Introduction to Maxima for Economics

Josef Leydold Martin Petry

September 26, 2011

Institute for Statistics and Mathematics, WU Wien

The following verions of the *Maxima* CAS has been used when writing this manual:

Maxima	5.25.1
wxMaxima	11.04.0

Contents

1 Introduction

Maxima is a *computer algebra system* (CAS) like Maple® and Mathematica®. Thus it is a powerful tool for the manipulation of symbolic and numerical expressions, including differentiation, integration, vectors, matrices, … and so on. It also provides commands for plotting functions, curves and data in two and three dimensions. In opposition to other CAS *Maxima* has a quite long history and is open source software. From the *Maxima* manual:

> *Maxima* is a descendant of Macsyma, the legendary computer algebra system developed in the late 1960s at the MIT. It is the only system based on that effort still publicly available and with an active user community. Macsyma was revolutionary in its day, and many other systems were inspired by it. In 1998, Professor W. F. Schelter of the University of Texas obtained permission form the Department of Energy to release the Macsyma source code under the GNU Public License, and in 2000 he initiated the Maxima project at SourceForge to maintain and develop Macsyma, now called *Maxima*.

Maxima is available on many operating systems including MS Windows®, Mac OSX and Linux. The software can be downloaded from

`http://maxima.sourceforge.net/`.

There one can find links to more information as well as to related software.

Maxima also allows to define your own commands. Thus it is easy to *Maxima*'s capabilities and adjust it to own's need. Thus you find contributed "packages" that have been developed and made publicly available by users of this CAS.

The obvious advantage of open source software (OSS) is that it can be downloaded and used for free. Nevertheless, there are also other gains for users: It allows to see how other people solved a similar problem (and you are free to copy and modify their solutions). In case of unexpected results one can investigate the sources and thus there is some chance to detect a possible error[1].

1.1 Maxima Interfaces

Maxima at its heart has a command line interface and by itself it is not capable of displaying formatted mathematics beyond the plain text level. For most users that has been grown up with graphical user interfaces (GUI) this is unfamiliar and may seem quite arcane. Fortunately, nowadays more fancy GUIs are available.

We will use the most popular one called *wxMaxima*. It is available at

`http://wxmaxima.sourceforge.net/`.

Its interface consists of a notebook with menus and a toolbar which allows an inexperienced user to pick an appropriate command. These commands as well as their outputs are printed into the notebook. On the other hand these commands can also directly be entered and edited and thus provide a powerful workspace for the experienced user. The notebook can then be saved at the

[1] The core functionality of Maxima is written in LISP.

end of a session and reloaded for later use. Thus it allows the user to construct a document that consists of text, calculations, and plots.

In this tutorial we simulate the output as it is given by *wxMaxima*. Moreover, we will explain some features of this interface. However, we mainly describe the commandline interface of *Maxima*. Exploring the toolbar is left to the reader.

Alternatives GUIs are *Xmaxima* which is shipped with *Maxima*. It is less fancy but more stable than *wxMaxima* that is currently being actively developed. A nice interface is also provided by *Texmacs*, a WYSIWYG word processing system that allows to include *Maxima* code and prints its output. Last but not least, an really excellent *Emacs* mode has been written for *Maxima*.

1.2 How to Read this Tutorial

The sections in each chapter first shows a table of (new) *Maxima* commands.

Command	*Description*
`cmd`	a command

In addition we may need to change some special variables that control the behavior of *Maxima*'s output or evaluations. These are listed in the following way.

Variable	*Default*	*Description*
`var`	**`true`**	a special variable

Examples of *Maxima* code are printed in a yellow box. The output mimics the output of the *wx-Maxima* program.

```
(%i1) input;
(%o1) output
```

Maxima code is running text is printed using **`courier`**.

Keyboard buttons are shown as $\boxed{\text{KEY}}$. Notice that $\boxed{\text{SHIFT}}$+$\boxed{\text{ENTER}}$ means that one has first press and hold the $\boxed{\text{SHIFT}}$ button and then press the $\boxed{\text{ENTER}}$ button.

Important
Caveats and possible errors are marked by an exclamation mark inside a triangle.

Each Section closes with a small set of exercises.

The tutorial organized in the following way. Chapter 2 presents the basic commands and concept that are necessary to run the *Maxima* system. Here we mainly focus on numerical calculations but we also point to possible errors. In Chapter 3 we discuss the principles of symbolic computations. The remaining chapters then deal with special topics, like Plotting (Chap. 4), Linear Algebra (Chap. 6), and Calculus (Chap. 7).

The topics in this manual are strongly influenced by typical applications in economics. Thus we present some relevant examples in Chapter 9.

Important
This manual gives insight into a small subset of the *Maxima* system. In addition, we tried to make the presentation as simple as possible and thus we did not explain all the details. Thus the reader is encouraged to read the online documentation (see Sect. 2.9) for a comprehensive description of the presented commands.

2 Basics

Maxima is a computer algebra system that accepts some input, evaluates the given commands and returns the results. As with all CAS we have to be familiar with the syntactical rules of this system.

2.1 Getting Started

Symbol	*Description*
`%i1`	Input (input prompt)
`%o1`	Output of evaluation (output prompt)
`;`	End-of-input marker
`$`	End-of-input marker. Output will be suppressed

Start *Maxima* or *wxMaxima*. Then you can enter your commands that must be terminated by a semicolon `;` .Observe that your *input* will be automatically prefixed by `%i1`. The given string is then evaluated by *Maxima* according to rules that are basically based on pattern matching. The result is then printed and the *output* is prefixed by `%o1`. It can be used to refer to the result in further computations, see Sect. 2.8 below.

A command may also be terminated by the special symbol `$` instead of a semicolon. Then the output of your result is suppressed, i.e., *Maxima* evaluates your input expression but does not show its results. This can be quite convenient when an intermediate result is very long and fills several pages. Here is a simple example.

```
(%i1) 2+3;
(%o1) 5
(%i2) 4*5$
```

As you can observe your input and output with be enumerated. It is also possible to put more than one command on one line. On the other hand an command can also be spread over two or more lines (thus the command has to be terminated by means of `;` or `$`).

```
(%i3) 2+3; 4*5;
(%o3) 5
(%o4) 20
(%i5) 2+3*(4+5)
      +6*7-8/4;
(%o5) 69
```

2.2 wxMaxima

wxMaxima provides a more convenient interface. It shows a *notebook* where one can insert an expression. This notebook is organized using co called *cells*. Notice that the bar on the left hand side indicates which input and output cells belong together. Whenever, a cell is currently edited or evaluated the color of this bar changes. (Some evaluations can be very time consuming.)

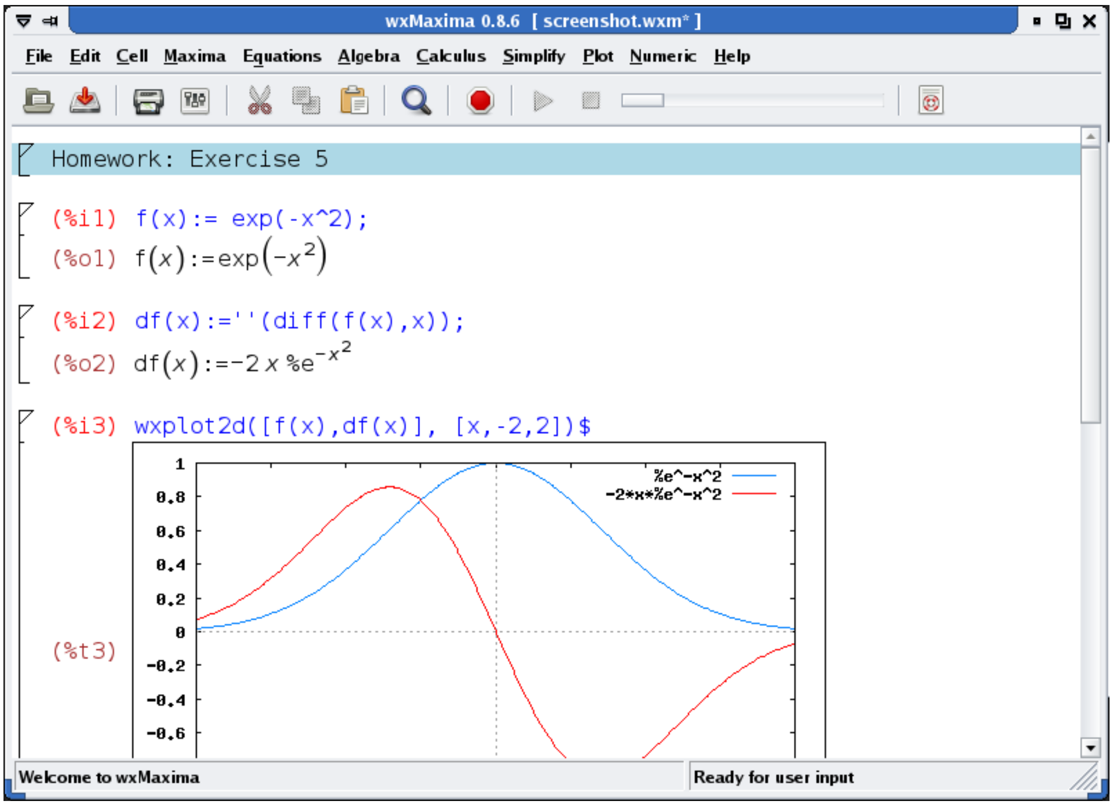
Important
One has to press SHIFT+ENTER to start the evaluation of your input. A missing ; character is then automatically inserted. Notice that pressing ENTER alone (without pressing SHIFT) only inserts a line break[1] even when a ; or **$** is present.

wxMaxima also provides a toolbar where one can select commands from menus. This is very convenient for the beginner as it disburdens one from memorizing the command syntax. (But it is cumbersome for the more experienced user.)

Using the notebook one can insert his or her commands almost everywhere on the page. Even further command can be edited and reevaluated. (Of course the number of the cell is then updated.) Thus the cells are not necessary in chronological order.

In addition one can insert sections headers and text to comment one's calculations. For this purpose use the corresponding entries in menu Cell.

When a *Maxima* session is finished one can save a notebook (using File->Save) and reload it again (using File->Open). Here is a typical *wxMaxima* notebook.



wxMaxima 0.8.6 [screenshot.wxm*]
File Edit Cell Maxima Equations Algebra Calculus Simplify Plot Numeric Help
%e^-x^2
-2*x*%e^-x^2
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
Welcome to wxMaxima
Ready for user input

```
Homework: Exercise 5

(%i1) f(x):= exp(-x^2);
(%o1) f(x):=exp(-x^2)

(%i2) df(x):=''(diff(f(x),x));
(%o2) df(x):=-2 x %e^(-x^2)

(%i3) wxplot2d([f(x),df(x)], [x,-2,2])$
(%t3)
```

[1] However, this default behavior of *wxMaxima* can be changed: tick the box **Enter evaluates cells** in the ***wxMaxima* configuration** window.

2.3 Maxima as a Calculator

Almost any mathematical expression is available in *Maxima*. The operators are the same as on your calculator, see Sections 2.4 and 2.6.

2.4 Basic Arithmetic Operators

Operator	*Description*
`+`	Addition
`-`	Subtraction
`/`	Division
`*`	Multiplication
`^`	Raise to power

Some examples are:

```
(%i1) 5+3;
(%o1) 8
(%i2) 22-7;
(%o2) 15
(%i3) 121/11;
(%o3) 11
(%i4) 25*4;
(%o4) 100
(%i5) 2^8;
(%o5) 256
(%i6) (2+3)^4-(5+6+7)/(2+4);
(%o6) 622
```

Maxima can compute very large numbers as the following example demonstrates.

```
(%i7) 2^100;
(%o7) 1267650600228229401496703205376
```

2.5 Constants

Constant	*Description*
`%e`	Euler's number $e = 2.71828\ldots$
`%pi`	$\pi = 3.14159\ldots$
`%i`	Imaginary unit $i = \sqrt{-1}$
`inf`	Positive infinity ∞
`minf`	Minus infinity $-\infty$

Maxima knows important numerical constants like e and π as well as $\pm\infty$.

2.6 Functions

Function	*Description*
`abs(x)`	Absolute value of **`x`**
`sqrt(x)`	Square root of **`x`**
`log(x)`	Natural logarithm (i.e, to base e) of **`x`**
`exp(x)`	Exponential function of **`x`**
`sin(x)`	Sine of **`x`**
`cos(x)`	Cosine of **`x`**
`tan(x)`	Tangent of **`x`**

As any calculator *Maxima* provides commands for computing functions.

```
(%i1) cos(%pi);
(%o1) −1
(%i2) exp(2);
(%o2) %e^2
(%i3) abs(-5);
(%o3) 5
```

Important
Notice that the arguments to **`sin`** and **`cos`** must be angles given in radians (i.e., where a right angle is $\frac{\pi}{2}$). If you need to do computations using degrees (i.e., where a right angle is 90°), then you have to convert degree d into radian x using $x = d\frac{\pi}{180}$.

The exponential function for an arbitrary base a can computed by means of the the **`^`** operator. Similarly the n-th root can be computed using **`^(1/n)`**. Notice that **`%e^x`** is an alternative for **`exp(x)`**.

```
(%i4) 10^4;
(%o4) 1000
(%i5) 1000^(1/3);
(%o5) 10
(%i6) 8^(2/3);
(%o6) 4
```

Maxima only provides the natural logarithm function (with base e). The common logarithm (base 10) can be computed using

$$\log_{10}(x) = \frac{\log(x)}{\log(10)}\,.$$

Important
Do not use **`ln(x)`** to compute the natural logarithm. *Maxima* does not know this function and thus it returns it unevaluated.

```
(%i7) log(%e);
(%o7) 1
(%i8) ln(%e);
(%o8) ln(%e)
```

2.7 Variables and User Defined Functions

operator	*Description*
`:`	Assignment operator
`:=`	Function definition operator
`kill(x)`	Removes all bindings (value, function) from **x**
`kill(all)`	Removes all bindings from all items

Variable	*Default*	*Description*
values	`[]`	List of user-defined variables
functions	`[]`	List of user-defined functions

Expressions like numbers can be stored in variables, i.e., a sequence of alphanumeric characters (that start with a letter). The alphanumeric characters are **A** through **Z**, **a** through **z**, **0** through **9**, **%**, and _. Examples of valid variable names are: **a**, **beta**, or **y_1**. Thus one can store the result of any evaluation by means of the assignment operator **:**. This operator evaluates its right-hand side and associates that *value* with the left-hand side. When the variable is evaluated in further computations, then it is replaced by its value. Here is a simple example.

```
(%i1) x;
(%o1) x
(%i2) x: 2+3;
(%o2) 5
(%i3) x;
(%o3) 5
(%i4) y: 2*x^2;
(%o4) 50
```

Important
Maxima is case-sensitive, that is, the identifiers **foo**, **FOO**, and **Foo** are distinct.

It also possible to extend *Maxima* with one's own functions by means of the function definition operator **:=**. Function names are similar to variable names but are followed by parenthesis **(...)** that contain a comma separated list of its arguments (which themselves are variables). The right-hand side of the function assignment operator **:=** (i.e., the function body) is never evaluated.

In the following example we define functions **sqr** for computing the square of a given expression and **sindeg** for the sine function where the angle is given in degrees. Notice the function body contains expression **x** which is note evaluated. Otherwise it would be replaced by value **5** which has been assigned in the previous example.

```
(%i5) sqr(x):= x^2;
(%o5) sqr(x) := x^2
(%i6) sqr(4);
(%o6) 16
(%i7) sindeg(x):= sin(x*%pi/180);
(%o7) sindeg(x) := sin(xπ/180)
(%i8) sindeg(45);
(%o8) 1/√2
```

$$sqr(x) := x^2 \qquad sindeg(x) := sin\left(\frac{x\pi}{180}\right) \qquad \frac{1}{\sqrt{2}}$$

Of course it is possible to define functions in two or more variables.

```
 (%i9) norm(x,y):= sqrt(x^2+y^2);
```
(%o9) $norm(x,y) := \sqrt{x^2+y^2}$
```
(%i10) norm(3,4);
(%o10) 5
```

The *system variables* **`values`** and **`functions`** contain a list of user defined variables and functions, resp.

```
(%i11) values;
```
(%o11) $[x,y]$
```
(%i12) functions;
```
(%o12) $[sindeg(x), sqr(x), norm(x,y)]$

Notice that both variables and functions remain persistent until the *Maxima* session is closed. Sometimes it is convenient to remove these again. This can be accomplished by function **`kill`**.

```
(%i13) kill(x);
```
(%o13) *done*
```
(%i14) values;
```
(%o14) $[y]$
```
(%i15) functions;
```
(%o15) $[sindeg(x), sqr(x), norm(x,y)]$
```
(%i16) kill(all);
```
(%o16) *done*
```
(%i17) values;
(%o17) [ ]
(%i18) functions;
(%o18) [ ]
```

Important
You **must not** use symbol **=** to define variables and functions. This symbol is used for defining equations, see Sect. 2.15.

```
(%i19) x=3;
```
(%o19) $x = 3$
```
(%i20) x;
```
(%o20) x

2.8 Use Output for Further Computations

Command	*Description*
`%`	Result of last evaluation
`%o`*i*	Result of *i*-th evaluation (content of output cell *i*)

The operator `%` refers to the output expression most recently computed my *Maxima*, whether or not it was displayed. In addition the result of the *i*-th computation is available by `%o`*i*.

```
(%i1) 2+3;
(%o1) 5
(%i2) % * 20;
(%o2) 100
(%i3) % - 10;
(%o3) 90
(%i4) %o1 - 10;
(%o4) −5
```

Important
Notice that `%` refers to the result of the *last* evaluation. As you can enter a command almost everywhere in a *wxMaxima* notebook this is **not** necessarily the content of the output cell just above your current input cell.

2.9 Help

Command	*Description*
`? cmd`	Show help page for command **`cmd`** (short for **`describe`**)
`?? cmd`	Search for inexact matches of **`cmd`** in help pages
`apropos("cmd")`	List of all *Maxima* names that contain substring **`cmd`**

Help pages for all commands and operators are available using the special symbols **`? cmd`** and **`?? cmd`**. The double question mark **`??`** can be used to search for string **`cmd`** in the manual.

```
(%i1) ? sqrt;
-- Function: sqrt (<x>)
    The square root of <x>. It is represented internally by
    `<x>^(1/2)'.  See also `rootscontract'.
    `radexpand' if `true' will cause nth roots of factors of a product
    which are powers of n to be pulled outside of the radical, e.g.
    `sqrt(16*x^2)' will become `4*x' only if `radexpand' is `true'.
There are also some inexact matches for `sqrt'.
Try `?? sqrt' to see them.
```

Important
It is important to insert a space between **`?`** and **`cmd`**. Otherwise the input is interpreted as an internal LISP variable which is simply returned. It also might result in a weird error message from the LISP interpreter.

```
(%i2) ?sqrt;
(%o2) sqrt
```

If you only remember a substring of a command name, then **apropos** is quite useful. It returns a list of all those *Maxima* names that contain this string.

```
(%i3) apropos("sqrt");
(%o3) [isqrt, sqrt, sqrtdispflag]
```

For an alternative method to access the manual within *wxMaxima* press the $\boxed{\text{F1}}$ button or use the Help button in the menu bar. It gives access to the whole library including an index and a search function.

2.10 System Variables

Command	*Description*
`ev(x,envir)`	Evaluate expression `x` with in given environment `envir`
`reset()`	Resets many (global) system variables

Maxima uses a set of *system variables* to control the behavior of the system. For example, in Sect. 2.11 below variable **fpprintprec** is used to control the printing of floating point numbers, and **numer** controls whether mathematical functions are evaluated in floating point or not.

One may use assign operator **:** to change value of system variables globally. Command **reset()** allows to reset many global system variables and some other variables, to their default values.

An alternative approach to changing and resetting system variables is the use of command **ev** which allows to evaluate an expression with locally changed system variables. An example with variable **numer** is given in Sect. 2.12 below.

2.11 Numbers

Command	*Description*
`123`	integer 123
`123.`	integer 123
`123.0`	floating point number 123
`123e0`	floating point number 123
`123b0`	big float number 123

Variable	*Default*	*Description*
`fpprec`	`16`	Number of significant digits for arithmetic of bigfloat numbers
`fpprintprec`	`0`	Number of digits when printing a float (default: all significant digits)

Maxima distinguishes between four different types of numbers:

- *integers*: $\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots$
- *rational numbers*: ratio of two integers
- *floating point numbers*: consist of a *mantissa* of (approximately) 16 decimal digits and an exponent to base 10, e.g., $1.234567890123456 \cdot 10^5$. In common speech these are called *decimal numbers* and usually written without the exponent; in our example 123456.7890123456.
- *arbitrary precision floating point numbers* (called *bigfloat numbers*: floating point numbers where the size of the mantissa can be set to some fixed but arbitrary number (using the system variable **fpprec**.

In addition, *Maxima* can handle some special constants, see Sect. 2.5.

Floating point numbers can be entered either as a decimal number with at least one digit after the decimal point, e.g., **`123.0`**, or using the scientific notation, e.g., **`123e0`**.

Integers and rational numbers are stored without loss of precision while this is not possible for floating point numbers. They can be seen as an approximation to real numbers. Notice that the decimal expression of real numbers (as elements from $\mathbb{R}$) may have an infinite number of digits as in $\frac{1}{7} = 0.14285714285714 28\ldots$ or $\sqrt{2} = 1.414213562373095\ldots$. When stored as floating point numbers only a limited number of digits can be stored and one looses precision. Additions and subtractions of floating point numbers then may results in further loss of precision due to cancellation errors. The following example demonstrates the difference between rational numbers and floating point numbers.

```
(%i1) 2/10 * 11 - 2 - 4/20;
(%o1) 0
(%i2) 2.0/10.0 * 11.0 - 2.0 - 4.0/20.0;
(%o2) 1.665334536937735 10^-16
```

Important
In opposition to decimal representation modern computer use a binary representation of numbers by means of the *binary digits* **`0`** and **`1`** called *bits*. As a consequence numbers like 0.2 cannot be stored exactly due an infinite binary representation.

Maxima tries to do all its evaluation as exact as possible. Consequently, *Maxima* reduces rational numbers or simplifies numerical expression where possible but does not convert to floating point numbers unless forced to do so (see Sect. 2.12 below). In particular *Maxima* also returns special numbers as results of computations.

```
(%i3) 19 / 3;
(%o3) 19/3
(%i4) 2^1000;
(%o4) 10715086071862673209484250490[242 digits]429831652624386837205668069376
(%i5) sqrt(2);
(%o5) √2
(%i6) 14 / 6;
(%o6) 7/3
(%i7) sqrt(12);
(%o7) 3√2
(%i8) sqrt(8);
(%o8) 2^(3/2)
(%i9) exp(3);
(%o9) %e^3
(%i10) atan(1);
(%o10) π/4
(%i11) tan(%pi/4);
(%o11) 1
```

When we use floating point numbers instead, we get a less precise result, i.e., stored as floating point numbers. As the first line of the following example shows, it is often sufficient to insert just one floating point number in order to obtain a floating point answer. As the last line shows, the answer can sometimes a bit weird.

```
(%i12) 19.0 / 3;
(%o12) 6.333333333333333
(%i13) 2.0^1000;
(%o13) 1.07150860718627 10^301
(%i14) sqrt(2.0);
(%o14) 1.414213562373095
(%i15) sqrt(12.0);
(%o15) 3.464101615137754
(%i16) exp(3.0);
(%o16) 20.08553692318767
(%i17) atan(1.0);
(%o17) .7853981633974483
(%i18) tan(%pi/4.0);
(%o18) tan(0.25 * %pi)
```

⚠ We again stress that the difference between integers and floating point numbers is important. In the inputs in the following two lines of code are different and thus are handled differently!

```
(%i19) 123;
(%o19) 123
(%i20) 123.0;
(%o20) 123.0
```

Sometimes it can be annoying when 16 digits of floating point numbers are printed. This can be controlled by setting system variable **`fpprintprec`**.

```
(%i21) fpprintprec: 4$
(%i22) sqrt(2.0);
(%o22) 1.141
(%i23) fpprintprec: 0$
(%i24) sqrt(2.0);
(%o24) 1.414213562373095
```

2.12 Numeric Results

Command	*Description*
`float(x)`	Convert **`x`** to floating point number
`bfloat(x)`	Convert **`x`** to big float
`set_display(ascii)`	Set display mode to plain ASCII
`set_display(xml)`	Set display mode to XML (default)

Variable	*Default*	*Description*
`numer`	**`false`**	if true, then mathematical functions are evaluated in floating point

When we are interested in numeric results we can set the system variable **`numer`** to **`true`**. Then all evaluations are performed in floating point numbers. This can also be done *locally* using the command **`ev`**.

```
(%i1) ev(19/3, numer:true);
(%o1) 6.333333333333333
```

In many situations we can simplify the notation. **`numer`** is interpreted as **`numer:true`**. Thus **`ev`** may be omitted, too. The following works as well.

```
(%i2) 19/3, numer;
(%o2) 6.333333333333333
(%i3) sin(4), numer;
(%o3) −0.75680249530793
(%i4) %pi, numer;
(%o4) 3.141592653589793
```

An alternative approach is to use the **`float`** command.

```
(%i5) float(19/3);
(%o5) 6.333333333333333
(%i6) float(sin(4));
(%o6) −0.75680249530793
(%i7) float(%pi);
(%o7) 3.141592653589793
```

wxMaxima tries to print *Maxima*'s output in a nice manner. Number are printed into one line. Suppose you need all digits of 100! or the first 500 digits of π. Then not all digits are displayed which may not be what you want.

```
 (%i8) 100!;
 (%o8) 933262154439441526816992388562[98 digits]916864000000000000000000000000
 (%i9) fpprec: 500$
(%i10) bfloat(%pi);
(%o10) 3.141592653589793238462643383[443 digits]8857527248912279381830119491b0
(%i11) reset()$
```

However, it is possible to switch back to *Maxima*'s native output format using command **`set_display(ascii)`**. Then all digits are printed albeit in a not so nice form. The backslash sign **`\`** at the end of each line indicates that it is continued on the next. Do not forget to reset the display format again by means of **`set_display(xml)`**;

```
(%i12) set_display(ascii)$
(%i13) 100!;
(%o13) 93326215443944152681699238856266700490715968264381621468592963895217599\
 99322991560894146397615651828625369792082722375825118521091686400000000000000000\
 000000000
(%i14) fpprec: 500$
(%i15) bfloat(%pi);
(%o15) 3.141592653589793238462643383279502884197169399375105820974944592307816\
 40628620899862803482534211706798214808651328230664709384460955058223172535940 8\
 12848111745028410270193852110555964462294895493038196442881097566593344612847 5\
 64823378678316527120190914564856692346034861045432664821339360726024914127372 4\
 58700660631558817488152092096282925409171536436789259036001133053054882046652 1\
 38414695194151160943305727036575959195309218611738193261179310511854807446237 9\
 962749567351885752724891227938183011949 1b0
(%i16) reset()$
(%i17) set_display(xml)$
```

2.13 Lists

Command	*Description*
`[...]`	Marks a list or enclose subscripts of a list
`length(L)`	Length of list `L`
`append(...)`	Concatenate given lists
`delete(x,L)`	Remove `x` from list `L`

Lists are used to combine several items into a single object. They are displayed by enclosing its elements between square brackets. A simple method to create list by means of square brackets `[...]`. A list may even contain other lists as its entries. It may also consists of one or zero elements.

```
(%i1) L: [1,4,9,16,25,[36,49]];
(%o1) [1,4,9,16,25,[36,49]]
(%i2) emptylist: [];
(%o2) []
```

An element of a list can be accessed by its index enclosed in square brackets. The first element has index `1` and the largest index must not exceed the length of the list.

```
(%i3) L[4];
(%o3) 16
(%i4) length(L);
(%o4) 6
(%i5) L[3]: -99;
(%o5) −99
(%i6) L;
(%o6) [1,4,−99,16,25,[36,49]]
```

There exist a couple of commands for manipulating lists. Concatenating lists (append elements of one list to another) and deleting elements from a list are the most important functions.

```
(%i7) append(L,[1,2,3]);
(%o7) [1,4,−99,16,25,[36,49],1,2,3]
(%i8) delete(1,L);
(%o8) [4,−99,16,25,[36,49]]
```

Notice that these commands return a new list and do not modify the existing one. For the latter purpose one has to replace the original list.

```
 (%i9) L: delete(4,L);
 (%o9) [1,−99,16,25,[36,49]]
(%i10) L;
(%o10) [1,−99,16,25,[36,49]]
```

2.14 Strings

Command	*Description*
`"a string"`	Character string

Character strings are enclosed in double quote marks `"` for input. Strings may contain any characters, including embedded tab, newline, and carriage return characters.

```
(%i1) s1: "This is a String.";
(%o1) This is a String.
```

2.15 Equations

Command	*Description*
`=`	Equation operator
`solve(eqn,x)`	Solve equation **`eqn`** with respect to **`x`**
`lhs(eqn)`	Left-hand side of equation **`eqn`**
`rhs(eqn)`	Right-hand side of equation **`eqn`**
`allroots(polyn)`	Numerical approximations of roots of polynomial **`polyn`** of one variable

