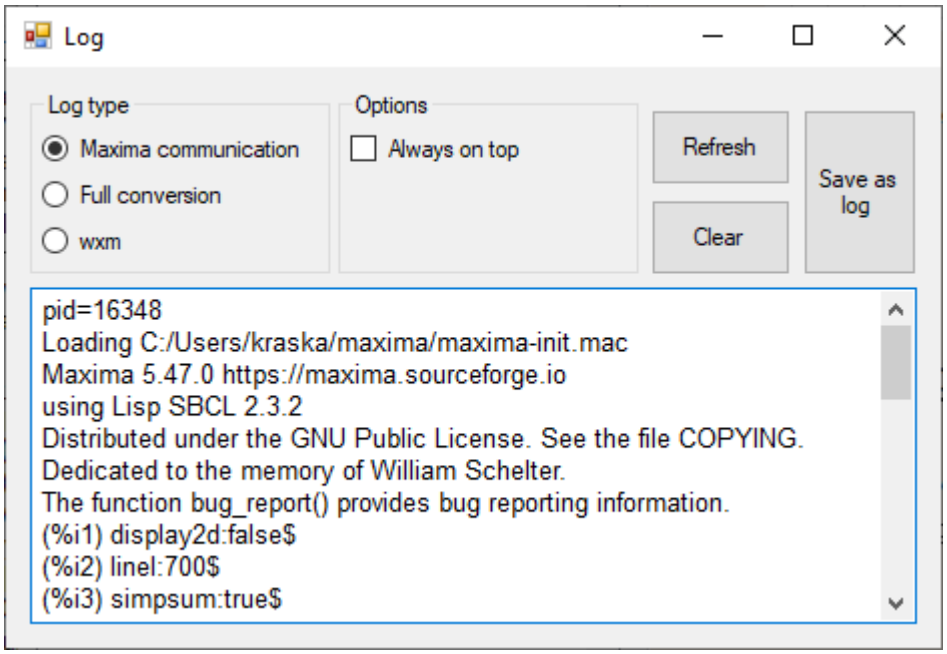
Usually translation of names to Maxima and back to SMath should yield the original name. The function `MaximaLog()` reveals what happens behind the scene. It shows the most recent input/output pair of the Maxima session.




To inspect the translation steps use the debug window under **Insert> Maxima> Debug.**

## 1.9 The Log Window

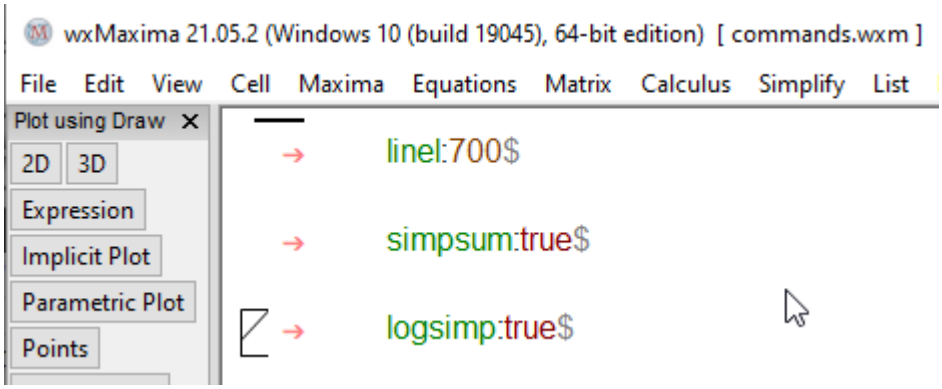
The communication between SMath and Maxima is logged by default. The log window is called either from the canvas by `MaximaLog("all")` or via **Insert**> **Maxima**> **Log**.




There are three types of logs:

- **Maxima communication**: This is a concatenation of the strings exchanged between SMath and Maxima
- **Full conversion**: For each request to Maxima, this log shows
  - The SMath expression in text format
  - The translation to Maxima
  - The response from Maxima
  - The translation to SMath in text format
- **wxm**: This contains the commands sent to Maxima formatted as wxMaxima input.

The button Save as.. offers a dialog to save the log. In the case of wxm format, the log is sent to wxMaxima (the standard GUI of Maxima).



## 1.10 MaximaLog()

```
MaximaLog(■)
```

shows the last input and output of Maxima with the respective labels.

```
MaximaLog("all")
```

displays Maxima input and output of the running Maxima session beginning from the last restart or cleanup or manual log reset. The output may be quite large. In this case, it is better to use the interactive Log window. This log is a simple concatenation of the strings exchanged between SMath and Maxima.

# 2 Computer Algebra

## Contents



The computer algebra features of Maxima expand the symbolic capabilities of SMath. Maxima has a variety of simplification functions.


*simp.sm*

## 2.1 Algebra

Display of Maxima results is best with **Context menu**> **Optimization**> **None**.

### Factor and Expand

$$\text{factor}(10!) = 2^8 \cdot 3^4 \cdot 5^2 \cdot 7$$

$$\text{expandwrt}\left((x+y)^4, x\right) = y^4 + 4 \cdot x \cdot y^3 + 6 \cdot x^2 \cdot y^2 + 4 \cdot x^3 \cdot y + x^4$$

$$\text{factor}\left(x^6 - 1\right) = (x-1) \cdot (x+1) \cdot \left(x^2 - x + 1\right) \cdot \left(x^2 + x + 1\right)$$

$$u := \text{expand}\left((x+y)^6\right)$$

$$u = y^6 + 6 \cdot x \cdot y^5 + 15 \cdot x^2 \cdot y^4 + 20 \cdot x^3 \cdot y^3 + 15 \cdot x^4 \cdot y^2 + 6 \cdot x^5 \cdot y + x^6$$

### Complex Numbers

$$\text{ratsimp}\left(e^{i \cdot \pi}\right) = -1$$

$$\text{polarform}\left(\frac{1+i}{1-i}\right) = e^{\frac{i \cdot \pi}{2}}$$

### Big Floats

`MaximaDefine()` is used to set values in the background Maxima session. A common usage is to set control variables:

`fpprec` Precision for floating point operations

`fpprintprec` Precision for output of floating point values

```
MaximaDefine(fpprec; 48) = 48
```

$$\text{bfloat}(e) = 2{,}718281828459045235360287471352662497757247093 7 \cdot 10^0$$

$$\text{fpprec} = 48$$

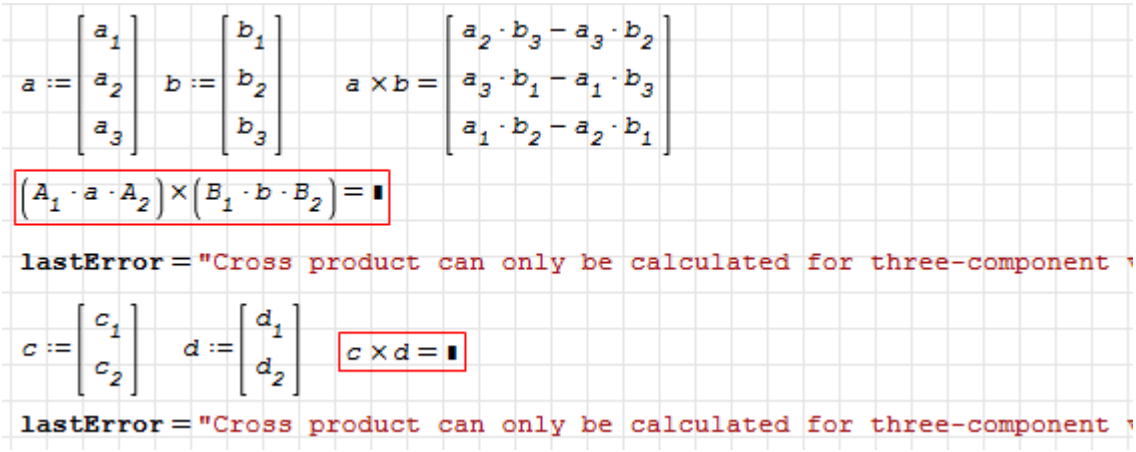
```
fpprec := 3
MaximaDefine(fpprec) = 3
```

$$\text{bfloat}(e) = 2{,}72 \cdot 10^0$$

$$\text{fpprec} = 3$$

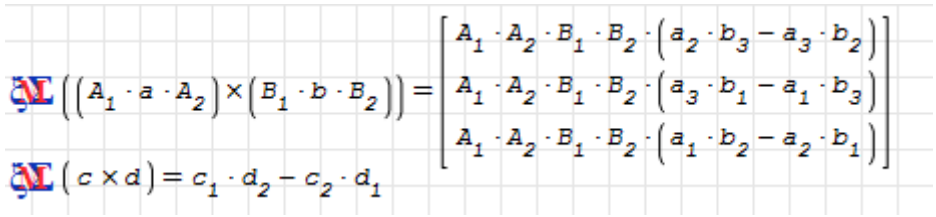
## Cross Product

The native SMath cross product has two limitations: It can't handle vectors which are multiplied by undefined variables, because in contrast to Maxima, SMath doesn't assume that such a factor is a scalar. The other restriction is that it can't handle 2D vectors.

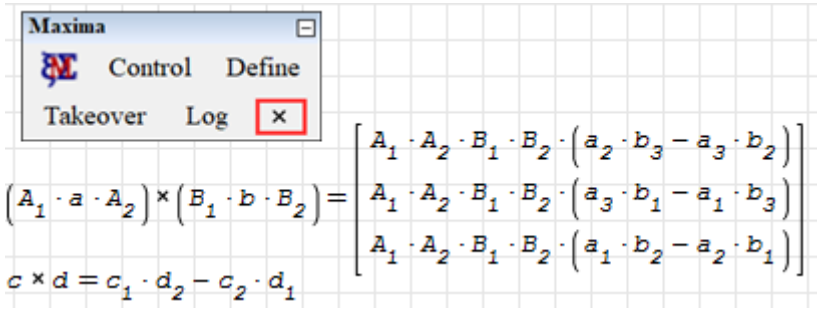



The Maxima plugin provides two ways to handle this.

- Wrap the native cross product with the `Maxima()` function:

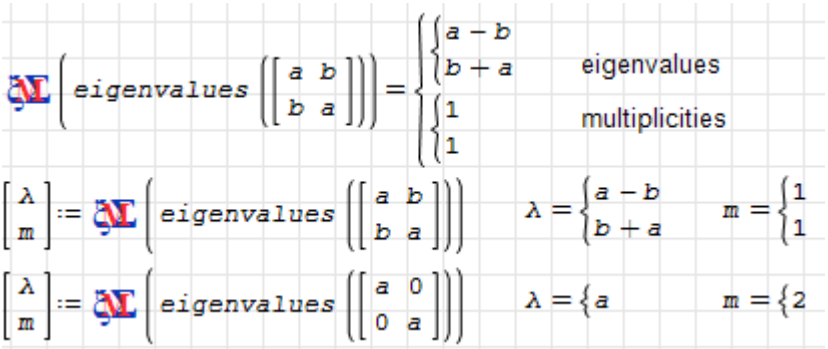


- Use the dedicated function `Cross()` with a permanent custom operator form (not affected by `MaximaTakeover()`)

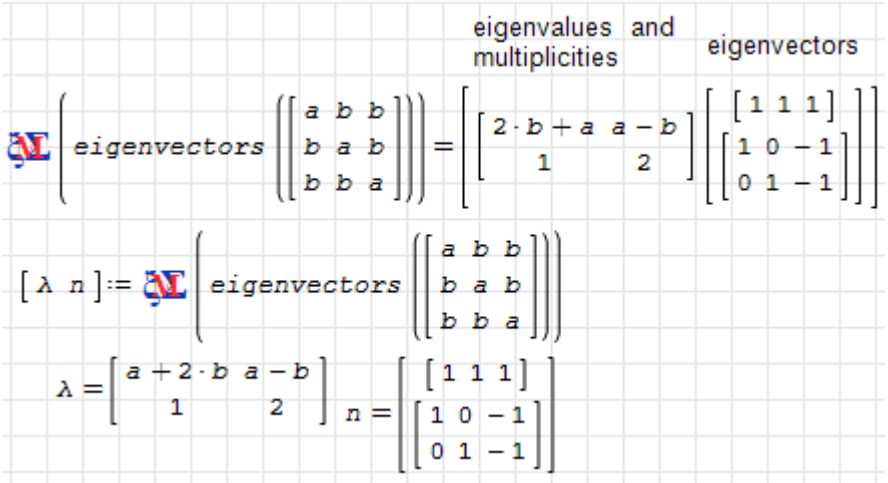



## Eigenvalues

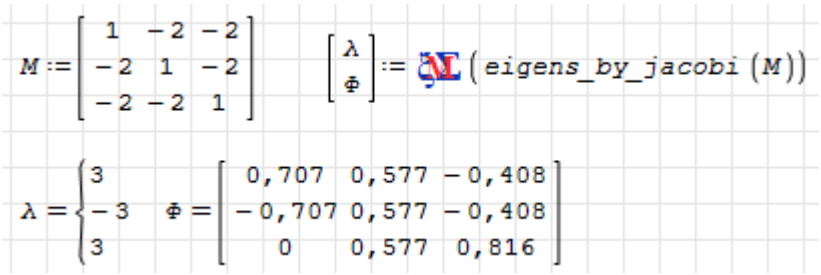
Maxima has functions for extracting eigenvalues and eigenvectors symbolically. For each eigenvalue it's multiplicity and eigenvectors are given in a nested list and matrix structure:




The Eigenvectors are returned as row vectors. For multiple eigenvalues the vectors are returned as rows in a matrix.




There is also a numeric procedure for eigenvalue problems: `eigens_by_jacobi()`. It returns a list of eigenvalues and the corresponding spectral matrix with it's columns being the normalized eigenvectors.

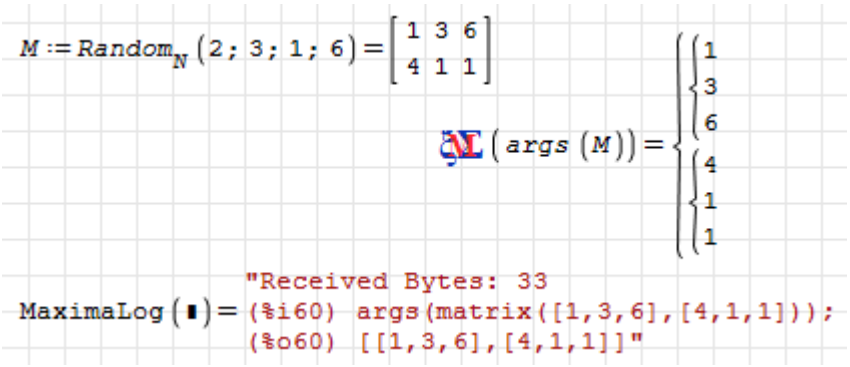


The spectral matrix rotates the matrix to diagonal form with the eigenvalues populating the diagonal

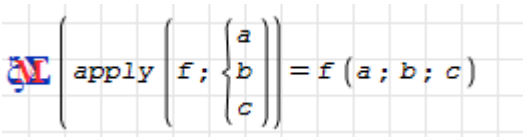
## 2.2 Manipulations of Terms

`args()` extracts the arguments of a function.

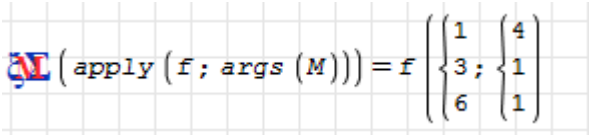
An application is the conversion of a matrix into a list of lists (which is sometimes needed for specifying graphics objects). In Maxima a matrix is represented as the function `matrix([...],[...]...)` with the rows represented by lists. `args()` extracts the rows of a matrix as a list of lists:



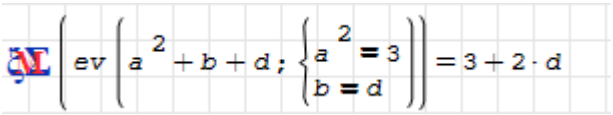

`apply(f, list)` converts the elements of `list` into arguments of function `f`.




A combination of both functions allows to apply the rows of a matrix as arguments of a function. Some graphics functions like `label()` take arbitrary numbers of lists as arguments. This is easy to generate from a matrix using the `args()` and `apply()` combination:



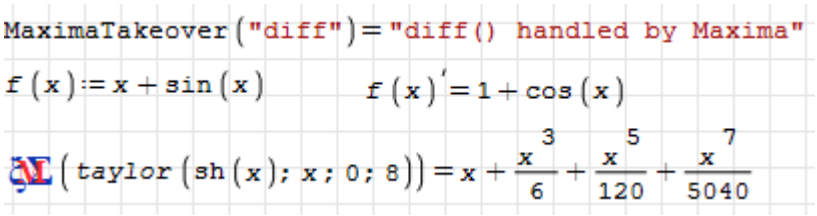

The function `ev()` applies substitutions to an expression. This goes beyond the SMath function `at()`, because it can replace complex expressions, not just variables:



## 2.3 Calculus

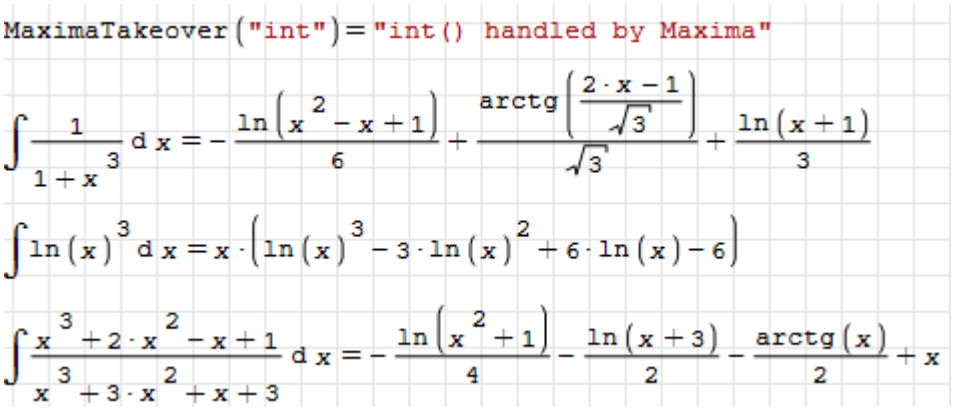
### Symbolic Derivatives

SMath can handle symbolic derivatives on it's own. Should that fail in particular cases, you can switch to Maxima. This also enables the prime operator form, which exists in SMath but is not functional without the Maxima plugin. The function behind this operator is `diff()`.

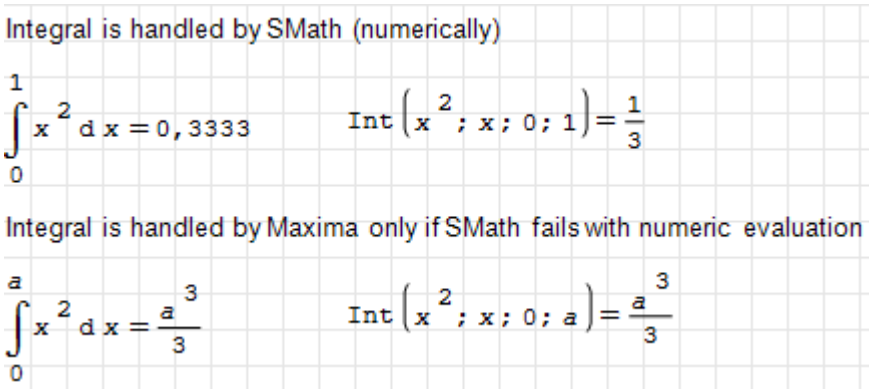



### Symbolic Integrals

Once the integral functions are taken over, SMath can handle indeterminate integrals symbolically.

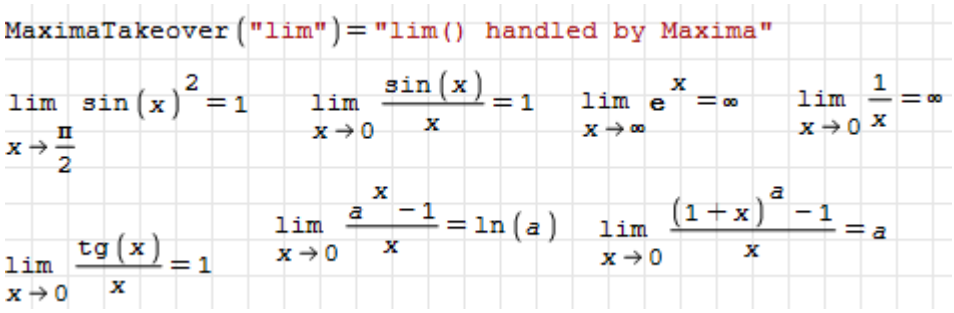



SMath will always try to handle determinate integrals numerically. Only if SMath fails, the takeover by Maxima will take effect. If you want to enforce handling by Maxima, you can use the capitalized function form `Int(f, x, a, b)`.

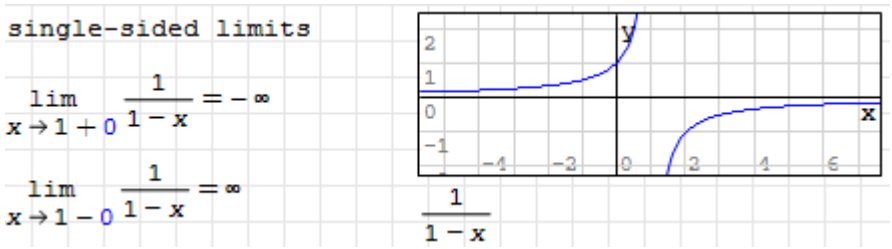


## Symbolic Limits

SMath has a native limit operator, yet normally that is not functional. Takeover by Maxima enables symbolic limits.




**Single-sided limits** are specified by adding `-'0` or `+'0` (a zero marked as unit) to the limit point. Using plus/minus a simple zero was possible in early SMath versions. Yet now there is no chance for the plugin to prevent SMath preprocessing from removing the zero. However, marking the zero as unit does the trick.



# 3 Plotting with Maxima

## Contents



The Maxima plugin provides access to Maxima's draw package[1]. It's back-end is the flexible and powerful open source graphics software Gnuplot[2] . The graphics are saved in temporary or user-specified files.

**Unique selling points** of Maxima graphics:

- Logarithmic plots (which you otherwise would only get with the ZedGraph region, which is very complicated to handle)
- Shaded 3D plots

There are two options to place Maxima plots on the sheet:

**Draw functions and Image region** The functions `Draw2D()` and `Draw3D()` produce 2D and 3D plots respectively and return the file name used for storage. This name goes to the placeholder of an Image region, a general purpose region for display of graphics from files or matrix variables. This region is provided by the *Image Region* plugin.

Use this for complete control by graphics commands.

**Draw region** These are dedicated Maxima plot regions, found under **Insert> Maxima> Draw2D and Draw3D**. They provide a setting menu and some moderate (slow) interactivity for scaling and rotating plots.

Use this for convenience: Interactive sizing, scaling and rotation and less commands required.

Both options share the same graphics features. In both cases, the user provides a list of objects and options to control the plot.

When demonstrating features, we mostly use the `Draw2D()/3D()` functions, because they don't have hidden settings.

The graphics can be produced as PNG, SVG or PDF files.

[1] riotorto.users.sourceforge.net/Maxima/gnuplot
[2] www.gnuplot.info

**PNG** is fastest but has sometimes visual glitches (incomplete display of characters)

**SVG** is good for export. Rendering on the canvas suffers from wrong text alignment in the image region and noenhanced text mode in the Draw regions. This means that texts like "a^b" and "c_2" aren't handled correctly.

**PDF** Best option for high quality display and export.

## 3.1 Your First `Draw2D()` Plot

1. Create an Image region. **Insert> Image**. A region with a placeholder appears. The actual size of that region is not important for now.
2. In the placeholder, add the Draw2D() functions with a list of commands as argument.

   Make sure to use the boolean equal for commands like `title = ''My first plot''`
3. Leave the region (click next to it) to see the result. The default image size is 300 x 240 pixels in 2D and 300 x 300 in 3D. Yet the image is scaled to the actual size of the image region.
4. Adjust the region to match the image size: **Context menu (left mouse button)> Reset to original size**.


```
Draw2D([title = "My first plot", explicit(sin(x), x, -π, π)])
```

Plot commands consist of objects and options.

- Global options affect the whole scene. Their position in the command list is not relevant.
- Local options affect subsequent objects. They must precede the object they are supposed to affect.

The options have to be written as equations with the boolean equality symbol `option=value`.

In the Draw-Description snippet (see section 3.3) you find comprehensive information on the available objects and options.

Some of the options are explained in the subsequent sections and demonstrated in the examples for the graphics objects in chapters 4 and 5.

## 3.2 Draw2D() and Draw3D() Functions

The functions `Draw2D()` and `Draw3D()` actually produce an image file and returns it's name. The image region takes a file name in the placeholder. This file is read and displayed.

You can use additional arguments to specify a file name and the size of the graphics.

```
Draw2D(commands)
Draw2D(commands; filename)
Draw2D(commands; {width, height})
Draw2D(command; filename; {width, height})
```

**commands** List of objects and options.

**filename** (default: "pdf") format and storage location of the graphics file. If just an extension is given ("png", "svg" or "pdf"), then a temporary file name is generated in the specified format.

Relative filenames (without full path specification) are relative to the current directory. It is advisable to set the current directory to the document directory. This yields portable documents.

```
pwd := CurrentDirectory(DocumentDirectory(""))
pwd = "C:\FHB\Software\SMath\SMath Skript\Images\"
```

**width, height** (default 300 x 240 for `Draw2D()` and 300 x 300 for `Draw3D()`) size of the graphics file in pixels. In the Image region, the graphics is scaled to the size of the region. Yet to keep consistent font sizes, do it the other way around:

- Specify size as needed in the draw function
- Adjust the region to match the image size: **Context menu> Reset to original size**.

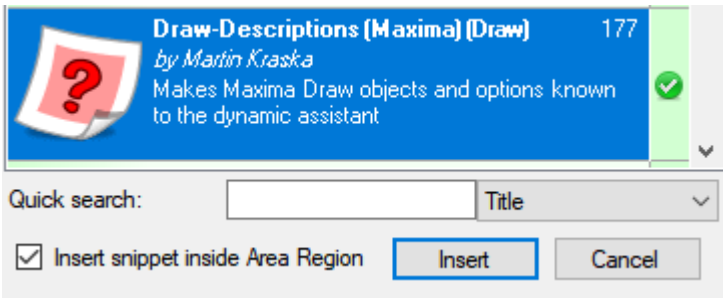
## 3.3 The Draw-Descriptions Snippet

Creating graphics with Maxima is all about knowing the available objects and options. SMath doesn't know these by default, so that the dynamic assistant doesn't help.

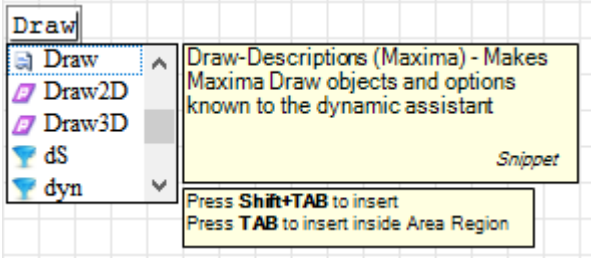
The Draw-Descriptions snippet (snippets are collapsible area regions with SMath regions) contains dummy definitions with attached descriptions of all objects and options in German and English. Once this snippet is inserted, the dynamic assistant knows them all.

Insert the snippet:

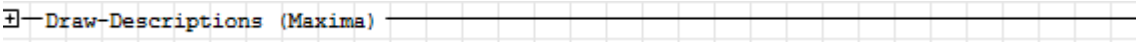
1. Set the cursor to where you want the snippet (it appears just as a thin line)
2. Open the snippets page of the extension manager **Tools> Snippet Manager**
3. Locate and select the snippet "Draw Descriptions" in the list. If it is not there, you might need to switch to the online gallery and get it from there.
4. Make sure to select "insert snippet inside Area Region" and press Insert



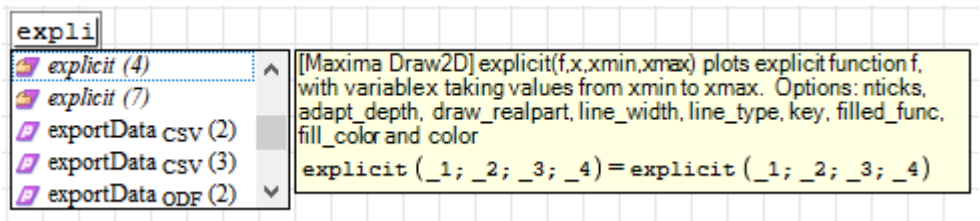

The dynamic assistant knows about locally stored snippets. So it is easy to insert the Draw-descriptions using the keyboard:




The snippet is inserted as a collapsed area region:

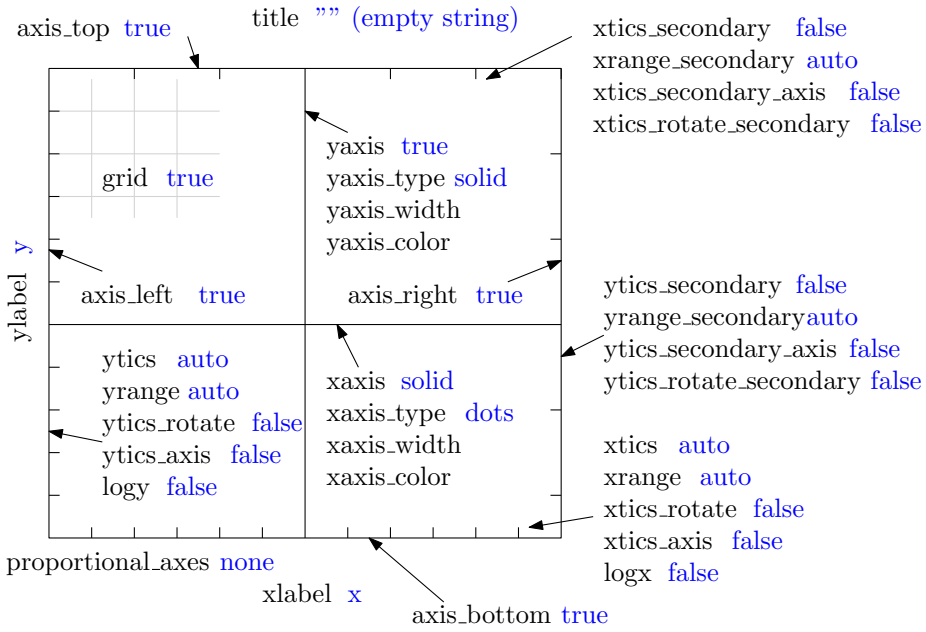


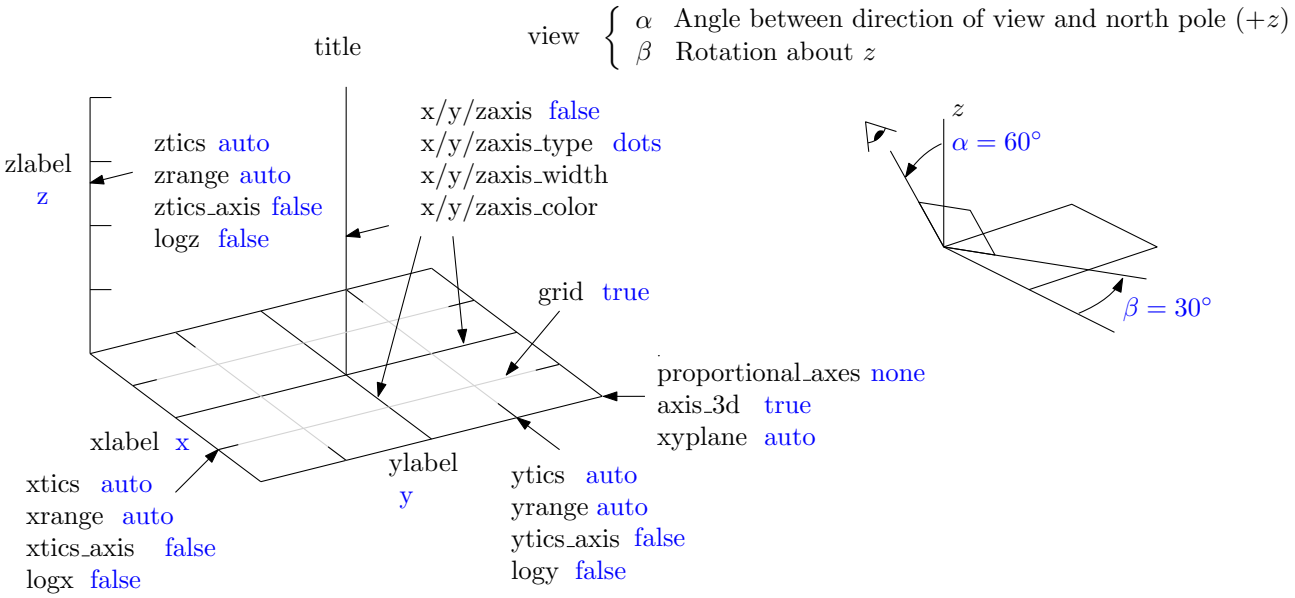

You can open the area region by clicking the + and browse the available objects and options. The real comfort of the snippet is that you avoid typos and get reference on available arguments and options.



## 3.4 Axes and Grid

The following images summarize the options for control of the axes and the grid. The default values are given in blue.




The grid is active by default for the x and y axes.

Text labels are produced in Gnuplot's enhanced mode with full Unicode support (UTF8): `a_b` yields $a_b$, `a^b` yields $a^b$.

This applies for all types of text:

- Axes labels (`xlabel`, `ylabel`, `zlabel`, `xlabel_secondary`, `ylabel_secondary`)
- Title (option `title`)
- Text labels at arbitrary coordinates (object `label()`).

## 3.5 Point and Line Types

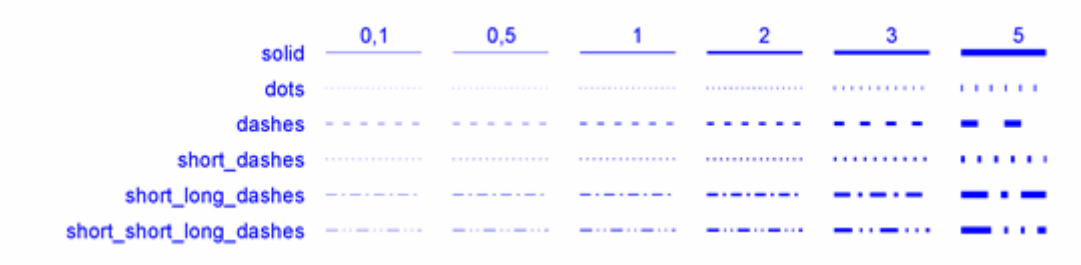



Table 3.1: Examples for `line_type` and `line_width`.

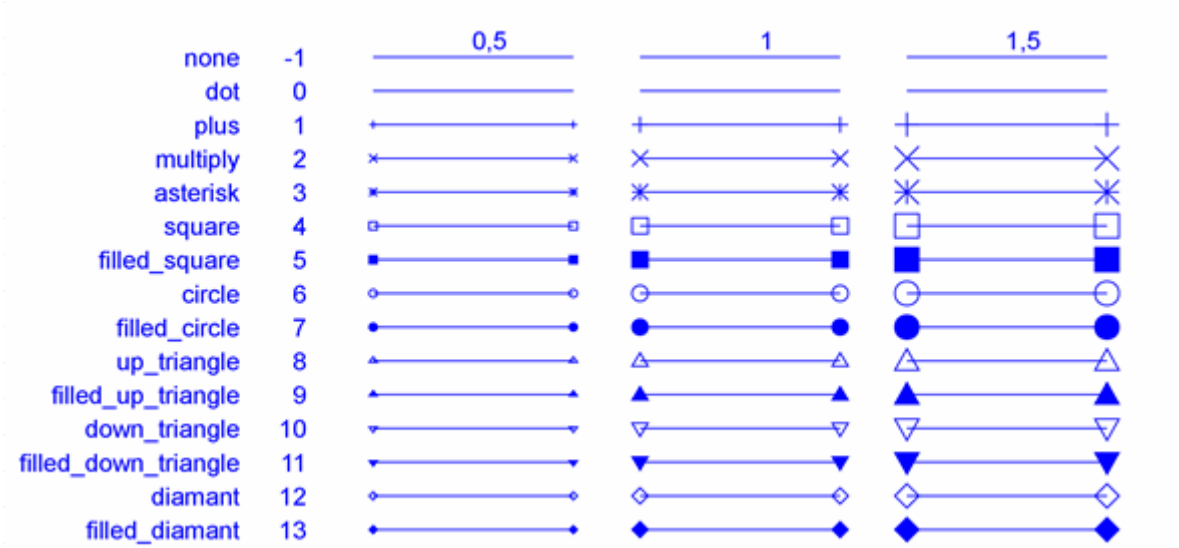



Table 3.2: Examples for `point_type` and `point_size`.