Command **`solve`** tries to solve algebraic equations. It returns a list of solution equations. (Use **`float`** when numeric solutions are required.)

```
(%i1) solve(x^2+3*x-1=0, x);
(%o1) $[x = -\frac{\sqrt{13}+3}{2}, x = \frac{\sqrt{13}-3}{2}]$
(%i2) float(%);
(%o2) [x = −3.302775637731995, x = .3027756377319946]
```

Finding roots and zeros of expression are very important special applications of solving equations. Thus if the expression **`expr`** that given as an argument to **`solve`** is not an equation, then the equation **`expr=0`** is assumed in its place. Thus we also obtain the roots of the above polynomial by the following command.

```
(%i3) sol: solve(x^2+3*x-1, x);
```
(%o3) $[x = -\frac{\sqrt{13}+3}{2}, x = \frac{\sqrt{13}-3}{2}]$

The solutions can be reverted into "non-equations" or used in further computations by means of command **`ev`**. An alternative way is to use command **`rhs`** to extract the expression from the right-hand side of the equation.

```
(%i4) ev(x, sol[1]);
```
(%o4) $-\frac{\sqrt{13}+3}{2}$
```
(%i5) rhs(sol[2]);
```
(%o5) $\frac{\sqrt{13}-3}{2}$

The variable can also be a more complex expression like a function.

```
(%i6) solve(exp(f(x))=10^y, f(x));
```
(%o6) $[f(x) = log(10)\, y]$

Systems of two or more expressions as well as their variables must be encapsulated in lists. Each solution is then also returned as list. (Thus we get a list of lists.) Notice that we can assign equations to variables.

```
(%i7) eq1: 3*x^2-y^2=2;
```
(%o7) $3x^2 - y^2 = 2$
```
(%i8) eq2: x^2+y^2=2;
```
(%o8) $x^2 + y^2 = 2$
```
(%i9) solve([eq1,eq2], [x,y]);
```
(%o9) $[[x = -1, y = -1], [x = -1, y = 1], [x = 1, y = -1], [x = 1, y = 1]]$

`solve` returns its solution as equations. These solutions can be used in further computations by means of **`ev`**.

```
(%i10) s: solve(x^3+2*x^2-5*x-6=0, x);
```
(%o10) $[x = -3, x = -1, x = 2]$
```
(%i11) z: ev(%pi^x, s[2]);
```
(%o11) $\frac{1}{\pi}$

Important
`solve` does not always find (all) solutions of a system of equations. Then an empty list is returned. In the following example the "obvious" solution **`[x=1,y=1]`** is not found.

```
(%i12) solve([x^(-1/2)*y^(1/2)=1, x^(1/2)*y^(-1/2)=1], [x,y]);
```
(%o12) $[\,]$
```
(%i13) ev([x^(-1/2)*y^(1/2)-1, x^(1/2)*y^(-1/2)-1], [x=1,y=1]);
```
(%o13) $[0, 0]$

In general there are no closed form solutions for the roots of polynomials of degree 5 or larger. Then command **`allroots`** may be used to get at least a numerical approximation of the roots.

```
(%i14) solve(x^5-x^4+2*x^3+x^2-x+5, x);
```

$$\text{(\%o14)}\quad [0 = x^5 - x^4 + 2 * x^3 + x^2 - x + 5]$$

```
(%i15) allroots(x^5-x^4+2*x^3+x^2-x+5);
```

$$\text{(\%o15)}\quad [x = 1.07479\,\%i + .970613, x = .970613 - 1.07479\,\%i, x = -1.1828,\\ x = 1.41457\,\%i + .120788, x = .120788 - 1.41457\,\%i]$$

2.16 Complex Numbers

Command	*Description*
`rectform(x)`	Express complex number **`x`** in Cartesian form $a + b\,i$
`polarform(x)`	Express complex number **`x`** in polar form $r\,e^{i\theta}$
`realpart(x)`	Real part of a complex number **`x`**
`imagpart(x)`	Imaginary part of a complex number **`x`**
`trigsimp(x)`	Simplifies expressions that contain trigonometric functions

Maxima performs many of its computation over the field of complex numbers. Thus the solutions may contain imaginary numbers.

```
(%i1) solve(x^2-4*x+13=0,x);
```

$$\text{(\%o1)}\quad [x = 2 - 3\,\%i, x = 3\,\%i + 2]$$

While this is quite convenient for many applications it may be annoying when we are interested in purely real solutions. The situation may become even worse as the following example demonstrates. Here the solution is computed by means of Cardano's formula. It looks like complex numbers.

```
(%i2) s: solve(x^3-x+1/3=0,x);
```

$$\text{(\%o2)}\quad [x = \frac{\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}}{3\left(\frac{\%i}{2\,3^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}} + \left(\frac{\%i}{2\,3^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}\left(-\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right),$$

$$x = \left(\frac{\%i}{2\,3^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}\left(\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}\right) + \frac{-\frac{\sqrt{3}\,\%i}{2} - \frac{1}{2}}{3\left(\frac{\%i}{2\,3^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}},$$

$$x = \left(\frac{\%i}{2\,3^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}} + \frac{1}{3\left(\frac{\%i}{2\,3^{\frac{3}{2}}} - \frac{1}{6}\right)^{\frac{1}{3}}}]$$

However, all three solutions are real numbers. We can try to transform these complex numbers into Cartesian form, i.e., into the representation $a + b\,i$ where $a \in \mathbb{R}$ and $b \in \mathbb{R}$ are called *real* and *imaginary* part, respectively. One then can try to reduce the imaginary part to **`0`** by means of command **`trigsimp`**.

```
(%i3) trigsimp(rectform(s));
```
(%o3) $[x = \frac{\sqrt{3}\,\sin\left(\frac{5\,\pi}{18}\right) - \cos\left(\frac{5\,\pi}{18}\right)}{\sqrt{3}}, x = -\frac{\sqrt{3}\,\sin\left(\frac{5\,\pi}{18}\right) + \cos\left(\frac{5\,\pi}{18}\right)}{\sqrt{3}}, x = \frac{2\,\cos\left(\frac{5\,\pi}{18}\right)}{\sqrt{3}}]$
```
(%i4) float(%);
```
(%o4) $[x = .3949308436346985, x = -1.137158042603258, x = .7422271989685593]$

If we do not use **trigsimp** or if it is not possible to reduce the imaginary part to **0**, then a tiny imaginary part might remain when we convert the solutions to floating point numbers due to round-off errors. It can be transformed to a purely real solution by using their real parts. However, we must be confidence that the solution is not a complex number.

```
(%i5) float(rectform(s));
```
(%o5) $[x = .3949308436346984 - 1.110223024625157\ 10^{-16}\ \%i,$
$x = -4.163336342344337\ 10^{-17}\ \%i - 1.137158042603257, x = .7422271989685593]$
```
(%i6) realpart(%);
```
(%o6) $[x = .3949308436346984, x = -1.137158042603257, x = .7422271989685593]$

2.17 Loading Packages

Command	*Description*
load(pkg)	Evaluate content of file **pkg** (“Load package”)

Like many computing environments *Maxima* provides a basic set of commands (core) and offers a variety of add-on packages for special tasks. This keeps starting time and memory consumptions at a minimum and allows large functionality. Moreover, it is thus possible to extend the capabilities of the system. Such packages are either automatically loaded on demand (by means of a list of commands for which packages have to be loaded) or the packages have to be explicitly loaded. For the latter we have to **load** the corresponding files.

```
(%i1) load(vect);
(%o1) /usr/share/maxima/5.22.1/share/vector/vect.mac
```

Notice that the loaded functions are added to the list of user-defined functions and can be displayed and removed using **functions** and **kill(all)**, respectively.

2.18 Writing Your Own Package

Command	*Description*
/*...*/	Comment: all characters between these delimiters are ignored

When working with *Maxima* one may use a set of functions or a complex variable definitions quite frequently. The code for these functions can be collected in a file and loaded into one’s *Maxima* session by means of command **load** as described in Sect. 2.17.

Important
The file must have extension **.mac**.

For example, start your favorite editor, enter the line

```
/* Euclidean norm of vector [x,y] */
norm(x,y):= sqrt(x^2+y^2);
```

and store the content in a file named (e.g.) **`mydefs.mac`**. This file can then loaded into one's *Maxima* session and function **`norm`** can be used just like any other *Maxima* function.

```
(%i1) norm(3,4);
(%o1) norm(3,4)
(%i2) load("mydefs.mac");
(%o2) mydefs.mac
(%i3) norm(3,4);
(%o3) 5
```

The line **`/*...*/`** in the above example is ignored when the file is loaded. It is **strongly recommended** to add such **comments** into one's files. If files get longer or the coded functions get more sophisticated it helps to maintain one's code.

Important
Do not forget the end-of-input marker **`;`** or **`$`** at the end of each piece of code.

Important
Use a string (here **`"mydefs.mac"`**) as argument for **`load`** and do not forget the file extension (**`.mac`**) when you want to load your own file with *Maxima* code.

2.19 LISP

LISP (from *LISt Processing*) is an old programming language and has be developed in 1958 at MIT. It was very popular in the *Artificial Intelligence* research community.

The core of *Maxima* is written in LISP. Thus when a expression is entered it is parsed and evaluated by the underlying LISP interpreter. Fortunately, we do not need any knowledge of this language. However, there are two incidences where you might come in contact with the interpreter. The first one has already been mentioned in Sect 2.9 (Help) when one types **`?sqrt`**. The second one occurs in case of a syntax errors. Usually *Maxima* will try to explain your error. However, sometimes the LISP interpreter gets confused and responses with a rather obscure error message. Then fix your input and ignore the content of this message.

— Exercises

1. Store the so-called golden ration, $\frac{1}{2}(1+\sqrt{5})$, in variable **`gs`**. In addition compute its numerical value.
2. Verify, that the golden ratio is a root of $x^2 - x - 1$. (Do not use **`solve`**!)
3. Compute all roots of $x^2 - x - 1$.
4. Compute **`(-1)^(1/3)`**. How do you get all (complex) cubic roots of -1?
5. Compute the first 100 digits of π. (Do not forget to reset the precision of big float arithmetic!)
6. Define function **`ln`** that computes the natural logarithm.
7. Define a function **`log10(x)`** that computes the common logarithm of **`x`**. Numerical input should be evaluated in floating point. Compute $\log_{10}(1000)$.
8. Define a function **`heron(a,b,c)`** that computes the area of a triangle with sides a, b, and c by means of Heron's formula
$$\text{Area} = \frac{1}{4}\sqrt{(a^2+b^2+c^2)^2 - 2(a^4+b^4+c^4)}\,.$$
9. Get all details about routine **`solve`**.
10. Get a list of all *Maxima* names that contain the (sub-) string **`solve`**.
11. Clear your working space. Remove all your variable and function definitions.

3 Symbolic Computation

3.1 Symbolic Calculator

In *Maxima* everything is an expression, including variables, functions, mathematical expressions, objects and programming constructs. Numbers are thus just a special case. Hence *Maxima* can solve algebraic equations symbolically in the same way as we do it by hand.

```
(%i1) solve(a*x^2+b*x+c=0,x);
```
(%o1) $[x = -\frac{\sqrt{b^2-4\,a\,c}+b}{2\,a},\ x = \frac{\sqrt{b^2-4\,a\,c}-b}{2\,a}]$

Of course the coefficients and variables of this quadratic equation can be more complex expressions.

```
(%i2) solve((s-t)*f(y)^2+abs(s+t)*f(y)+(s+t)=0, f(y));
```
(%o2) $[f(y) = \frac{|t+s|+\sqrt{5\,t^2+2\,s\,t-3\,s^2}}{2\,t-2\,s},\ f(y) = -\frac{\sqrt{5\,t^2+2\,s\,t-3\,s^2}-|t+s|}{2\,t-2\,s}]$

When an expression is evaluated, then *Maxima* substitutes all expressions to which values are assigned but leaves all other (unknown) expressions left as-is.

```
(%i3) foo(3*bar+2*bar) + bar^2 + log(%e);
```
(%o3) $foo(5\,bar)+bar^2+1$

Important
Notice that due to this design of every CAS *Maxima* cannot detect any typo. If one mistypes a variable or function name, then the invalid name is simply returned as-is.

In Sect. 2.7 we have introduced the assignment operators `:` and `:=`. Of course they can be used to assign any expression to a variable or function. Known variables are then substituted while unknown expressions are used as-is. *wxMaxima* tries to print these variables in a nice way. E.g., names of Greek letters are printed by means of the corresponding symbol.

```
(%i4) x: (a+beta)*(a-beta);
```
(%o4) $(a-\beta)\,(\beta+a)$
```
(%i5) y: x-a^2;
```
(%o5) $(a-\beta)\,(\beta+a)-a^2$
```
(%i6) foo(u):= log(u)-x+alpha;
```
(%o6) $foo(u) := log(u)-x+\alpha$
```
(%i7) foo(y);
```
(%o7) $log\left((a-\beta)\,(\beta+a)+2\,a^2\right)-(a-\beta)\,(\beta+a)+\alpha$

Important
When an expression has been assigned to some variable, say **x**, then it remains persistent until it is **kill**ed (or *Maxima* is stopped). Now suppose you are running a *Maxima* session on your computer for hours and use it as a symbolic calculator every now and then, then you might forget about such assignments and you might get dubious results or obscure error messages[1]. From the authors experience this is a very annoying method of "shooting yourself in the foot". So it is strongly recommended to **kill** variable and function definitions that are obsolete.

```
 (%i8) /* ... Do a lot of computations ... */
       /* Now we want to solve an equation in x and y and define: */
       eq: a*x+b*y=c;
```
(%o8) $b\left((a-\beta)(\beta+a)-a^2\right)+a(a-\beta)(\beta+a)=c$
```
 (%i9) /* Oops! We expected: */                       kill(all)$
(%i10) eq: a*x+b*y=c;
```
(%o10) $b\,y+a\,x=c$

3.2 Expand, Factor, and Simplify

Command	*Description*
expand(expr)	Expand expression **expr**
ratexpand(x)	Expand rational expression **expr** (more efficient than **expand**)
factor(expr)	Factor expression **expr**
ratsimp(expr)	Simplifies a (rational) expression **expr**
fullratsimp(expr)	Apply **ratsimp** and similar until no further changes occur
radcan(expr)	Simplifies expression which contains exponentials, logs, and radicals
num(expr)	Numerator of **expr**
denom(expr)	Denominator of **expr**
trigsimp(expr)	Simplifies expression that contains trigonometric functions
trigreduce(expr)	Combines products and powers of trigonometric functions

When *Maxima* evaluates expressions it usually does not perform all "obvious" expansions or simplifications. It also does not try to transform its results into a "nice" form, either. However, *Maxima* provides a couple of commands that try to do particular simplifications or transformations of an expression. Each of these applies several rules which embody conventional notions of simplicity. Simplification of expression is quite a difficult job for a computer algebra system even though there are established routines for standard simplification procedures.

Products of sums can be multiplied out by means of **expand** and **ratexpand**. To the contrary one can **factor** an expression into irreducible terms (which are called *prime*, e.g., prime numbers).

```
(%i1) (x+y)*(x-y);
```
(%o1) $(x-y)(x+y)$
```
(%i2) expand(%);
```
(%o2) x^2-y^2
```
(%i3) factor(%);
```
(%o3) $-(y-x)*(y+x)$

[1]In the worse case you might receive an erroneous results that looks correct at a first glance.

```
(%i4) (x+y)^4;
```
(%o4) $(y+x)^4$

```
(%i5) expand(%);
```
(%o5) $y^4+4\,x\,y^3+6\,x^2\,y^2+4\,x^3\,y+x^4$

```
(%i6) factor(%);
```
(%o6) $(y+x)^4$

```
(%i7) factor(-8*y-4*x+z^2*(2*y+x));
```
(%o7) $(2\,y+x)\,(z-2)\,(z+2)$

```
(%i8) factor(20!);
```
(%o8) $2^{18}*3^8*5^4*7^2*11*13*17*19$

Numerator of rational terms are also split into their respective terms. Command **`ratexpand`** also cancels out common divisors in rational expressions.

```
(%i9) e: (x+y)^2/(x^2-y^2);
```
(%o9) $\dfrac{(y+x)^2}{x^2-y^2}$

```
(%i10) expand(e);
```
(%o10) $\dfrac{y^2}{x^2-y^2}+\dfrac{2\,x\,y}{x^2-y^2}+\dfrac{x^2}{x^2-y^2}$

```
(%i11) ratexpand(e);
```
(%o11) $-\dfrac{y}{y-x}-\dfrac{x}{y-x}$

Maxima provides commands to find equivalent but simpler expression for more complex one. Thus it applies several rules which embody conventional notions of simplicity. Simplification of expression is quite a difficult job for a computer algebra system even though there are established routines for standard simplification procedures.

```
(%i12) (x^2-y^2)/(x+y)^2;
```
(%o12) $\dfrac{x^2-y^2}{(y+x)^2}$

```
(%i13) ratsimp(%);
```
(%o13) $-\dfrac{y-x}{y+x}$

```
(%i14) (%e^x-1)/(1+%e^(x/2));
```
(%o14) $\dfrac{\%e^x-1}{\%e^{x/2}+1}$

```
(%i15) radcan(%);
```
(%o15) $\%e^{x/2}-1$

Commands **`num`** and **`denom`** allow to decompose a ratio into its numerator and its denominator.

```
(%i16) num(a/b);
```
(%o16) a

```
(%i17) denom(a/b);
```
(%o17) b

```
(%i18) (3+cos(x)^4+8*sin(x)+4*(cos(x)^2-sin(x)^2)
       -6*cos(x)^2*sin(x)^2+sin(x)^4)/(8*cos(x)^3);
```
(%o18) $\dfrac{3+cos(x)^4+8\,sin(x)+4\,(cos(x)^2-sin(x)^2)-6\,cos(x)^2\,sin(x)^2+sin(x)^4}{8*cos(x)^3}$

```
(%i19) trigsimp(%);
```
(%o19) $\dfrac{sin(x)+cos(x)^4}{cos(x)^3}$

Important
Maxima offers many more tools to simplify, expand, factor, and contract symbolic expressions. System variables allow a fine-grained adjustment of manipulation. Some of these methods are offered by *wxMaxima*'s menubar. However, it is beyond the scope of this tutorial to describe all the details.

3.3 Verbs and Nouns

operator	*Description*
'expr	Prevents evaluation of expression **expr**
''expr	Causes evaluation of expression **expr**

Maxima distinguishes between *nouns* and *verbs*. A *verb* is an operator which can be executed. A noun is an operator which appears as a symbol in an expression, without being executed. By default, function names are verbs.

The single quote operator ' prevents evaluation of an expression. This can be useful if one needs to "mask" a global variable. Applied to a function call it returns the noun form of the function call.

```
(%i1) y: 2;
(%o1) 2
(%i2) 'y^2 = y^2;
(%o2) y^2 = 4
(%i3) f(x):= x^2;
(%o3) f(x) := x^2
(%i4) f(a);
(%o4) a^2
(%i5) 'f(a);
(%o5) f(a)
```

The quote-quote operator '' (two single quote marks) causes evaluation of an expression **expr** when evaluation is otherwise suppressed, such as in function definitions. The value of **expr** is then substituted for **expr** in the input expression. Applied to the operator of an expression, the quote-quote operator changes the operator from a noun to a verb (if it is not already a verb).

```
(%i6) f(x):= x^2+y^2;
(%o6) f(x) := x^2 + y^2
(%i7) f(x):= x^2+''y^2;
(%o7) f(y) := x^2 + 2^2
```

This is quite convenient when the result of a previous computation should be used as function body of a new function. In the following example, we define a function **`f`** and its derivative **`fx`** which we compute by means of command **`diff`**[2].

```
(%i8) f(x):= exp(-x^2);
```
(%o8) $f(x) := exp(-x^2)$
```
(%i9) fx(x):= ''(diff(f(x),x));
```
(%o9) $fx(x) := -2\,x\,\%e^{-x^2}$

Important
⚠
Notice that the single quote operator `'` and the quote-quote operator `''` act on the next variable or function only. In particular **`'f(x)`** only prevents symbol **`f`** from being evaluated but not its argument **`x`**. When the respective operator should prevent or cause evaluation for a more complex expression, e.g. of **`f(x)`**, then it must be enclosed in parentheses, **`'(f(x))`**.

The following examples demonstrates the behavior. Here only the symbol **`diff`** is evaluated (with value **`diff`**) not the entire expression **`diff(f(x),x)`** as in the example above.

```
(%i10) f(x):= exp(-x^2);
```
(%o10) $f(x) := exp(-x^2)$
```
(%i11) fx(x):= ''diff(f(x),x);
```
(%o11) $fx(x) := diff(f(x), x)$

Here is a simple example for the quote operator.

```
(%i12) y: 2$
(%i13) f(x):= exp(-x^2)$
(%i14) f(y);
```
(%o14) $\%e^{-4}$
```
(%i15) 'f(y);
```
(%o15) $f(2)$
```
(%i16) f('y);
```
(%o16) $\%e^{-y^2}$
```
(%i17) '(f(y));
```
(%o17) $f(y)$

[2] See Sect. 7.3 for command **`diff`**.

3.4 Logical Operators

operator	*Description*
`=`	Equation operator
`equal(x,y)`	Equivalence
`#`	Negation of equation operator
`<`	Less than
`<=`	Less than or equal
`>`	Greater than
`>=`	Greater than or equal
`and`	Conjunction operator
`or`	Disconjunction operator
`not`	Negation operator
`true`	True statement
`false`	False statement
`unknown`	Statement with unknown logical value
`is(x)`	Attempts to determine whether predicate `x` is `true` or `false`

Operator `is` attempts to determine the logical value of a predicate. If the predicate is provably then the respective value `true` or `false` is returned. Otherwise, `is` returns `unknown`.

The operators `and`, `or`, and `not` force evaluation of its operands, i.e., similar to a call to `is`.

```
(%i1) 3>4;
(%o1) 3 > 4
(%i2) is(3>4);
(%o2) false
(%i3) not(3>4);
(%o3) true
(%i4) 3>4 or 7<8;
(%o4) true
(%i5) is(x>0);
(%o5) unknown
```

Important
Operators `=` and `#` test for syntactical equality, in contrast to the equivalence operator `equal`. Expressions can be equivalent and not syntactically equal.

```
(%i6) a: (x+1)*(x-1);
(%o6) (x − 1) (x + 1)
(%i7) b: x^2-1;
(%o7) x^2 − 1
(%i8) is(a=b);
(%o8) false
(%i9) is(equal(a,b));
(%o9) true
```

3.5 Conditional Evaluation

operator	*Description*
`if ... then ... else ...`	Conditional evaluation

Functions like the absolute value of (real) number are defined by different function terms for different subintervals of its domain, e.g.,

$$\text{abs}(x) = \begin{cases} x, & \text{if } x \geq 0, \\ -x, & \text{if } x < 0. \end{cases}$$

We can implement such functions using conditional evaluation

```
if (cond) then (val_1) else (val_0).
```

There **`cond`** is a logical statement composed by means of logical operators (Sect. 3.4). If condition **`code`** is proved to be **`true`**, then the expression evaluates to **`val_1`**. Otherwise, if it is **`false`**, then the expression evaluates to **`val_0`**. If **`is(cond)`** is **`unknown`**, then the whole expression remains unevaluated.

```
(%i1) fabs(x):= if (x>=0) then x else -x;
(%o1) if x >= 0 then x else -x
```

3.6 Assumptions and Facts

operator	*Description*
`assume(pred)`	Adds predicate **`pred`** to list of "facts"
`declare(x,prop)`	Assigns property **`prop`** to symbol **`x`**
`facts()`	Retrieve list of assumptions
`forget(pred)`	Remove assumption **`pred`**

We work with symbols since we are interested in general solutions to our problems. Nevertheless, we have some knowledge about the variables and parameters in our models. For example, the arguments of a utility function are assumed to be nonnegative real numbers. The parameter of a Cobb-Douglas production function fulfill some inequalities (to ensure concavity). However, when doing symbolic computations *Maxima* always assumes the general case and thus often does not provide a solution (e.g., because there might be infinitely many in the field of complex numbers). Moreover, for some computations (e.g., when we want to compute the integral $\int x^a dx$) *Maxima* asks about further properties of given symbols (e.g., whether $a-1$ is zero or not in $\int x^a dx$), because otherwise *Maxima* cannot give an closed form solution. For this purpose we can make assumptions about particular symbols. Thus there is no need for *Maxima* to ask.

Assumptions can be added as an logical expressions using **`assume`**. When *Maxima* evaluates a logical expression, then it takes care about our assumptions. Command **`facts`** retrieves the list of our assumptions. We can use **`forget`** to remove assumptions again.

```
(%i1) is(x>=0);
(%o1) unknown
(%i2) is(U(x)>=0);
(%o2) unknown
(%i3) assume(x>0,U(x)>0);
(%o3) [x > 0, U(x) > 0]
(%i4) is(x>=0 and U(x)>=0);
(%o4) true
(%i5) facts();
(%o5) [x > 0, U(x) > 0]
```

It is possible to add more assumptions and remove some of them.

```
(%i6) assume(y>0);
(%o6) [y > 0]
(%i7) facts();
(%o7) [x > 0, U(x) > 0, y > 0]
(%i8) forget(U(x)>0);
(%o8) [U(x) > 0]
(%i9) facts();
(%o9) [x > 0, y > 0]
```

Important
When we need to **`assume`** equality, then we must use **`equal`** but not the equality operator **`=`**.

```
(%i10) assume(a=1);
 assume: argument cannot be an '=' expression; found a=1
 assume: maybe you want 'equal'.
  -- an error. To debug this try: debugmode(true);
(%i11) assume(equal(a,1));
(%o11) [equal(a,1)]
```

Unfortunately *Maxima* is not very clever in deducing the logical value of a statement.

```
(%i12) assume(x+y<1);
(%o12) [-y - x + 1 > 0]
(%i13) facts();
(%o13) [x > 0, y > 0, -y - x + 1 > 0]
(%i14) is(x*y>0);
(%o14) true
(%i15) is(x<1);
(%o15) unknown
```

However, *Maxima* does detect redundant and inconsistent predicates (as long as **`is(pred)`** is evaluated to **`false`**).

```
(%i16) facts();
(%o16) [x > 0, y > 0, −y − x + 1 > 0]
(%i17) assume(x>0);
(%o17) [redundant]
(%i18) assume(x<0);
(%o18) [inconsistent]
(%i19) facts();
(%o19) [x > 0, y > 0, −y − x + 1 > 0]
```

In addition to state assumptions *Maxima* allows to assign properties to symbols. For example we may declare symbol **n** to be an integer.

```
(%i20) declare(n,integer);
(%o20) done
(%i21) facts();
(%o21) [x > 0, y > 0, −y − x + 1 > 0, kind(n, integer)]
```

3.7 Local Evaluations

Command	*Description*
ev(x,envir)	Evaluate expression **x** with in given environment **envir**

Maxima uses a couple of system variables for a fine-grained control of the system. E.g., in Sect. 2.11 we have seen that the variable **fpprec** controls the number of significant digits in bigfloat numbers. Now suppose that we want to compute the first 50 digits of Euler's number e but not want to modify **fpprec** globally. This can be performed by changing its value locally within function **ev**.

```
(%i1) ev(bfloat(%e), fpprec:100);
(%o1) 2.7182818284590452353602874713526624977572470937b0
```

If **ev** is not used as part of another expression, e.g., in a function, then it is sufficient just to type its arguments.

```
(%i2) bfloat(%e), fpprec:100;
(%o2) 2.7182818284590452353602874713526624977572470937b0
```

— Exercises

12. Multiply out $(a+b)^{10}$.

13. Factor $a^{10}-b^{10}$ and $a^{10}+b^{10}$.

14. Simplify $\dfrac{a^{10}-b^{10}}{a^2-b^2}$.

15. Find all solutions x of

$$x^2+2\,(\sin(\alpha)+\cos(\alpha))\,x+\sin(2\,\alpha)+1=0.$$

16. Define function `sgn(x)` the determines the sign of number `x`. Compute `sgn(x)` for -2, 0, and π.

$$\operatorname{sgn}(x)=\begin{cases}-1, & \text{if } x<0,\\ 0, & \text{if } x=0,\\ 1, & \text{if } x>0.\end{cases}$$

17. Add a minimal set of predicates to your facts such that $\alpha>0$, $\beta>0$, $\alpha+\beta<1$, $\alpha\,\beta>0$, $(\alpha-1)\,(\beta-1)>0$, $\alpha\,(1-\alpha)>0$, $\beta\,(1-\beta)>0$, and $\alpha\,(\alpha-1)<0$, $\beta\,(\beta-1)<0$ evaluate all to `true`. What are then the results of `is(alpha>-1/2)`, `is(alpha>1/2)` and `is(alpha>1)`? Remove your assumptions when you are done.

18. Define function `numeric(x,prec)` which evaluates expression `x` in bigfloat with precision `prec` (number significant digits). Use this function and compute the first 30 digits of $1/e$.

4 Plotting

Plotting graphs of univariate an bivariate functions is an important tool to investigate the properties of such functions. For this task *Maxima* uses external plotting packages. Thus one can choose among two powerful plotting devices[1]:

- *gnuplot* (the default)
  See `http://www.gnuplot.info/` for further information.
- *xmaxima* (formerly called *openmath*)

Maxima's plotting functions calculate a set of points and pass them to the plotting package together with a set of commands. All these points and plotting commands are saved into the file `maxout.gnuplot` which is located in your home folder.