## 3.6 Colors

The color values for the options `background_color`, `color`, `palette` and `x/y/zaxis_color` can be specified in different ways:

- String with hexadecimal RGB value `"#rrggbb"`. Each digit (byte) represent a hexadecimal number 0-9, a-f. Each two bytes represent decimal integers from 0-255. `"#000000"` is black and `"#ffffff"` is white.
- String with hexadecimal RGBA value `"#rrggbbtt"`. The tt bytes denote transparency `00` is opaque (non-transparent) and `ff` is full transparency. Note that SMath native plots use the ARGB color scheme "`#ttrrggbb`".
- As color name (string or undefined variable name)

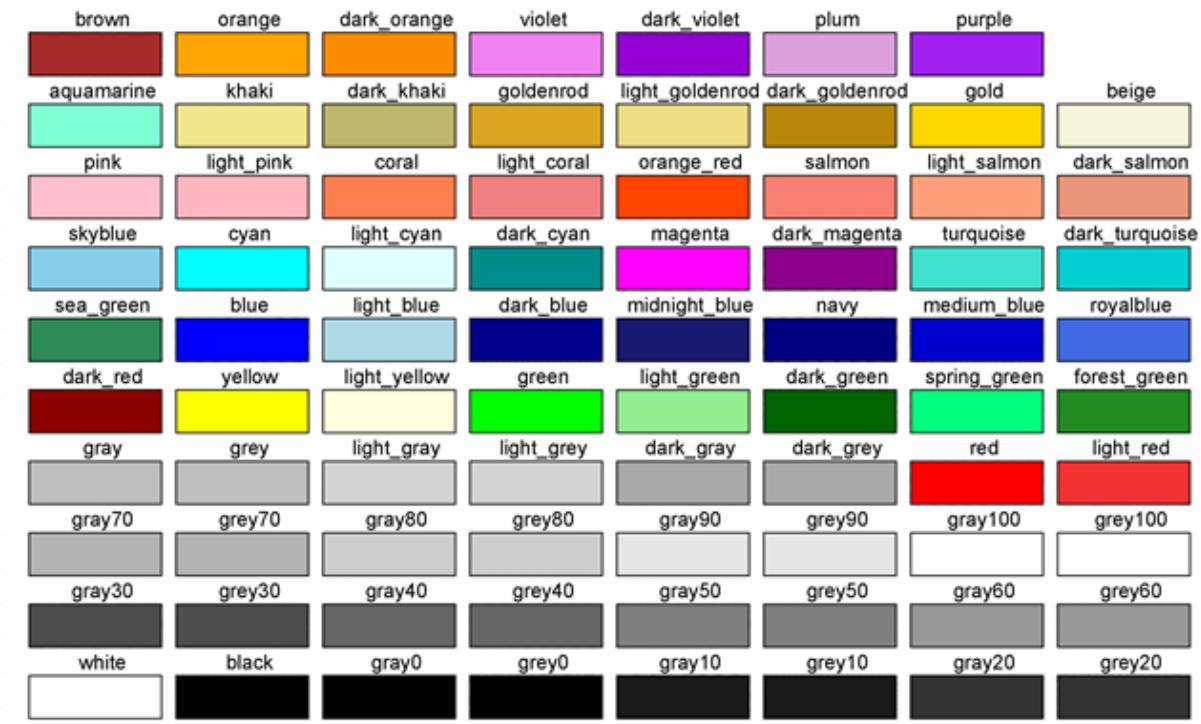



Table 3.3: Valid `color` names. Other colors must be specified as RGB or RGBA hex strings.

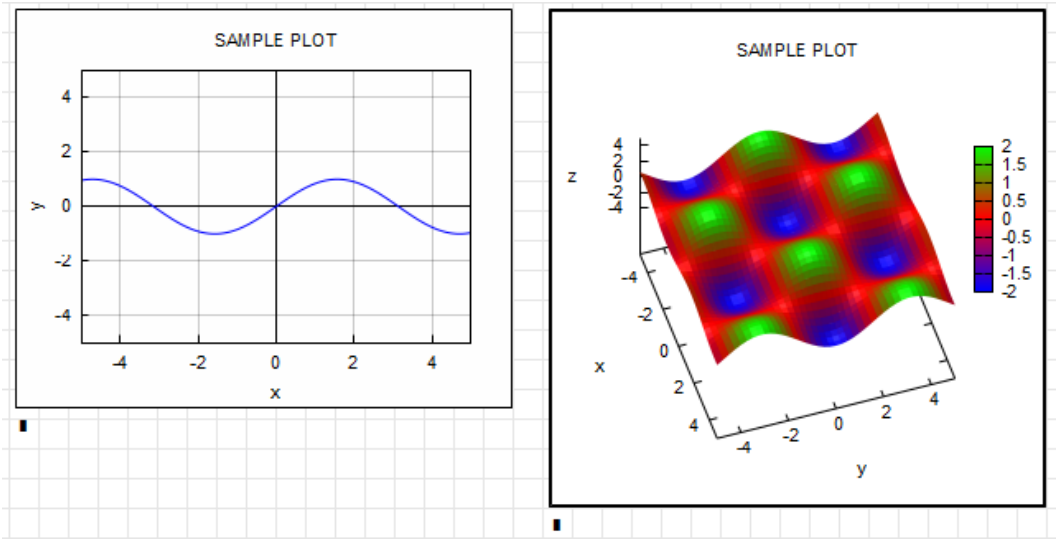
## 3.7 Maxima Draw Regions

The Maxima plugin provides dedicated regions for graphics, where the graphics commands directly go to the placeholder. They are called *Draw regions*. The advantages of this approach:

- Mouse control for pan, zoom and rotation of the axes (yet quite slow)
- The size of the generated graphics is adapted to the region size (which you can drag with the mouse)
- Settings dialog for axes and diagram properties. This reduces the amount of commands required in the input field and of lookup work to find the proper commands.

**Insert> Maxima> Draw2D** or **Draw3D**

As long as the input placeholder is empty, a sample plot is shown (left: **Draw2D**, right: **Draw3D**:



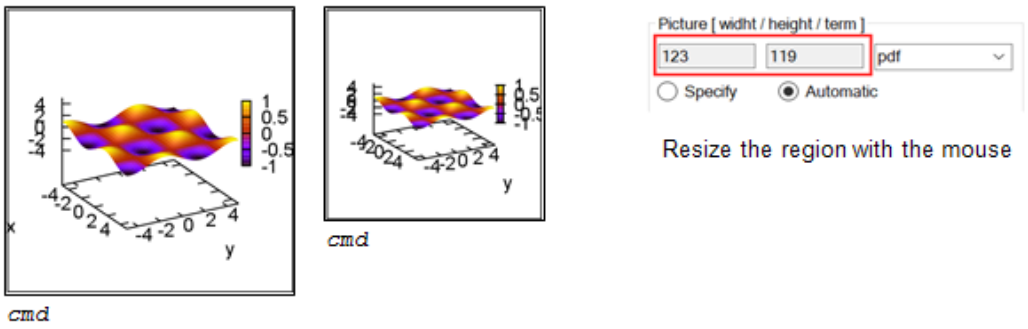

The placeholder takes a list with draw objects and options just as the first argument of the functions `Draw2D()` and `Draw3D()` (see section (3.2)).

**The context menu** also has interesting options:

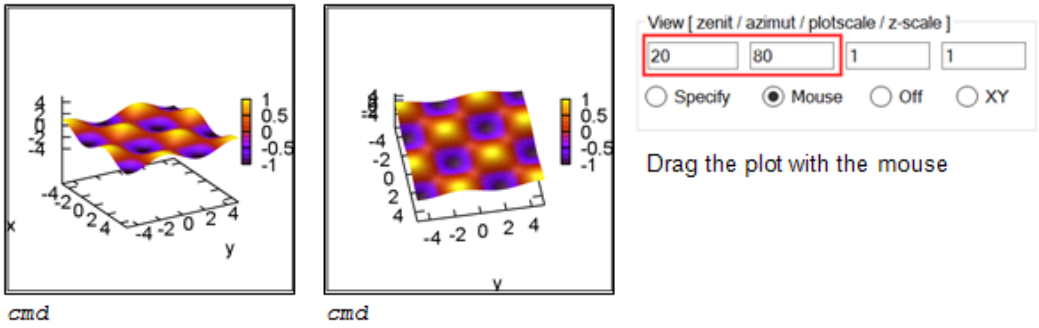
- (Un)select "Display input data" to hide the graphics commands,
- (Un)select the outer border of the region,
- Export graphics to a file (this actually just saves the existing temporary file to a user-specified location

### Mouse Control

**Sizing**: Drag the control points at the border to resize the image. Upon release, the graphics is re-computed with a new size. If you want to change the font size, use the settings dialog.

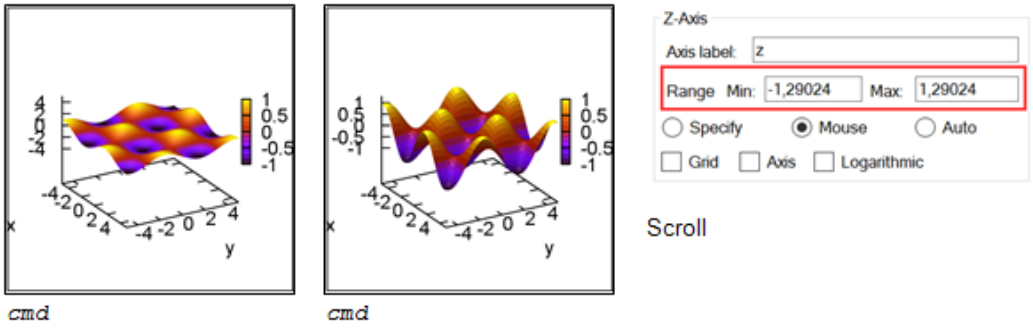



**Rotating**: Drag the image to rotate the plot. Upon release, the graphics is re–computed with the new orientation.

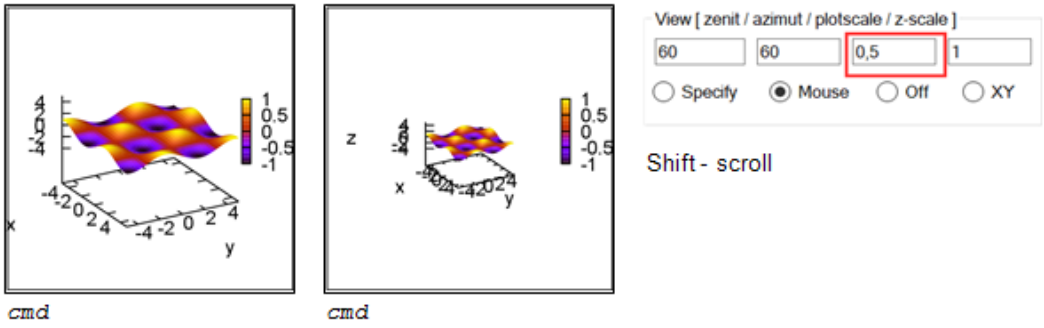


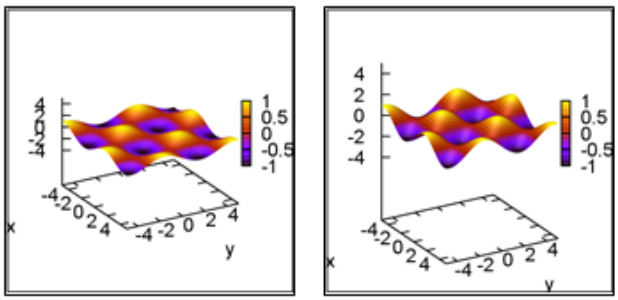

**Scaling (axes limits)**: Scroll (with the mouse wheel). Scroll: Modifies the axes limits.

Scroll up reduces the limits (expands the objects), scroll down increases the limits (shrinks the objects). This acts on axes whith range control set to "Mouse" in the Plot settings form. By default, just the vertical axis is activated.




**Scaling (bounding box)**: Shift-scroll changes the size of the bounding box in 3D plots relative to the region size. Ctrl-scroll scales the bounding box vertically. The axes limits aren't affected by these settings.

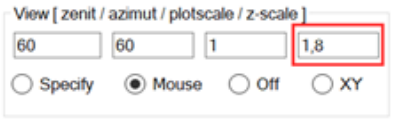





cmd




cmd




Ctrl - scroll

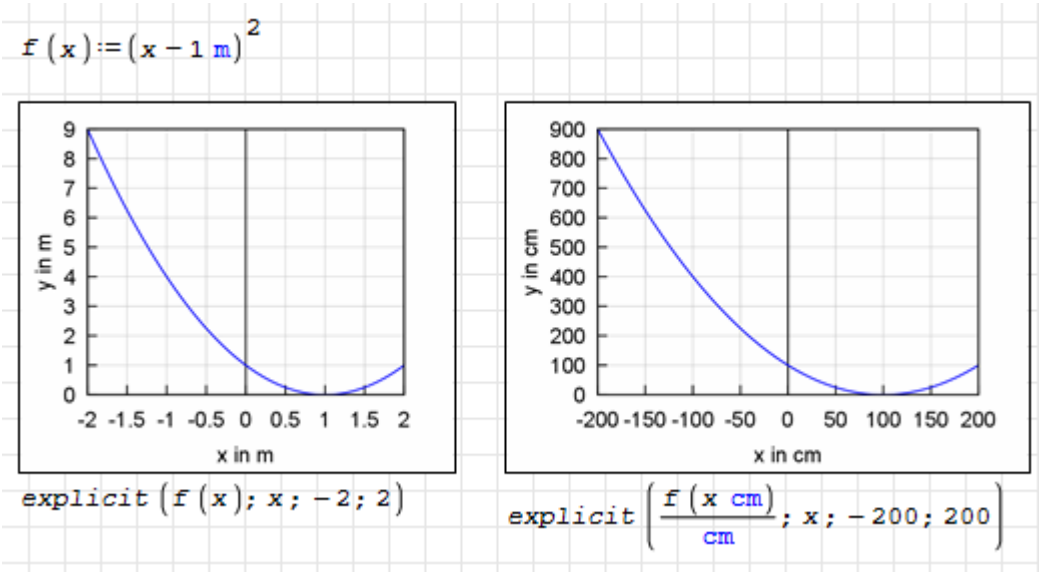
## 3.8 Units in Graphics

SMath internally converts any units to base units before storage. Maxima generally treats units as symbolic constants. In plot functions, any units are replaced by the number 1.

Thus, any base units in plot expressions are optional.

To use a different unit for the independent variables, multiply them by this unit wherever they appear in the expression.

To use a different units for the value of an expression, divide it by this unit.




Adjust the axes labels accordingly using the `xlabel` and `ylabel` options or the settings menu of the Draw regions.

# 4 2D Graphics Objects

## Contents



 *Draw2d3d-EN.sm* (takes some time to recalculate)

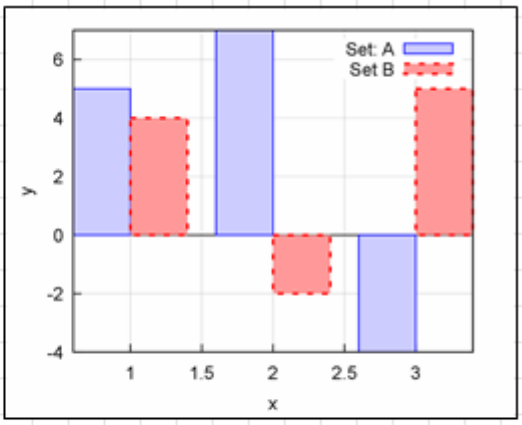
Here we demonstrate the available objects for 2D graphics, i.e. for use with `Draw2D()` or the Draw2D region. For some of the objects, we also explain some of the most important options. The Draw Description snippet is a good source on available options.

## bars(b1; b2; b3...)

This is a low level object for bar charts. The individual bars are given by lists with position $x$, height $h$ and width $b$:

$$\begin{cases} x \\ h \\ b \end{cases}$$

```
cm := [key = "Set: A"
       fill_color = blue
       fill_density = 0,2
       bars [[0,8  1,8  2,8]
             [5    7   -4 ]
             [0,4  0,4  0,4]]
       key = "Set B"
       fill_color = red
       fill_density = 0,4
       line_width = 2
       bars [[1,2  2,2   3,2]
             [4   -2    5  ]
             [0,4  0,4  0,4]]
       xaxis = true]
```




```
Draw2D(cm)
```

Specifying the bars as individual arguments is uncomfortable. You can use Maxima functions to extract the arguments from a matrix:

```
apply(f; args([[1 2 3]
               [4 5 6]
               [7 8 9]])) = f([1  4  7]
                              [2; 5; 8]
                              [3  6  9])
```

## boxplot_description(*M*, option1,...)

Box-and-whisker-plots visualize parameters of random distributions (minimum, maximum, median, lower and upper quartile). The option range allows to set limits for outliers in terms of the interquartile range.

*M* Data matrix $m \times n$, each of the $n$ columns is a sample of $m$ values.

`box_width` (default 3/4): relative width of the boxes, must be within [0,1].

`box_orientation` (default `vertical`): possible values: `vertical` and `horizontal`.

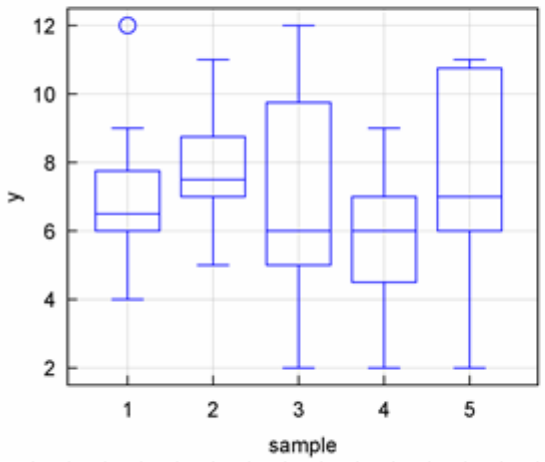
`range` (default $\infty$): Sets a factor $f$, such that values which are by more than $f$ times the interquartile range off the box are considered to be outliers. These are marked with circles and not taken into account for the minimum and maximum indicators. Usually $f = 1,5$.

`outliers_size` (default 1): size factor for the outlier-circles.

The function is defined in the Maxima package *descriptive*, which is not loaded by default in the Maxima plugin.

In the example we are sampling throws of two dice.

```
load("descriptive") = "C:/maxima-5.47.0/share/maxima/5.47.0/sh
samples := 5 values := 10
header := [1..samples]^T
M := Random_N(values; samples; 1; 6) + Random_N(values; samples; 1; 6)
```




| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 4 | 7 | 10 | 6 | 8 |
| 12 | 8 | 5 | 6 | 11 |
| 6 | 9 | 2 | 9 | 10 |
| 6 | 6 | 3 | 7 | 2 |
| 6 | 7 | 12 | 8 | 6 |
| 7 | 8 | 10 | 7 | 6 |
| 9 | 7 | 9 | 3 | 11 |
| 6 | 11 | 5 | 6 | 6 |
| 7 | 9 | 5 | 2 | 3 |
| 8 | 5 | 7 | 4 | 11 |

M

```
Draw2D({boxplot_description(M; range = 1,5)
        xlabel = "sample"})
```

If you want to specify several options to `boxplot_description`, then you might consider Maxima's mapping features:

$$\text{apply}\left(f; \begin{Bmatrix} a \\ b \\ c \end{Bmatrix}\right) = f(a; b; c)$$

## ellipse(xc; yc; a; b; ang; delta;)

Draws an ellipse centered at `xc`, `yc` with the radii `a` and `b`, starting at angle `ang` until angle `ang+delta`.

Caution! In Maxima 5.47 `delta` is multiplied by 2.

### Options

`nticks` Number of sampling points

`transparent` (true) filled or unfilled sector. Actual transparency must be set by `fill_color` in RGBA format.

`fill_color` (red) fill color in RGBA format.

`border` (true) draw the border

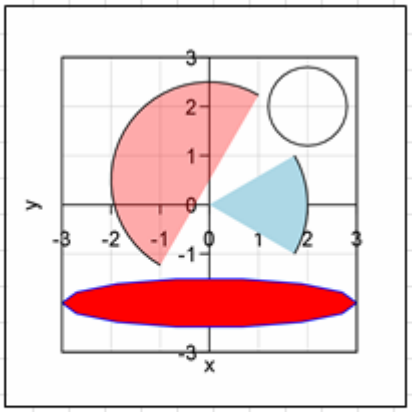
`line_type` (solid) line type for the border

`line_width` line width for the border

`key` (none) legend entry

`color` (blue) line color

```
obj := [ ellipse(0; -2; 3; 0,5; 0; 360)
         fill_color = lightblue
         color = black
         line_width = 1
         ellipse(0; 0; 2; 2; -30; 30)
         fill_color = "#ff0000aa"
         ellipse(0; 0,5; 2; 2; 60; 90)
         transparent = true
         ellipse(2; 2; 0,8; 0,8; 0; 180)
         yrange = {-3; 3}
         proportional_axes = xy
         xtics_axis = true
         ytics_axis = true ]

Draw2D(obj; {200; 200})
```



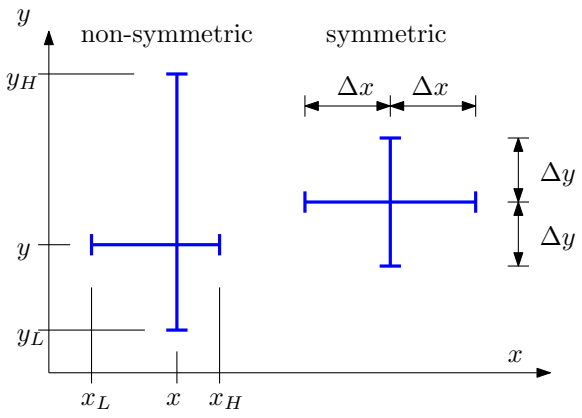
## errors(l1; l2;...)

Plots error bars. The arguments are lists, depending on the type of the error bars.

error_type = x : (x, y, Δx) or (x, y, $x_u$, $x_o$)

error_type = y : (x, y, Δy) or (x, y, $y_u$, $y_o$)

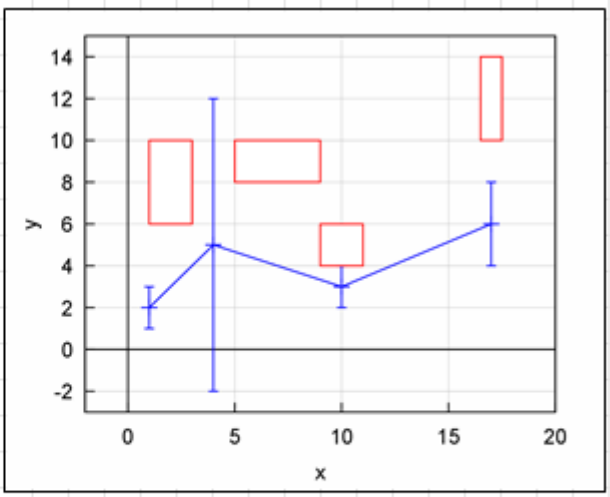
error_type = xy
error_type = boxes : (x, y, Δx, Δy) or (x, y, $x_u$, $x_o$, $y_u$, $y_o$)




### Options

`error_type` Type of error bars (`x`, `y`, `xy` or `boxes`)

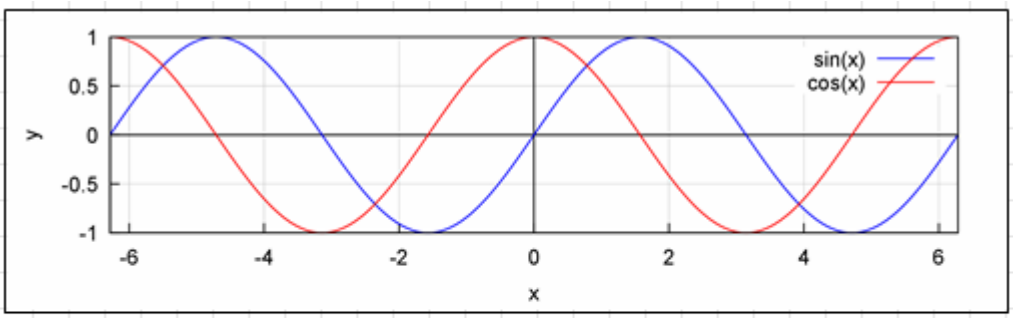
```
obj := { error_type = y
         points_joined = true
         errors = args( [[1 2 1]
                         [4 5 7]
                         [10 3 1]
                         [17 6 2]] )
         color = red
         error_type = boxes
         errors = args( [[2  8  1  2]
                         [7  9  2  1]
                         [10 5  1  1]
                         [17 12 1/2 2]] )
         xrange = {-2
                   20}
         yrange = {-3
                   15}
```




Draw2D(obj)

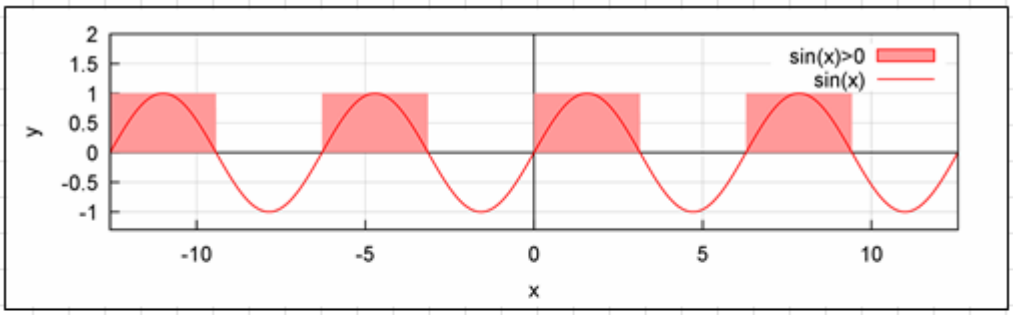
## `explicit`$(f(x);\ x;\ x_{\min};x_{\max})$

Plots an explicit function $y(x)$ within a given range of the variable $x$.




```
Draw2D({key = "sin(x)"
        explicit(sin(x); x; -2·π; 2·π)
        key = "cos(x)"                  ; {500
        color = red                       150
        explicit(cos(x); x; -2·π; 2·π)})
```

Use the option `filled_func` to shade areas between functions (here we use $y = 0$ as limit)




```
Draw2D({filled_func = 0
        fill_color = red
        fill_density = 0,4
        key = "sin(x)>0"
        explicit(charfun(sin(x)>0); x; -4·π; 4·π)
        key = "sin(x)"                              ; {500
        color = red                                   150
        filled_func = false
        explicit(sin(x); x; -4·π; 4·π)
        yrange = {-1,3
                  2})
```

## histogram_description(data; option1; option2...)

Creates the graphics elements for a histogram plot of a sample `data`. The specific options must be given as separate arguments (not as a list). Use `apply()` to map from a list to function arguments

**Options specific to histogram_description** .

`nclasses` (`10`): number of classes or list with the limits and an optional number of classes :

$$\begin{cases} x_{\min} \\ x_{\max} \\ n_k \end{cases} \text{ or } \begin{cases} x_{\min} \\ x_{\max} \end{cases}$$

`frequency` (`absolute`) scaling of the class values:

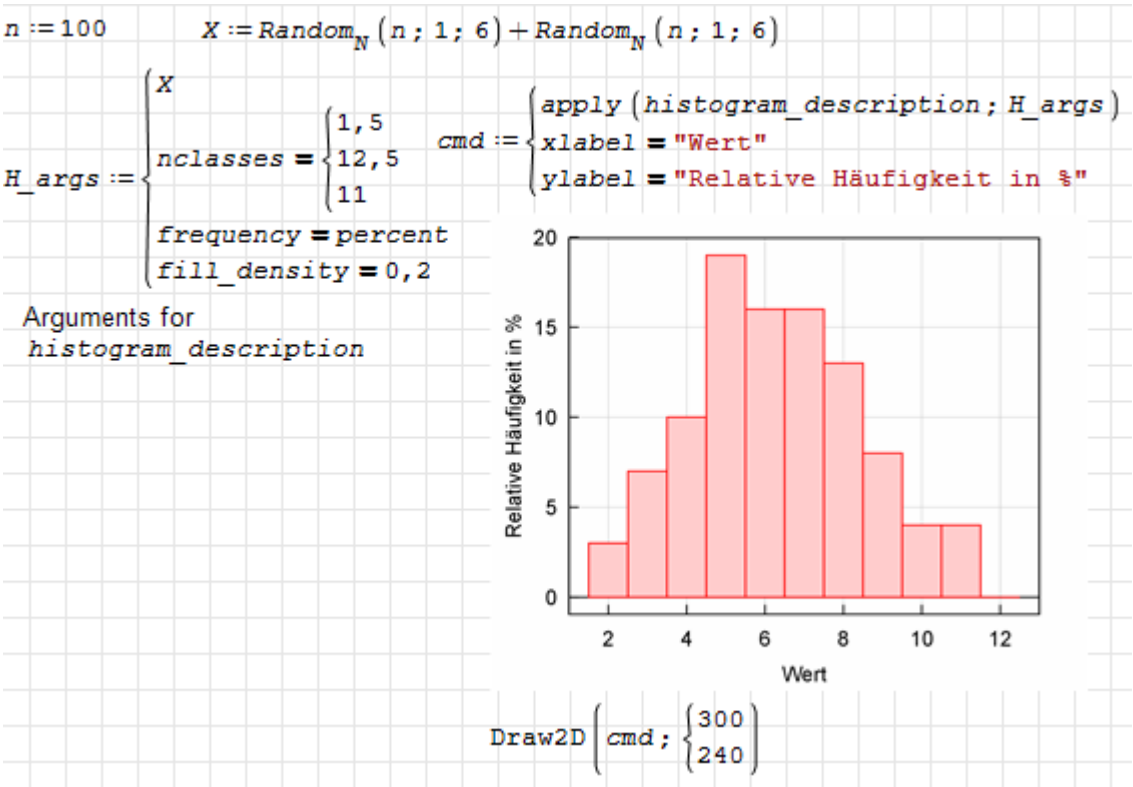
`absolute` Number of values in the class

`relative` Fraction of values in the class

`percent` Fraction of values in %

`density` Mean density of the distribution (fraction divided by width of the classes)

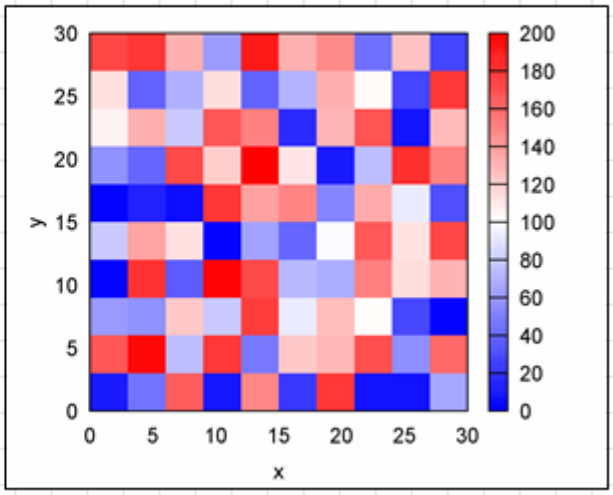
`htics` (`auto`) Tick location on the horizontal axis. Possible values, `auto`, `endpoints`, `intervals` or a list with labels



## image(Matrix; xmin; ymin; xmax; ymax)

Plots matrices of real values as color values (heat map) in the $xy$-plane with given axes limits.

```
obj := { image(Random_N(10; 10; 0; 200); 0; 0; 30; 30)
       { proportional_axes = xy
       { palette = { blue
                   { white
                   { red
Draw2D(obj)
```



## implicit(f(x; y); x; xmin; xmax; y; ymin; ymax)

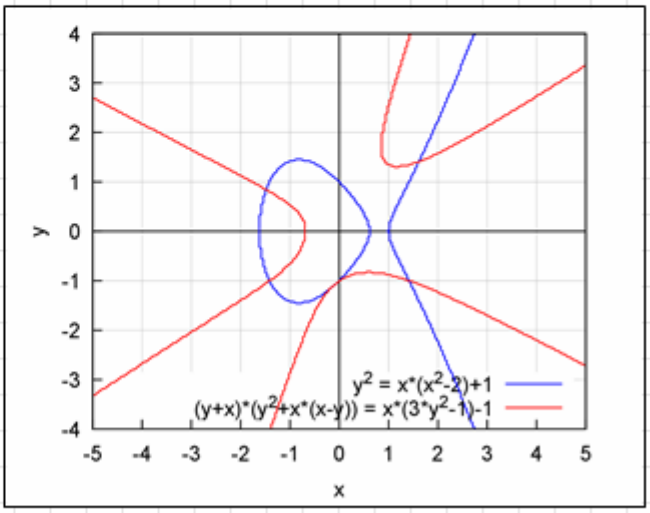
Plots implicit functions $f(x, y) = 0$ using the Marching squares algorithm. The sampling grid is defined by the options

`ip_grid` Number of sampling intervals in $x$- and $y$-direction

`ip_grid_in` Number of secondary interval

$f_1 := \left(y^2 = x^3 - 2 \cdot x + 1\right)$ $f_2 := \left(x^3 + y^3 = 3 \cdot x \cdot y^2 - x - 1\right)$ $r_x := 5$ $r_y := 4$

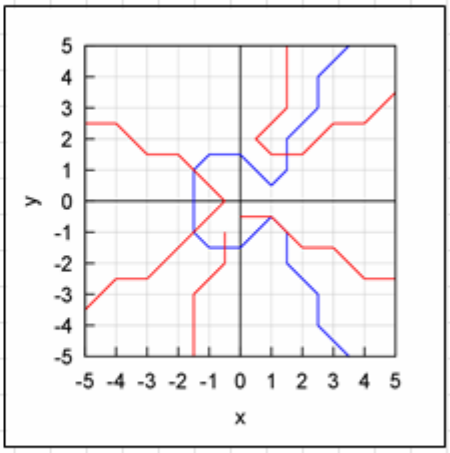
$$obj := \begin{cases} key = string\left(f_1\right) \\ implicit\left(f_1; x; -r_x; r_x; y; -r_y; r_y\right) \\ key = string\left(f_2\right) \\ color = red \\ implicit\left(f_2; x; -r_x; r_x; y; -r_y; r_y\right) \\ proportional\_axes = xy \\ key\_pos = bottom\_right \\ xtics = 1 \\ ytics = 1 \end{cases}$$




$Draw2D\left(obj; \begin{cases} 320 \\ 250 \end{cases}\right)$

$f_1 := \left(y^2 = x^3 - 2 \cdot x + 1\right)$ $f_2 := \left(x^3 + y^3 = 3 \cdot x \cdot y^2 - x - 1\right)$ $r_x := 5$ $r_y := 5$

$$obj := \begin{cases} ip\_grid = \begin{cases} 10 \\ 10 \end{cases} \\ ip\_grid\_in = \begin{cases} 1 \\ 1 \end{cases} \\ implicit\left(f_1; x; -r_x; r_x; y; -r_y; r_y\right) \\ color = red \\ implicit\left(f_2; x; -r_x; r_x; y; -r_y; r_y\right) \\ proportional\_axes = xy \\ xtics = 1 \\ ytics = 1 \end{cases}$$




$Draw2D\left(obj; \begin{cases} 220 \\ 220 \end{cases}\right)$

## label(list1, list2,...)

Inserts text labels at specified locations. The arguments are lists of three entries

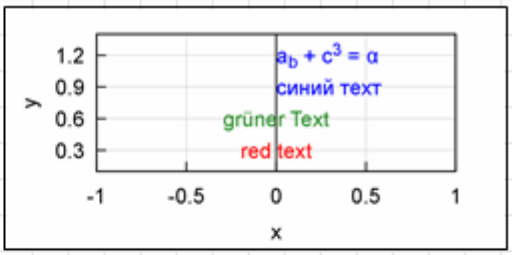
$$\begin{cases} \text{text} \\ x \\ y \end{cases}$$

### Options

`label_alignment` horizontal alignment: `center` (default) or `left`, `right`

`label_orientation horizontal` (default) or `vertical`

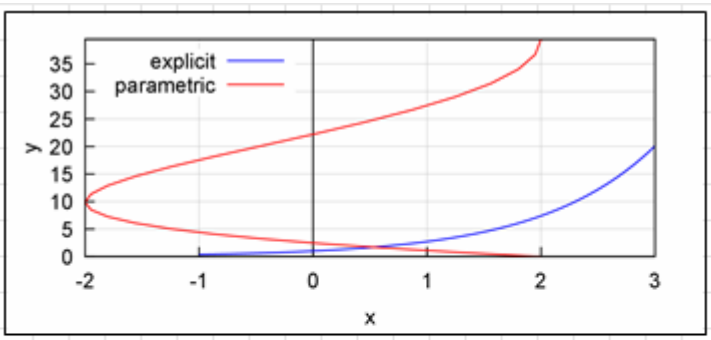
```
obj := { yrange = {0,1
                   1,4
       ytics = 0,3
       color = red
       label ({"red text"
               0
               0,3})
       color = forest_green
       label ({"grüner Text"
               0
               0,6})
       color = blue
       label_alignment = left
       apply(label; args({"синий текст"  0 0,9
                          "a_b + c^3 = α" 0 1,2}))
       xrange = {-1
                 1
```




```
Draw2D(obj; {250
             120})
```

## parametric(x(t); y(t); t; tmin; tmax)

Parametric functions.




```
Draw2D({key = "explicit"
        explicit(e^x; x; -1; 3)
        color = red
        key = "parametric"
        parametric(2·cos(r); r^2; r; 0; 2·π)
        key_pos = top_left}; {350
                              160})
```

## piechart_description(data; option1; option2...)

Pie chart. `data` is a list or a vector. The options need to be provided as function arguments.