Within *wxMaxima* you can easily switch between these plotting packages when the plot menu is used. Notice that both packages, *gnuplot* and *xmaxima* open a window that shows the requested plot. Therefore, *wxMaxima* offers a third device

- *inline*

that draws the plots into the notebook rather than a separate window. By right clicking on the inline plot and choosing a destination folder you can save it as an image file.

In this manual we only describe package *gnuplot*.

The standard commands for plotting in *Maxima* are **`plot2d`** and **`plot3d`**. For the *inline* device *wxMaxima* uses special functions **`wxplot2d`** and **`wxplot3d`**, respectively.

In general there are three types of plots: *explicit*, *implicit*, and *parametric* plots. In fact the syntax listed in the sections below are nearly identical.

1. **Explicit:** The function is given in terms of the input values. For example the functions $y = f(x) = x^2$ defined in $\mathbb{R}$, or $z = f(x,y) = sin(x\,y)$ defined over the rectangle $[-2,2] \times [-3,2]$.
2. **Implicit:** The function is implicitly given by solving an equation of the form $F(x,y) = 0$. An example is $x^2 + y^2 - 1 = 0$ in the unit square $[0,1] \times [0,1]$.
3. **Parametric:** The plotted curve is given as a path, that is a map $(0,1) \to \mathbb{R}^2$. For example, the parabola $y = x^2$ can be parametrized by using a free parameter named t, and setting $x = t$ and $y = t^2$.

[1]Do not be afraid. Both packages are installed together with *Maxima*.

4.1 Plots in 2D

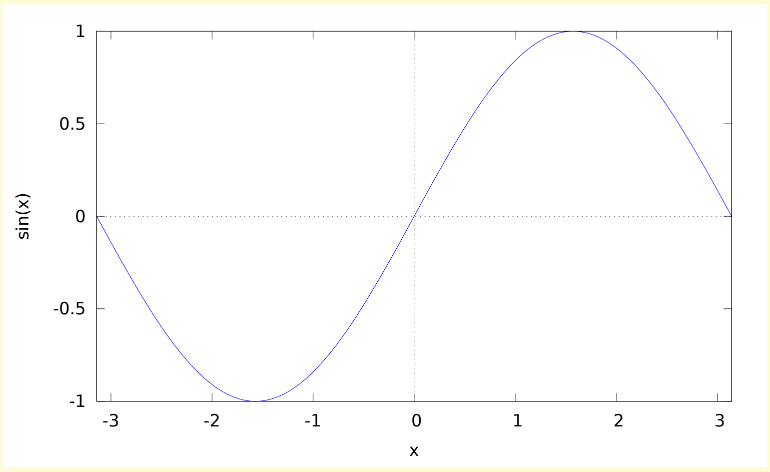
4.1.1 Explicit: Univariate Functions

Command	*Description*
`plot2d(funct,x_range,opts)`	Plot graph of a function of one variable

Option	*Default*	*Description*
`[x,x_min,x_max]`	mandatory	Domain `x_range` of variable `x`
`[y,y_min,y_max]`	automatic	Range of vertical axis (“y-axis”)

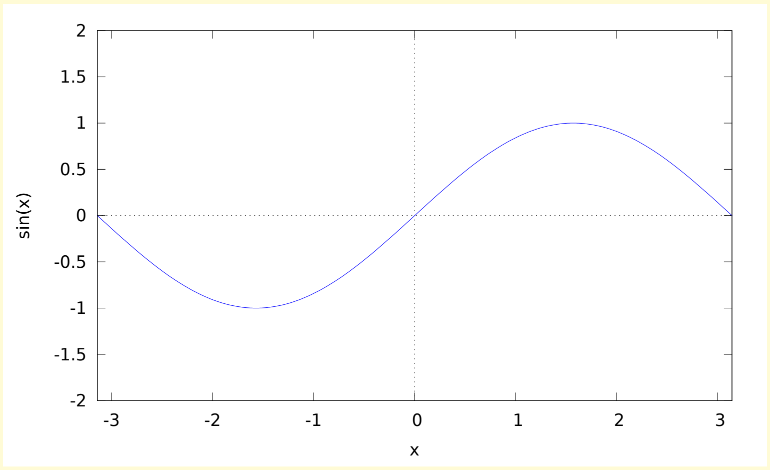
The most common use of **plot2d** is to draw the graph of a function. The first argument **funct** is an expression that describes the function in question. The second argument **x_range** is mandatory and is a list **[x, x_min, x_max]** that contains the name of the variable (usually **x** but other variable names are also possible), and the minimum and maximum values, **x_min** and **x_max**, respectively. In addition options **opts** may be given as third, forth, . . . , argument, each of which is a list enclosed in square brackets, see Sect. 4.3.

```
(%i1) plot2d(sin(x), [x,-%pi,%pi]);
(%o1)
```



The range for the vertical axis can be adjusted by an optional argument with the form **[y, y_min, y_max]** (the keyword **y** is always used for the vertical axis). If that option is used, the plot will show that exact vertical range, independently of the values reached by the plot. If the vertical range is not specified, it will be set up according to the minimum and maximum values of the second coordinate of the plot points.

```
(%i2) plot2d(sin(x), [x,-%pi,%pi], [y,-2,2]);
(%o2)
```



4.1.2 Discrete Points

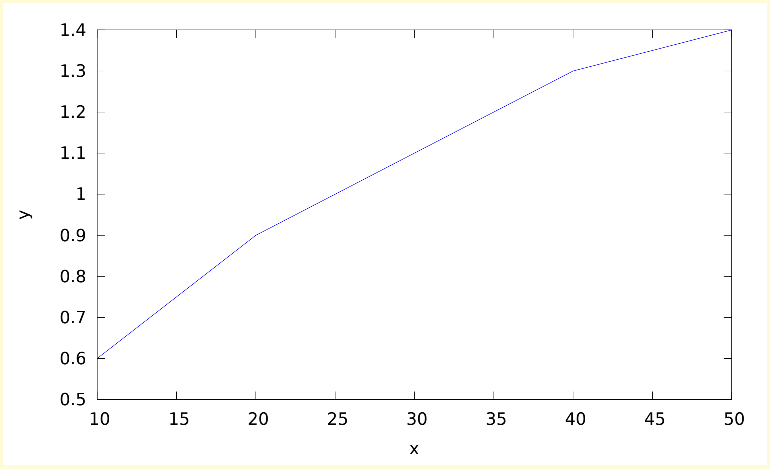
Command	*Description*
`plot2d([discrete,[x_coords,...],[y_coords,...]],opts)`	Plot points
`plot2d([discrete,[[x_coord,y_coord],...],opts)`	Plot points

Option	*Default*	*Description*
`style`	**`lines`**	Plotting style

A plot can also be defined in discrete form, i.e., a set of points with given coordinates. A discrete plot is defined by a list starting with the keyword **`discrete`**, followed by one or two lists of values. If two lists are given, they must have the same length; the first list will be interpreted as the x coordinates of the points to be plotted and the second list as the y coordinates. If only one list is given after the discrete keyword, each element on the list should also be a list with two values that correspond to the x and y coordinates of a point.

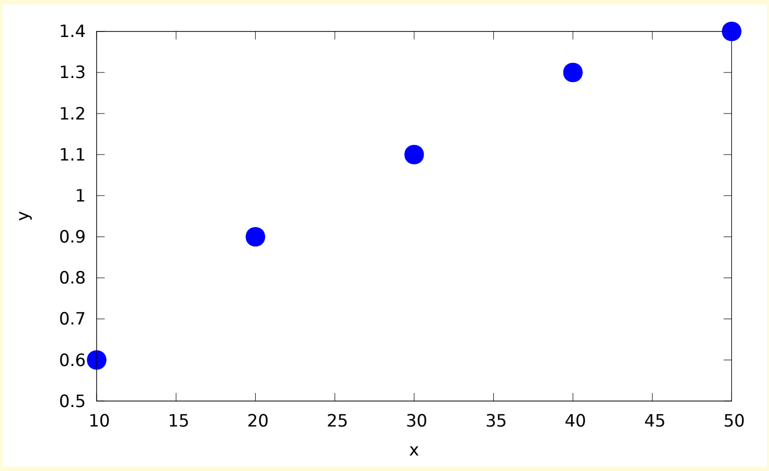
The first example shows the points joined by lines.

```
(%i3) plot2d([discrete, [10,20,30,40,50], [.6,.9,1.1,1.3,1.4]]);
(%o3)
```



The second example just plots the points. Notice that this time we have given the points in single list where each point is a list of two elements.

```
(%i4) plot2d([discrete, [[10,.6],[20,.9],[30,1.1],[40,1.3],[50,1.4]]],
      [style,points]);
(%o4)
```



4.1.3 Parametric Plots

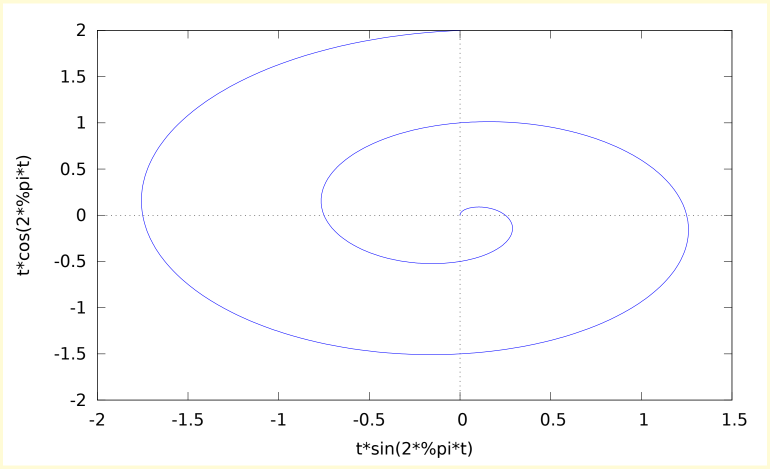
Command	*Description*
`plot2d([parameter,x_coord,y_coord,t_range],opts)`	Parametric plot

Option	*Default*	*Description*
`[t,t_min,t_max]`	mandatory	Domain **`t_range`** of “parameter” **`t`**
`nticks`	`29`	Number of points used for plot

A parametric plot is defined by a list starting with the keyword **`parametric`**, followed by two expressions which are functions of the “parameter” for each coordinate, and a range for the parameter. The range for the parameter must be a list with the name of the parameter (usually **`t`** but other names are also possible) followed by its minimum and maximum values. The plot will show the path traced out by the point with coordinates given by the two expressions or functions, as parameter **`t`** increases from **`t_min`** to **`t_max`**.

Option **`nticks`** sets the number of points shown in the plot. It controls the smoothness of the parametric curve.

```
(%i5) plot2d([parametric, t*sin(2*%pi*t), t*cos(2*%pi*t), [t,0,2],
       [nticks, 1000]]);
(%o5)
```

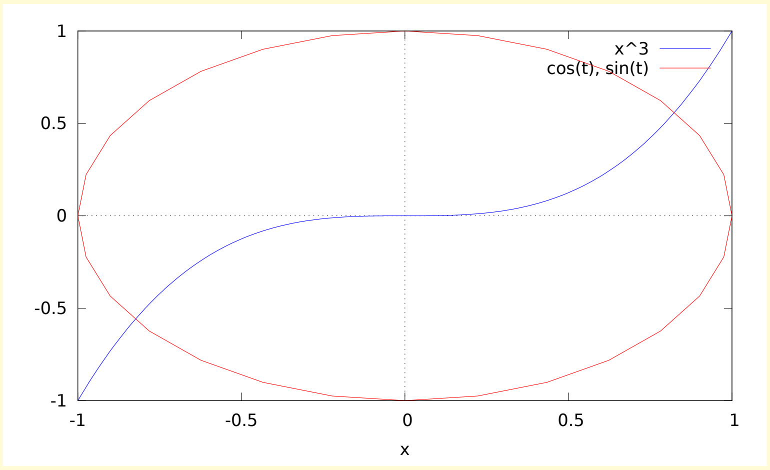


4.1.4 Combining Plots

Command	*Description*
`plot2d([plot_1,...,plot_n],opts)`	Displays several plots

It is possible to combine two or more plots into one picture. The plots have to be given as a list. Here is an example which combines the graph of $f(x) = x^3$ and a parametric plot of a circle.

```
(%i6) plot2d([x^3, [parametric,cos(t),sin(t),[t,-%pi,%pi]]],
       [x,-1,1]);
(%o6)
```



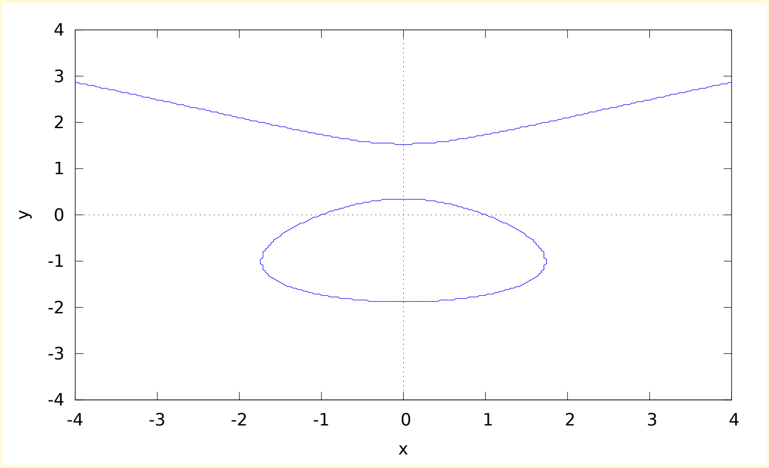
4.1.5 Implicit: Univariate Functions

Command	*Description*
`implicit_plot(funct,x_range,y_range,opts)`	Implicit plot

For plotting implicit functions we must load the package **`implicit_plot`**.

The syntax of an implicit plot is a little bit different from command **`plot2d`**. It requires that intervals for both variables involved in the relation are be specified.

```
(%i7) load(implicit_plot)$
(%i8) implicit_plot(x^2-y^3+3*y-1=0, [x,-4,4], [y,-4,4]);
(%o8)
```



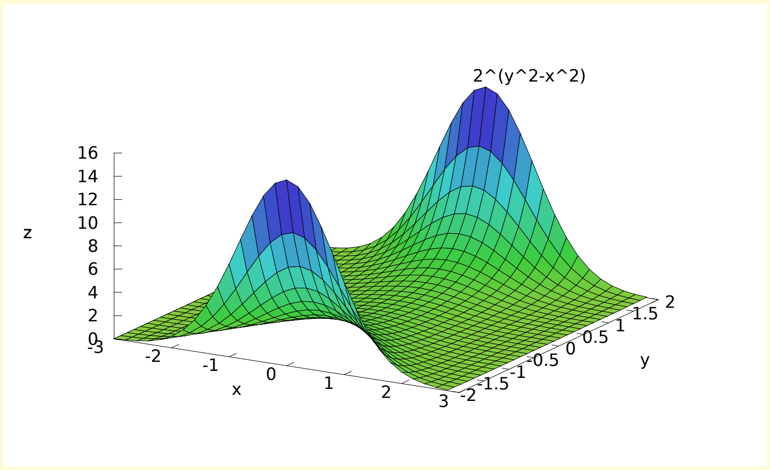
4.2 Plots in 3D

4.2.1 Graphs of Bivariate Functions

Command	*Description*
`plot3d(funct,x_range,y_range,opts)`	Plot graph of bivariate function **`funct`**

In general the plot commands for plots in three dimensions are similar to the command in two dimensions. We focus on graphs of bivariate functions. The only modification needed is the addition of the second variable. Furthermore, the mouse can be used to rotate the plot looking at the surface from view points. Just click with your mouse on the image and keep the mouse button pressed while moving the mouse. (Unfortunately it depends on your OS which mouse button has to be used.)

```
(%i1) plot3d(2^(-x^2+y^2),[x,-3,3],[y,-2,2]);
(%o1)
```

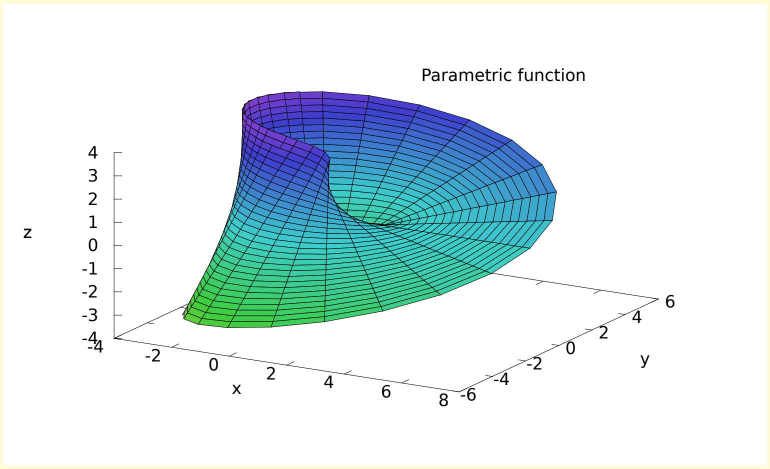


4.2.2 Parametric Plots

Command	*Description*
`plot3d([x,y,z],s_range,t_range,opts)`	Parametric plot in 3d

The next example demonstrates a parametric plot. The surface is given by three expression for the three coordinates as functions of two “parameters” often called s and t. It shows the famous Möbius strip.

```
(%i2) moeb_x: cos(s)*(3+t*cos(s/2))$
(%i3) moeb_y: sin(s)*(3+t*cos(s/2))$
(%i4) moeb_z: t*sin(s/2)$
(%i5) plot3d([moeb_x,moeb_y,moeb_z], [s,-4,4], [t,-4,4]);
(%o5)
```

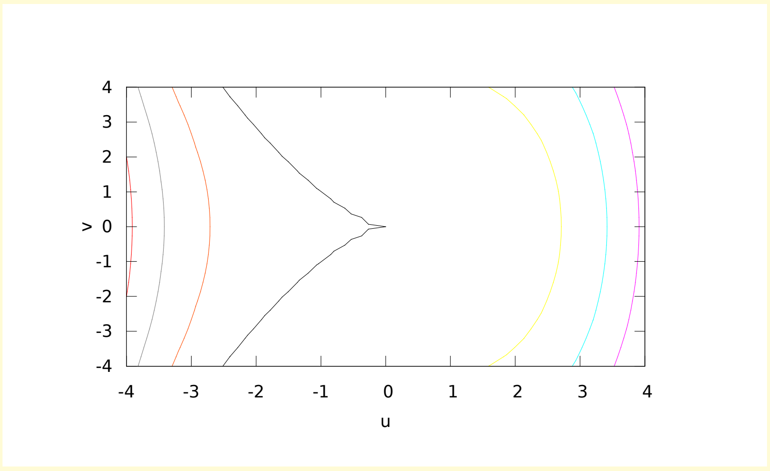


4.2.3 Contour Plots

Command	*Description*
`contour_plot(funct,x_range,y_range,opts)`	Contour plot

A contour plot consists of curves where the given function yields the same value. Unfortunately, one has to pass options to *gnuplot* to set the number of contour levels. The contour plot in the following example shows 12 levels.

```
(%i6) contour_plot(u^3+v^2, [u,-4,4], [v,-4,4], [legend,false],
       [gnuplot_preamble,"set cntrparam levels 12"]);
(%o6)
```



4.3 Plot Options

Maxima offers a vast set of options creating plots. Thus we only describe the most important ones. Please see Section “Plotting” in the *Maxima* manual. Options are always given as list with the option name as its first entry: `[opt, value, ...]`.

4.3.1 Annotation

Option	*Default*	*Description*
`[xlabel, "text"]`	`"x"`	Label for x axis
`[ylabel, "text"]`	`"y"`	Label for y axis
`[zlabel, "text"]`	`"z"`	Label for z axis

With these options one can insert any text for the labels of the two (`plot2d`) and three axes (`plot3d`), respectively.

Option	*Default*	*Description*
`[legend, "text1", "text2", ...]`	automatic	Labels for curves are shown
`[legend, false]`		suppress legend

When a more than one curve is shown in plot (two or more graphs or points in `plot2d`, or for `contour_plot`), then these curves are drawn with different lines (usually in different colors). A legend in the right upper labels these lines. By default, the names of the expressions or functions will be used, or the words `discrete1`, `discrete2`, …, for discrete sets of points. Displaying the legend can be suppressed by means of `[legend,false]`.

4.3.2 Plotting Range

Option	*Default*	*Description*
`[x, x_min, x_max]`	mandatory	Range for x coordinate
`[y, y_min, y_max]`		Range for y coordinate
`[z, z_min, z_max]`	optional	Range for z coordinate

When a function is plotted using **`plot2d`**, then the range for the variable is obligatory. The range of the vertical axes is automatically computed, it will be set up according to the minimum and maximum values of the second coordinate of the plot points. This may sometimes not appropriate. Then the vertical range can be adjusted using option **`y`**.

Similarly, when using **`plot3d`** the ranges of two variables must be given and the range of the third coordinate (z) is computed automatically or my be adjusted using option **`z`**.

Notice that the variables of the functions given to **`plot2d`** and **`plot3d`** need not be called **`x`** and **`y`**. Then options **`x`** and **`y`** can be still used to adjust the plotting range of the first coordinates. In particular, the plotting range may be even larger than the given function plot.

4.3.3 Smoothness Parameters

Option	*Default*	*Description*
`[nticks, integer]`	**`29`**	Initial number of points in **`plot2d`**

When plotting functions with **`plot2d`**, we can control the smoothness of the drawn curve by setting the number of points by means of option **`nticks`**. For explicit plots this is just the initial number of plots that will be automatically increased when the given function has great variation. For parametric plots this is the actual number of points that will be shown for the plot. This often can result in curve that does not look as smooth as expected. In this case it is recommended to increase **`nticks`**.

Option	*Default*	*Description*
`[grid, integer, integer]`	**`[30,30]`**	Number of grid points in **`plot3d`**

The command **`plot3d`** evaluates the given function over a given grid. This grid can be specified by setting the number of points to use in the x and y directions for three-dimensional plotting. May be warned that increasing these numbers may result in very long computation times.

4.3.4 Plotting Styles

Option	*Default*	*Description*
`[style, type_1, type_2, ...]`	automatic	Plotting styles
`[style, [type_1], [type_2], ...]`	automatic	Plotting styles

Gnuplot selects some style for each function or set of data points such these can be distinguished in one plot. However, this may always what your want. E.g., you want to join your data points with line segments. Option **`style`** allows to specify such styles manually. For each function or data set a style must be given. Each style can be either

`lines`	for line segments,
`points`	for isolated points,
`linespoints`	for segments and points, or
`dots`	for small isolated dots.

The first three styles accept an optional parameter[2] (integer) that specifies the line width and point size, respectively. Then each style has the be a list: `[lines,2]` sets the style to `lines` of width `2`.

Option	*Default*	*Description*
`[color, color_1, color_2,...]`	automatic	Line/point color

In `plot2d` and `implicit_plot` we can defines the color (or colors) for the various curves using option `color`. The colors for each curve are taken from the given list in sequential order.

In `plot3d`, this option defines the colors used for the mesh lines of the surfaces, when no palette is being used; one side of the surface will have color `color_1` and the other `color_2` (or the same color if there is only one color).

Available colors are: **`blue`**, **`red`**, **`green`**, **`magenta`**, **`black`**, **`cyan`**.

Option	*Default*	*Description*
`[point_type, type_1, type_2, ...]`	automatic	Point type

When plotting points we can choose among a list of possible point types. The point type for each data set are taken from the given list in sequential order.

Available types are: **`bullet`**, **`circle`**, **`plus`**, **`times`**, **`asterisk`**, **`box`**, **`square`**, **`triangle`**, **`delta`**, **`wedge`**, **`nabla`**, **`diamond`**, **`lozenge`**.

4.3.5 Meshes and Shading

Option	*Default*	*Description*
`[mesh_lines_color, color]`	`black`	Color used for drawing mesh lines
`[mesh_lines_color, false]`		Switch off mesh lines
`[palette, [palette_1], ...]`		Colors for shading surface
`[palette, false]`		Switch of shading of surface

Command `plot3D` uses two technique to draws a surface: It draws a mesh according to the given grid bounds. Then the squares of this mesh are filled using a given color palette. Thus the colors indicate function values (as on any map of a mountain) or it may mimic shading.

The colors of the surface are given by option `palette`. Each palette is a list with a keyword followed by four numbers. The keyword specifies which of the three attributes (`hue`, `saturation` or `value`) will be increased according to the values of z. The first three numbers, which must be between 0 and 1, define the basis color to be assigned to the minimum value of z. The last number indicates the increase corresponding to the maximum value of z.
Default for the first plot is: `[hue, 0.25, 0.7, 0.8, 0.5]`

If palette is given the value `false`, the surfaces will not be shaded but represented with a mesh of curves only. In that case, the colors of the lines will be determined by the option `color` (see Sect. 4.3.4).

Otherwise the color of mesh lines is set my option `mesh_lines_color`. It accepts the same colors as for the option `color`. It can also be given a value `false` to eliminate completely the mesh lines.

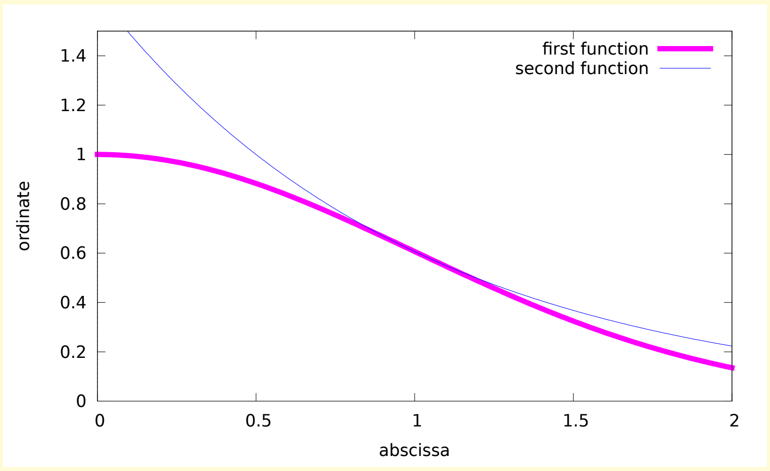
4.3.6 Examples for `plot2d`

In our first example we plot two functions denoted by `first function` and `second function`. Labels for x and y coordinates are `abscissa` and `ordinate`, respectively. We use 100 initial points

[2] In fact there are even more optional parameters, see plot option `style` in Section Plotting of the manual for more details.

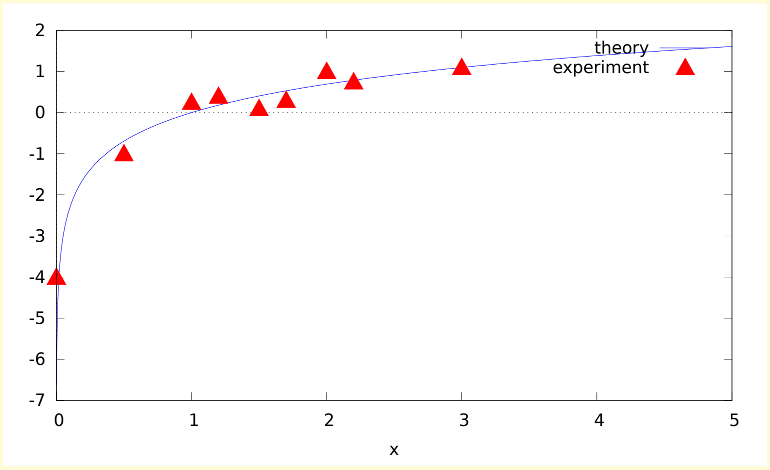
for each plot and restrict the vertical axis to the interval $[0, 1.5]$. The first function is plotted by a thick magenta line while the second function is plotted by a thin blue one.

```
(%i1) plot2d([exp(-x^2/2), sqrt(%e)*exp(-x)], [x,0,2],
       [y,0,1.5], [nticks, 100],
       [legend,"first function", "second function"],
       [xlabel, "abscissa"], [ylabel, "ordinate"],
       [style, [lines,10],[lines,1]], [color, magenta,blue]);
(%o1)
```



The next example shows data points together with an approximating function.