The function returns a list of lists. The second entry of top level list is the actual diagram object. Therefore, the function has to be used with index 2 as in the example.

The diagram is assembled using `ellipse()` objects, Therefore it suffers from a bug in Maxima 5.47 (sector angle is double the specified value).

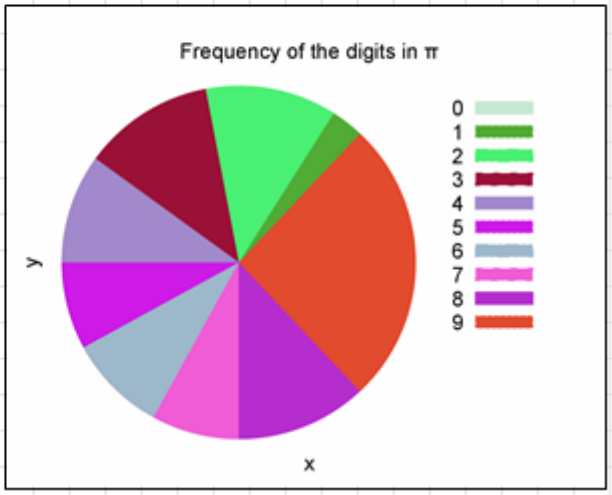
### Options

`sector_colors` (random) list with sector colors

`pie_center` $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ center of the pie

`pie_radius` (1) radius of the pie

```
msg : load(descriptive)
S : read_matrix(file_search("pidigits.data"))
p : [title = "Frequency of the digits in π",
     piechart_description(S)[2],
     proportional_axes = xy,
     xrange = [-1, 1.8],
     [axis_top = false, axis_bottom = false, axis_left = false,
      axis_right = false, xtics = false, ytics = false, xaxis = false]]
```



## points(matrix)

Plot points from

- a matrix of two columns (first column: $x$-values, second column: $y$-values) or
- a matrix of two rows and more than two columns (first row: $x$-values, second row: $y$-values) or
- a row or column vector (entries taken as $y$-value, index taken as $x$-value)

### Options

`point_size (1)`

`point_type` (plus). Point type given as number or as name, see section (3.5).

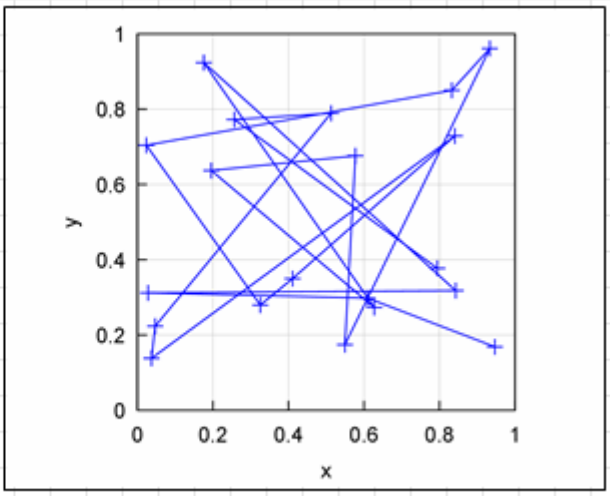
`points_joined (false)`:

`false` no additional lines

`true` connect points by lines

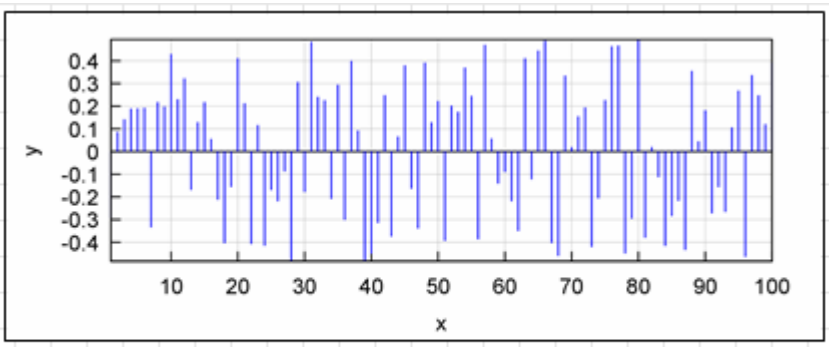
`impulses` vertical lines to the $x$-axis

```
cm := [points_joined = true
       points(Random(20; 2))
       proportional_axes = xy
       xrange = [0
                 1
       yrange = [0
                 1
Draw2D(cm)
```




Random $y$values plotted over their index

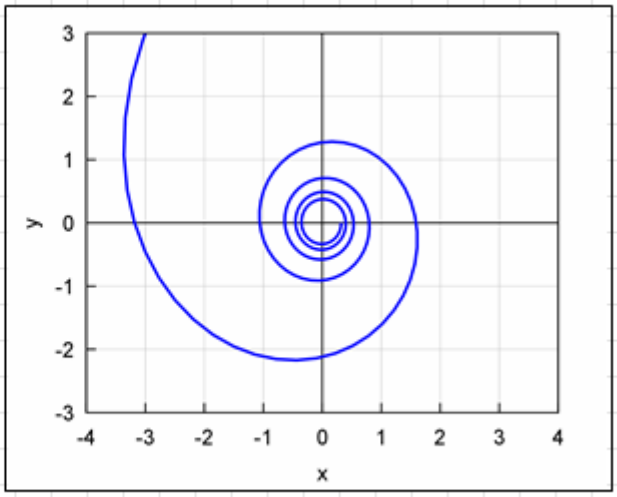
```
Draw2D([points_joined = impulses ; [400
        points(Random(100; 1) - 0,5)   [160)
```



## polar(r; phi; min; max)

2D-curve, given in polar coordinates $r(\varphi)$.

```
cmd := { nticks = 200
       { xrange = { -4
       {          {  4
       { yrange = { -3
       {          {  3
       { line_width = 2
       { polar(10/w; w; 1; 10·π)
Draw2D(cmd)
```



# polygon()

Filled polygon. Arguments:

- Two lists: one containing the $x$-values and the second containing the $y$-values.

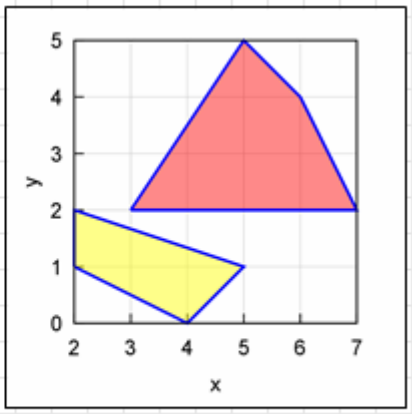
```
apply(polygon; args([[2 2 5 4]
                     [1 2 1 0]])) = polygon([2 2 5 4]; [1 2 1 0])
```

- One lists of $n$ lists: each containing the $x$and $y$-values of a vertex.

```
polygon(args([[3 2]
              [7 2]
              [6 4]
              [5 5]])) = polygon([[3 2]; [7 2]; [6 4]; [5 5]])
```

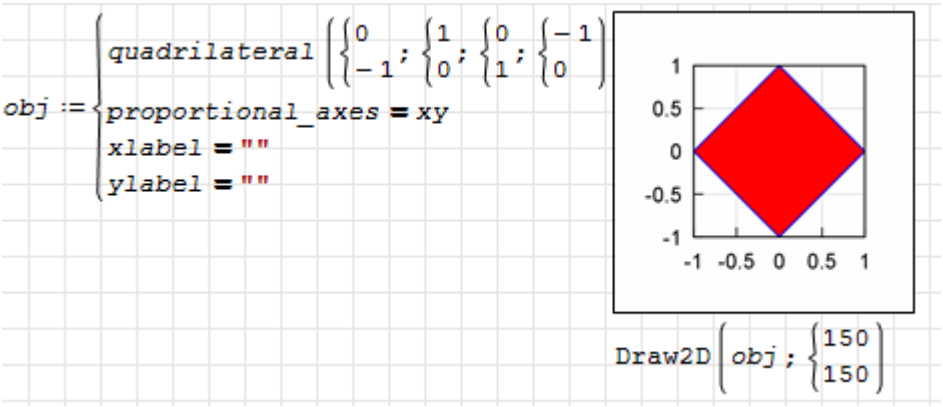
Here we demonstrate both versions.

```
obj := {line_width = 2
        fill_color = "#ffff0088"
        apply(polygon; args([[2 2 5 4]
                             [1 2 1 0]]))
        fill_color = "#ff000088"
        polygon(args([[3 2]
                      [7 2]
                      [6 4]
                      [5 5]]))
        proportional_axes = xy}
Draw2D(obj; [200 200])
```



## quadrilateral(p1; p2; p3; p4)

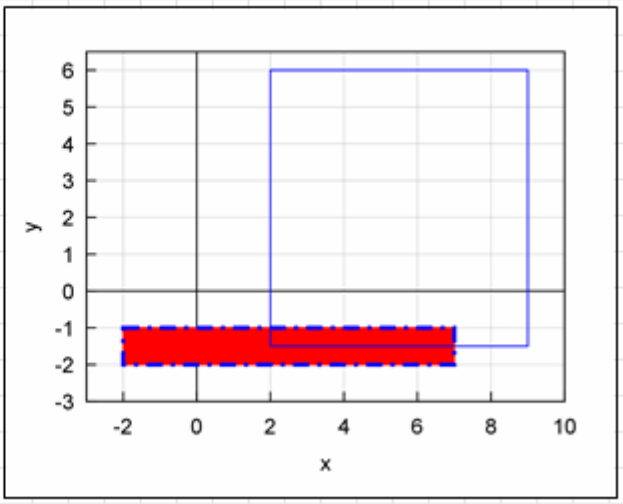
This is an alternative to a polygon with four points. Yet the input is different, each argument is a two-element list with the vertex coordinates.



## rectangle(p1; p2)

Rectangle, drawn between two points given as coordinate lists.

```
cm := { line_width = 3
      { line_type = short_long_dashes
      { fill_color = red
      { rectangle ( {-2 ; {7 )
                  {-2   {-1
      { transparent = true
      { line_type = solid
      { line_width = 1
      { rectangle ( {9 ; {2
                  {6   {-1,5
      { xrange = {-3
                 {10
      { yrange = {-3
                 {6,5
      { proportional_axes = xy
```



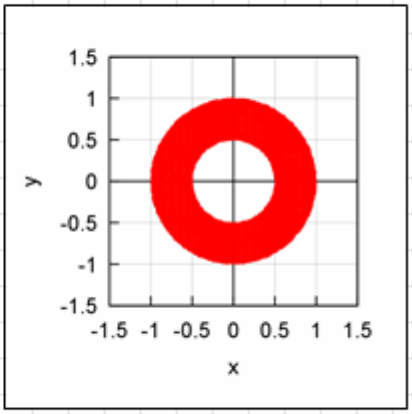

```
Draw2D ( cm )
```

## region(condition; x; xmin; xmax; y; ymin; ymax)

Fill a region within the given limits, where it meets a given logical condition.

$$R := \left(\frac{1}{4} < \left(x^2 + y^2\right) < 1\right)$$

$$obj := \begin{cases} x\_voxel = 30 \\ y\_voxel = 30 \\ region\left(R; x; -\frac{3}{2}; \frac{3}{2}; y; -\frac{3}{2}; \frac{3}{2}\right) \\ proportional\_axes = xy \end{cases}$$

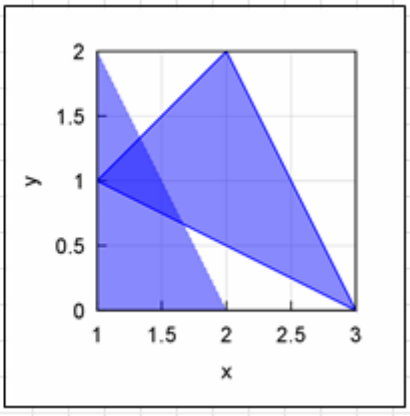



$$Draw2D\left(obj; \begin{pmatrix} 200 \\ 200 \end{pmatrix}\right)$$

## triangle(p1; p2; p3)

Triangle given by three lists of vertex coordinates.

```
p := Draw2D(⟨fill_color = "#0000ff88"
             triangle({1;1}; {2;2}; {3;0})
             proportional_axes = xy
             border = false
             triangle({1;0}; {1;2}; {2;0})⟩; {200;200})
```



## vector(p; d)

Arrow with vector components `d` starting at point `p`. Both arguments are lists with 2 coordinate entries.

### Options

`head_angle` (45) Angle between the arrow heads and the segment in °

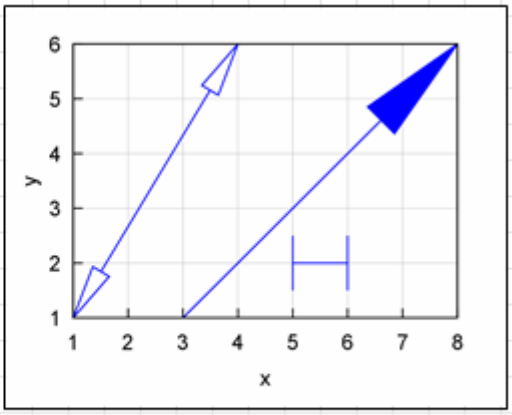
`head_both` (`false`) A single head if `false` or two heads if `true`

`head_length` (2) Head length in units of the $x$-axis

`head_type` (`filled`): `filled`, `nofilled`: open, `empty`: closed but not filled

`unit_vectors` (`false`)if true, the vector is scaled to length 1.

```
obj := { proportional_axes = xyz
       { head_angle = 10
       { vector([3;1];[5;5])
       { head_both = true
       { head_length = 1
       { head_type = empty
       { vector([1;1];[3;5])
       { unit_vectors = true
       { head_angle = 90
       { vector([5;2];[3;0])
```




```
Draw2D(obj; [250;200])
```

# 5 3D Graphics Objects

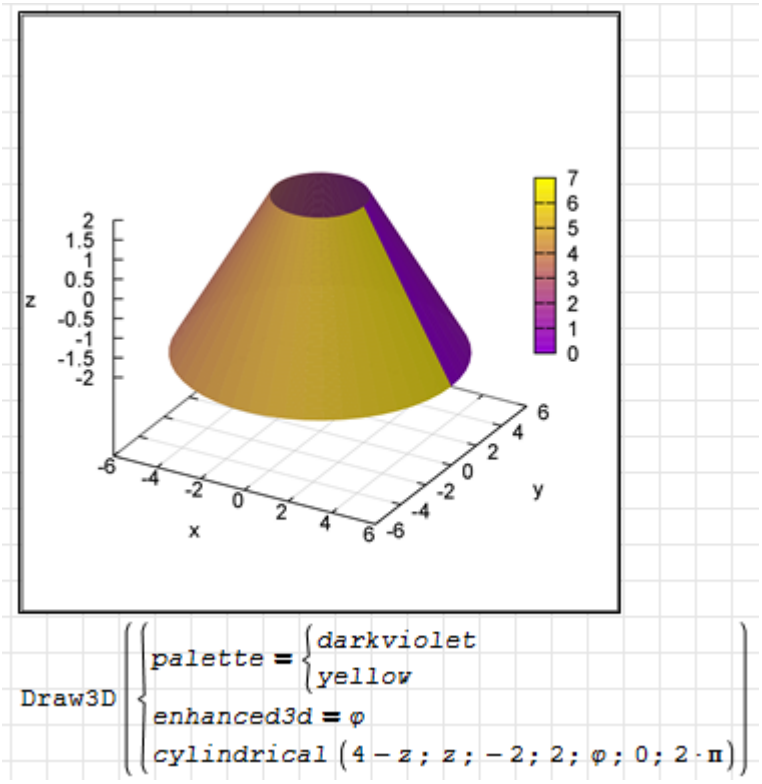
## Contents



Here we demonstrate the available objects for 3D graphics, i.e. for use with `Draw3D()` or the Draw3D region. For some of the objects, we also explain some of the most important options. The Draw Description snippet is a good source on available options.

## cylindrical(r(z;φ);z;$z_{min}$;$z_{max}$;φ;$φ_{min}$;$φ_{max}$)

This function plots surfaces given by the radius as a function of $z$ and of the azimuth angle $\varphi$.



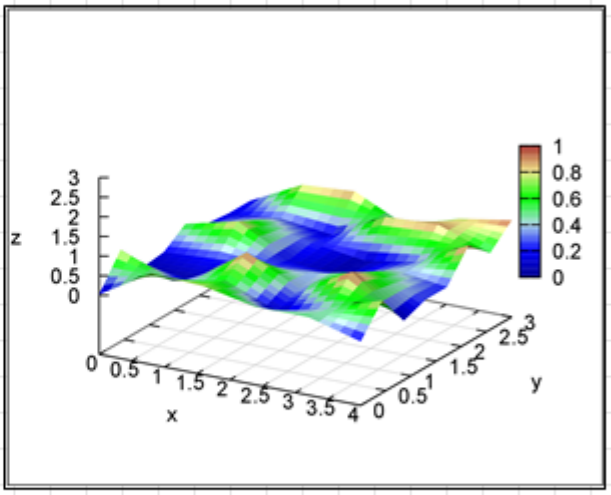

This is an example of how to color the plot based on the angular coordinate and based on a custom palette.

## elevation_grid($\underline{M}$; $x_0$; $y_0$; $w$; $h$)

3D representation of a $m \times n$ matrix of $z$-values. $M_{11}$ corresponds to $x = x_0$ and $y = y_0 + h$, $M_{mn}$ corresponds to $x = x_0 + w$ and $y = y_0$.

The example shows a $8 \times 6$ random matrix with a custom color palette. In Gnuplot, the faces have uniform color. In order to have color gradients on a face, the `interpolate_color` option can be used.

```
obj := { elevation_grid (Random (8; 6); 0; 0; 4; 3)
       { ztics = 0,5
       { palette = { darkblue
       {           { blue
       {           { lightblue
       {           { green
       {           { khaki
       {           { brown
       { zrange = { 0
       {          { 3
       { cbtics = 0,2
       { interpolate_color = { 5
       {                     { 5

Draw3D (obj ; { 300
              { 240 )
```



## explicit(z(x; y); x; xmin; xmax; y; ymin; ymax)

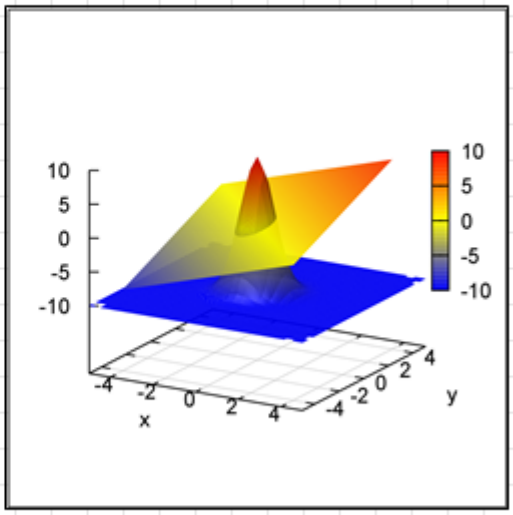
Plots explicit functions $z(x, y)$ within given ranges of the variables $x$ and $y$.

**Options:**

**palette** list of colors. (color) The value from enhanced3D is uniformly mapped to this list.

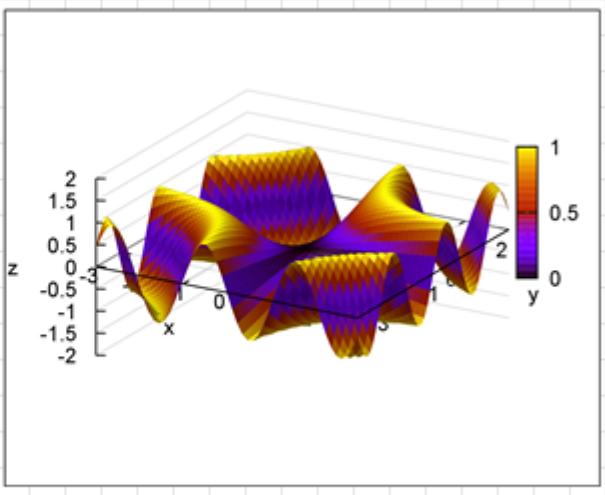
**enhanced3d** $(z)$ how to generate values for coloring. Specified as list with the entries $f(x, y, z), x, y, z$

```
gauss := explicit(20·e^(-x^2-y^2) - 10; x; -5; 5; y; -5; 5)
plane := explicit(x + y; x; -4; 4; y; -4; 4)
obj := { interpolate_color = true
         xu_grid = 30
         yv_grid = 30
         palette = {blue, yellow, red}
         gauss
         plane
         view = {70, 30}
         zrange = {-10, 10}
Draw3D(obj; {250, 250})
```




Here we use the default color map to color by the distance from the $xy$-plane $f = |z|$.

```
f := explicit(sin(x·y); x; -3; 3; y; -3; 3)
cmd := { enhanced3d = {|z|, x, y, z}
         f
         zrange = {-2, 2}
         cbtics = 0,5
         xyplane = 0
         grid = true
Draw3D(cmd; {300, 240})
```

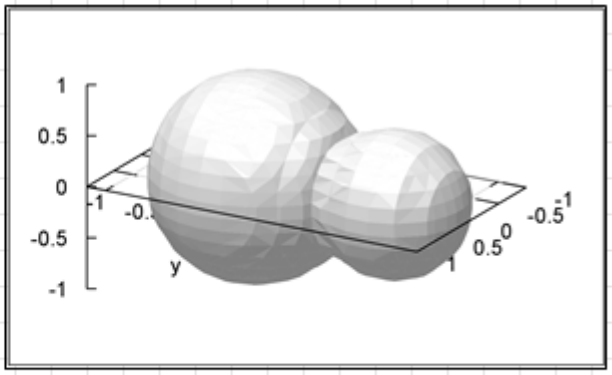


## `implicit`$(f(x,y,z);x;x_{\min};x_{\max};y;y_{\min};y_{\max};z;z_{\min};z_{\max})$

Visualization of implicit functions $f(x,y,z)=0$ within the given range of the variables.

- Number of intervals for the Marching Cubes Algorithm is set by the options `x_voxel`, `y_voxel`, `z_voxel`.

```
f := ((x^2 + y^2 + z^2 - 1) · (x^2 + (y - 1,5)^2 + z^2 - 0,5) = 0,015)

obj := { palette = { white
                   { white
       { x_voxel = 15
       { y_voxel = 15
       { z_voxel = 15
       { implicit (f; x; -1; 1; y; -1,2; 2,3; z; -1; 1)
       { user_preamble = "set view 70, 120, 2.8, 1.4"
       { colorbox = false
       { proportional_axes = xyz
       { xyplane = 0

Draw3D ( obj ; { 300
               { 180 )
```



## label(list1, list2,...)

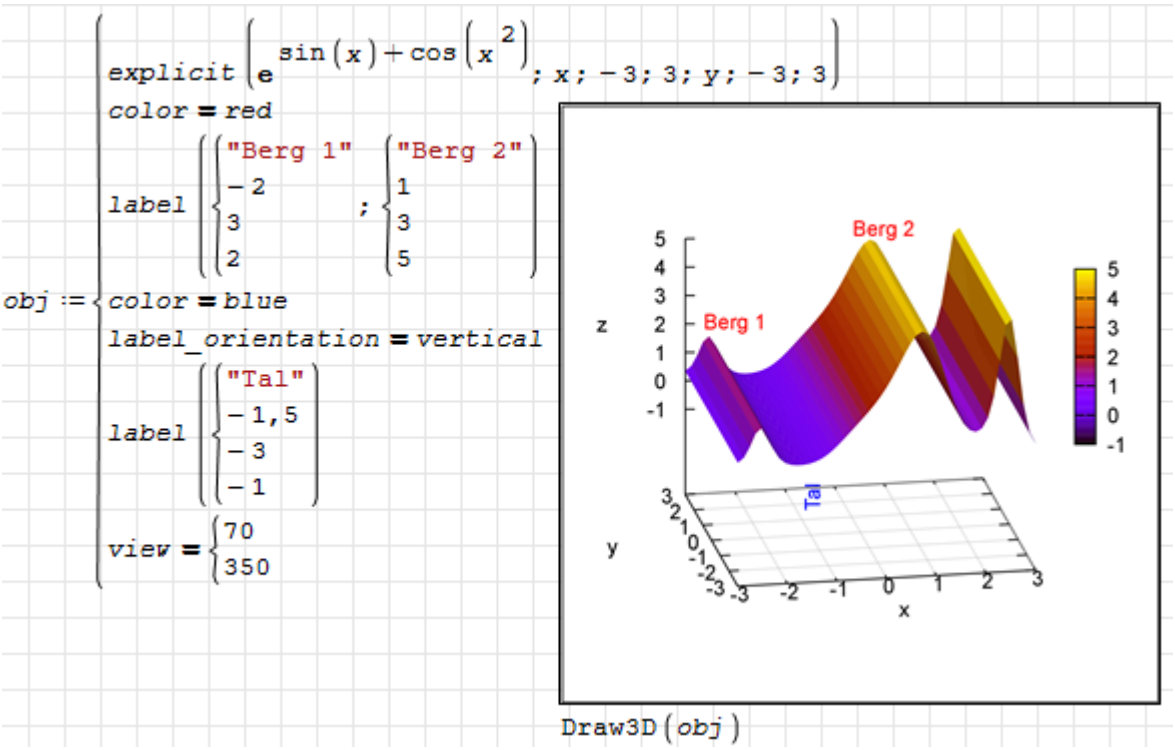
Inserts text labels at specified locations. The arguments are lists of four entries

$$\begin{cases} \text{text} \\ x \\ y \\ z \end{cases}$$

### Options

`label_alignment` horizontal alignment: `center` (default) or `left`, `right`

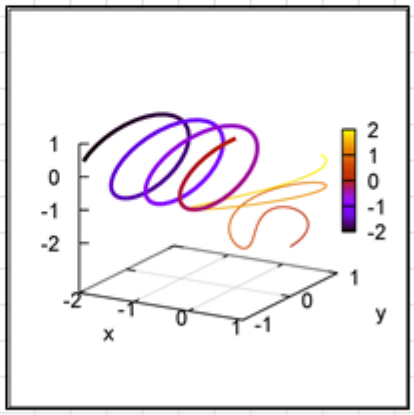
`label_orientation` `horizontal` (default) or `vertical`



## parametric(fx; fy; fz; t; tmin; tmax)

Parametric curve, given by three functions of parameter $t$. The sampling density is controlled by option `nticks`.

```
obj := { enhanced3d = {u
                       u
       nticks = 100
       parametric((cos(5·u))^2; sin(7·u); u − 2; u; 0; 2)
       line_width = 3
       parametric(t; sin(10·t); cos(10·t); t; −2; 0)
       view = {70
               30
       xtics = 1
       ytics = 1
       ztics = 1
       cbtics = 1
```




```
Draw3D(obj; {200
             200)
```

## parametric_surface(fx;fy;fz;u;umin;umax;v;vmin;vmax)

Parametric surface, given by three functions of two parameters. The following example is inspired by the Gnuplot examples page. The sampling density of the parameters is controlled by the options `xu_grid` and `yv_grid`.

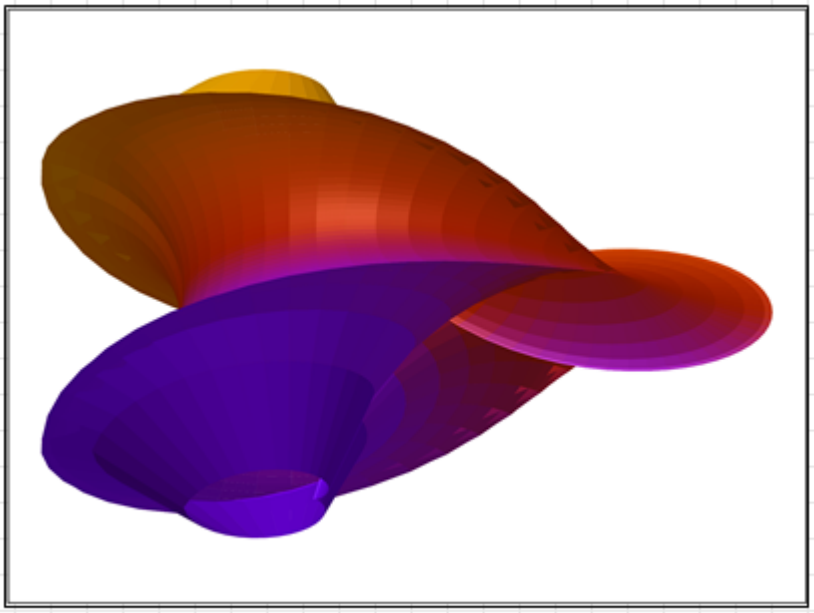
$$a := 1$$

$$fx := \frac{2 \cdot a \cdot (\cos(u) + u \cdot \sin(u)) \cdot \sin(v)}{1 + u^2 \cdot (\sin(v))^2}$$

$$fy := \frac{2 \cdot a \cdot (\sin(u) - u \cdot \cos(u)) \cdot \sin(v)}{1 + u^2 \cdot (\sin(v))^2}$$

$$fz := a \cdot \ln\left(tg\left(\frac{v}{2}\right)\right) + \frac{2 \cdot \cos(v)}{1 + u^2 \cdot (\sin(v))^2}$$

```
obj := { axis_3d = false
       { colorbox = false
       { xu_grid = 51
       { yv_grid = 51
       { user_preamble = {"set view 120, 357, 1.75, 1.0"
       { parametric_surface(fx; fy; fz; u; -4,5; 4,5; v; 0,05; π-0,05)
```



```
Draw3D(obj; {400
            {300)
```

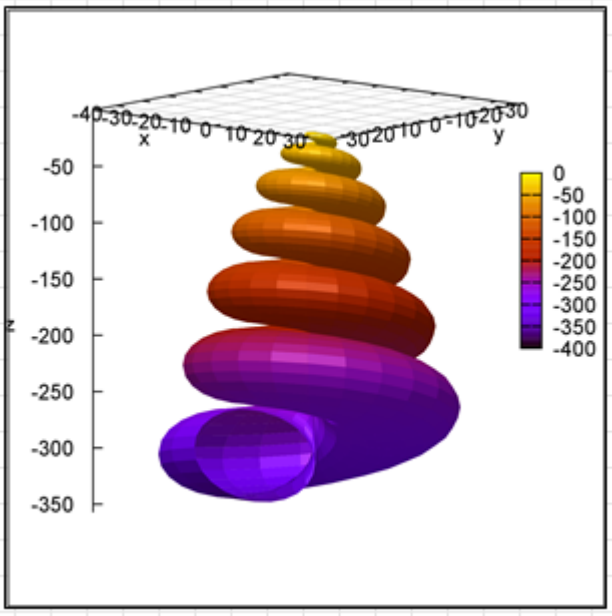
This example is from the Maxima documentation:

$$fx := \frac{u}{2} \cdot \cos(u) \cdot (\cos(v) + 1)$$

$$fy := \frac{u}{2} \cdot \sin(u) \cdot (\cos(v) + 1)$$

$$fz := u \cdot \sin(v) - \left(\frac{u+3}{8} \cdot \pi\right)^2 - 20$$

$$obj := \begin{cases} user\_preamble = \{\text{"set view 100, 320, 1, 1.5"} \\ xu\_grid = 150 \\ yv\_grid = 25 \\ parametric\_surface(fx; fy; fz; u; 0; 13 \cdot \pi; v; -\pi; \pi) \\ xyplane = 0 \end{cases}$$




Draw3D(*obj*)

## points(matrix)

Plot points at $(x, y, z)$ given by

- a $3 \times n$ matrix, each column is a point or
- a $n \times 3$ matrix, each row is a point.

### Options

`enhanced3d (color)` coloring of the points. If you specify a list of two entries.$j$ is the index of the point in the matrix (row index) and $f(j)$ is an arbitrary function of that index

$$\begin{cases} f(j) \\ j \end{cases}$$

`point_size (1)`

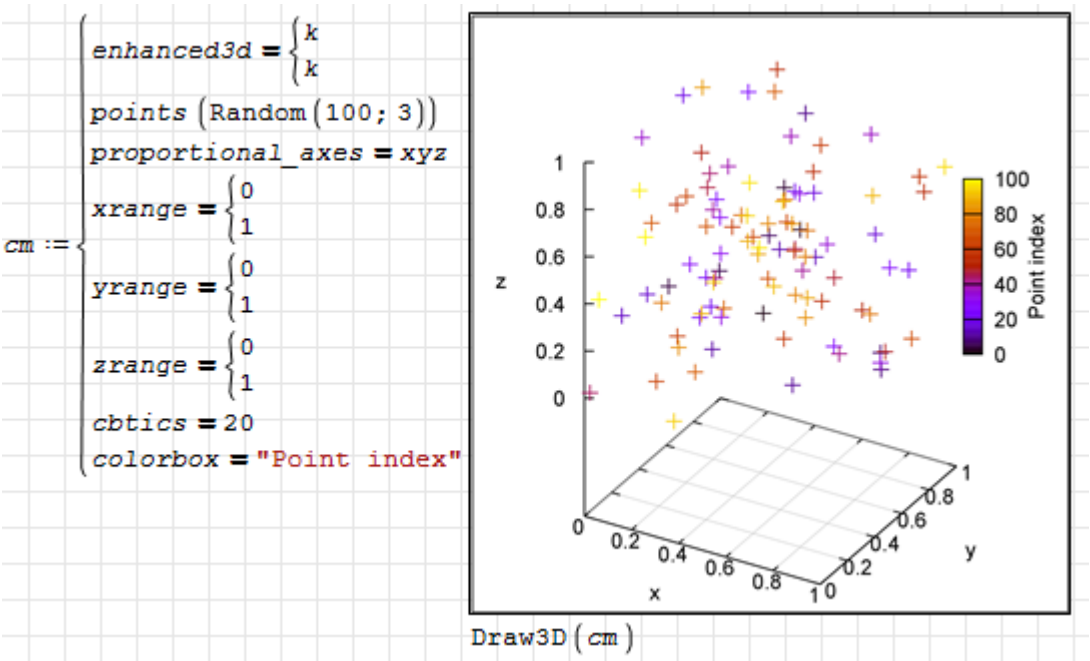
`point_type` (plus). Point type given as number or as name.