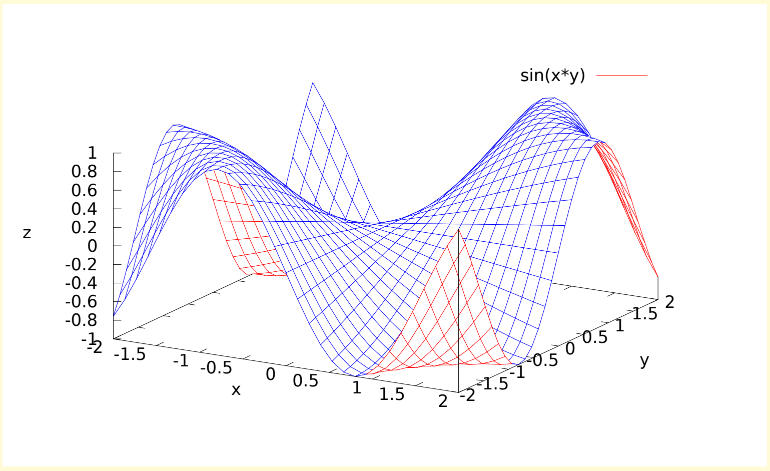
```
(%i2) data_list: [[0,-4], [0.5,-1], [1,0.25], [1.2,0.4], [1.5,0.1],
       [1.7,0.3], [2,1], [2.2,0.75], [3,1.1]]$
(%i3) plot2d([log(x), [discrete, data_list]], [x,0,5],
       [legend, "theory", "experiment"],
       [style, [line,2], [points]],
       [color, blue, red], [point_type, triangle]);
(%o3)
```



4.3.7 Examples for `plot3d`

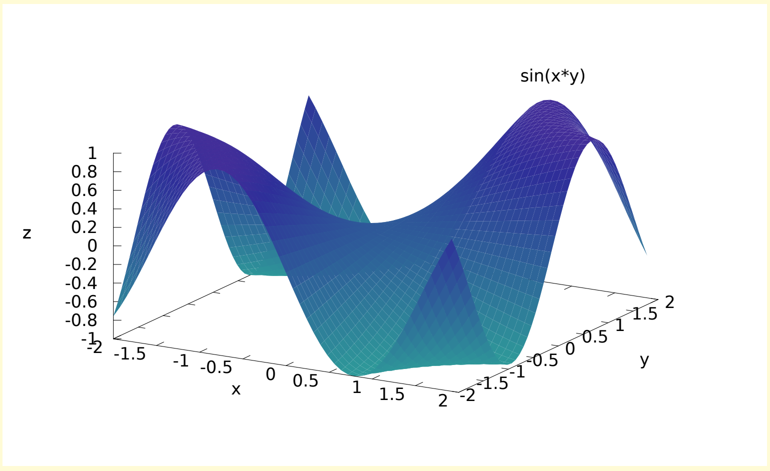
In the first example we only use a mesh but do not use shading. We use blue color for the upper side of the surface and red for its lower side. (When we only gave one color, then both sides are plotted with that color.)

```
(%i4) plot3d(sin(x*y), [x,-2,2], [y,-2,2],
       [palette,false], [color, red, blue]);
(%o4)
```



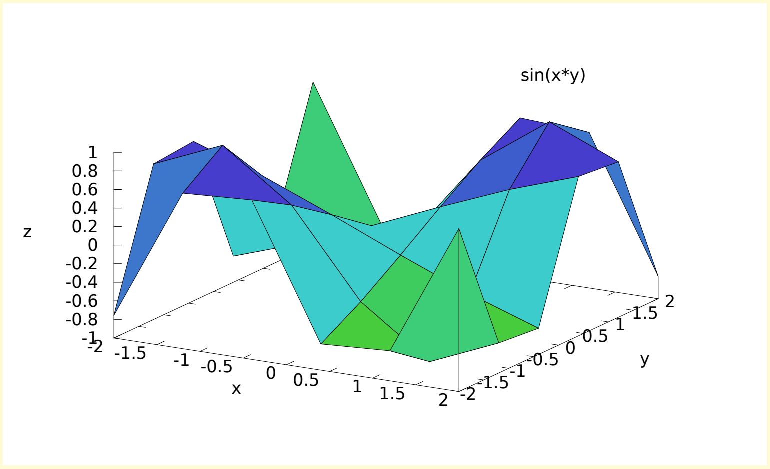
The next example shows a plot without mesh. In addition the shading is different from the default setting and we use a finer grid.

```
(%i5) plot3d(sin(x*y), [x,-2,2], [y,-2,2], [grid, 50,50],
       [mesh_lines_color,false], [palette, [hue, 0.5,0.7,0.6, 0.2]]);
(%o5)
```



The last example shows the influence of option **`grid`**. Here we use a very coarse mesh.

```
(%i6) plot3d(sin(x*y), [x,-2,2], [y,-2,2], [grid, 5,5]);
(%o6)
```



— Exercises

19. Plot the graph of the density of the standard normal distribution, $f(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$, in the range $[-3,3]$. Label the y axis with **`density`**.

20. Plot the graphs of the density of the standard normal distribution, $f(x) = \frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$, and the Cauchy distribution, $g(x) = \frac{1}{\pi(1+x^2)}$, in the range $[-3,3]$. Use a thick red line for the Cauchy distribution, and a thin black line for the normal distribution. The lines should be annotated with **`Gaussian distribution`** and **`Cauchy distribution`**, respectively. The vertical axis should be labeled **`density`**.

21. Plot the graphs of the power functions x^n for $n = -2,-1,0,1,2$ in the range $[0,2]$.

22. We are given the set of data points $\{(.25,-1.03), (.5,-0.63), (.75,-0.28), (1.,0.), (1.25,0.22), (1.5,0.38), (1.75,0.47)\}$. Plot these points together with function $\log(x)$ in the range $[0,2]$. Choose an appropriate range for the vertical axis.

23. Plot the curve $t \mapsto \begin{pmatrix}\sin(4\pi t)\\ \cos(4\pi t)\end{pmatrix}$ for $t \in [0,4]$. What do you see and what do you expect?

24. Plot the graph of the binormal distribution, $f(x,y) = \frac{1}{2\pi}\exp\left(-\frac{1}{2}(x^2+y^2)\right)$.

(a) Use default setting.

(b) Use a blue mesh, no shading.

(c) Use a mesh that is blue on the upper surface and cyan on the lower one.

(d) Use shading only but no mesh.

5 The Draw Package

The abilities of *Maxima*'s **`plot`** command (see Chap. 4) are a bit restricted and do not satisfy all needs. Moreover, additional parameters for the underlying plotting routine have a syntax that is different from that of *Maxima*.

Package **`draw`** provides a more powerful and flexible plotting (and drawing) tool. In addition more complex pictures can be created by means of vector graphics[1]. However, it only supports the *Gnuplot* plotting routine. It thus is a (powerful) *Maxima–Gnuplot* interface. Before we can use the package it must be loaded as described in Sect. 2.17 on p. 18.

```
(%i1) load(draw)$
```

It out of the scope of this tutorial to give a detailed description of the package. Thus we refer to the *Maxima* online help and the corresponding pages in *Maxima*'s online manual which also includes a link to more elaborate examples.

The layout of this chapters roughly follows that of Chap. 4 (Plotting).

5.1 Draw Scenes

Command	*Description*
`gr2d(opts,graphic_objects)`	Builds an object for a 2D scene
`gr3d(opts,graphic_objects)`	Builds an object for a 3D scene
`draw(scenes,opts)`	Plot (a series of) scenes
`draw2d(opts,graphic_object)`	Shortcut for **`draw(gr2d(opts,graphic_object))`**
`draw3d(opts,graphic_object)`	Shortcut for **`draw(gr3d(opts,graphic_object))`**

Roughly spoken the **`draw`** package works as follows. Scenes are described in **`gr2d`** or **`gr3d`** objects, which are then passed to function **`draw`**. If more than one scene is described, a multiplot will be generated. Thus it is possible to construct quite complex pictures. If only one scene is build, then **`draw2d(...)`** and **`draw3d(...)`** can be used as these are equivalent to **`draw(gr2d(...))`** and **`draw(gr3d(...))`**, respectively.

Type **`? gr2d`** and **`? gr3d`** to get a list of available graphic objects in two and three dimensions.

Options for these functions (*graphics options*) are given as **`key=value`** pairs.

Scenes in the **`gr2d`** and **`gr3d`** constructor are interpreted sequentially: graphic options affect those graphic objects placed on its right. Some graphic options affect the global appearance of the scene and may be located at any position or given to the **`draw`** command (*global options*).

[1]In *Vector graphics* a figure is composed of elementary geometric objects like lines, polygons, circles, arcs, etc., which are stored by the coordinates of their characteristic points and characteristic parameters like the radius of a circle.

5.2 Graphic Objects in 2D

Two dimensional scenes can be build by means of the **`gr2d`** scene constructor. There are a couple of graphic objects available. For plotting graphs of functions we have:

- **`explicit`**: The function is given in terms of the input values, e.g., **`f(x):=x^2`**.
- **`implicit`**: The function is implicitly given by the solution of an equation of the form $F(x, y) = 0$, e.g., **`x^2+y^2-1=0`**.
- **`parametric`**: The plotted curve is given as a path in $\mathbb{R}^2$, e.g., **`[sin(t*%pi),cos(t*%pi)]`**.

Besides these there are some low level graphic objects for adding lines or annotating graphs. The following objects have self-explanatory names. For details use *Maxima*'s online help, e.g., **`? points`**.

- **`label`**: Places text on the plot.
- **`bars`**
- **`ellipse`**
- **`points`**
- **`polygon`**
- **`quadrilateral`**
- **`rectangle`**
- **`triangle`**
- **`vector`**: Draws an arrow.
- **`region`**: Plots a region on the plane defined by inequalities.
- **`image`**: Plot a bitmap graphic.

In the following we give a short description of the most important graphic objects.

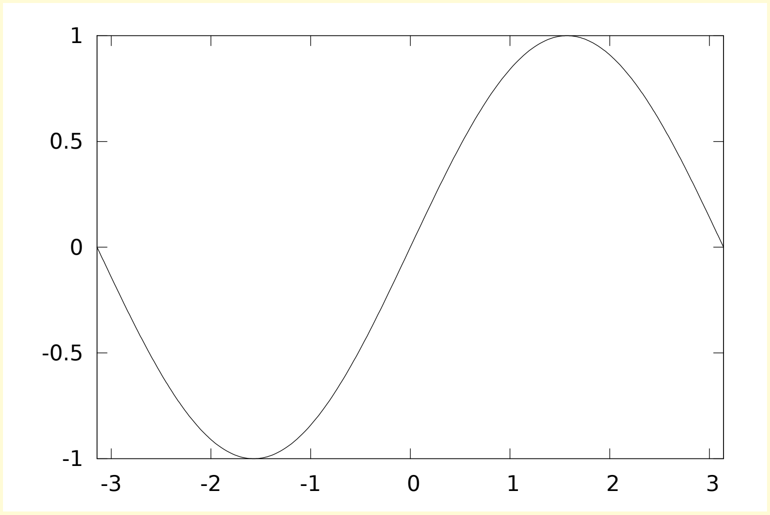
5.2.1 Explicit Functions

Graphic Object	*Description*
`explicit(func, x,xmin,xmax)`	Explicit function with variable **`x`** in 2D

Option	*Default*	*Description*
`xrange=[xmin,xmax]`	automatic	Range of horizontal axis ("y-axis")
`yrange=[ymin,ymax]`	automatic	Range of vertical axis ("y-axis")
`color`	**`black`**	Set color of curve

Plots of explicit functions can be created by means of graphic object **`explicit`**. Notice that in contrast to the **`plot2d`** command, the domain argument are not given as a list but as arguments to the **`explicit`** call. Of course the argument need not be called **`x`**.

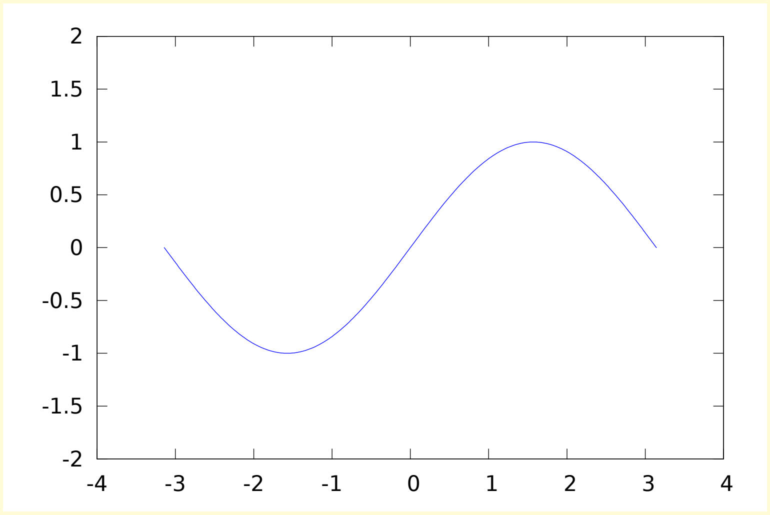
```
(%i1) draw2d(explicit(sin(x), x,-%pi,%pi));
(%o1)
```



The range for the horizontal and vertical axes are computed automatically from the given domain of the function and the range of the function values. They can be adjusted by the respective options **`xrange`** and **`yrange`**. Notice that these are *global options* and may be specified at any position.

The color of the drawn curve can be specified by option **`color`**. This is a *local option* and thus it only has affect on graphic objects that are placed in its right in the argument list of **`grd2d`**. Thus in the following example **`color`** has to be set in before adding the explicit function object.

```
(%i2) draw2d(color=blue, explicit(sin(x), x,-%pi,%pi),
      xrange=[-4,4], yrange=[-2,2]);
(%o2)
```

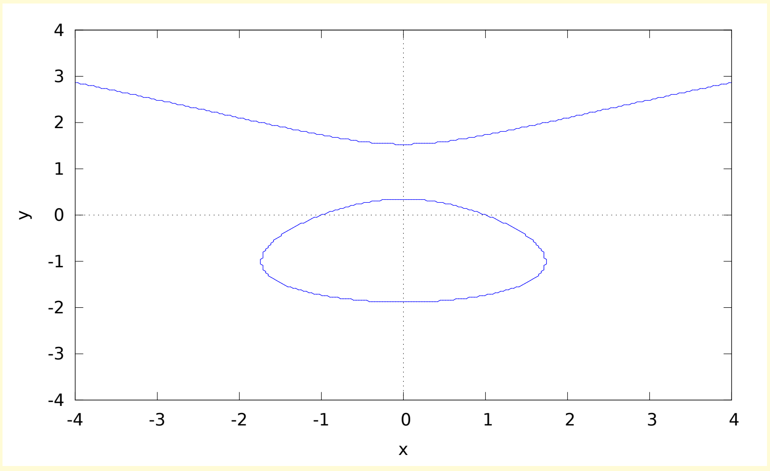


5.2.2 Implicit Functions

Graphic Object	*Description*
`implicit(feq, x,xmin,xmax, y,ymin,ymax)`	Implicit function given by equation **`feq`** with variables **`x`** and **`y`** in 2D

Implicit functions are defined by an equality relation of the form $F(x,y) = 0$, e.g., **`x^2-y^3+3*y-1=0`**. It is given similar to the explicit functions but in contrast to **`explicit`**, **`implicit`** requires that the domain for both variables involved have to be specified. The first variable listed in the **`implicit`** call will be plotted on the x-axis and the second variable is plotted on the y-axis. Of course the two arguments need not be called **`x`** and **`y`**.

```
(%i3) draw2d(implicit(x^2-y^3+3*y-1=0, x,-4,4, y,-4,4));
(%o3)
```



5.2.3 Parametric Plots

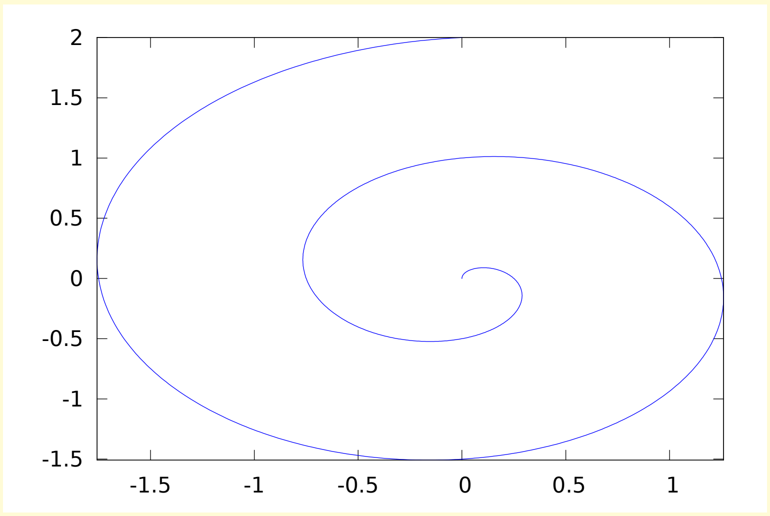
Graphic Object	*Description*
`parametric(xfun,yfun, t,tmin,tmax)`	Parametric function with parameter **`t`** in 2D

Option	*Default*	*Description*
`nticks=integer`	**`29`**	Number of points used for plot

A parametric plot is defined by a pair of expressions for the x and y coordinates, respectively, and a variable (called *parameter*) for each of these two functions (often called **`t`** but other names are also possible). The plot will show the path traced out by the point with coordinates given by the two expressions as parameter **`t`** increases from **`tmin`** to **`tmax`**.

Option **`nticks`** sets the number of points shown in the plot. It controls the smoothness of the parametric curve.

```
(%i4) draw2d(color=blue, nticks=1000,
      parametric(t*sin(2*%pi*t),t*cos(2*%pi*t), t,0,2));
(%o4)
```



5.2.4 Discrete Points

Graphic Object	*Description*
`points([[x_coord,y_coord],...])`	Points in 2D
`points([x_coords,...],[y_coords,...])`	Points in 2D

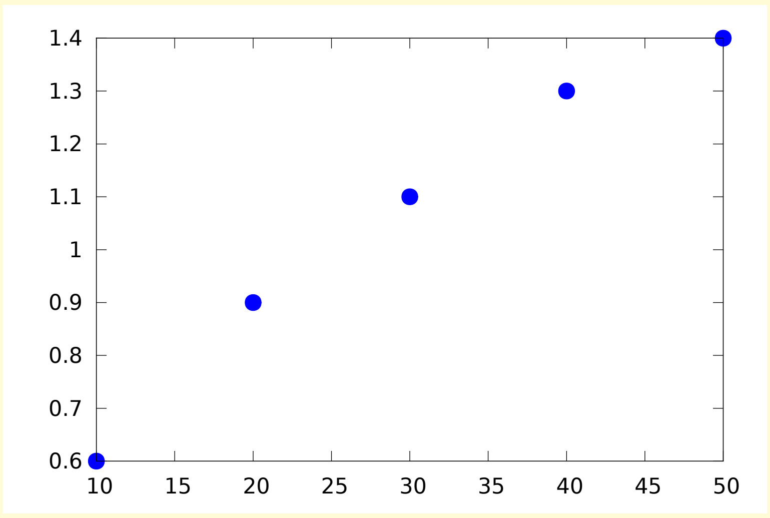
Option	*Default*	*Description*
`point_type`	**`plus`** (1)	Type of point
`point_size`	1	Size of plotted point.
`points_joined`	**`false`**	If **`true`**, then the points are joined by lines.

A plot can also be defined in discrete form, i.e., a set of points with given coordinates. These can be given either by a list of lists of values **`[x,y]`**, or by two lists of equal length of the x and y coordinates of each point, respectively.

By default *Maxima* produces a scatter plot of the points. The style and size of the plotted points can be changed by the respective options **`point_type`** and **`point_size`**. Point types can be specified either by a number or the name of the plot style: **`none`** (−1), **`dot`** (0), **`plus`** (1), **`multiply`** (2), **`asterisk`** (3), **`square`** (4), **`filled_square`** (5), **`circle`** (6), **`filled_circle`** (7), **`up_triangle`** (8), **`filled_up_triangle`** (9), **`down_triangle`** (10), **`filled_down_triangle`** (11), **`diamant`** (12), **`filled_diamant`** (13).

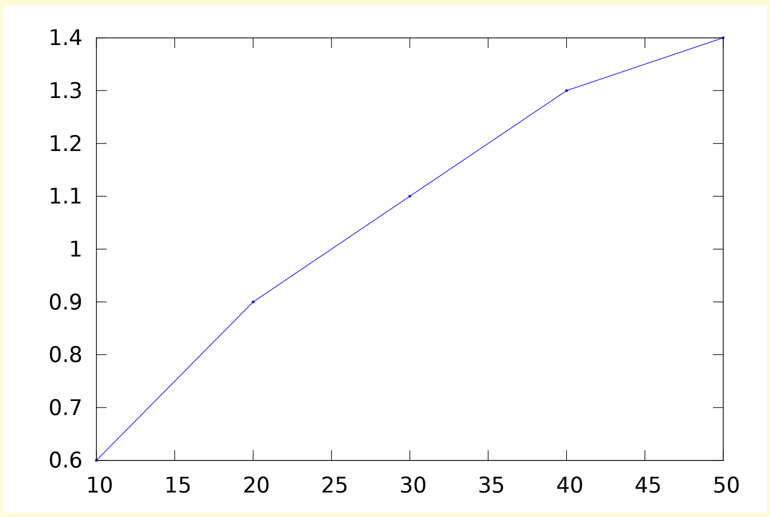
The first example shows a scatter plot where each point is indicated by a blue filled circle. The points are given as a list of 2-tuples.

```
(%i5) draw2d(color=blue, point_type=filled_circle, point_size=2,
      points([[10,.6],[20,.9],[30,1.1],[40,1.3],[50,1.4]]));
(%o5)
```



In the second example the points are joined by lines. Moreover, we suppress plotting a mark for each of the points by using plot style **none**. The points are given as two lists of the x and y coordinates, respectively.

```
(%i6) draw2d(color=blue, point_type=none, points_joined=true,
      points([10,20,30,40,50],[.6,.9,1.1,1.3,1.4]));
(%o6)
```

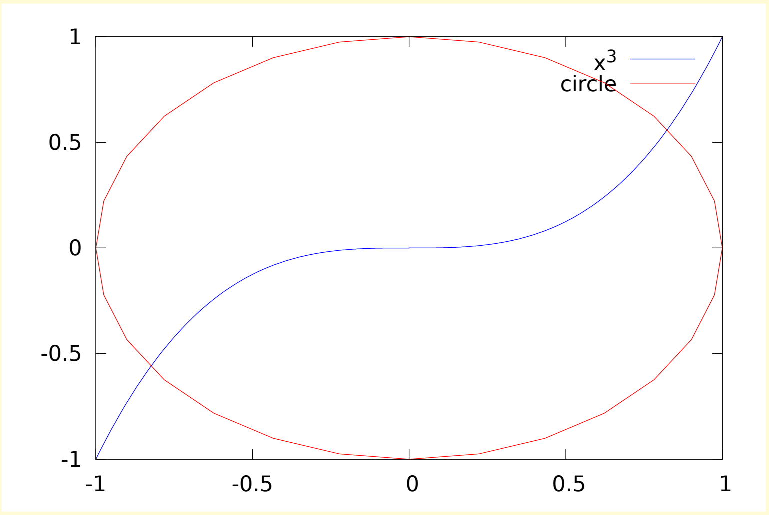


5.2.5 Combining Plots

Option	*Default*	*Description*
`key="Text"`	`""` (empty string)	Name of function in the legend

It is possible to combine two or more graphic objects in one picture. These are just given as arguments to the **`gr2d`** or **`draw2d`** call. Local graphic options may be given between the graphic objects. Recall that these local options affect all those graphic objects that are placed on its right. Option **`key`** can be used to add the name of the function or other graphic objects into the legend of the scene. Here is an example which combines the graph of $f(x) = x^3$ (blue) and a parametric plot of a circle (red).

```
(%i7) draw2d(color=blue,key="x^3", explicit(x^3, x,-1,1),
      color=red,key="circle", parametric(cos(t), sin(t), t,-%pi,%pi));
(%o7)
```



5.3 Graphic Objects in 3D

Most of what we have discussed in Sect. 5.2 about 2D graphics also applies here. Three dimensional scenes can be build by means of the **`gr3d`** scene constructor. Its syntax is similar to that of the **`gr2d`**. The main difference between these constructors is an additional third variable. Again there are a couple of available graphic objects. For plotting graphs of functions we have:

- **`explicit`**: The function is given in terms of the input values, e.g., **`f(x,y):=x^2+y^2`**.
- **`implicit`**: The function is implicitly given by the solution of an equation of the form $F(x,y,z) = 0$, e.g., **`x^2+y^2+z^2-1=0`**.

Curves and surfaces in $\mathbb{R}^3$ can be produced by the following graphic objects:

- **`parametric`**: A parametric curve in $\mathbb{R}^3$, e.g., **`[sin(t*%pi),cos(t*%pi),t]`**.
- **`parametric_surface`**: A parametric surface in $\mathbb{R}^3$.

Besides these there are some low level graphic objects for adding lines or annotating graphs. The following objects have self-explanatory names. For details use *Maxima*'s online help, e.g., **`?points`**.

- **`label`**: Places text on the plot.

- **mesh**: Draws a quadrangular mesh in 3D.
- **points**
- **quadrilateral**
- **triangle**
- **tube**
- **vector**: Draws an arrow.

There are many, many options available for controlling the appearance of 3D graphics. Although we will describe the most important ones, we refer to the *Maxima* manual.

Notice that the 3D plots the mouse can be used to rotate the plot looking at the surface from view points. Just click with your mouse on the image and keep the mouse button pressed while moving the mouse. (Unfortunately it depends on your OS which mouse button has to be used.)

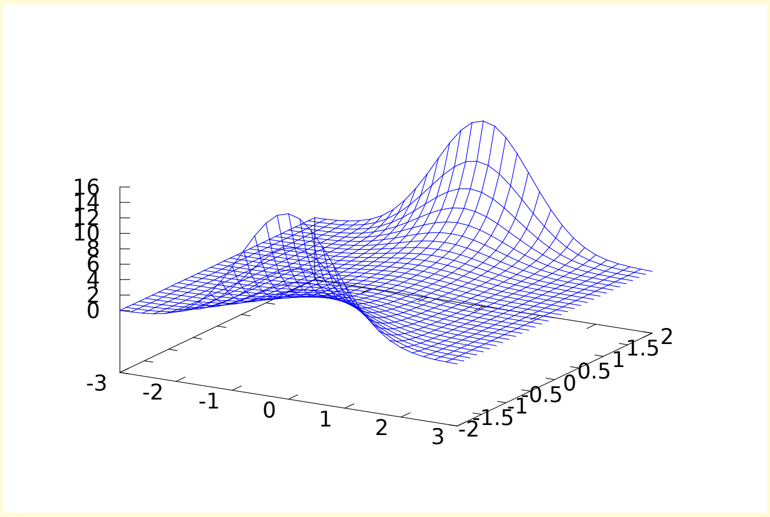
5.3.1 Explicit Functions (Graphs of Bivariate Functions)

Graphic Object	*Description*
explicit(func, x,xmin,xmax, y,ymin,ymax)	Explicit function with variables **x,y**

Option	*Default*	*Description*
surface_hide	**false**	If **true**, hidden parts are not plotted
enhanced3d	**none**	Get colored surface
colorbox	**true**	If **true**, a color scale is drawn

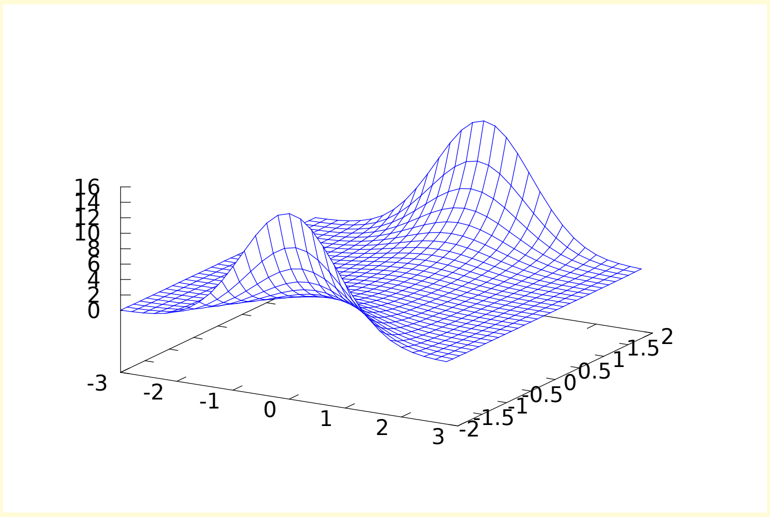
Similarly to univariate functions we get the graph of a bivariate functions using graphical object **explicit**. Note that in opposition to **plot3d** only a mesh in drawn.

```
(%i1) draw3d(explicit(2^(-x^2+y^2), x,-3,3, y,-2,2));
(%o1)
```



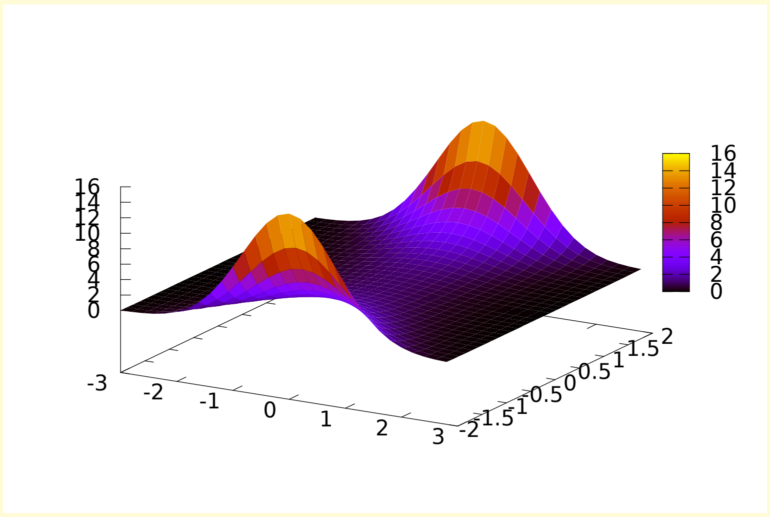
Option **`surface_hide`** controls whether the surface is opaque or not. In the first case hidden parts are not drawn.

```
(%i2) draw3d(explicit(2^(-x^2+y^2), x,-3,3, y,-2,2),
      surface_hide=true);
(%o2)
```