`points_joined (false)`:

`false` no additional lines

`true` connect points by lines (color is averaged between adjacent points)

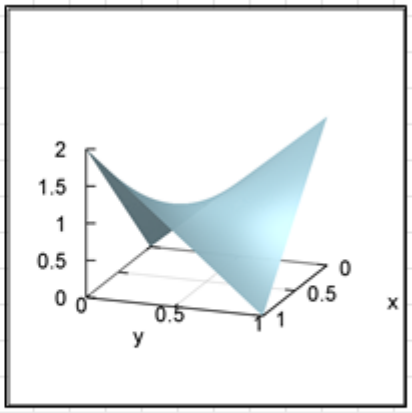
`impulses` vertical lines to the $x$-axis



## quadrilateral(p1; p2; p3; p4)

This draws a quadrilateral. Each argument is a three-element list with the vertex coordinates.

```
cm := { palette = { lightblue
                  { lightblue
      { quadrilateral ( { 0   { 1   { 0   { 1
                        { 0 ; { 0 ; { 1 ; { 1
                        { 0   { 2   { 2   { 0
      { xyplane = 0
      { interpolate_color = true
      { colorbox = false
      { view = { 70
               { 110
      { xtics = 0,5
      { ytics = 0,5
```

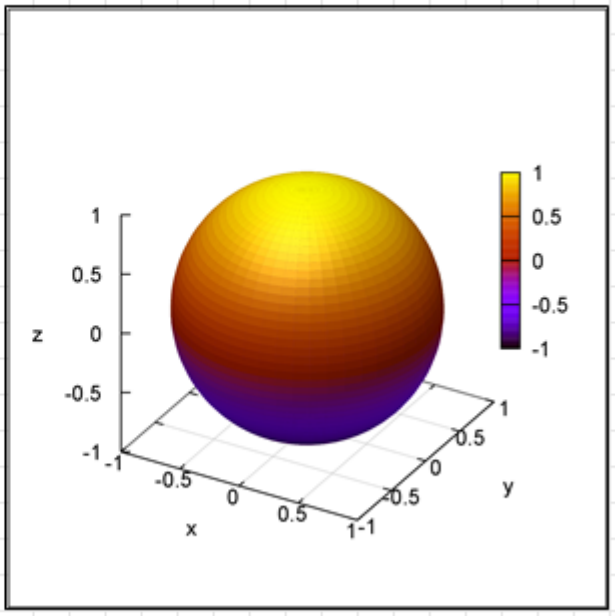



```
Draw3D ( cm ; { 200
              { 200 )
```

## spherical(r; a; amin; amax; z; zmin; zmax)

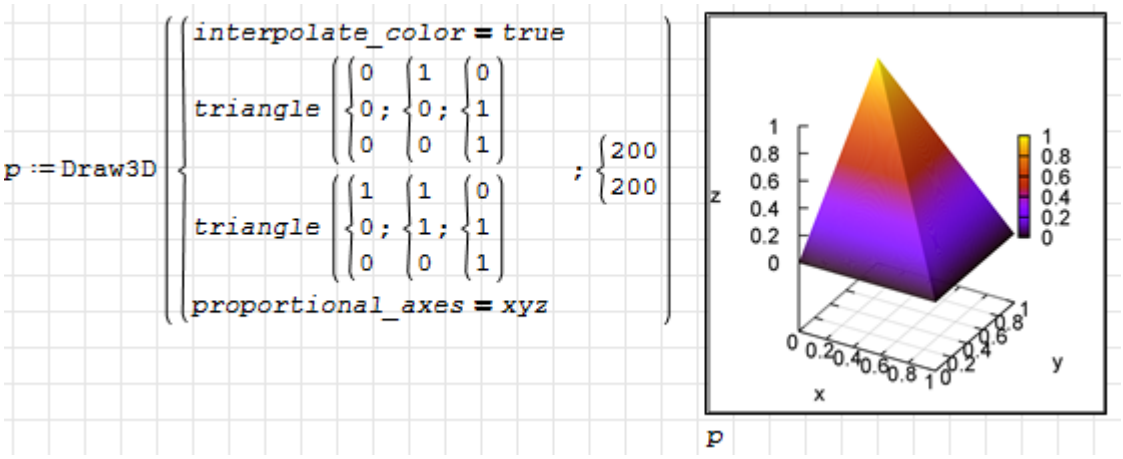
Surface given by a radius $r(\alpha, \beta)$ in spherical coordinates with $\alpha$ being the azimuth angle counted from the $x$-axis and $\beta$ being the angle from zenith (north pole).

```
cmds := { user_preamble = {"set view 60, 30, 1.0, 1.2"
        | spherical (1; azi; 0; 2·π; zen; 0; π)
        | proportional_axes = xyz
        | xtics = 0,5
        | ytics = 0,5
        | xyplane = -1
Draw3D (cmds)
```



## triangle(p1; p2; p3)

Triangle given by three lists of vertex coordinates.



## tube(x; y; z; r; p; min; max)

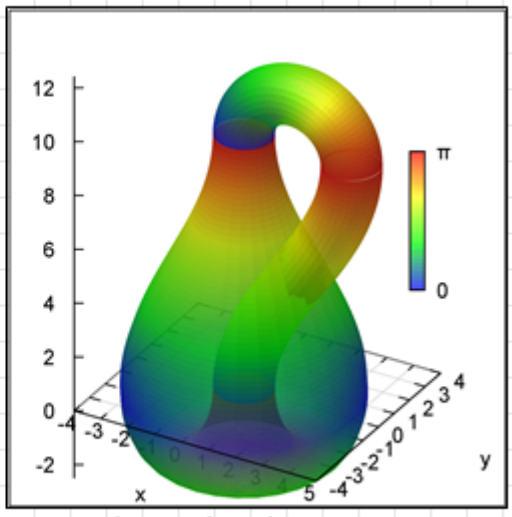
Tubular surface, given by center line coordinates $x, y, z$ and radius $r$ as function of the path parameter $p$.

The example is a Klein bottle. The individual parts have colors assigned, which are only relevant for the surface grid.

```
a := 2,5   b := 1,5   c := 2   d := 3

bottle := color = blue
          tube(0; 0; (-a)·sin(t); a + b·cos(t); t; 0; π/2)
          tube(0; 0; (-a)·sin(t); a + b·cos(t); t; π/2; π)
          color = red
          tube(0; 0; d·t; a + b·cos(t); t; 0; π)
          color = forest_green
          tube(c - c·cos(t); 0; d·t; a - b; t; 0; π)
          color = magenta
          tube(c - c·cos(t); 0; d·π + c·sin(t); a - b; t; 0; π)
```

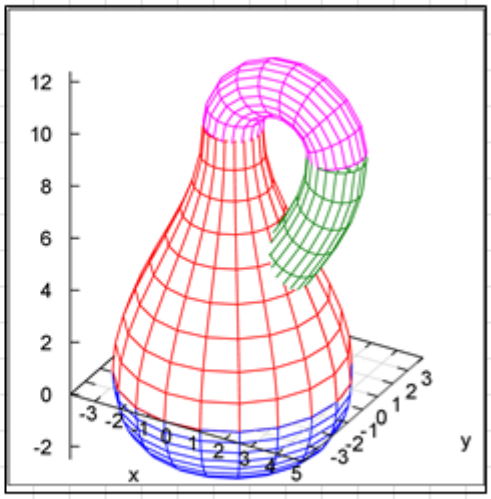
Here we use a non-standard palette for coloring by path parameter.

```
obj := user_preamble = {"set style fill transparent solid 0.7"
                        "set view 60, 30, 1.3, 1.1"
       xu_grid = 50
       yv_grid = 29
       enhanced3d = {t
                     t
       palette = {blue
                  green
                  yellow
                  red
       xyplane = 0
       proportional_axes = xyz
       cbtics = set({"0" ; "π"
                     0   ; π})
       bottle

Draw3D(obj; {250
             250})
```



The same plot with surface grid instead of shading. Shading is disabled using `enhanced3D=none`. The surface grid respects the color specification.

```
obj := { user_preamble = {"set view 60, 30, 1.3, 1.1"
       | enhanced3d = none
       | xu_grid = 10
       | yv_grid = 20
       | xyplane = 0
       | surface_hide = true
       | proportional_axes = xyz
       | cbtics = set ({"0" ; {"π"
       |                {0   ;  π}
       | bottle

Draw3D (obj ; {250
               {250)
```



## vector(p; d)

Arrow with vector components `d` starting at point `p`. Both arguments are lists with 2 coordinate entries.

### Options

`head_angle` (45) Angle between the arrow heads and the segment in °

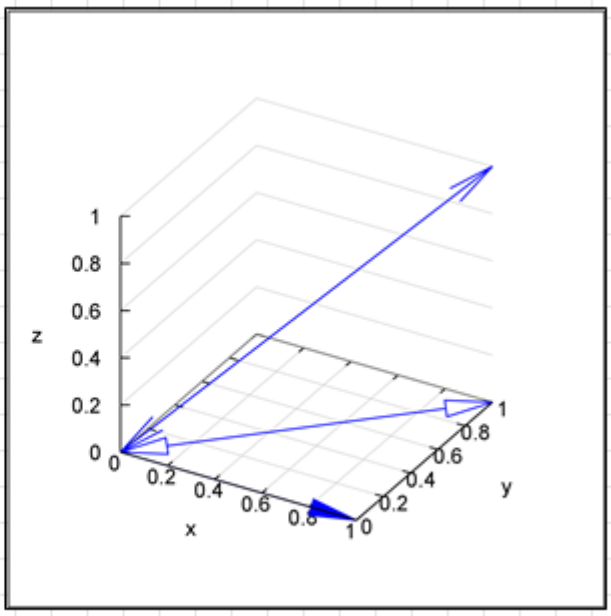
`head_both` (`false`) A single head if `false` or two heads if `true`

`head_length` (2) Head length in units of the $x$-axis

`head_type` (`filled`): `filled`, `nofilled`: open, `empty`: closed but not filled

`unit_vectors` (`false`)if true, the vector is scaled to length 1.

```
obj := { proportional_axes = xyz
         head_length = 0,1
         head_angle = 10
         vector( {0;0;0} , {1;0;0} )
         head_both = true
         head_type = empty
         vector( {0;0;0} , {1;1;0} )
         head_type = nofilled
         vector( {0;0;0} , {1;1;1} )
         xyplane = 0
         grid = true }
Draw3D(obj)
```



# 6 Curve Fitting

## Contents



*Fit.sm*

The function `Fit()` is an interface to Maxima's *lsquares* package for fitting mathematical models to data. It is augmented by `MSE()` and `Residuals()` for evaluation of the quality of the fit and `Assign()` for converting the result represented by equations into assignments.

`Fit(data; vars; eqn; pars; init; tol)`

`data` (matrix) Given data. Each line is a a data point, each column is a variable.

`vars` (list) Names of variables, one for each column in `data`.

`eqn` Equation or expression (which will augmented by = 0), which contains the Variables listed in vars `vars` and the model parameters listed in `pars`.

`pars` (list) Names of the free model parameters in `eqn`.

`init` (list) Initial values for the numeric search for optimal parameter values.

`tol` (optional number) Tolerance for the precision of the model parameters.

The function returns a list of Boolean equations `parameter = value`. This list can be used for value assignment using the function `Assign()` , e.g. if you want to plot the model curve.

The quality of the curve fit can be evaluated using the functions `MSE()` and `Residuals()`, which return the mean square error and the deviation between model and data respectively.

Note that the fit is a numeric procedure. The result may depend on the initial values of the parameters.

## 6.1 Fit of a Cubic Parabola to 5 Random Points

A cubic parabola $f(x) = ax^3 + bx^2 + cx + d$ is to be fitted to five points with $x_i = 1 \ldots 5$ and $y_i$ being random values between 0 and 1.

First, the data matrix is created by combining the vectors with the $x$-values and the $y$-values. Then, the model function $f(x)$ is defined. The free parameters must be undefined variables. If in doubt, apply the function `Clear()` to the parameters.

$$D := \text{augment}([1..5]; \text{Random}(5)) \qquad D = \begin{bmatrix} 1 & 0,461 \\ 2 & 0,736 \\ 3 & 0,406 \\ 4 & 0,849 \\ 5 & 0,845 \end{bmatrix}$$

$$\text{Clear}(a; b; c; d) = 1$$

$$f(x) := a \cdot x^3 + b \cdot x^2 + c \cdot x + d$$

Now the Fit() function is applied. Display the result with **Context menu> Optimization> None**. Otherwise the results aren't displayed properly.

Data and column names — model — parameters and initial guess — optimized parameter values

$$F := \text{Fit}\left(D; \begin{pmatrix} x \\ y \end{pmatrix}; y = f(x); \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}; \begin{pmatrix} 1, \\ 3, \\ 1, \\ 1, \end{pmatrix}\right) = \begin{pmatrix} a = 0,01308076422217354 \\ b = -0,10245656098800907 \\ c = 0,3052002065923548 \\ d = 0,2820408383091464 \end{pmatrix}$$

Context menu> Optimization> None

`Assign()` is used to convert the equations from `Fit()` into parameter assignments. Thus the model function can be plotted using the result of the fit.

The quality of the fit is evaluated by the model-data-deviations (residuals) and by the mean square error (MSE). These functions take the same arguments like `Fit()`.

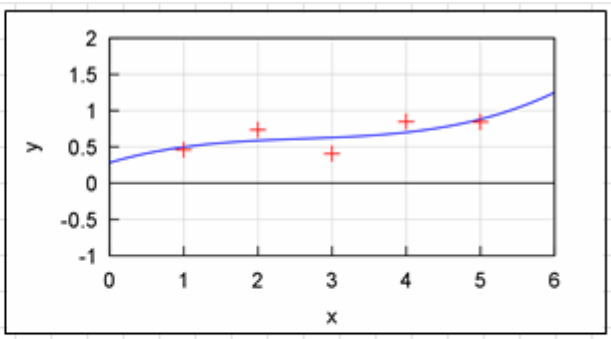
$$\text{Assign}(F) = \begin{pmatrix} 0,0131 \\ -0,1025 \\ 0,3052 \\ 0,282 \end{pmatrix}$$

Quality of the fit

$$R := \text{Residuals}\left(D; \begin{pmatrix} x \\ y \end{pmatrix}; y = f(x)\right)$$

$$R = \begin{bmatrix} -0,0371 \\ 0,149 \\ -0,223 \\ 0,149 \\ -0,0371 \end{bmatrix}$$

$$\text{MSE}\left(D; \begin{pmatrix} x \\ y \end{pmatrix}; y = f(x)\right) = 0,0193$$




$$\text{Draw2D}\left(\begin{pmatrix} \text{explicit}(f(x); x; 0; 6) \\ \text{color} = \text{red} \\ \text{points}(D) \\ \text{yrange} = \begin{pmatrix} -1 \\ 2 \end{pmatrix} \end{pmatrix}; \begin{pmatrix} 300 \\ 160 \end{pmatrix}\right)$$

## 6.2 Fit of a Circle to 4 Points

Find a circle with center point $x_x, y_c$ and radius $R$ such that it is an optimal fit to four given points.

The circle is represented by an implicit equation $f(x, y) = 0$. The involved variables are cleared, just to be sure they are undefined.

$$D := \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 2 & 1 \\ 2 & 3 \end{bmatrix} \qquad \text{Clear}(x_c; y_c; R; x; y) = 1$$

$$f(x; y) := (x - x_c)^2 + (y - y_c)^2 - R^2$$

There is no solution such that all points are on the perimeter of the circle, so an approximate solution is to be found. The function `Fit()` is used to determine the parameters of the circle such that the mean square error of $f(x, y) = 0$ at the data points is minimized.

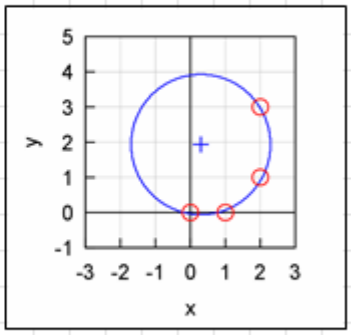
$$F := \text{Fit}\left(D; \begin{Bmatrix} x \\ y \end{Bmatrix}; f(x; y) = 0; \begin{Bmatrix} x_c \\ y_c \\ R \end{Bmatrix}; \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix}\right) = \begin{cases} x_c = 0,29924286819971646 \\ y_c = 1,9337810843605383 \\ R = 1,990876602322017 \end{cases}$$

Context menu> Optimization> None

The resulting equations are converted to assignments. In the plot the implicit equation is used directly. Center and data points are added to the plot.

$$\text{Assign}(F) = \begin{Bmatrix} 0,299 \\ 1,93 \\ 1,99 \end{Bmatrix} \qquad x_c = 0,299 \qquad y_c = 1,93 \qquad R = 1,99$$

$$cmd := \begin{cases} \text{implicit}(f(x; y); x; -3; 3; y; -1; 5) \\ \text{points}([x_c \; y_c]) \\ \text{color} = \text{red} \\ \text{point\_type} = \text{circle} \\ \text{points}(D) \\ \text{proportional\_axes} = \text{xy} \end{cases}$$




$$\text{Draw2D}\left(cmd; \begin{Bmatrix} 170 \\ 160 \end{Bmatrix}\right)$$

For an exact solution, the function $f(x, y)$ should be zero at the data points. The deviation is shown using `Residuals()`. This isn't the real distance between points and perimeter but it is approximately proportional to it.

$$\text{Residuals}\left(D; \begin{Bmatrix} x \\ y \end{Bmatrix}; f(x; y) = 0\right) = \begin{bmatrix} -0,1345 \\ 0,267 \\ -0,1991 \\ 0,0658 \end{bmatrix} \qquad f(D_{1\,1}; D_{1\,2}) = -0,1345$$

# 7 Algebraic Equations

## Contents



*Solve.sm*

The function `Solve()` is an interface to Maxima's `solve()` function. It's advantages are:

- Symbolic solution of linear and non-linear systems of equations.
- No need for initial guesses or search intervals.
- Simple handling of multiple solutions.
- The function can handle units of measurement.

Sometimes the symbolic expressions become too complex and `Solve()` doesn't find a solution. Then you can try the native SMath solver `solve()` or `roots()` or the function `FindRoot()` from the *Nonlinear Solvers* Plugin.

`Solve(eqn,vars)`

`eqn` equation or list or vector of equations. If the expressions don't contain a Boolean equal sign, then implicitly = 0 is added.

`vars` variable or list or vector of variables to solve for.

The solution comes as an equation or a list of equations `name=value`. If there is more then one solution, these are given as Matrix of equations or lists of equations.

The solution can be converted into assignments using the function `Assign()`. In case of multiple solutions you need to decide, which one to use for assignment. Select the solution by using the appropriate element index with the solution matrix.

Note: Always use symbolic – or even better, no – optimization/evaluation for results display. The equations contain undefined variables and those can't be displayed with numeric optimization.

# 7.1 Scalar Equations

## Single Solution

Single equations and variables can be specified as scalar expressions without using lists or matrices.

$$\texttt{Solve}\left(a\cdot x+b=y;\ a\right)=a=\frac{y-b}{x}$$

In the case of a single solution the result is a single equation without any wrapping structure. To actually assign the solution to the variable, use `Assign()`:

$$L:=\texttt{Solve}\left(a\cdot x+b=y;\ a\right)\qquad L=a=\frac{y-b}{x}$$

$$\texttt{Assign}\left(L\right)=\frac{y-b}{x}\qquad a=\frac{y-b}{x}$$

Alternatively, you can use the substitution function `at()` from the *Custom Functions* plugin.

$$\texttt{Clear}\left(a\right)=1$$

$$a:=a\Big|_{L}$$  a:at Tab a → L

$$a=a=\frac{y-b}{x}$$

## Multiple Solutions

Multiple solutions for a single variable are displayed as a list of equations. You can select one of them by using an element index.

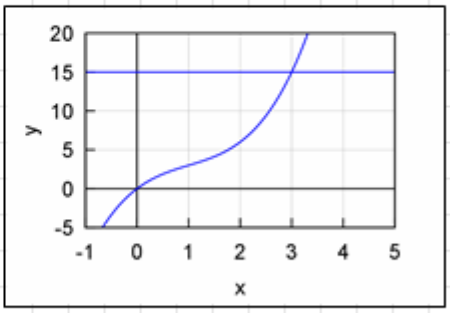
$$\texttt{Clear}\left(x;\ y\right)=1$$

$$f\left(x\right):=x^{3}-3\cdot x^{2}+5\cdot x$$

$$L:=\texttt{Solve}\left(f\left(x\right)=15;\ x\right)$$

$$L=\begin{cases}x=-i\cdot\sqrt{5}\\ x=i\cdot\sqrt{5}\\ x=3\end{cases}$$

$$p:=\begin{cases}\texttt{explicit}\left(f\left(x\right);\ x;\ -1;\ 5\right)\\ \texttt{explicit}\left(15;\ x;\ -1;\ 5\right)\\ \texttt{yrange}=\begin{cases}-5\\ 20\end{cases}\end{cases}$$




$$\texttt{Draw2D}\left(p;\ \begin{cases}220\\ 150\end{cases}\right)$$

While solving and selecting solutions, the variables must be undefined. Delete eventual values using `Clear()`.

| | | |
|---|---|---|
| $\text{Assign}\left(L_1\right) = -i \cdot \sqrt{5}$ | $x = -2{,}24 \cdot i$ | $\text{Clear}\left(x\right) = 1$ |
| $\text{Assign}\left(L_2\right) = i \cdot \sqrt{5}$ | $x = 2{,}24 \cdot i$ | $\text{Clear}\left(x\right) = 1$ |
| $\text{Assign}\left(L_3\right) = 3$ | $x = 3$ | $\text{Clear}\left(x\right) = 1$ |

# 7.2 Systems of Equations

## Single Solution

Systems of equations are given as vectors or lists. Also the variables are given as vectors or lists. In the case of a single solution, it is returned as a list of equations, one for each variable.

$$\texttt{Clear}(a)=1$$

$$L:=\texttt{Solve}\left(\begin{bmatrix}3\cdot x+4\cdot y=7\\ 2\cdot x+a\cdot y=13\end{bmatrix};\begin{bmatrix}x\\ y\end{bmatrix}\right)\qquad L=\begin{cases}x=\dfrac{-52+7\cdot a}{-8+3\cdot a}\\ y=\dfrac{25}{-8+3\cdot a}\end{cases}$$

`Assign()` converts all equations in the argument (no matter what structure that is) to assignments.

$$\texttt{Assign}(L)=\begin{cases}\dfrac{-52+7\cdot a}{-8+3\cdot a}\\ \dfrac{25}{-8+3\cdot a}\end{cases}\qquad x=\frac{-52+7\cdot a}{-8+3\cdot a}\qquad y=\frac{25}{-8+3\cdot a}$$

The variant with `at()` does not work out of the box, because L doesn't actually store the result but rather the command to create it.

`at()` expects equations as specification for substitution.

$$\texttt{Clear}(x;y)=1$$

$$\begin{bmatrix}x\\ y\end{bmatrix}:=\left.\begin{bmatrix}x\\ y\end{bmatrix}\right|_{L}\qquad\begin{bmatrix}x\\ y\end{bmatrix}=\begin{bmatrix}x\\ y\end{bmatrix}$$

$$\texttt{lastError}=\texttt{"Cannot parse.}\\ \texttt{[2] unknowns: x, y}\\ \texttt{[1] values: Solve(mat(3*x+4*y=7,2*x+a*y=13,2,1),mat(x,y,2,1))."}$$

If the expression is converted to displayed form (this is what the function `simp()` does), then `at()` can handle the assignment.

$$\texttt{simp}(a\#):=\left|\texttt{str2num}(\texttt{num2str}(a\#))\right.$$

$$\begin{bmatrix}x\\ y\end{bmatrix}:=\left.\begin{bmatrix}x\\ y\end{bmatrix}\right|_{\texttt{simp}(L)}\qquad\begin{bmatrix}x\\ y\end{bmatrix}=\begin{bmatrix}\dfrac{-52+7\cdot a}{-8+3\cdot a}\\ \dfrac{25}{-8+3\cdot a}\end{bmatrix}$$

## Multiple Solutions

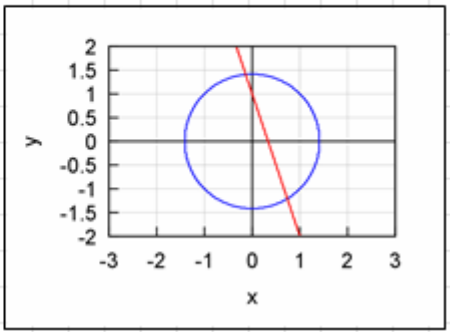
Here we show how to build the system of equations one by one. This can be useful if you want to explain the meaning of these equations separately.

Note that assigning an equation to a variable requires brackets around the equation. The assignment operator := has higher priority than the Boolean equal.

The plot shows that there are two solutions:

$$Gl_1 := \left(x^2 + y^2 = 2\right) \qquad Gl_2 := (3 \cdot x + y = 1)$$

$$p := \begin{cases} \text{implicit}\left(Gl_1; x; -3; 3; y; -2; 2\right) \\ color = red \\ \text{implicit}\left(Gl_2; x; -3; 3; y; -2; 2\right) \\ proportional\_axes = xy \end{cases}$$




$$Gl_1 := \left(x^2 + y^2 = 2\right) \qquad Gl_2 := (3 \cdot x + y = 1)$$

$$L := \text{Solve}\left(Gl; \begin{bmatrix} x \\ y \end{bmatrix}\right) \qquad L = \left[\begin{cases} x = -\frac{-3+\sqrt{19}}{10} \\ y = \frac{1+3\cdot\sqrt{19}}{10} \end{cases} \quad \begin{cases} x = \frac{3+\sqrt{19}}{10} \\ y = -\frac{-1+3\cdot\sqrt{19}}{10} \end{cases}\right]$$

Multiple solutions with multiple variables are returned as a row vector of lists. The solutions can be accessed via an element index.

$$\text{Assign}\left(L_1\right) = \begin{cases} -\frac{-3+\sqrt{19}}{10} \\ \frac{1+3\cdot\sqrt{19}}{10} \end{cases} \qquad x = -\frac{-3+\sqrt{19}}{10} \qquad y = \frac{1+3\cdot\sqrt{19}}{10}$$

$$x = -0{,}136 \qquad y = 1{,}408 \qquad \text{Clear}(x; y) = 1$$

$$\text{Assign}\left(L_2\right) = \begin{cases} \frac{3+\sqrt{19}}{10} \\ -\frac{-1+3\cdot\sqrt{19}}{10} \end{cases} \qquad x = \frac{3+\sqrt{19}}{10} \qquad y = -\frac{-1+3\cdot\sqrt{19}}{10}$$

$$x = 0{,}736 \qquad y = -1{,}208 \qquad \text{Clear}(x; y) = 1$$

# 8 Ordinary Differential Equations

## Contents



*ODE2.sm*

Maxima has many features for ordinary differential equations (ODE). Some of them are:

- The function `ODE.2()` for symbolic solution of ODE up to second order.
- The plot functions for direction fields in the Maxima package *drawdf*.

## 8.1 Function `ODE.2()`

The function `ODE.2()` solves ordinary differential equations up to second order.

`ODE.2(ode;f(t),t)`

`ode` differential equation

`f(t)` unknown function

`t` independent variable

The result is a Boolean equation which can be used by `Assign()` for assignment. Depending on the order of the ODE, the solution contains one or two integration constants, named `c` (first order) or `k1` and `k2` (second order).

This is demonstrated for the initial value problem

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}w(t) + \omega^2 w(t) = 0, \quad w(0) = 1, \quad \frac{\mathrm{d}}{\mathrm{d}t}w(t)\bigg|_{t=0} = 0$$

First, the general solution is obtained:

$$L := ODE_2\left(\frac{\mathrm{d}^2}{\mathrm{d}t^2}w(t) + \omega^2 \cdot w(t) = 0;\ w(t);\ t\right)$$

$$L = \begin{cases} \text{"}\omega \text{ is assumed to be positive."} \\ w(t) = k1 \cdot \sin(t \cdot \omega) + k2 \cdot \cos(t \cdot \omega) \end{cases}$$

The message "$\omega$ is assumed to be positive" means that Maxima internally asked for the sign of $\omega$. The plugin captures such questions and always responds with $+$. The message is integrated in the result for transparency. This way the user can see that the assumption was made.

The user can store such assumptions in advance and thus avoid questions by Maxima.

$$\mathrm{M}(assume(\omega > 0)) = \{\omega > 0$$

$$L := ODE_2\left(\frac{\mathrm{d}^2}{\mathrm{d}t^2}w(t) + \omega^2 \cdot w(t) = 0;\ w(t);\ t\right)$$

$$L = w(t) = k1 \cdot \sin(t \cdot \omega) + k2 \cdot \cos(t \cdot \omega)$$

The return value is a Boolean equation. Using `Assign()` this equation can be converted into a definition of the unknown function. The solution contains one or two constants of integration depending on the order of the ODE. Their name is `c` (first order) or `k1` and `k2` (second order).

$$\mathtt{Assign}(L) = k1 \cdot \sin(t \cdot \omega) + k2 \cdot \cos(t \cdot \omega)$$

$$w(t) = k1 \cdot \sin(t \cdot \omega) + k2 \cdot \cos(t \cdot \omega)$$

The constants are determined from the initial conditions $w(0) = 1$ and $w'(0) = 0$. Note that you can't write the latter as $\mathrm{d}w(0)/\mathrm{d}t$ because $w(0)$ isn't a function of $t$ any more. Thus the derivative yields zero. Therefore, the derivative has to be computed first and then the value $t = 0$ has to be substituted using `at()`.

$$w(0) = k2 \qquad \frac{d}{dt}(w(0)) = 0$$

$$\frac{d}{dt}(w(t)) = \omega \cdot (\cos(t \cdot \omega) \cdot k1 - \sin(t \cdot \omega) \cdot k2)$$

$$\left.\frac{d}{dt}(w(t))\right|_{t=0} = \omega \cdot k1$$

Solution and assignment are written in a single formula.

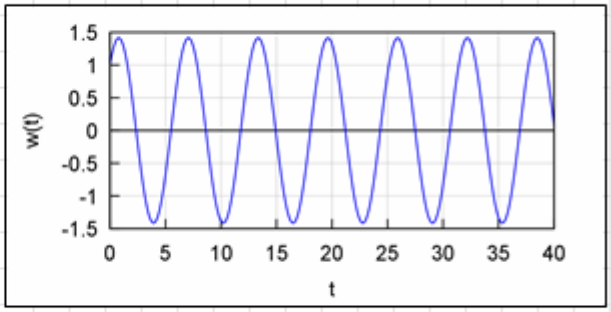
$$\text{Assign}\left(\text{Solve}\left(\left\{\begin{matrix} w(0) = 1 \\ \left.\frac{d}{dt}(w(t))\right|_{t=0} = 1 \end{matrix}\right\}; \begin{Bmatrix} k1 \\ k2 \end{Bmatrix}\right) = \begin{Bmatrix} \frac{1}{\omega} \\ 1 \end{Bmatrix}\right)$$

$$w(t) = \frac{1}{\omega} \cdot \sin(t \cdot \omega) + 1 \cdot \cos(t \cdot \omega)$$

In order to plot the solution, $\omega$ must be given a value.

$$\omega := 1$$

$$P := \begin{cases} \text{explicit}(w(t); t; 0; 40) \\ \text{xlabel} = \text{"t"} \\ \text{ylabel} = \text{"w(t)"} \\ \text{yrange} = \begin{Bmatrix} -1{,}5 \\ 1{,}5 \end{Bmatrix} \end{cases}$$




$$\text{Draw2D}\left(P; \begin{Bmatrix} 300 \\ 150 \end{Bmatrix}\right)$$

# 9 Advanced Topics

## Contents

## 9.1 Custom Definitions and Translations

Custom definitions and translations are stored in the config file *maxima.xml*, which resides in the top level maxima plugin directory. If the Development tools plugin is installed, then you can open the plugin directory via **Insert> Development Tools> Plugin's folder> MaximaPlugin.**

Note that such configurations are local to your SMath installation. Documents which make use of such features aren't portable, i.e. they won't work on other computers. Yet it enables you to experiment with translations without access to the source code.

### Example

Let's assume we want to add handling of the SMath function `round(x,n)` which is not available in Maxima. In Maxima, we have just `round(x)`, which rounds to zero decimal places.

First, we have to define a helper function in Maxima, to which we later translate `round(x,n)`:

```
round2(x,n):=round(x*10^n)/10^n$
```

We add this definition to *maxima.xml*:

```
<CustomLoadCommands>
    <Command>round2(x,n):=round(x*10^n)/10^n$</Command>
</CustomLoadCommands>
```

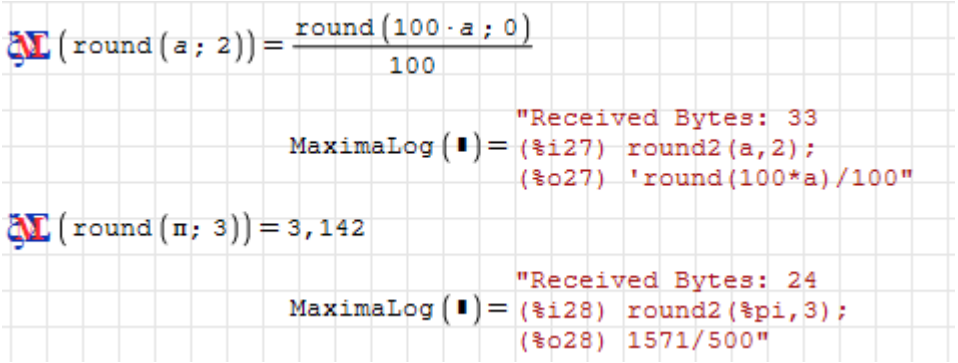
Second, we need to add translation rules to map `round(x,n)` to `round2(x,n)`:

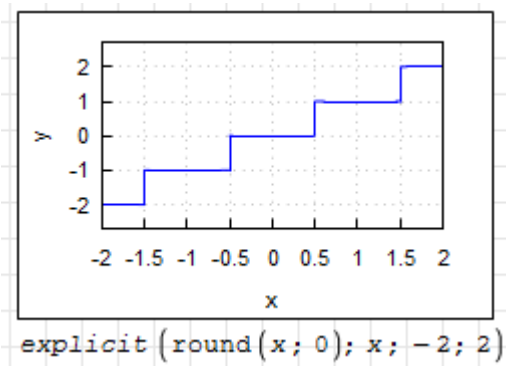
```
<ConvertFromSMathToMaxima>
    <Expression regex="round\(" replace="round2("></Expression>
</ConvertFromSMathToMaxima>
```

As it may happen that a symbolic expression returned from Maxima contains `round(x)`, we also want to add a translation into the SMath equivalent `round(x,0)`. This is a bit more involved, as we have to isolate the argument of `round()`, in order to add a second one:

```
<ConvertFromMaximaToSMath>
    <Expression regex="round\((.+)\)" replace="round($1,0)"></Expression>
</ConvertFromMaximaToSMath>
```

Note that by the time you are reading this, the above translation has been implemented in the plugin.






The plot above is generated using a Maxima Draw2D() plot region.

# Bibliography

M. Kraska. *SMath Studio mit Maxima. Einführung und Referenz für Version 0.99.7561*. Technische Hochschule Brandenburg. 2020. DOI: 10.25933/opus4-2946.