For colored surfaced one has to use option **`enhanced3d`**. The simplest way is to use value **`true`**. For an advanced usage of this option we refer to the corresponding man page. The color scale can be suppressed by mean of option **`colorbox=false`**.

```
(%i3) draw3d(enhanced3d=true, explicit(2^(-x^2+y^2), x,-3,3, y,-2,2));
(%o3)
```

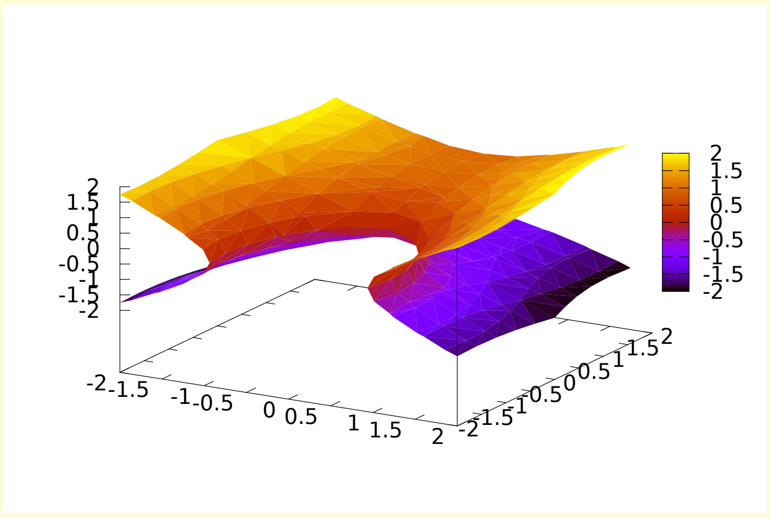


5.3.2 Implicit Functions

Graphic Object	*Description*
`implicit(feq, x,xmin,xmax, y,ymin,ymax, z,zmin,zmax)`	Implicit function

Implicit functions are defined by an equality relation of the form $F(x,y,z) = 0$. It is given similar to the explicit functions but in contrast to **`explicit`**, **`implicit`** requires that the domain for all three variables involved have to be specified. The next example shows the implicit function $x^2 + sin(y) = z^2$ in the domain $[-2,2] \times [-2,2] \times [-2,2]$. When **`enhanced3d`** is **`true`**, then the colors of the surface depend on the z-values.

```
(%i4) draw3d(implicit(x^2+sin(y)=z^2, x,-2,2, y,-2,2, z,-2,2));
(%o4)
```



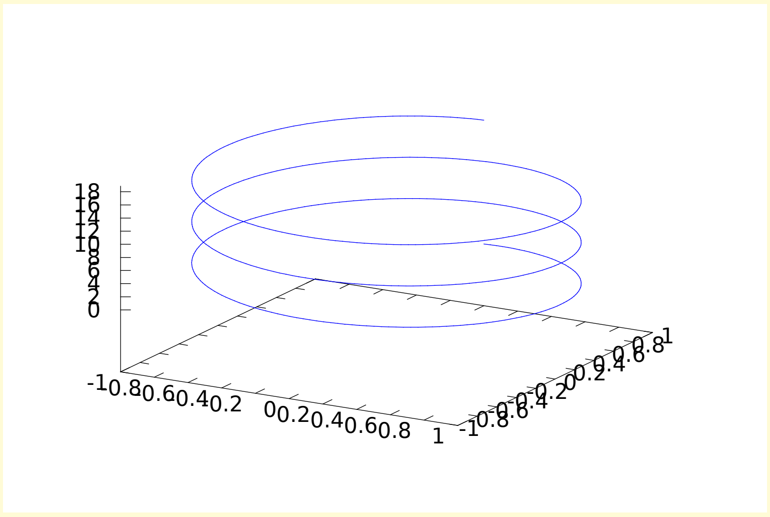
5.3.3 Parametric Plots

Graphic Object	*Description*

`parametric(xfun,yfun,zfun, t,tmin,tmax)`
`parametric_surface(xfun,yfun,zfun, s,smin,smax, t,tmin,tmax)`	

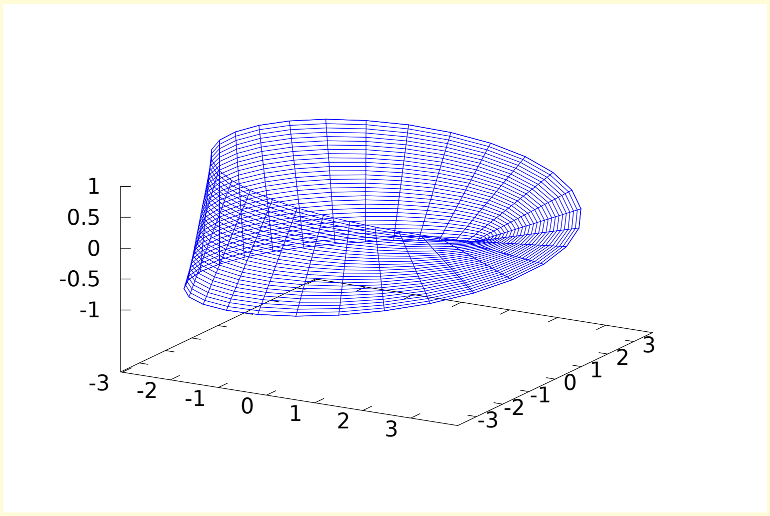
Graphic object **`parametric`** also allows to draw curves in 3D. However, in opposition to the 2D graphic object (see 5.2.3 on p. 48) it now requires the additional argument **`zfun`** for the 3rd dimension. Here is a simple example that draws a helix.

```
(%i5) draw3d(nticks=500, parametric(sin(t),cos(t),t, t,0,6*%pi));
(%o5)
```



A parametric surface in 3D is defined by functions for the x, y, and z coordinates, respectively. Each is a function in two variables (called *parameters*). Such a surfaces is represented by graphic object **`parametric_surface`**. The next example shows the famous Möbius strip.

```
(%i6) moeb_x: cos(s)*(3+t*cos(s/2))$
(%i7) moeb_y: sin(s)*(3+t*cos(s/2))$
(%i8) moeb_z: t*sin(s/2)$
(%i9) draw3d(parametric_surface(moeb_x,moeb_y,moeb_z, s,-%pi,%pi,
      t,-1,1));
(%o9)
```



5.3.4 Contour Plots

Option	*Default*	*Description*
`contour`	**`none`**	Enable contour plots
`contour_levels`	5	Number or location of contour levels

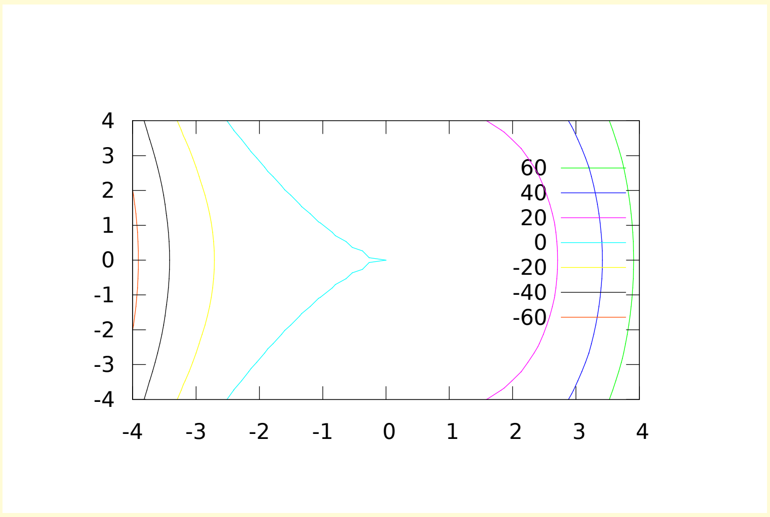
Contour curves of a function $f(x,y)$ are lines where $f(x,y) = c$ for some constants c. In the **`draw`** package, contour plots can be created for **`explicit`** functions using graphic option **`contour`**. Possible values are:

- **`none`**: no contour lines are plotted.
- **`base`**: contour lines are projected on the xy plane.
- **`surface`**: contour lines are plotted on the surface.
- **`both`**: two contour lines are plotted: on the xy plane and on the surface.
- **`map`**: contour lines are projected on the xy plane, and the view point is set just in the vertical.

The number of contour lines can be controlled by graphic option **`contour_levels`**. It either can be a single number for the number of levels, or can be given explicitly a list or set of numbers, see its man page for details.

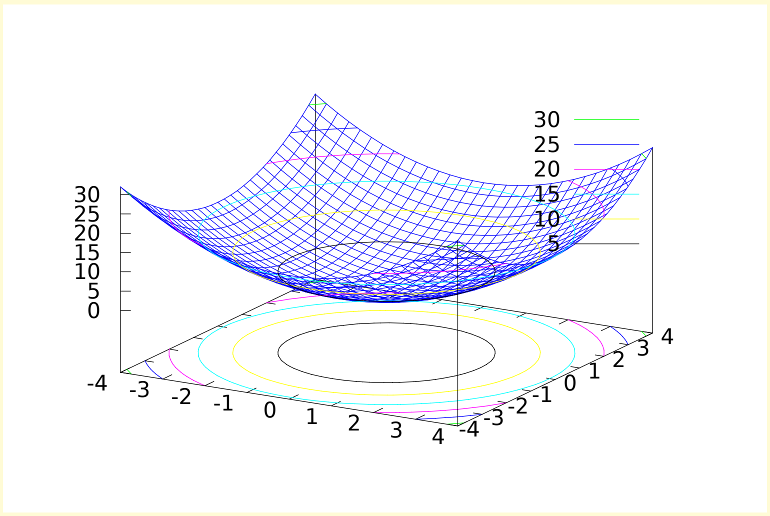
The following example reproduces the contour plot from Sect. 4.2.3 on p. 38. Notice that we have to use **`map`** for this purpose.

```
(%i10) draw3d(explicit(x^3+y^2, x,-4,4, y,-4,4),
       contour=map, contour_levels=12);
(%o10)
```



In addition we can also draw contour lines directly on the **`surface`** of the function graph as well on the **`base`** (the xy plane), or **`both`**. Here is an example for **`both`** with function $f(x,y) = x^2 + y^2$.

```
(%i11) draw3d(explicit(x^2+y^2, x,-4,4, y,-4,4),
       contour=both, contour_levels=15);
(%o11)
```

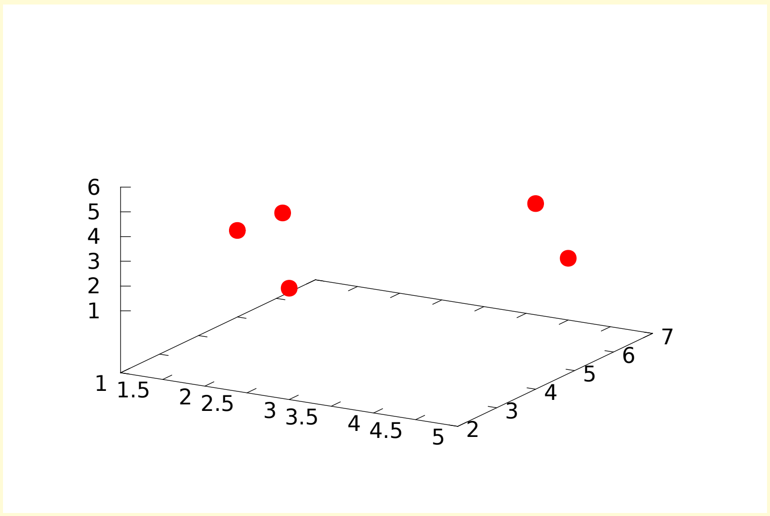


5.3.5 Discrete Points in 3D

Graphic Object	*Description*
`points([[x_coord,y_coord,z_coord],...])`	Points in 3D
`points([x_coords,...],[y_coords,...],[z_coords,...])`	Points in 3D

It is also possible to plot discrete plots in three dimensions. The commands and options described in Sect. 5.2.4 on p. 49. In the following example the data points $(1,5,2)$, $(2,4,4)$, $(3,2,3)$, $(4,7,1)$, $(5,4,6)$ are plotted.

```
(%i12) xc:[1,2,3,4,5]$
(%i13) yc:[5,4,2,7,4]$
(%i14) zc:[2,4,3,1,6]$
(%i15) draw3d(color=red, point_type=filled_circle, points(xc,yc,zc));
(%o15)
```



5.4 Graphic Options

Package **draw** offers a vast set of options creating and controlling plots. Thus we only describe the most important ones. Please see Section “Draw” in the *Maxima* manual.

Graphic Options are always given as **key=value** pairs.

5.4.1 Global Settings

Command	*Description*
set_draw_defaults(opts,...)	Set user graphics options

Function **set_draw_defaults** allows to set global defaults for a *Maxima* session. Calling this function without arguments removes user defaults again.

The following piece of code changes the default color for plots to **red** and turns on plotting a grid on the xy plane.

```
(%i1) set_draw_defaults(color=red, grid=true)$
```

5.4.2 Annotation and Axes

Graphic Object	*Description*
label(["Text",x_coord,y_coord],...])	Write label(s) in 2D
label(["Text",x_coord,y_coord,z_coord],...])	Write label(s) in 3D

One can use graphic object **label** to write any text anywhere in the plot.

Option	*Default*	*Description*
title="Text"	**""** (empty string)	Main title for scene
xlabel="Text"	**""** (empty string)	Label for x axis
ylabel="Text"	**""** (empty string)	Label for y axis
zlabel="Text"	**""** (empty string)	Label for z axis
xtics	**auto**	Tic marks on x axis
ytics	**auto**	Tic marks on y axis
ztics	**auto**	Tic marks on z axis

Option **title** adds a title for the scene. Options **xlabel**, **xlabel**, and **xlabel** inserts a label for the corresponding axes. Options **xtics**,**ytics**,and **ztics** controls the way how tic marks are drawn. For possible values see the corresponding man pages.

Option	*Default*	*Description*
key="Text"	**""** (empty string)	Name of function in the legend

Option **key** inserts **Text** to annotate a given function in the legend. For each function plot a **key** must be given before the graphic object. The entry into the legend for further objects can be suppressed by setting **key=""**.

Option	*Default*	*Description*
`xaxis`	**`false`**	If **`true`**, the x axis is shown drawn
`yaxis`	**`false`**	If **`true`**, the y axis is shown drawn
`zaxis`	**`false`**	If **`true`**, the z axis is shown drawn (3D only)
`grid`	**`false`**	If **`true`**, a grid will be drawn on the xy plane

By default there are no coordinate axes in plots of scenes. These can be drawn by adding options **`xaxis=true`**, **`yaxis=true`** and/or **`zaxis=true`**. There are further options that controls the style, width of color of the corresponding axis, see the man pages of these graphic options for details.

5.4.3 Plotting Range

Option	*Default*	*Description*
`x=[x_min,x_max]`	**`auto`**	Range for x coordinate
`y=[y_min,y_max]`	**`auto`**	Range for y coordinate
`z=[z_min,z_max]`	**`auto`**	Range for z coordinate

These options set the range of the plotting region for each direction.

Notice that in case of (explicit or implicit) functions the name of these graphic options are independent of the particular name of the variables for the function, i.e., **`x`**, **`y`**, **`z`** just refer to the first, second, and third coordinate (direction).

5.4.4 Smoothness Parameters

Option	*Default*	*Description*
`nticks=integer`	**`29`**	Initial number of points used for 2D function plots

When plotting **`explicit`** or **`parametric`** functions, we can control the smoothness of the drawn curve by setting the number of points by means of option **`nticks`**. For **`explicit`** plots this is just the initial number of plots that will be automatically increased when the given function has great variation. For **`parametric`** plots this is the actual number of points that will be shown for the plot. This often can result in curve that does not look as smooth as expected. In this case it is recommended to increase **`nticks`**.

Option	*Default*	*Description*
`xu_grid=integer`	**`30`**	Number of coordinates of the first variable (3D)
`yv_grid=integer`	**`30`**	Number of coordinates of the second variable (3D)

For **`explicit`** functions and **`parametric_surface`** in 3D a grid of sample points is computed. The number of points in each coordinate can be specified by means of the two options **`xu_grid`** and **`yv_grid`**. Thus the given function has to be evaluated in **`xu_grid`** times **`yv_grid`** points.

Option	*Default*	*Description*
`ip_grid=[integer,integer]`	**`[50,50]`**	Number of grid points for the first sampling in implicit plots
`ip_grid_in=[integer,integer]`	**`[5,5]`**	Number of grid points for the second sampling

Plots for **`implicit`** graphic objects are created in a two step procedure. In the first step the coarse structure of the implicit plot is computed. In a second step this structure is refined by means of a local grids. The smoothness of **`implicit`** graphic objects is thus controlled by means of **`ip_grid`** and **`ip_grid_in`**. It may be necessary to increase the values of **`ip_grid`** for very rugged functions.

5.4.5 Export Graphics

Option	*Default*	*Description*
`terminal`	`'screen`	Selects the terminal for plotting
`file_name="file"`	`"maxima_out"`	Name of file for saving graphical output

Option `terminal` allows to set the output device for graphics. By default all graphics are drawn on the screen. However, plots also can be exported into some file by setting one of the following terminals:

- `'pdf`
- `'eps`
- `'png`
- `'gif`
- `'svg`

The output is then stored in a file set by option `file_name`. The respective extension is then automatically added to the file name.

Here is a simple example where we save the plot of the cosine function in PNG format in file `cosine.png`. Notice that one must not add extension `.png`.

```
(%i2) draw2d(explicit(cos(x),x,-2*%pi,2*%pi),
      terminal='png, file_name="cosine");
```

5.4.6 Style and Colors

Option	*Default*	*Description*
`point_type`	`plus` (1)	Type of point (see Sect. 5.2.4)
`point_size`	1	Size of plotted point

Option	*Default*	*Description*
`line_type`	`solid`	Type of line (`solid` or `dots`)
`line_width`	1	Width of plotted line

Option	*Default*	*Description*
`color`	`blue`	Color lines, points and labels

Plotting of points and lines can be controlled by various options. Colors can be set by means of option `color` and can given as names or in hexadecimal RGB code. Available colors are (for a complete list see the man page): `gray`, `blue`, `red`, `green`, `magenta`, `black`, `cyan`, `yellow`.

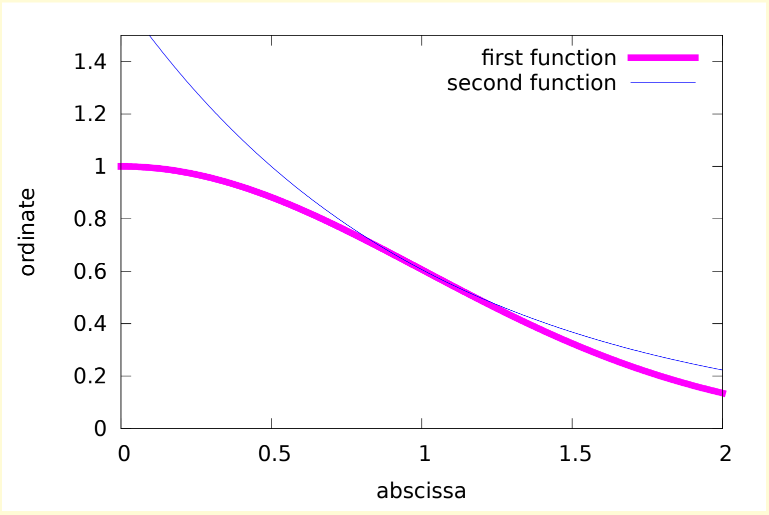
Option	*Default*	*Description*
`surface_hide`	`false`	If `true`, hidden parts are not plotted in 3D
`enhanced3d`	`none`	Get colored surface in 3D
`palette`	`color`	Indicates how to map gray levels onto color components

By default 3D graphical objects are drawn as transparent meshes. If the surface should be opaque use option `surface_hide=true`. For colored surfaced one has to use option `enhanced3d` to enable colored surfaces and option `palette` to set a particular color set, see the corresponding man pages for details.

5.4.7 Examples for `draw2d`

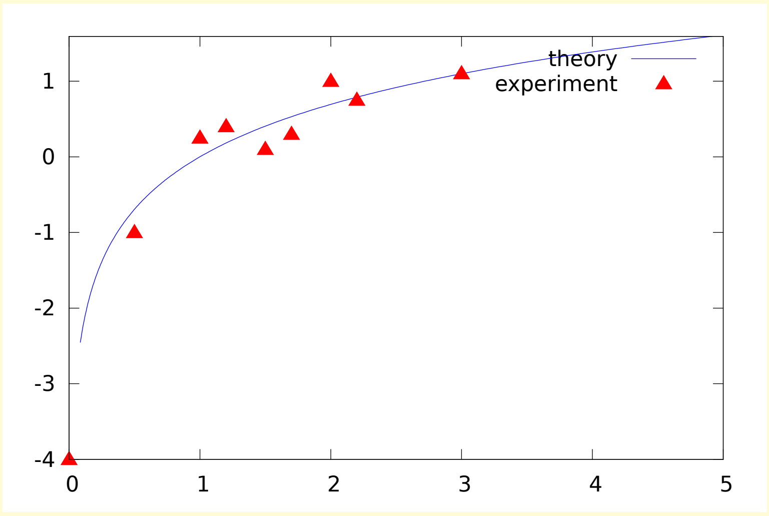
In our first example we plot two functions denoted by **`first function`** and **`second function`**. Labels for x and y coordinates are **`abscissa`** and **`ordinate`**, respectively. We use 100 initial points for each plot and restrict the vertical axis to the interval $[0,1.5]$. The first function is plotted by a thick magenta line while the second function is plotted by a thin blue one.

```
(%i3) draw2d(nticks=100, yrange=[0,1.5],
      xlabel="abscissa", ylabel="ordinate",
      color=magenta, line_width=10, key="first function",
      explicit(exp(-x^2/2), x,0,2),
      color=blue, line_width=1,key="second function",
      explicit(sqrt(%e)*exp(-x), x,0,2));
(%o3)
```



The next example shows data points together with an approximating function.

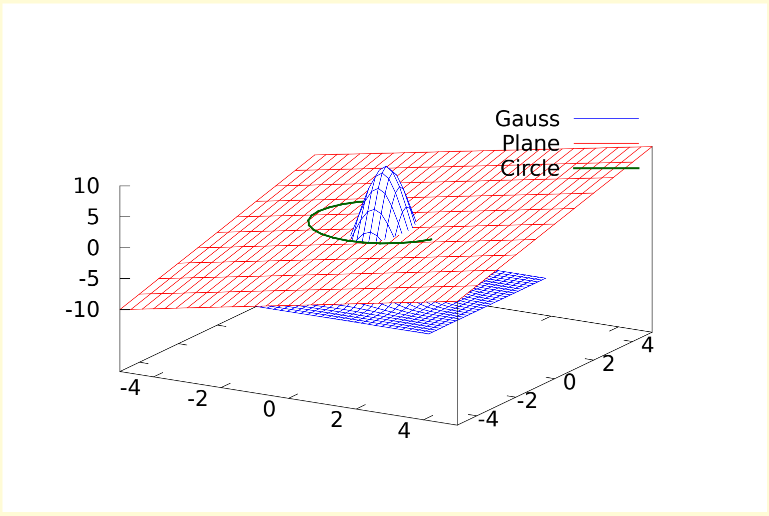
```
(%i4) datalist: [[0,-4], [0.5,-1], [1,0.25], [1.2,0.4], [1.5,0.1],
      [1.7,0.3], [2,1], [2.2,0.75], [3,1.1]]$
(%i5) draw2d(color=blue, key="theory", explicit(log(x), x,0,5),
      color=red, point_type=filled_up_triangle, point_size=3,
      key="experiment", points(datalist));
(%o5)
```



5.4.8 Examples for `draw3d`

The following example demonstrates that it us also possible to combine two or more plots in 3D.

```
(%i6) draw3d(key="Gauss", color=blue,
      explicit(20*exp(-x^2-y^2)-10,x,-3,3,y,-3,3),
      key="Plane", color=red, yv_grid=10,
      explicit(x+y,x,-5,5,y,-5,5),
      key="Circle", color=dark_green, line_width=3,
      parametric(2*sin(t),2*cos(t),1, t,0,2*%pi),
      surface_hide=true);
(%o6)
```



— Exercises

25. Plot the graph of the density of the standard normal distribution, $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, in the range $[-3,3]$. Label the y axis with **`density`**.

26. Plot the graphs of the density of the standard normal distribution, $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, and the Cauchy distribution, $g(x) = \frac{1}{\pi(1+x^2)}$, in the range $[-3,3]$. Use a thick red line for the Cauchy distribution, and a thin black line for the normal distribution. The lines should be annotated with **`Gaussian distribution`** and **`Cauchy distribution`**, respectively. The vertical axis should be labeled **`density`**.

27. Plot the graphs of the power functions x^n for $n = -2, -1, 0, 1, 2$ in the range $[0,2]$. Use different colors for each function.

28. We are given the set of data points $\{(.25,-1.03), (.5,-0.63), (.75,-0.28), (1.,0.), (1.25,0.22), (1.5,0.38), (1.75,0.47)\}$. Plot these points together with function $\log(x)$ in the range $[0,2]$. Choose an appropriate range for the vertical axes.

29. Plot the curve $t \mapsto \begin{pmatrix} \sin(4\pi t) \\ \cos(4\pi t) \end{pmatrix}$ for $t \in [0,4]$. What do you see and what do you expect?

30. Plot the graph of the binormal distribution, $f(x,y) = \frac{1}{2\pi} \exp\left(-\frac{1}{2}(x^2+y^2)\right)$.

6 Linear Algebra

6.1 Vectors

Operator	*Description*
`u+v`	Sum of vectors **`u`** and **`v`**
`u-v`	Difference of vectors **`u`** and **`v`**
`s*v`	Multiply vector **`v`** with scalar **`s`**
`u.v`	Scalar product of vectors **`u`** and **`v`** (also called dot product or inner product)

Vectors are implemented in *Maxima* by means of lists, i.e., their elements enclosed in square brackets **`[...]`**, see Sect. 2.13. Vector addition and multiplication with a scalar are performed by operators **`+`**, **`-`** and **`*`**. The scalar product is computed by the dot operator **`.`**.

```
(%i1) [1,0,1]+[1,2,3];
(%o1) [2,2,4]
(%i2) [1,0,1]-[1,2,3];
(%o2) [0,−2,−2]
(%i3) 5*[1,2,3];
(%o3) [5,10,15]
(%i4) [1,0,1].[1,2,3];
(%o4) 4
```

Important
Notice that **`*`** performs a pointwise multiplication.

```
(%i5) [1,2,3]*[1,2,3];
(%o5) [1,4,9]
```

The *k*-th element of a vector can be retrieved using **`[k]`** as the following example shows.

```
(%i6) x: [1,2,3,4];
(%o6) [1,2,3,4]
(%i7) x[3];
(%o7) 3
```

Notice that one can also use **`[k]`** as an index for any expression and create a vector with symbolic components.

```
(%i8) y[3];
(%o8) y_3
(%i9) z: [z[1],z[2]];
(%o9) [z_1,z_2]
```

6.2 Matrices

Command	*Description*
`matrix(row_1,...,row_n)`	Create a rectangular matrix with rows `row_1`,...,`row_n`

Operators	*Description*
`A+B`	Sum of matrices `A` and `B`
`A-B`	Difference of matrices `A` and `B`
`s*A`	Multiply matrix `A` with scalar `s`
`A.B`	Product of matrices `A` and `B`
`A^^n`	`n`-th power of matrix `A`, i.e., `A.A.` $\cdots$ `.A`
`A^^(-1)`	Inverse of matrix `A`

Matrices are created by means of command **`matrix`** which takes the row vectors of the matrix as its arguments. These must have the same length.

```
(%i1) A: matrix([1,2],[3,4]);
```
$$(\%o1)\quad \begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$$
```
(%i2) B: matrix([0,1],[2,3]);
```
$$(\%o2)\quad \begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix}$$
```
(%i3) A+B;
```
$$(\%o3)\quad \begin{bmatrix}1 & 3\\ 5 & 7\end{bmatrix}$$
```
(%i4) 5*A;
```
$$(\%o4)\quad \begin{bmatrix}5 & 10\\ 15 & 20\end{bmatrix}$$
```
(%i5) A.B;
```
$$(\%o5)\quad \begin{bmatrix}4 & 7\\ 8 & 15\end{bmatrix}$$
```
(%i6) A^^2;
```
$$(\%o6)\quad \begin{bmatrix}7 & 10\\ 15 & 22\end{bmatrix}$$

Remind that the inverse of A is given by A^{-1}.

```
(%i7) A^^(-1);
```
$$(\%o7)\quad \begin{bmatrix}-2 & 1\\ \frac{3}{2} & -\frac{1}{2}\end{bmatrix}$$

Important
Notice that `*` and `^` perform a pointwise multiplication and exponentiation, respectively.

```
(%i8) A*B;
```
$$(\%o8)\quad \begin{bmatrix}0 & 2\\ 6 & 12\end{bmatrix}$$
```
(%i9) A^2;
```
$$(\%o9)\quad \begin{bmatrix}1 & 4\\ 9 & 16\end{bmatrix}$$

The i-th rows of a matrix **A** can be retrieved by means of index `[i]`. Element a_{ij} can be retrieved or replaced using index `[i,j]`.

```
(%i10) A: matrix([1,2],[3,4]);
```
(%o10) $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$
```
(%i11) A[2];
```
(%o11) $[3,4]$
```
(%i12) A[2,1];
```
(%o12) 3
```
(%i13) A[2,1]: 10;
```
(%o13) 10
```
(%i14) A;
```
(%o14) $\begin{bmatrix}1 & 2\\10 & 4\end{bmatrix}$

Important
Vector are usually printed as row vectors. However, *Maxima* is quite inconsistent when treating vectors. Depending on the given expression it is interpreted as row vector or as column vector.

```
(%i15) A: matrix([1,2],[3,4]);
```
(%o15) $\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix}$
```
(%i16) x: [1,2];
```
(%o16) $[1,2]$
```
(%i17) A.x;
```
(%o17) $\begin{bmatrix}5\\11\end{bmatrix}$
```
(%i18) x.A;
```
(%o18) $\begin{bmatrix}7 & 10\end{bmatrix}$

6.3 Operations on Matrices

Command	*Description*
`diagmatrix(n,x)`	Diagonal matrix of size **`n`** with the diagonal elements **`x`**
`ident(n)`	Identity matrix of order **`n`**
`transpose(A)`	Transpose of matrix **`A`**
`invert(A)`	Inverse of matrix **`A`** (same as **`A^^(-1)`**)
`determinant(A)`	Determinant of matrix **`A`**

(Multiples of) identity matrices can be constructed by means of **`diagmatrix`** and **`ident`**.

```
(%i1) ident(2);
```
(%o1) $\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$
```
(%i2) diagmatrix(2,3);
```
(%o2) $\begin{bmatrix}3 & 0\\0 & 3\end{bmatrix}$

```
(%i3) A: matrix([1,2],[3,4]);
```
(%o3) $\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix}$

```
(%i4) transpose(A);
```
(%o4) $\begin{bmatrix}1 & 3\\ 2 & 4\end{bmatrix}$

```
(%i5) invert(A);
```
(%o5) $\begin{bmatrix}-2 & 1\\ \frac{3}{2} & -\frac{1}{2}\end{bmatrix}$

```
(%i6) determinant(A);
```
(%o6) -2

6.4 Rank of a Matrix

Command	*Description*
`rank(A)`	Rank of matrix **`A`**, i.e., dimension of the column space of **`A`**
`nullity(A)`	Corank of matrix **`A`**, i.e., dimension of the null space of **`A`**
`columnspace(A)`	Basis for the column space of **`A`**
`nullspace(A)`	Basis for the null space of **`A`**

Maxima provides commands for computing image and kernel of a matrix.

```
(%i1) A: matrix([1,2,3],[4,5,6],[7,8,9]);
```
(%o1) $\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}$

```
(%i2) rank(A);
```
(%o2) 2

```
(%i3) nullity(A);
```
(%o3) 1

```
(%i4) columnspace(A);
```
(%o4) $\mathit{span}\left(\begin{bmatrix}1\\ 4\\ 7\end{bmatrix}, \begin{bmatrix}2\\ 5\\ 8\end{bmatrix}\right)$

```
(%i5) nullspace(A);
```
(%o5) $\mathit{span}\left(\begin{bmatrix}-3\\ 6\\ 3\end{bmatrix}\right)$

6.5 Lists (Revised)

Command	*Description*
`length(L)`	Length of list **`L`**
`first(L)`	First element of list **`L`**
`second(L)`	Second element of a list **`L`**
`rest(L,n)`	Returns **`L`** with its first **`n`** elements removed
`last(L)`	Last element of list **`L`**

In Sect. 2.13 we already have learned about lists enclosed in square brackets `[...]`. But in fact lists are the basic building block for LISP and thus for *Maxima*. Almost all data types, including functions are lists. So we can extract any elements from this list.

```
(%i1) first(f(x,y,z));
(%o1) x
(%i2) rest(f(x,y,z));
(%o2) f(y,z)
(%i3) last(f(x,y,z));
(%o3) z
```

Commands **`nullspace`** and **`columnspace`** return the bases of the corresponding spaces. The elements of these bases can be extracted using list operations.

```
(%i4) A: matrix([1,2,3],[4,5,6],[7,8,9]);
```

$$\text{(\%o4)}\quad \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}$$

```
(%i5) CS: columnspace(A);
```

$$\text{(\%o5)}\quad span\left(\begin{bmatrix}1\\4\\7\end{bmatrix}, \begin{bmatrix}2\\5\\8\end{bmatrix}\right)$$

```
(%i6) first(CS);
```

$$\text{(\%o6)}\quad \begin{bmatrix}1\\4\\7\end{bmatrix}$$

```
(%i7) second(CS);
```

$$\text{(\%o7)}\quad \begin{bmatrix}2\\5\\8\end{bmatrix}$$

6.6 Linear Equations

A system of linear equations is a collections of linear equations involving the same set of variables. A general system of m equations with n unknowns can be written as

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + \cdots + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + \cdots + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + \cdots + & a_{mn}x_n & = & b_m \end{array}$$

with $x_1, \ldots, x_n$ are the unkowns, $a_{11}, \ldots, a_{mn}$ are the coefficients of the linear equalities and $b_1, \ldots, b_m$ are the constant on the right-hand side. This notations is equivalent to the matrix equation of the form

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}$$

where $\mathbf{A}$ is called the coefficient matrix of this system of linear equations, vector $\mathbf{x}$ contains the unknowns, and vector $\mathbf{b}$ contains all the constants:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n}x_n \\ a_{21} & a_{22} & \cdots & a_{2n}x_n \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn}x_n \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}.$$

Finding solutions of such linear equations is an importing part for many tools. For a quadratic (i.e. $n \times n$) regular coefficient matrix $\mathbf{A}$ one may use its inverse $\mathbf{A}^{-1}$: $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. For this purpose one can apply **`A^^(-1)`** (see Sect. 6.2) or **`invert(A)`** (see Sect. 6.3).

```
(%i1) A: matrix([1,2,2],[3,3,8],[0,7,4]);
```
(%o1) $\begin{bmatrix} 1 & 2 & 2 \\ 3 & 3 & 8 \\ 0 & 7 & 4 \end{bmatrix}$
```
(%i2) b: [-1,-15,2];
```
(%o2) $[-1, -15, 2]$
```
(%i3) A^^(-1).b;
```
(%o3) $\begin{bmatrix} 1 & 2 & -3 \end{bmatrix}$

Notice that there exist other more efficient methods for solving such linear equations. See the *Maxima* manual for such functions.

Command **`solve`** offers a general method that works for any number of equations an unknown, see also Sect. 2.15.

```
(%i4) eq1: x[1]+2*x[2]+2*x[3]=-1;
```
(%o4) $2x_3 + 2x_2 + x_1 = -1$
```
(%i5) eq2: 3*x[1]+3*x[2]+8*x[3]=-15;
```
(%o5) $8x_3 + 3x_2 + 3x_1 = -15$
```
(%i6) eq3: 7*x[2]+4*x[3]=2;
```
(%o6) $4x_3 + 7x_2 = 2$
```
(%i7) solve([eq1,eq2,eq3], [x[1],x[2],x[3]]);
```
(%o7) $[[x_1 = 1, x_2 = 2, x_3 = -3]]$

`solve` also can handle under-determined systems of equations as the following example shows. The symbol **`%r1`** is introduced to represent arbitrary parameters in the solution. A particular solution can be found by replacing **`%r1`** by some number.

```
(%i8) solve([eq1,eq2], [x[1],x[2],x[3]]);
```
(%o8) $[[x_1 = -\frac{10\,\%r1 + 27}{3}, x_2 = \frac{2\,\%r1 + 12}{3}, x_3 = \%r1]]$
```
(%i9) %, %r1=2;
```
(%o9) $[[x_1 = -\frac{47}{3}, x_2 = \frac{16}{3}, x_3 = 2]]$

Notice that the set of all solutions of an under-determined systems of equations forms an affine space which can be described by a particular solution of $\mathbf{Ax} = \mathbf{b}$ and the nullspace of $\mathbf{A}$ (which is the set of solutions of the homogeneous equation $\mathbf{Ax} = 0$). Its basis can be obtained by command **`nullspace`**, see Sect. 6.4.

```
(%i10) nullspace(matrix([1,2,2],[3,3,8]));
```
(%o10) $span\left(\begin{bmatrix} 10 \\ -2 \\ -3 \end{bmatrix}\right)$

For the sake of completeness we also remark that one can compute the echelon form of a given matrix.

```
(%i11) echelon(A);
```
(%o11) $\begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & -\frac{2}{3} \\ 0 & 0 & 1 \end{bmatrix}$

6.7 Eigenvalues and Eigenvectors

Command	*Description*
`charpoly(A,x)`	Characteristic polynomial of A
`eigenvalues(A)`	Eigenvalues of matrix A
`eigenvectors(A)`	Eigenvectors of matrix A

Command **charpoly(A,x)** returns the characteristic polynomial for matrix **`A`** with respect to variable **`x`**.

```
(%i1) A: matrix([1,2,1],[1,4,4],[7,3,5]);
```
(%o1) $\begin{pmatrix} 1 & 2 & 1 \\ 1 & 4 & 4 \\ 7 & 3 & 5 \end{pmatrix}$

```
(%i2) charpoly(A,x);
```
(%o2) $-7(4-x)+((4-x)(5-x)-12)(1-x)-2(-x-23)+3$

Eigenvalues can be computed by means of command **`eigenvalues`**. It returns a list of two elements. The first one is the list of eigenvalues while the second element is a list of their corresponding algebraic multiplicities.

```
(%i3) B: matrix([1,0],[6,6]);
```
(%o3) $\begin{pmatrix} 1 & 0 \\ 6 & 6 \end{pmatrix}$

```
(%i4) eigenvalues(B);
```
(%o4) $[[1,6],[1,1]]$

```
(%i5) eigvals: eigenvalues(B)[1];
```
(%o5) $[1,6]$

```
(%i6) multiplicities: eigenvalues(B)[2];
```
(%o6) $[1,1]$

Command **`eigenvectors`** computes both eigenvalues and the eigenvector. It returns a list of two elements: the result of **`eigenvalues`** and a list of the corresponding eigenspaces. Each eigenspace is represented by a list that contain its basis vectors ("eigenvectors").

```
 (%i7) es: eigenvectors(B);
```
(%o7) $[[[1,6],[1,1]],[[[1,-\frac{6}{5}]],[[0,1]]]]$
```
 (%i8) eigvals: es[1][1];
```
(%o8) $[1,6]$
```
 (%i9) multiplicities: es[1][2];
```
(%o9) $[1,1]$
```
(%i10) vectors: es[2];
```
(%o10) $[[[1,-\frac{6}{5}]],[[0,1]]]$

Notice that multiple eigenvalues are listed only once as the following example shows. In following example the algebraic multiplicity of the unique eigenvalue 3 is 2 while the dimension of the corresponding eigenspace (i.e., its geometric multiplicity) is only 1.

```
(%i11) C: matrix([3,1],[0,3]);
```
(%o11) $\begin{pmatrix}3 & 1\\ 0 & 3\end{pmatrix}$
```
(%i12) eigenvalues(C);
```
(%o12) $[[3],[2]]$
```
(%i13) eigenvectors(C);
```
(%o13) $[[[3],[2]],[[[1,0]]]]$

In the next example algebraic and geometry multiplicities coincide

```
(%i14) D: matrix([3,0],[0,3]);
```
(%o14) $\begin{pmatrix}3 & 0\\ 0 & 3\end{pmatrix}$
```
(%i15) eigenvectors(D);
```
(%o15) $[[[3],[2]],[[[1,0],[0,1]]]]$

Important
Command **`eigenvalue`** simply computes the roots of the characteristic polynomial. In the case of symmetric matrices we known that all eigenvalues are real. However, the result may look like complex numbers. These the imaginary numbers can be removed as described in Sect. 2.16.

— Exercises

31. Let $\mathbf{x} = \begin{pmatrix}1\\2\\3\end{pmatrix}$ and $\mathbf{y} = \begin{pmatrix}4\\5\\6\end{pmatrix}$. Compute $\mathbf{x}+\mathbf{y}$, $\mathbf{x}-\mathbf{y}$, $\mathbf{x}'\mathbf{y}$, and $3\mathbf{x}$.

32. Let $\mathbf{A} = \begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$ and $\mathbf{A} = \begin{pmatrix}-1&0&2\\-2&1&3\\2&4&-1\end{pmatrix}$.

Compute $\mathbf{A}+\mathbf{B}$, $\mathbf{A}-\mathbf{B}$, $3\mathbf{A}$, $\mathbf{A}\,\mathbf{B}$, $\mathbf{A}^2$, $\mathbf{B}^{-1}$, $|\mathbf{A}|$, and $\mathbf{A}'$. What happens when one tries to compute $\mathbf{A}^{-1}$?

33. Let $\mathbf{A} = \begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}$ and $\mathbf{A} = \begin{pmatrix}-1&0&2\\-2&1&3\\2&4&-1\end{pmatrix}$.

Compute rank and corank of matrices **A** and **B**.
Compute null space and columns space of **A** and **B**.
Compute the linear combination $\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2$ of the base vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ of the column space of **A**.

34. Solve the following systems of equations:

(a)
$$\begin{aligned}2x_1+3x_2+4x_3&=2\\4x_1+3x_2+x_3&=10\\x_1+2x_2+4x_3&=5\end{aligned}$$

(b)
$$\begin{aligned}2x_1+2x_2+x_3+3x_4&=10\\3x_1+5x_2+2x_3-x_4&=30\\x_1+2x_2+x_3-x_4&=12\end{aligned}$$

(c)
$$\begin{aligned}2x_1+10x_2+4x_3+9x_4&=1\\x_1+6x_2+5x_3+3x_4&=1\\3x_1+16x_2+9x_3+11x_4&=-1\\x_1+5x_2+2x_3+5x_4&=2\\x_2+3x_3&=4\end{aligned}$$

35. Compute all eigenvalues and eigenvectors of $\mathbf{A} = \begin{pmatrix}1&2\\3&4\end{pmatrix}$. Store the list of eigenvalues in `eigval`. Store the eigenvectors to the first and second eigenvalue in `v[1]` and `v[2]`, repectively.

36. Compute all eigenvalues and eigenvectors of $\mathbf{A} = \begin{pmatrix}1&2&1\\2&1&3\\1&3&1\end{pmatrix}$ numerically.

Notice that the matrix is symmetric (and thus all eigenvalues are real).

37. Compute all eigenvalues and eigenvectors of $\mathbf{A} = \begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$ and $\mathbf{B} = \begin{pmatrix}1&1&1\\0&1&1\\0&0&1\end{pmatrix}$.

What are the algebraic and geometric multiplicities of the corresponding eigenvalues?

7 Calculus

7.1 Limits

Command	*Description*
`limit(expr,x,val)`	Limit of expression **`expr`** when variable **`x`** approaches **`val`**
`limit(expr,x,val,dir)`	Limit of **`expr`** when **`x`** approaches **`val`** from direction **`dir`**

Constant	*Description*
`infinity`	Complex infinity
`ind`	Bounded indefinite result
`und`	Undefined result

Command **`limit`** computes the limit of a given expression. It requires an expression, name of the variable with the limit point to approach. It also computes limits to **`inf`** (∞) and **`minf`** ($-\infty$), respectively. When computing the limit *Maxima* assumes that **`x`** is real variable. The output can also be one of ∞, $-\infty$, **`infinity`** (complex infinity), **`ind`** (indefinite but bounded), or **`und`** (undefined). Notice that **`infinity`** is returned when the limit of the absolute value of the expression is positive infinity, but the limit of the expression itself is neither ∞ nor $-\infty$.

```
(%i1) limit(sin(x)/x,x,0);
(%o1) 1
(%i2) limit(atan(x),x,minf);
(%o2) -π/2
(%i3) limit(log(x),x,0);
(%o3) infinity
(%i4) limit(sin(1/x),x,0);
(%o4) ind
(%i5) limit(sin(1/x)/x,x,0);
(%o5) und
```

Important
Do not mix up result **`infinity`** with ∞. The latter is represented by **`inf`** in *Maxima*.

It is also possible to compute limits from above and below by adding a forth argument. It may have the respective values **`plus`** for limits from above and **`minus`** for limits from below.

```
(%i6) limit(log(x),x,0,plus);
(%o6) -∞
(%i7) limit(x/abs(x),x,0,plus);
(%o7) 1
(%i8) limit(x/abs(x),x,0,minus);
(%o8) -1
(%i9) limit(x/abs(x),x,0);
(%o9) und
```

7.2 Sums and Products

Command	*Description*
sum(expr,i,i_0,i_1)	Sum of the values of **expr** with index **i**
product(expr,i,i_0,i_1)	Product of the values of **expr** with index **i**

Variable	*Default*	*Description*
simpsum	**false**	When **true**, simplification rules for sums are applied
simpproduct	**false**	When **true**, simplification rules for products are applied

Command **sum** evaluates the sum of the first argument **expr** where index variable **i** is substituted for all values **i_0,...,i_i**.

```
(%i1) sum(z(i),i,1,5);
```
(%o1) $z(5)+z(4)+z(3)+z(2)+z(1)$
```
(%i2) sum(z[i],i,1,5);
```
(%o2) $z_5+z_4+z_3+z_2+z_1$
```
(%i3) sum(i^3,i,1,10);
```
(%o3) 3025

Notice that **i_1** $\geq$ **i_0** must hold. Otherwise **0** is returned. Upper and lower bound my also be expressions including **minf** and **inf**.

```
(%i4) sum(z[i],i,1,n);
```
(%o4) $\sum_{i=1}^{n} z_i$
```
(%i5) sum(z[i],i,1,inf);
```
(%o5) $\sum_{i=1}^{\infty} z_i$

Maxima tries to compute (simplify) sums when the system variable **simpsum** is **true**.

```
(%i6) sum(1/4^i,i,1,inf);
```
(%o6) $\sum_{i=1}^{\infty} \frac{1}{4^i}$
```
(%i7) sum(1/4^i,i,1,inf), simpsum;
```
(%o7) $\frac{1}{3}$
```
(%i8) sum(1/i^2,i,1,inf);
```
(%o8) $\sum_{i=1}^{\infty} \frac{1}{i^2}$
```
(%i9) sum(1/i^2,i,1,inf), simpsum;
```
(%o9) $\frac{\pi^2}{6}$

Analogously one can compute products by means of command **`product`**.

```
(%i10) product(i^3,i,1,10);
(%o10) 47784725839872000000
(%i11) product(z(i),i,1,minf);
```
(%o11) $\prod_{i=1}^{\infty} z(i)$
```
(%i12) product(x+i*(i+1)/2,i,1,4);
```
(%o12) $(x+1)(x+3)(x+6)(x+10)$

7.3 Derivatives and Total Differentials

Command	*Description*
`diff(expr,x)`	Derivative of **`expr`** with respect to **`x`**
`diff(expr,x,n)`	**`n`**-th derivative of **`expr`** with respect to **`x`**
`diff(expr,x_1,n_1,x_2,n_2,...)`	Mixed partial derivatives
`diff(expr)`	Total differential of **`expr`**
`del(x)`	Represents differential of **`x`**

Maxima is capable of computing derivatives. It provides a single command **`diff`** for this purpose. **`diff(expr,x)`** differentiates the expression **`expr`** with respect to variable **`x`**. Optionally, a third argument may be given for derivatives of higher order. For example **`expr,x,2`** yields the second derivative of **`expr`** with respect to variable **`x`**.

```
(%i1) diff(x^a,x);
```
(%o1) $a\,x^{a-1}$
```
(%i2) diff(x^a,x,2);
```
(%o2) $(a-1)\,a\,x^{a-2}$

Mixed partial derivatives can be computed either by nesting **`diff`** calls or listing all variables as arguments.

```
(%i3) diff(diff(x^3*y^2,x),y);
```
(%o3) $6\,x^2\,y$
```
(%i4) diff(x^3*y^2,x,1,y,1);
```
(%o4) $6\,x^2\,y$
```
(%i5) diff(x^3*y^2,x,2,y,1);
```
(%o5) $12\,x\,y$

Important
Notice that for mixed partial derivatives each variable name must be immediately followed by the order of the respective derivative.

When we call **`diff`** with only one argument (i.e., without passing any variable names), then *Maxima* computes the total differential of the given expression. The differentials of the variables are represented by symbol **`del`**.

```
(%i6) diff(sin(y*x));
```
(%o6) $x\cos(x\,y)\,del(y)+y\cos(x\,y)\,del(x)$

If we prefer **dx** and **dy** instead, we have to substitute accordingly as shown in the next example.

```
(%i7) diff(sin(y*x));
```
(%o7) $x\,cos(xy)\,del(y) + y\,cos(xy)\,del(x)$
```
(%i8) %, del(x)=dx, del(y)=dy;
```
(%o8) $dx\,y\,cos(x\,y) + dy\,x\,cos(x\,y)$

The quote operator `'` may be used to prevent *Maxima* from immediately compute the derivative.

```
(%i9) 'diff(exp(-x^2),x)=diff(exp(-x^2),x);
```
(%o9) $\frac{d}{dx}\%e^{-x^2} = -2\,x\,\%e^{-x^2}$

7.4 Dependencies and Chain Rule

Command	*Description*
depends(f,x)	Declare functional dependency of **f** from **x**

Variable	*Default*	*Description*
derivabbrev	**false**	When **true**, symbolic derivatives are displayed as subscripts
dependencies	**[]**	List of functional dependencies

We can also work with symbolic derivatives, i.e., the derivatives some function f. However, any dependencies of f must be represented explicitly.

```
(%i1) diff(f(x,y),x);
```
(%o1) $\frac{d}{dx}f(x,y)$
```
(%i2) diff(f,x);
```
(%o2) 0

However, *Maxima* allows to declare dependencies among variables and **diff** takes fully care of this.

```
(%i3) depends(f,x);
```
(%o3) $[f(x)]$
```
(%i4) diff(f,x);
```
(%o4) $\frac{d}{dx}f$

By default derivatives are displayed in Leibniz notation $\frac{df}{dx}$ (which should read $\frac{\partial f}{\partial x}$ in case of partial derivatives). However, when we set the system variable **derivabbrev** to **true**, then symbolic derivatives are displayed as subscript as in f_x.

```
(%i5) derivabbrev: true$
(%i6) diff(f,x);
```
(%o6) f_x
```
(%i7) derivabbrev: false$
```

Now suppose that x itself depends on some other variable t. The we may compute the derivative of the compound function $f(x(t))$ with respect to t by means of the chain rule:

$$\frac{df(x(t))}{dt} = \frac{d}{dx}f(x(t))\,\frac{d}{dt}x(t)$$

We try the following:

```
(%i8) diff(f(x(t)),t);
```
(%o8) $\frac{d}{dt}f(x(t))$

However, this does not work. The solution is to declare a dependency of x from t (notice that already declared f being depend from t).

```
(%i9) depends(x,t);
```
(%o9) $[x(t)]$
```
(%i10) diff(f,t);
```
(%o10) $\left(\frac{d}{dx}f\right)\left(\frac{d}{dt}x\right)$

Do not forget to remove the dependencies when you do not need it any more. Notice **dependencies** contains a list of all dependencies declared so far. They can be removed using **kill**

```
(%i11) dependencies;
```
(%o11) $[f(x), x(t)]$
```
(%i12) kill(f)$
(%i13) dependencies;
```
(%o13) $[x(t)]$

Here is a more advanced function. Assume we need the derivatives of $f(x(t),t)$ with respect to x and t.

```
(%i14) depends(f,[x,t]);
```
(%o14) $[f(x,t)]$
```
(%i15) depends(x,t);
```
(%o15) $[x(t)]$
```
(%i16) diff(f,x);
```
(%o16) $\frac{d}{dx}f$
```
(%i17) diff(f,t);
```
(%o17) $\left(\frac{d}{dx}f\right)\left(\frac{d}{dt}x\right)+\frac{d}{dt}f$
```
(%i18) kill(all)$
```

When computing total derivatives *Maxima* treats every unknown symbol as variable. However, we may declare some of these variables as constants using command **declare**, see Sect 3.6.

```
(%i19) diff(a*x^2+b*y^2);
```
(%o19) $2\,b\,y\,del(y)+2\,a\,x\,del(x)+y^2\,del(b)+x^2\,del(a)$
```
(%i20) declare([a,b],constant);
```
(%o20) *done*
```
(%i21) diff(a*x^2+b*y^2);
```
(%o21) $2\,b\,y\,del(y)+2\,a\,x\,del(x)$

7.5 Jacobian Matrix and Gradient

Command	*Description*
`jacobian(f,x)`	Jacobian matrix of vector-valued function **`f`** with respect to vector **`x`**

In *Maxima* Vector-values functions, i.e., functions from $\mathbb{R}^n \to \mathbb{R}^m$, are represented by vectors of (single-values) functions.

```
(%i1) f(x,y):= [x^2+y^2, x^2-y^2];
```
(%o1) $f(x,y) := [x^2+y^2, x^2-y^2]$

The derivative of such functions is called the Jacobian matrix.

```
(%i2) jacobian(f(x,y),[x,y]);
```
(%o2) $\begin{bmatrix} 2\,x & 2\,y \\ 2\,x & -2\,y \end{bmatrix}$

Here is a more symbolic example.

```
(%i3) f(x,y):= [f[1](x,y),f[2](x,y)];
```
(%o3) $f(x,y) := [f_1(x,y), f_2(x,y)]$
```
(%i4) jacobian(f(x,y),[x,y]);
```
(%o4) $\begin{bmatrix} \frac{d}{dx} f_1(x,y) & \frac{d}{dy} f_1(x,y) \\ \frac{d}{dx} f_2(x,y) & \frac{d}{dy} f_2(x,y) \end{bmatrix}$

The gradient of a single-valued function can be computed as Jacobian. Notice that the function must be given as a list with one element. The result is a $1 \times n$-matrix.

```
(%i5) jacobian([x^3+y^2],[x,y]);
```
(%o5) $\begin{bmatrix} 3\,x^2 & 2\,y \end{bmatrix}$

7.6 Taylor Series

Command	*Description*
`taylor(expr,x,a,n)`	Expands **`expr`** in a Taylor series of order **`n`** in variable **`x`** around **`a`**
`powerseries(expr,x,a)`	General form of the power series expansion
`niceindices(expr)`	Rename indices of sums and products in **`expr`**

A Taylor series is a representation of a function as an infinite sum of terms calculated from the values of its derivatives at a single point a.

$$f(x) = \sum_{k=0}^{\infty} \frac{d^k f}{dx^k}(a)\,(x-a)^k$$

The n-th order truncated taylor series (also known as the n-th Taylor polynomials) consists of the first terms up to degree n.

$$f(x) \approx \sum_{k=0}^{n} \frac{d^k f}{dx^k}(a)\,(x-a)^k = f(a) + f'(a)\,(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f(n)(a)}{n!}(x-a)^n$$

It can be easily computed by means of command **`taylor`**. In the following example we compute the truncated Taylor series of order 3 of e^x with expansion point 0.

```
(%i1) taylor(exp(x),x,0,3);
(%o1) /T/  1 + x + x^2/2 + x^3/6 + ...
```

The leading symbol $/T/$ indicates a Taylor series. The trailing ellipses . . . represents truncation. The resulting expression can be used in further calculations. For this purpose the truncated Taylor series is first transformed into a polynomial that is represented by a rational expression. It is indicated by symbol $/R/$.

```
(%i2) T(x):= ''(taylor(exp(x),x,0,3));
(%o2) /T/  T(x) := 1 + x + x^2/2 + x^3/6 + ...
(%i3) T(3);
(%o3) /R/  13
(%i4) T(y);
(%o4) /R/  (y^3 + 3 y^2 + 6 y + 6)/6
```

Taylor series of functions in two or more variables are obtained by using vectors for **`x`** and point **`a`**. Here is a simple example.

```
(%i5) taylor(exp(x^2+y),[x,y],[0,1],2);
(%o5) /T/  %e + %e (y - 1) + (2%e x^2 + %e (y - 1)^2)/2 + ...
```

Maxima can also handle infinite Taylor series. Command **`powerseries`** creates the infinite sum.

```
(%i6) powerseries(sin(x),x,0);
```
(%o6) $\sum_{i1=0}^{\infty} \frac{(-1)^{i1} x^{2i1+1}}{(2i1+1)!}$

```
(%i7) niceindices(%);
```
(%o7) $\sum_{i=0}^{\infty} \frac{(-1)^{i} x^{2i+1}}{(2i+1)!}$

Maxima automatically uses the indices **`i1`**, **`i2`**, ... Command **`niceindices`** replaces these to the more familiar symbols **`i`**, **`j`**, ...

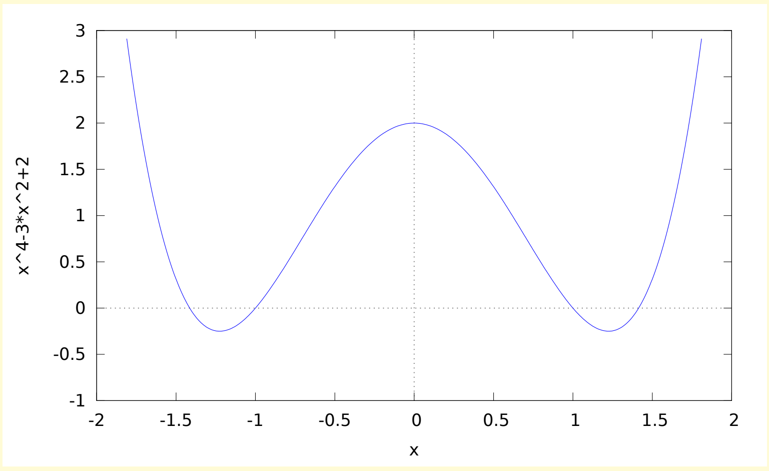
7.7 Minima and Maxima of Univariate Functions

Extrema of a univariate function f can be found by the following well-known method:

1. Find the critical points of f, i.e., points x^* with $f'(x^*) = 0$.
2. Compute the second derivative f'' and check its sign at these critical points.
   - If $f''(x^*) > 0$, then x^* is a local minimum.
   - If $f''(x^*) < 0$, then x^* is a local maximum.
   - If $f''(x^*) = 0$, then we need higher order derivatives at x^* for a decision.

Assume we have function $f(x) = x^4 - 3x^2 + 2$ and need to compute all local extrema. It is always a good idea to plot the graph of the function.

```
(%i1) f(x):= x^4-3*x^2+2;
```
(%o1) $f(x) := x^4 - 3x^2 + 2$

```
(%i2) plot2d(f(x), [x,-2,2], [y,-1,3]);
(%o2)
```


In order to find all critical points we have to compute the first derivative f' and find all its roots.

```
(%i3) fx: diff(f(x),x);
```
(%o3) $4x^3 - 6x$

```
(%i4) crit: solve(fx=0,x);
```
(%o4) $[x = -\frac{\sqrt{3}}{\sqrt{2}}, x = \frac{\sqrt{3}}{\sqrt{2}}, x = 0]$

Next we have to evaluate the second derivative f'' at all these critical points and check the signs of the results.

```
(%i5) fxx: diff(f(x),x,2);
(%o5) 12 x^2 - 6
(%i6) fxx, crit[1];
(%o6) 12
(%i7) fxx, crit[2];
(%o7) 12
(%i8) fxx, crit[3];
(%o8) -3
```

Consequently, we have two local minima at $x_1 = -\frac{\sqrt{3}}{\sqrt{2}}$ and $x_2 = \frac{\sqrt{3}}{\sqrt{2}}$, and a local maximum at $x_3 = 0$.

7.8 Minima and Maxima of Multivariate Functions

Command	*Description*
`hessian(f,x)`	Hessian matrix of function `f` with respect to vector `x`

Extrema of a multivariate function f can be found by an analogous recipe:

1. Find the critical points of f, i.e., points $\mathbf{x}^*$ with $\nabla f(\mathbf{x}^*) = 0$.
2. Compute the Hessian matrix f'' and check its definiteness at these critical points.
   – If $f''(\mathbf{x}^*)$ is positive definite, then $\mathbf{x}^*$ is a local minimum.
   – If $f''(\mathbf{x}^*)$ is negative definite, then $\mathbf{x}^*$ is a local maximum.
   – If determinant $|f''(\mathbf{x}^*)| = 0$, then we need higher order derivatives at $\mathbf{x}^*$ for a decision.
   – Otherwise, $f''(\mathbf{x}^*)$ is indefinite and $\mathbf{x}^*$ is a saddle point.

The Hessian matrix of a function f consists of all second partial derivatives.

$$f''(x_1,\dots,x_n) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Command **`hessian`** computes the Hessian matrix for a given function.

```
(%i1) hessian(x^3+4*x*y+y^2,[x,y]);
      [6 x  4]
(%o1) [      ]
      [ 4   2]
```

Here is an example with symbolic derivatives.

```
(%i2) depends(g,[x,y])$
(%i3) hessian(g,[x,y]);
```

(%o3) $\begin{bmatrix} \frac{d^2}{dx^2}g & \frac{d^2}{dx\,dy}g \\ \frac{d^2}{dx\,dy}g & \frac{d^2}{dy^2}g \end{bmatrix}$

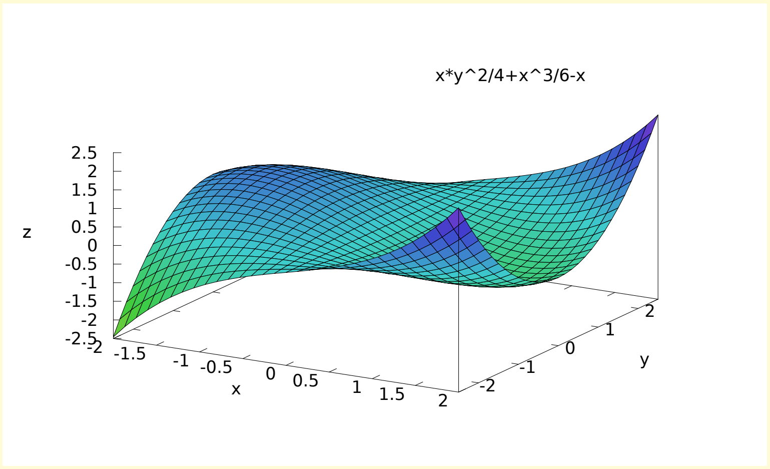
Computing local extrema is much more difficult for functions with two or more variables than for univariate functions. So consider the following function.

```
(%i4) f(x,y):=1/6*x^3 - x + 1/4*x*y^2;
```

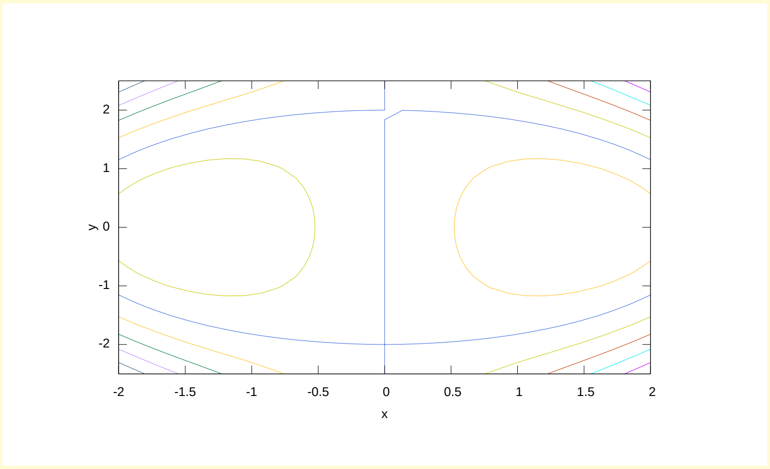
(%o4) $f(x,y) := \frac{1}{6}x^3 - x + \frac{1}{4}x\,y^2$

Again it is a good idea to plot the graph of the function. (Use the mouse to rotate the graph and get views from different viewpoints.)

```
(%i5) plot3d(f(x,y), [x,-2,2], [y,-2.5,2.5]);
(%o5)
```



```
(%i6) contour_plot(f(x,y), [x,-2,2], [y,-2.5,2.5], [legend,false],
      [gnuplot_preamble, "set cntrparam levels 12"]);
(%o6)
```



First we have to find the critical points of f. Thus we compute the gradient of f and have to solve a system of equations.

```
(%i7) [fx,fy]: [diff(f(x,y),x), diff(f(x,y),y)];
```
(%o7) $[\frac{y^2}{4}+\frac{x^2}{2}-1, \frac{x\,y}{2}]$
```
(%i8) crit: solve([fx=0,fy=0],[x,y]);
```
(%o8) $[[x=-\sqrt{2}, y=0], [x=\sqrt{2}, y=0], [x=0, y=2], [x=0, y=-2]]$

Next we need the Hessian matrix of f.

```
(%i9) hess: hessian(f(x,y),[x,y]);
```
(%o9) $\begin{bmatrix} x & \frac{y}{2} \\ \frac{y}{2} & \frac{x}{2} \end{bmatrix}$

Remind there are two method for estimating the definiteness of a symmetric matrix $\mathbf{A}$. The first looks at the eigenvalues of the matrix:

- $\mathbf{A}$ is positive definite $\Leftrightarrow$ all eigenvalues of $\mathbf{A}$ are positive.
- $\mathbf{A}$ is negative definite $\Leftrightarrow$ all eigenvalues of $\mathbf{A}$ are negative.
- $\mathbf{A}$ is indefinite $\Leftrightarrow$ $\mathbf{A}$ has positive and negative eigenvalues.

```
(%i10) H1: ev(hess, crit[1]);
```
(%o10) $\begin{bmatrix} -\sqrt{2} & 0 \\ 0 & -\frac{1}{\sqrt{2}} \end{bmatrix}$
```
(%i11) eigenvalues(H1);
```
(%o11) $[[-\frac{1}{\sqrt{2}}, -\sqrt{2}], [1,1]]$
```
(%i12) H3: ev(hess, crit[3]);
```
(%o12) $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
```
(%i13) eigenvalues(H3);
```
(%o13) $[[-1,1],[1,1]]$

Hence $\mathbf{x}_1 = (-\sqrt{2},0)'$ is a local maximum and $\mathbf{x}_3 = (0,2)'$ is a saddle point.

The second method makes use of the leading principle minors of a symmetric matrix $\mathbf{A}$. The k-th leading principle minor of $\mathbf{A}$ is the determinant of the left upper $k \times k$ submatrix.

$$M_k = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \dots & a_{kk} \end{vmatrix}$$

We then have

- $\mathbf{A}$ is positive definite $\Leftrightarrow$ $M_k > 0$ for all $k = 1, \dots n$.
- $\mathbf{A}$ is negative definite $\Leftrightarrow$ $(-1)^k M_k > 0$ for all $k = 1, \dots n$.
- $\mathbf{A}$ is indefinite $\Leftrightarrow$ $M_n = |\mathbf{A}| \neq 0$ but none of the above cases holds.

Notice that for a 2×2 matrix this reduces to

- **A** is positive definite $\Leftrightarrow a_{11} > 0$ and $|\mathbf{A}| > 0$.
- **A** is negative definite $\Leftrightarrow a_{11} < 0$ and $|\mathbf{A}| > 0$.
- **A** is indefinite $\Leftrightarrow |\mathbf{A}| < 0$.

```
(%i14) H2: ev(hess, crit[2]);
```
(%o14) $\begin{bmatrix} \sqrt{2} & 0 \\ 0 & \frac{1}{\sqrt{2}} \end{bmatrix}$
```
(%i15) [H2[1,1], determinant(H2)];
```
(%o15) $[\sqrt{2}, 1]$
```
(%i16) H4: ev(hess, crit[4]);
```
(%o16) $\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$
```
(%i17) [H4[1,1], determinant(H4)];
```
(%o17) $[0, -1]$

Hence $\mathbf{x}_2 = (\sqrt{2}, 0)'$ is a local minimum and $\mathbf{x}_4 = (0, -2)'$ is a saddle point.

7.9 Integration

Command	*Description*
`integrate(expr,x)`	Indefinite integral of **`expr`** with respect of **`x`**
`integrate(expr,x,a,b)`	Definite integral of **`expr`** with respect of **`x`** with limits **`a`** and **`b`**
`quad_qags(expr,x,a,b)`	Numerical approximation of integral over a finite interval

Command **`integrate`** attempts to symbolically compute the integral of a given expression. Thus a range of heuristics is employed to handle cases where closed form solutions are known. If one of these methods succeeds, then the integral is returned. Notice that **`integrate`** usually does not add an integration constant.

```
(%i1) integrate(sin(x)^3, x);
```
(%o1) $\frac{\cos(x)^3}{3} - \cos(x)$
```
(%i2) integrate(x/sqrt(b^2-x^2),x);
```
(%o2) $-\sqrt{b^2 - x^2}$
```
(%i3) integrate(exp(-x^2),x);
```
(%o3) $\frac{\sqrt{\pi}\, erf(x)}{2}$

The last example shows that the integral may contain functions names that are less common. In this case the so called *error function* is defined as $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$. It plays an important rôle in statistics. However, there might be even more obscure function that are not used in economics.

If **`integrate`** does not succeed, then the return value is the noun form of the integral or an expression containing one or more noun forms. The noun form is displayed with an integral sign.

```
(%i4) integrate(tan(log(x)),x);
```
(%o4) $\int \tan(\log(x)) dx$

`integrate` may need to know some property of a parameter in the integrand. Thus it will first consult the **`assume`** database (see Sect. 3.6), and, if the variable of interest is not there, **`integrate`** will ask the user. Depending on the question, suitable responses are **`yes;`** or **`no;`**, or **`pos;`**, **`zero;`**, or **`neg;`**.

```
(%i5) integrate(x^a,x);
      Is  a+1  zero or nonzero? zero;
(%o5) log(x)
```

We can avoid this question by adding the answer into our **`facts`**. Notice that we have to use **`equal`** rather then the symbol **`=`**. (Do not forget to **`forget`** of **`kill`** this assumption when it is not needed any more!)

```
(%i6) assume(equal(a,-1));
(%o6) [equal(a,−1)]
(%i7) integrate(x^a,x);
(%o7) log(x)
```

Command **`integrate`** also handles definite integrals. However, internally it is using an algorithm that works independently from that for indefinite integrals. Notice that the integral boundaries also may be **`minf`** and **`inf`**.

```
(%i8) integrate(cos(x)^2*exp(x),x,0,%pi);
```
(%o8) $\frac{3\%e^{\pi}}{5}-\frac{3}{5}$
```
(%i9) integrate(x^2*exp(-x^2),x,minf,inf);
```
(%o9) $\frac{\sqrt{\pi}}{2}$

The *QUADPACK* package provides a couple of functions for numerical approximations of definite integrals. We give a simple example with routine **`quad_qags`**. It returns a list of four elements:

- an approximation to the integral,
- the estimated absolute error of the approximation,
- the number integrand evaluations,
- an error code which is **`0`** if no problems were encountered.

```
(%i10) quad_qags(x^(1/2)*log(1/x),x,0,1);
```
(%o10) $[.4444444444444448, 1.110223024625157\ 10^{-15}, 315, 0]$

Please see the *Maxima* manual for details.

— Exercises

38. Compute the following limits:

$\lim\limits_{n\to\infty}\frac{n+1}{n-1}$, $\lim\limits_{n\to\infty}\left(1+\frac{1}{n}\right)^n$, $\lim\limits_{x\to0}\frac{\sin(x)}{x}$, $\lim\limits_{x\to0}\frac{\sin(\frac{1}{x})}{x}$, $\lim\limits_{x\to0}\sin\left(\frac{1}{x}\right)$, $\lim\limits_{x\to0}x\,\sin\left(\frac{1}{x}\right)$,
$\lim\limits_{x\to0}\exp\left(\frac{1}{x}\right)$, $\lim\limits_{x\to0^+}\exp\left(\frac{1}{x}\right)$, $\lim\limits_{x\to0^-}\exp\left(\frac{1}{x}\right)$.

39. Compute the first derivative of $f(x)=x^5$ as the limit of the difference quotient.

40. Compute and simplify $\sum\limits_{k=0}^{n}\binom{n}{k}$, where $\binom{n}{k}$ denotes the binomial coefficients.
(Use `apropos` to find a function that computes $\binom{n}{k}$.)

41. Compute $\sum\limits_{i=1}^{n}(a_i b_{n-i+1}-a_{n-i+1}b_i)$ for $n=10,10^2,10^3,10^4,10^5,10^6$.

42. Create a polynomial with roots $1,2,3,\dots,999,1000$.
(Compute the product of all linear factor.)

43. Compute all partial derivatives of first and second order of $f(x,y)=x^\alpha\,y^\beta$.

44. Compute the total derivative of $\log(f(x,y))$.

45. We are given a function $f(x,y,t)$. Assume that $y=y(t)$ and $x=x(y)$.
Compute $\frac{d}{dt}f(x,y,t)$ and $\frac{d^2}{dt^2}f(x,y,t)$. Display the derivatives both in Leibniz notation and as subscripts.

46. Compute the Jacobian matrix of the function $\mathbf{f}$ given by
$(f_1(x,y),f_2(x,y))'=(exp(-x^2-y^2),x^2+y^2)'$.

47. Compute the Taylor series of $\sqrt{1+x^2}$ about the point $a=0$.

48. Compute the Taylor polynomial of order 4 of $\sqrt{1+x^2}$ around the point $a=0$.
Plot both the function and the polynomial.

49. Compute the local extrema of $f(x)=\frac{x^2+1}{x}$. Plot the graph of the function.

50. Compute the local extrema of $f(x,y)=-x^2+xy+y^2$.
Plot the graph of the function near its extrema.

51. Compute the local extrema of $g(x,y)=2(y-x^2)^2+(1-x)^2$.
Plot the graph of the function near its extrema.

52. Compute the local extrema of $h(x,y)=\frac{1}{x}\ln(x)-y^2+1$.
Plot the graph of the function near its extrema.

53. Compute $\int x^3\,\sin(-x)dx$.

54. Compute $\int_0^\pi x^3\,\sin(-x)dx$.

55. Compute $\int_1^2\tan(\ln(x))dx$ numerically. Store the numerical result in variable **a**.

8 Ordinary Differential Equations (ODE)

8.1 Analytic Solutions of Ordinary Differential Equations

Command	*Description*
`ode2(eqn,dvar,ivar)`	Solves ODE of first and second order
`ic1(sol,xval,yval)`	Solves initial value problems of first order
`ic2(sol,xval,yval,dval)`	Solves initial value problems of second order

Constant	*Description*
`%c`	integration constant for first order ODEs
`%k1, %k2`	integration constants for second order ODEs

System Variable	*Default*	*Description*
`method`		Shows successful solution method

An *ordinary differential equation* is an equation where the unknown is not a number but a function. First-order differential equations contain first derivatives and can be written as

$$y' = f(x,y) \ .$$

Second-order differential equations contain first and second derivatives:

$$y'' = f(x,y,y') \ .$$

An example for a first-order derivative is

$$x^2\frac{dy}{dx} + 3yx = \frac{sin(x)}{x}$$

There are two ways to represent ordinary differential equations in *Maxima*. The simplest way is to represent the derivatives by **`'diff(y,x)`** and **`'diff(y,x,2)`**, respectively. The above ordinary differential equation would then be entered as

```
(%i1) x^2*'diff(y,x) + 3*y*x = sin(x)/x;
```
(%o1) $x^2\left(\frac{d}{dx}y\right) + 3yx = \frac{sin(x)}{x}$

Note that the derivative **`'diff(y,x)`** is quoted, to prevent it from being evaluated (to 0). The second way is to write $y(x)$ explicitly as a function of x. The above equation would then be entered as

```
(%i2) x^2*diff(y(x),x) + 3*y(x)*x = sin(x)/x;
```
(%o2) $x^2\left(\frac{d}{dx}y(x)\right) + 3y(x)x = \frac{sin(x)}{x}$

When we solve such an equation we have to find an appropriate function $y(x)$. Routine **ode2** tries to solve first and second-order ODEs. It takes three arguments: an ODE given by **eqn**, the dependent variable **dvar**, and the independent variable **ivar**. The function tries various integration methods to find an analytic solution. When successful, it returns either an explicit or implicit solution for the dependent variable. **%c** is used to represent the integration constant in the case of first-order equations. If routine **ode2** cannot obtain a solution for whatever reason, it returns **false**.

```
(%i3) ode2(x^2*'diff(y,x) + 3*y*x = sin(x)/x, y, x);
```
(%o3) $y = \frac{\%c - cos(x)}{x^3}$

Routine **ode2** stores information about the method used to solve the differential equations in variable **method**. The above differential equation has been recognized as a linear ODE.

```
(%i4) method;
```
(%o4) $linear$

In the case where $y(x)$ is given, then the independent variable must be given as function **y(x)**.

```
(%i5) ode2(x^2*diff(y(x),x) + 3*y(x)*x = sin(x)/x, y(x), x);
```
(%o5) $y(x) = \frac{\%c - cos(x)}{x^3}$

Initial value problems of first order can be solved by routine **ic1**. It takes a general solution to the equation as found by **ode2** as its first argument. The second argument **xval** gives an initial value for the independent in the form **x=x0** and the third argument **yval** gives the initial value for the dependent variable in the form **y=y0**.

Thus we get the solution of the initial value problem

$$x^2\frac{dy}{dx} + 3yx = \frac{sin(x)}{x}\,, \qquad y(\pi) = 0$$

in the following way:

```
(%i6) sol1: ode2(x^2*'diff(y,x) + 3*y*x = sin(x)/x, y, x);
```
(%o6) $y = \frac{\%c - cos(x)}{x^3}$
```
(%i7) ic1(sol1, x=%pi, y=0);
```
(%o7) $y = -\frac{cos(x) + 1}{x^3}$

Similarly we can use **ode2** to solve second-order differential equations. In this case the two integration constants are represented by **%k1** and **%k2**.

```
(%i8) sol2: ode2('diff(y,x,2) + y*'diff(y,x)^3 = 0, y, x);
```
(%o8) $\frac{y^3 + 6 * \%k1 * y}{6} = x + \%k2$

The corresponding initial value problem can be solved by means of **ic2**. Compared to **ic1** is has an additional forth argument **dval** that gives the initial value of the first derivative of the dependent variable with respect to the independent variable in the form **diff(y,x)=dy0**.

```
(%i9) ic2(sol2, x=0, y=0, 'diff(y,x)=2);
```
(%o9) $\frac{y^3 + 3 * y}{6} = x$

8.2 Linear Systems

Command	*Description*
`desolve(eqn, y)`	Solves linear ODE
`desolve([eqn_1,...,eqn_n],[y_1,..,y_n])`	Solves system of linear ODEs
`atvalue(expr,[x_1=a_1,...],c)`	Assigns value **`c`** to **`expr`** at point **`x=a`**

(Systems) of linear equations can be found by routine **`desolve`**. It is important that the functions $y_i(x)$ must be given in functional form. The solution of the linear second-order differential equation

$$y''(x) + y(x) = 2x$$

can be found as follows:

```
(%i1) desolve(diff(y(x),x,2)+y(x)=2*x, y(x));
```
(%o1) $y(x) = sin(x)\left(\frac{d}{dx}y(x)\Big|_{x=0} - 2\right) + y(0)\,cos(x) + 2\,x$

If **`desolve`** cannot find a solution to the given ODE, then **`false`** is returned.

If initial conditions at $x = 0$ are known, they can be supplied by using **`atvalue`**. Notice that the conditions can only be given at 0, and must be given before the equations are solved. It is recommended to remove your (global) assignment immediately after solving the initial value problem as otherwise it may cause obscure error messages or (even worse) wrong results.

```
(%i2) atvalue(y(x), x=0, 1)$
(%i3) atvalue(diff(y(x),x), x=0, 2)$
(%i4) desolve(diff(y(x),x,2)+y(x)=2*x, y(x));
```
(%o4) $y(x) = cos(x) + 2x$
```
(%i5) kill(y)$
```

Similarly systems of linear ODEs can be solved.

$$\begin{aligned} x'(t) &= x(t) - 2y(t) \\ y'(t) &= -x(t) + y(t) \end{aligned} \qquad (*)$$

```
(%i6) eqn1: diff(x(t),t)=x(t)-2*y(t);
```
(%o6) $\frac{d}{dt}x(t) = x(t) - 2y(t)$
```
(%i7) eqn2: diff(y(t),t)=-x(t)+y(t);
```
(%o7) $\frac{d}{dt}y(t) = y(t) - x(t)$
```
(%i8) desolve([eqn1,eqn2],[x(t),y(t)]);
```
(%o8) $[x(t) = \%e^t\left(\frac{(2(-2y(0)-x(0))+2x(0))sinh(\sqrt{2}t)}{2^{3/2}} + x(0)cosh(\sqrt{2}t)\right),$
$y(t) = \%e^t\left(\frac{(2y(0)+2(-y(0)-x(0)))sinh(\sqrt{2}t)}{2^{3/2}} + y(0)cosh(\sqrt{2}t)\right)]$

The initial value problem with $x(0) = 1$ and $y(0) = 1$ is then solved by

```
 (%i9) atvalue(x(t), t=0, 1)$
(%i10) atvalue(y(t), t=0, 1)$
(%i11) desolve([eqn1,eqn2],[x(t),y(t)]);
```
(%o11) $$[x(t) = \%e^t(cosh(\sqrt{2}t) - \sqrt{2}sinh(\sqrt{2}t)), \quad y(t) = \%e^t(cosh(\sqrt{2}t) - \frac{sinh(\sqrt{2}t)}{\sqrt{2}})]$$

Routine **desolve** used the so called Laplace transform to solve the given ODE. Thus the returned solution may look weird and is hardly useful. This is in particular the case when the coefficients of the linear differential equations are not constants.

8.3 Direction Fields

Command	*Description*
plotdf(dxdy,[x,y],opts)	Plot direction field of a single ODE
plotdf([dxdt,dydt],[x,y],opts)	Plot direction field of a set of two autonomous ODEs

Option	*Default*	*Description*
[trajectory_at,x,y]	empty	Starting point of an integral curve
[direction,dir]	**both**	direction of the independent variable to compute an integral curve (**forward**, **backward**, **both**)
[versus_t,num]	**0**	If **1**, curves are also plot as function of t
[x,x_min,x_max]	automatic	Range for x-axis
[y,y_min,y_max]	automatic	Range for y-axis

plotdf creates a plot of a vector field (direction field) of a first-order ODE or a system of two first-order ODEs. The directions field is a graphical representation of the solutions of an ODE.

8.3.1 Direction Fields of a Single First-Order ODE

To plot the direction field of a single ODE, the ODE must be written in the form

$$\frac{dy}{dx} = F(x,y)$$

and the function F should be given as the first argument for **plotdf**. The name of the independent and the dependent variable are given as a list as second argument. If these are called **x** and **y**, respectively, it may be omitted.

When called **plotdf** opens a new window that contains a the direction plot. The length of the draw vectors are proportional to the absolute value of the first derivative.

A direction field for the differential equation

$$y' = \exp(-x) + y$$

is created by the following *Maxima* code. Notice that it might be necessary to load package **plotdf** first.

```
 (%i1) load(plotdf)$
 (%i2) plotdf(exp(-x)+y)$
```

The resulting plot is shown in Fig. 8.1, l.h.s. The menu in the plot window has several options (not shown in the figure as its appearance heavily depends on computing environment) that allows to manipulate the plot. Clicking into the direction field on particular point adds a trajectory starting at that point, see Fig. 8.1, r.h.s. It is possible to draw two or more trajectories with different starting points. One can also enter staring points by means of option **trajectory_at**.

```
(%i3) plotdf(exp(-x)+y, [trajectory_at,2,-0.1])$
```

Option **direction** controls the direction of the independent variable that will be followed to compute an integral curve. Possible values are **forward**, to make the independent variable increase, **backward**, to make the independent variable decrease, or **both** that will lead to an integral curve that extends in both directions.

It is also possible to plot the integral curve as a function of the independent variable *t* in a second plot window. This can be switched on using the appropriate option in the menu of the plot window or by means of command line option **versus_t**.

For further options we refer to the corresponding man page.

Figure 8.2 shows the result of the next *Maxima* command.

```
(%i4) plotdf(exp(-x)+y, [trajectory_at,2,-0.1], [direction,forward],
       [versus_t,1])$
```

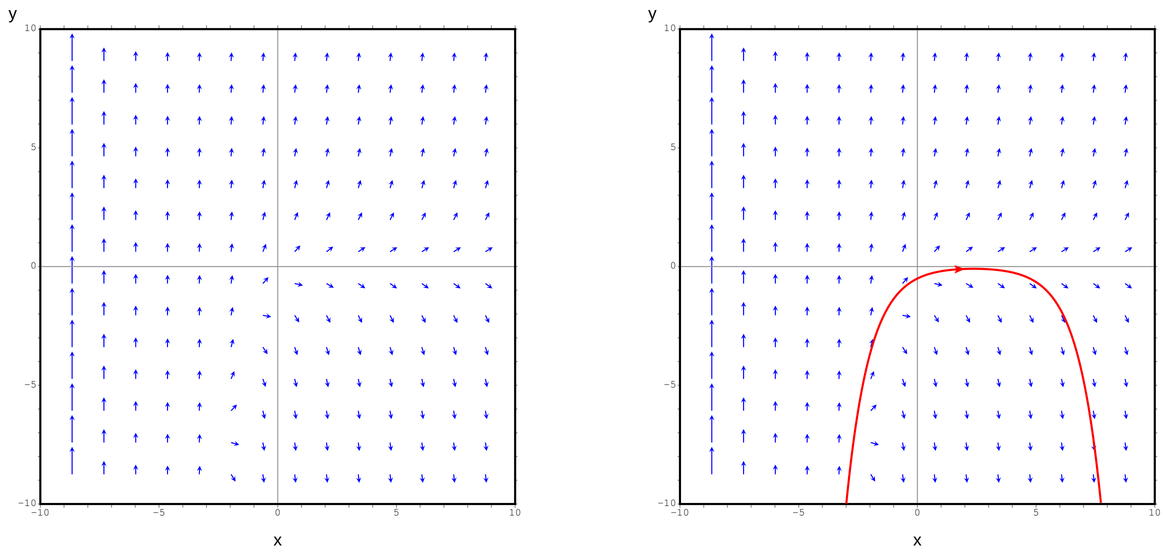


Figure 8.1: Direction field without (l.h.s.) and with trajectory (r.h.s.).

8.3.2 Direction Fields of Two Autonomous First-Order ODEs

To plot the direction field of a set of two autonomous ODEs, they must be written in the form

$$\frac{dx}{dt} = F(x,y), \quad \frac{dy}{dt} = G(x,y)$$

and the the first argument for **plotdf** should be a list with the two functions F and G, in that order; namely, the first expression in the list will be taken to be the time derivative of the variable represented on the horizontal axis, and the second expression will be the time derivative of the variable represented on the vertical axis. Those two variables do not have to be x and y, but if they

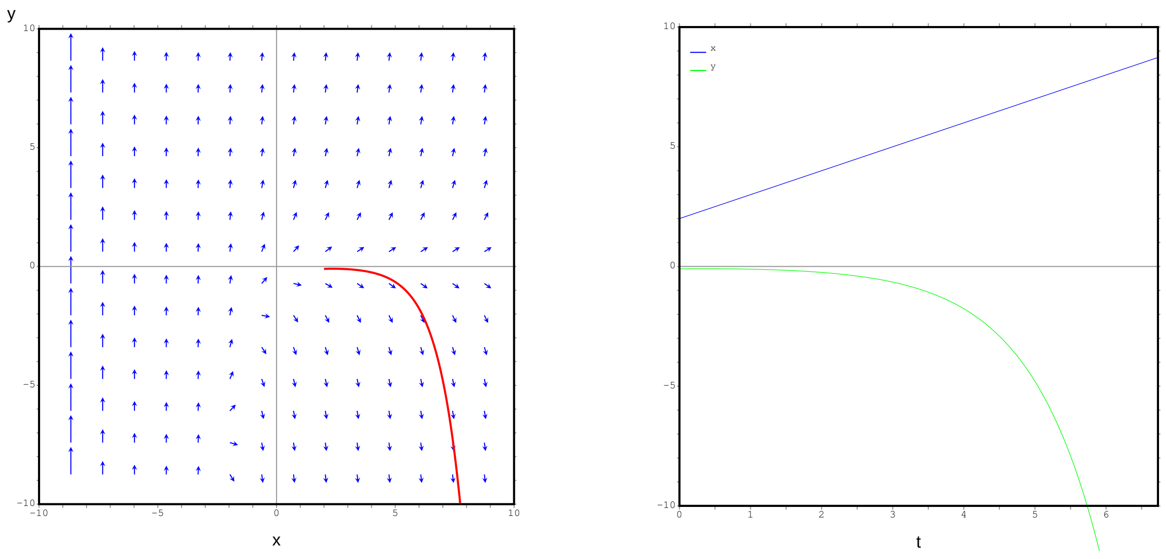


Figure 8.2: Direction field with trajectory starting at $x = 2$ and $y = -0.1$ (l.h.s.) and the plot against independent variable t (r.h.s.).

are not, then the second argument given to **`plotdf`** must be another list naming the two variables, first the one on the horizontal axis and then the one on the vertical axis.

Here is the code plotting the directory field for the system ($*$) in Sect. 8.2 on p. 91 above; see Fig. 8.3 for the result.

```
(%i5) plotdf([x-2*y, -x+y], [trajectory_at,1,1], [versus_t,1])$
```

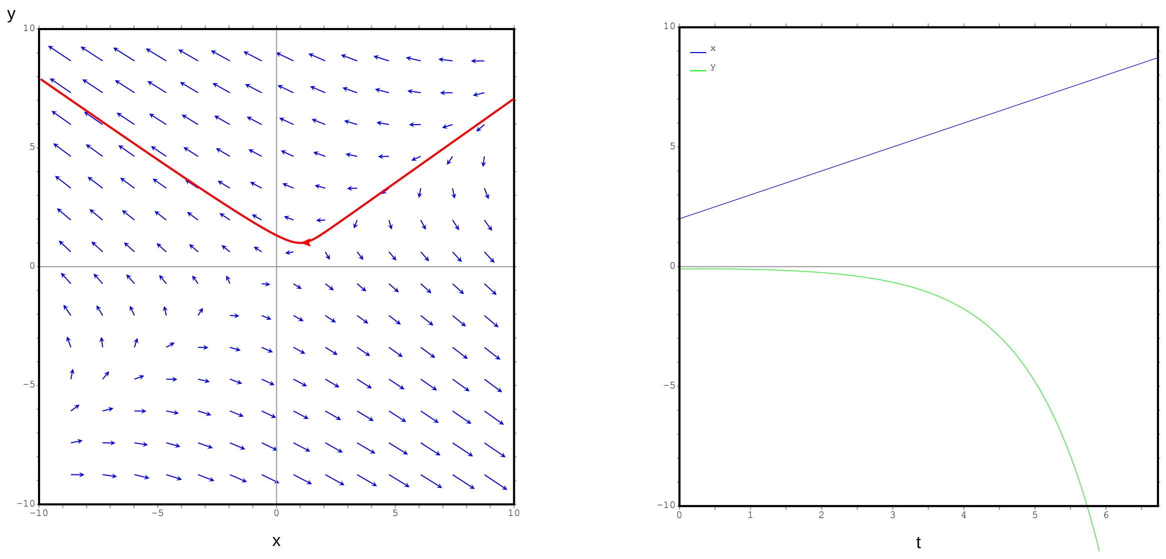


Figure 8.3: Direction field of two autonomous ODEs with trajectory starting at $x = 1$ and $y = 1$ (l.h.s.) and the plot against independent variable t (r.h.s.).

— Exercises

56. Compute the general solutions of the following ordinary differential equations by means of routine **`ode2`** as well as the corresponding initial value problems with initial values $y(1) = 1$. Determine the methods that are used to solve these ODEs.

(a) $y' - k\frac{y}{x} = 0$

(b) $x\,y' - (1 + y) = 0$

(c) $y' = x\,y$

(d) $y' + e^y = 0$

(e) $y' = y^2$

(f) $y' = \sqrt{x^3\,y}$

57. Compute the general solutions of the following ordinary differential equations by means of routine **`ode2`** as well as the corresponding initial value problems with the given initial values. Determine the methods that are used to solve these ODEs.

(a) $y'' + y' - 2y = 3$ with $y(0) = y'(0) = 1$.

(b) $y'' - 6y' + 9y = 0$ with $y(0) = 2$ and $y'(0) = 0$.

(c) $y'' + 2y' + 17y = 0$ with $y(0) = 0$ and $y'(1) = 1$.

58. Compute the general solutions of the following ordinary differential equations by means of routine **`desolve`** as well as the corresponding initial value problems with the given initial values.

(a) $x\,y' - (1 + y) = 0$ with $y(0) = 1$.

(b) $y'' + y' - 2y = 3$ with $y(0) = y'(0) = 1$.

(c) $y'' - 6y' + 9y = 0$ with $y(0) = 2$ and $y'(0) = 0$.

(d) $y'' + 2y' + 17y = 0$ with $y(0) = 0$ and $y'(1) = 1$.

59. Plot the direction fields and some trajectories for the following ordinary differential equations.

(a) $y' - \frac{y}{x} = 0$

(b) $x\,y' - (1 + y) = 0$

(c) $y' = x\,y$

(d) $y' + e^y = 0$

(e) $y' = y^2$

(f) $y' = \sqrt{x^3\,y}$ (Hint: use options **`x`** and **`y`**.)

60. Plot the direction fields and some trajectories for the following system of ordinary differential equations. Use plotting range $[0, 10] \times [0, 10]$.

$$\begin{aligned} x'(t) &= x(t)(1 - y(t)) \\ y'(t) &= -y(t)(1 - x(t)) \end{aligned}$$

9 Economic Examples

9.1 Monopoly

Assume that a local provider has a monopoly over the provision of rock concerts. The firm has estimated that its market demand curve can be drawn from the following equation

$$p = 124 - 6q\,.$$

The marginal revenue (MR) in this case would be $MR = 124 - 12q$.

```
(%i1) p(q):= 124-6*q;
(%o1) p(q) := 124 − 6 q
(%i2) MR(q):= ''(diff(q*p(q),q));
(%o2) MR(q) := 124 − 12 q
```

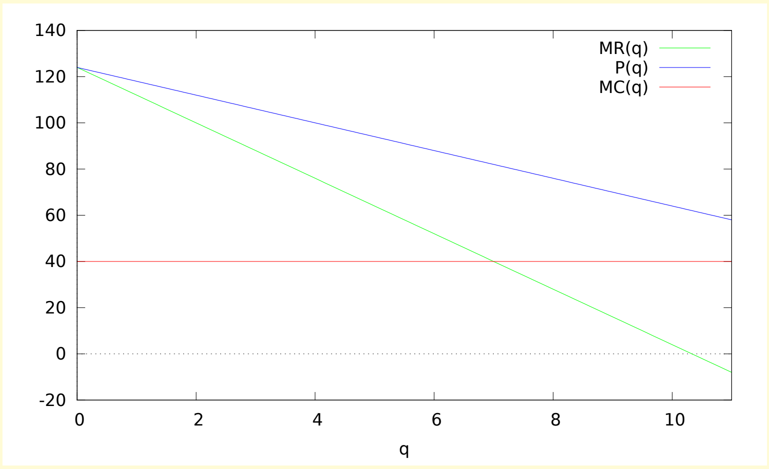
The firm's average and marginal cost are constant, in our case let $MC = AC = 40$. The solution in the single price case is found by setting $MR = MC$.

```
(%i3) MC(q):= 40;
(%o3) MC(q) := 40
(%i4) eq1: MR(q)=MC(q);
(%o4) 124 − 12 q = 40
(%i5) q0: ev(q, solve(eq1,q));
(%o5) 7
```

So we find $\hat{q} = 7$. The computation of the optimal price $\hat{p}$ and the optimal single price monopoly is then straightforward.

```
(%i6) p0: p(q0);
(%o6) 82
(%i7) pi(p,q):= p*q-40*q;
(%o7) π (p,q) := p q − 40 q
(%i8) pi(p0,q0);
(%o8) 294
```

```
(%i9) plot2d([MR(q),p(q),MC(q)],[q,0,11],
       [color,green,blue,red], [legend, "MR(q)","p(q)","MC(q)"]);
(%o9)
```



9.2 Profit Maximization

The notion of optimization of a function is central to economic analysis of rational behaviour. For example, firms are usually assumed to want to choose an output level that maximises profit and minimizes the cost. We solve like above an another problem of a monopolist who wants to maximize his output.

Assume a monopoly faces a demand function given by

$$p(q) = 50 - \frac{1}{2}q$$

and a total cost function of

$$TC(q) = 2 + 60\,q - 8\,q^2 + q^3\,.$$

In order to find the maximum profit output we first need to derive the total revenue function TR and the total profit function TP in the usual way.

(%i1) `p(q):= 50-1/2*q;`

(%o1) $p(q) := 50 - \frac{1}{2}\,q$

(%i2) `TC(q):= 2+60*q-8*q^2+q^3;`

(%o2) $TC(q) := 2 + 60\,q + (-8)\,q^2 + q^3$

(%i3) `TR(q):= ''(expand(p(q)*q));`

(%o3) $TR(q) := 50\,q - \frac{q^2}{2}$

(%i4) `TP(q):=''(expand(TR(q)-TC(q)));`

(%o4) $TP(q) := -q^3 + \frac{15\,q^2}{2} - 10\,q - 2$

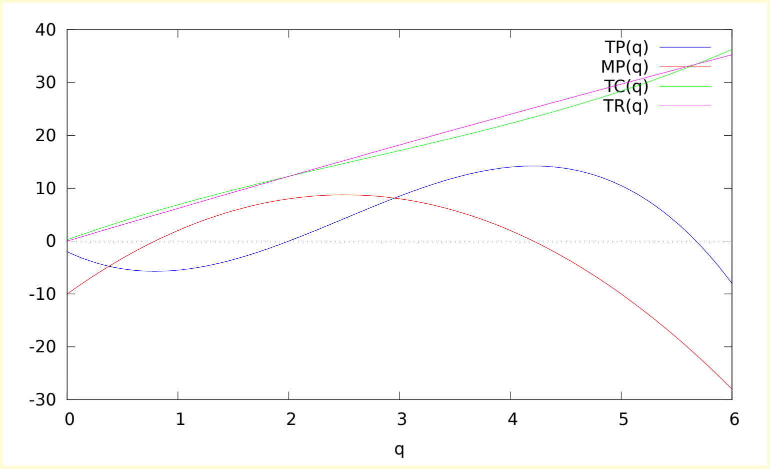
Since this function is not concave we apply the following method: Compute the candidate points for the (global) maximum and compare their total profits: the critical points of TP and the boundary points $q = 0$ (notice that q must satisfy the nonnegativity condition) and $q = \infty$ (that is, $\lim_{q\to\infty} TP(q)$). Notice that all critically points are feasible.

```
(%i5) cp: solve(diff(TP(q),q)=0,q);
```
$$(\%o5)\quad [q=-\frac{\sqrt{105}-15}{6}, q=\frac{\sqrt{105}+15}{6}]$$
```
(%i6) float(cp);
```
$$(\%o6)\quad [q=0.792174872340067, q=4.207825127659932]$$
```
(%i7) [TP(0),TP(rhs(cp[1])),TP(rhs(cp[2])),limit(TP(q),q,inf)], numer;
```
$$(\%o7)\quad [-2,-5.712313244682943,14.21231324468293,-inf]$$

Hence we have maximum profit $q^* = 4.2078$. By substitution we get maximum profit and optimal price.

```
(%i8) q0: rhs(cp[2]), numer;
(%o8) 4.207825127659932
(%i9) TP0: TP(q0);
(%o9) 14.21231324468293
(%i10) p0: p(q0);
(%o10) 47.89608743617003
```

At last we plot total profit and marginal profit as well as total revenue and total price (the later two are rescaled for the picture).

```
(%i11) plot2d([TP(q),diff(TP(q),q),TC(q)/8,TR(q)/8], [q,0,6],
       [legend, "TP(q)", "MP(q)", "TC(q)", "TR(q)"]);
(%o11)
```


9.3 Price Discrimination

Assume that a provider sales a good at two different prices

$$p_1(q_1) = 20 - 2\,q_1$$
$$p_2(q_2) = 10 - q_2$$

and has cost function

$$C(q_1, q_2) = (q_1 + q_2)^2 + 5$$

The provider wants to maximize its profit. One approach is to define two functions p_1 and p_2. However, here we want to demonstrate the possibility of vector valued functions in *Maxima*. Thus we need a vector $\mathbf{q} = (q_1, q_2)$ and a function $\mathbf{p}(\mathbf{q}) = \big(p_1(q_1, q_2),\ p_2(q_1, q_2)\big)$.

```
(%i1) q: [q[1], q[2]];
(%o1) [q1, q2]
(%i2) p(q):= [20-2*q[1], 10-q[2]];
(%o2) p(q) := [20 - 2 q1, 10 - q2]
(%i3) C(q):=(q[1]+q[2])^2+5;
(%o3) C(q) := (q1 + q2)^2 + 5
(%i4) TP(q):=p(q).q-C(q);
(%o4) TP(q) := p(q) · q - C(q)
```

In order to get the maximal profit we have to compute the stationary points of TP.

```
(%i5) MP(q):= ''([diff(TP(q),q[1]), diff(TP(q),q[2])]);
(%o5) MP(q) := [-2 (q2 + q1) - 4 q1 + 20, -2 (q2 + q1) - 2 q2 + 10]
(%i6) cq: solve(MP(q), q);
(%o6) [[q2 = 1, q1 = 3]]
```

May also may check whether the point indeed is a global maximum. Thus we show that TP is concave by computing all eigenvalues of its Hessian.

```
(%i7) hessian(TP(q), q);
       [-6  -2]
(%o7)  [-2  -4]
(%i8) float(eigenvalues(%));
(%o8) [[-7.23606797749979, -2.76393202250021], [1.0, 1.0]]
```

Hence we have optimal profit for $q^* = (3, 1)$.

```
(%i9) q0: ev(q, cq);
(%o9) [3, 1]
```

We also can easily verify that marginal revenues and marginal costs coincide.

```
(%i10) MR(q):= ''([diff((p(q)*q)[1],q[1]), diff((p(q)*q)[2],q[2])]);
(%o10) MR(q) := [20 - 4 q1, 10 - 2 q2]
(%i11) MC(q):= ''([diff(C(q),q[1]), diff(C(q),q[2])]);
(%o11) MC(q) := [2 (q2 + q1), 2 (q2 + q1)]
(%i12) MR(q0); MC(q0);
(%o12) [8, 8]
(%o13) [8, 8]
```

9.4 Cournot Duopoly

In the Cournot duopoly model we have a market with demand function

$$p(q_1,q_2) = a - b\,(q_1,q_2)$$

and two providers with respective cost functions

$$C_1(q_1) = d_1 + c_1\,q_1$$
$$C_2(q_2) = d_2 + c_2\,q_2$$

and profit functions

$$\pi_i(q_1,q_2) = p(q_1,q_2)\,q_i - C_i(q_i)\,,\quad i = 1,2$$

If we use vector valued function this can be coded as follows.

```
(%i1) q: [q[1],q[2]]$
(%i2) p(q):= a-b*(q[1]+q[2]);
(%o2) p(q) := a − b (q1 + q2)
(%i3) c:[c[1],c[2]]$  d:[d[1],d[2]]$
(%i4) C(q):= d+c*q;
(%o4) C(q) := d + c q
(%i5) pi(q):= p(q)*q - C(q);
(%o5) π(q) := p(q) q − C(q)
```

Each company tries to maxize its profit given the supply of its competitor. Thus we get the so called reaction function as

```
(%i6) R: [solve(diff(pi(q)[1],q[1]),q[1])[1],
      solve(diff(pi(q)[2],q[2]),q[2])[1]];
```

$$\text{(\%o6)}\quad [q_1 = -\frac{q_2\,b - a + c_1}{2\,b}, q_2 = -\frac{q_1\,b - a + c_2}{2\,b}]$$

Thus if each company will react on any change in supply of its opponent we only have equlibrium if both equalities $q_1 = -\frac{q_2\,b-a+c_1}{2\,b}$ and $q_2 = -\frac{q_1\,b-a+c_2}{2\,b}$ hold simultaniously. We then have reached a so called Nash equlibrium because none of the two companies can change his supply without decreasing its profit.

```
(%i7) solve(R,q);
```

$$\text{(\%o7)}\quad [[q_1 = \frac{a + c_2 - 2\,c_1}{3\,b}, q_2 = \frac{a - 2\,c_2 + c_1}{3\,b}]]$$

9.5 Cobb-Douglas Production Function

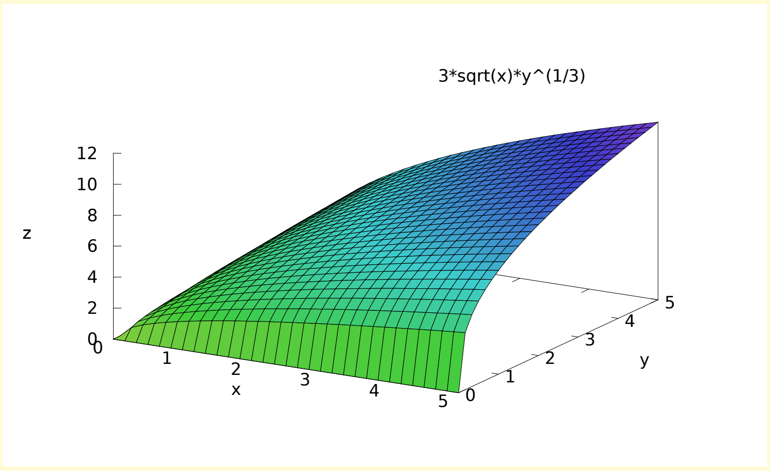
The Cobb Douglas function is widely used to represent the relationship of an output to inputs. The function was proposed by K. Wicksell and tested against statistical evidence by C. Cobb and P. Douglas. In general the function has the following form

$$u = c\,x^{\alpha}y^{\beta}$$

whereby u describes the total production, $x \geq 0$ the labor input, $y \geq 0$ the capital input and c the total factor productivity. The exponents $\alpha > 0$ and $\beta > 0$ are the output elasticities of labor and capital.

Here is example of an Cobb-Douglas function plot with $\alpha = \frac{1}{2}$, $\beta = \frac{1}{3}$ and constant $c = 3$.

```
(%i1) plot3d(3*x^(1/2)*y^(1/3), [x,0,5], [y,0,5]);
(%o1)
```



We can easily check that the Cobb-Douglas function is strictly concave whenever $\alpha + \beta < 1$.

```
(%i2) u(x,y):= c*x^alpha*y^beta;
```
(%o2) $u(x,y) := c\,x^{\alpha}\,y\,beta$
```
(%i3) hess: hessian(u(x,y), [x,y]);
```
(%o3) $\begin{bmatrix} (\alpha-1)\,\alpha\,c\,x^{\alpha-2}\,y^{\beta} & \alpha\,\beta\,c\,x^{\alpha-1}\,y^{\beta-1} \\ \alpha\,\beta\,c\,x^{\alpha-1}\,y^{\beta-1} & (\beta-1)\,\beta\,c\,x^{\alpha}\,y^{\beta-2} \end{bmatrix}$
```
(%i4) factor(determinant(hess));
```
(%o4) $-\alpha\,\beta\,(\beta+\alpha-1)\,c^2\,x^{2\,\alpha-2}\,y^{2\,\beta-2}$

Since $0y\alpha, \beta < 1$ and $0 < \alpha + \beta < 1$, we find for $x, y > 0$ for the leading principal minors:

$$\begin{array}{ll} (\alpha-1)\,\alpha\,c\,x^{\alpha-2}\,y^{\beta} & < 0 \\ -\alpha\,\beta\,(\beta+\alpha-1)\,c^2\,x^{2\,\alpha-2}\,y^{2\,\beta-2} & > 0 \end{array}$$

and thus $u(x,y)$ is strictly concave.

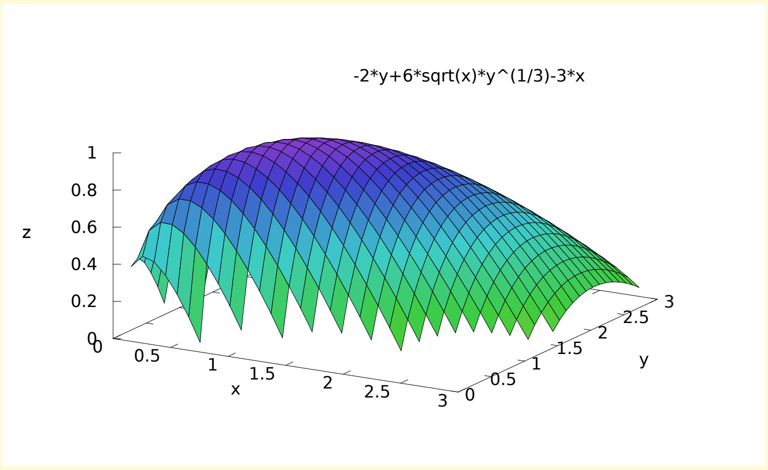
9.6 Stationary Points and Cobb-Douglas Function

Suppose we need the maximum of the function

$$f(x,y) = 6\,x^{\frac{1}{2}}\,y^{\frac{1}{3}} - 3\,x - 2\,y\,.$$

We already have seen in Sect. 9.5 that f is strictly concave, since $\alpha + \beta = \frac{5}{6} < 1$. Thus it remains to find stationary points.

```
(%i1) f(x,y):= 6*x^(1/2)*y^(1/3)-3*x-2*y;
```
(%o1) $f(x,y) := 6\,x^{1/2}\,y^{1/3} - 3\,x + (-2)\,y$
```
(%i2) plot3d(f(x,y), [x,0,3], [y,0,3], [z,0,1]);
```
(%o2)



```
(%i3) fx: diff(f(x,y), x);
```
(%o3) $\frac{3\,y^{1/3}}{\sqrt{x}} - 3$
```
(%i4) fy: diff(f(x,y), y);
```
(%o4) $\frac{2\,\sqrt{x}}{y^{2/3}} - 2$
```
(%i5) solve([fx=0,fy=0],[x,y]);
```
(%o5) $[\,]$

Unfortunately, **solve** does not find a solution. However, the following trick does the job. We tell *Maxima* that both varaibles are positive and solve one equation and substitute into the other one.

```
(%i6) assume(x>0,y>0);
```
(%o6) $[x > 0, y > 0]$
```
(%i7) solve(fx=0,x);
```
(%o7) $[x = y^{2/3}]$
```
(%i8) yo: solve(ev(fy,%)=0,y);
```
(%o8) $[y = 1]$
```
(%i9) xo: solve(ev(fx,yo)=0,x);
```
(%o9) $[x = 1]$

Thus the point $(1,1)$ maximizes f.

Now let us look at the general function

$$f(x,y) = c\,x^{\alpha}\,y^{\beta} - r\,x - w\,y$$

where $\alpha + \beta < 1$. Thus f is strictly concave and we only have to find the stationary points of f. However, if we want to solve both equalities simulatiously, we only get an error message.

```
(%i10) kill(all)$
(%i11) f(x,y):= c*x^a*y^b-r*x-w*y;
```
(%o11) $f(x,y) := c\,x^a\,y^b - r\,x + (-w)\,y$
```
(%i12) fx: diff(f(x,y), x);
```
(%o12) $a\,c\,x^{a-1}\,y^b - r$
```
(%i13) fy: diff(f(x,y), y);
```
(%o13) $b\,c\,x^a\,y^{b-1} - w$
```
(%i14) assume(x>0,y>0)$
(%i15) solve([fx=0,fy=0],[x,y]);
 algsys: tried and failed to reduce system to a polynomial in one
 variable; give up.
  -- an error. To debug this try: debugmode(true);
```

Thus let us try the same trick as above. However, we have to answer some more questions.

```
(%i16) solve(fx=0,x);
       Is  a  an integer? no;
       Is  a c r  positive, negative, or zero? pos;
```
(%o16)
$$[x = \frac{\left(\frac{r}{a\,c}\right)^{\frac{1}{a-1}}}{y^{\frac{b}{a-1}}}]$$
```
(%i17) yo: solve(ev(fy,%)=0,y);
       Is  (b+a-1)/(a-1) an integer? no;
       Is  a b c^2 r w positive, negative, or zero? pos;
```
(%o17)
$$[y = \left(\frac{w}{b\,c\left(\left(\frac{r}{a\,c}\right)^{\frac{1}{a-1}}\right)^a}\right)^{\frac{1}{b+a-1}-\frac{a}{b+a-1}}]$$
```
(%i18) xo: solve(ev(fx,yo)=0,x);
       Is  a^2 b c^3 r^2 w  positive, negative, or zero? pos;
```
(%o18)
$$[x = \left(\frac{r\left(\left(\frac{w}{b\,c\left(\left(\frac{r}{a\,c}\right)^{\frac{1}{a-1}}\right)^a}\right)^{\frac{1}{b+a-1}-\frac{a}{b+a-1}}\right)^b}{a\,c}\right)^{\frac{1}{a-1}}]$$

It works although the result is not very useful.

9.7 Constraint Optimization

Suppose we have to maximize a Cobb-Douglas production function

$$Y(K,L) = L^2K$$

under constraint

$$g(K,L) = K + L = 3.$$

Such problems can be solved by means of Lagrange multipliers. Thus we have to find the stationary points of the Lagrange function

$$\mathcal{L}(K,L,\lambda) = L^2K + \lambda(3 - L - K)$$

Although we can symbol **`L`** for both the Lagrange function $\mathcal{L}$ and the variable L (as *Maxima* distinguishes between functions and variables) it is recommended to use different symbols.

```
(%i1) Y(K,L):= K^2*L;
(%o1) K^2 L
(%i2) g(K,l):= K+L;
(%o2) K+L
(%i3) F(K,L,mu):= ''(Y(K,L)  + mu*(3-g(K,L)));
(%o3) F(L,K,μ) := K^2 L + μ(3 − K − L)
(%i4) gradF: jacobian([F(K,L,mu)],[K,L,mu])[1];
(%o4) [2 K L − μ, K^2 − μ, −L − K + 3]
(%i5) cp: solve(gradF);
(%o5) [[L = 1, K = 2, μ = 4], [L = 3, K = 0, μ = 0]]
```

Thus we have found two stationary points. For the next step we have to compute the bordered Hessian of $Y(K,L)$ in order to distinguishe between local maxima and local minima. Unfortunately, yet there is no function that computes the border Hessian and we have to do it ourselves. Recall that (in our case)

$$\bar{H} = \begin{bmatrix} 0 & g_K & g_L \\ g_K & \mathcal{L}_{KK} & \mathcal{L}_{KL} \\ g_l & \mathcal{L}_{LK} & \mathcal{L}_{LL} \end{bmatrix}$$

```
(%i6) H: matrix([0, diff(g(K,L),K), diff(g(K,L),L)],
      [diff(g(K,L),K), diff(F(K,L,mu),K,2), diff(F(K,L,mu),K,1,L,1)],
      [diff(g(K,L),L), diff(F(K,L,mu),L,1,K,1), diff(F(K,L,mu),L,2)]);
      [0   1    1 ]
(%o6) [1  2 L  2 K]
      [1  2 K   0 ]
(%i7) ev(determinant(H), cp[1]);
(%o7) 6
(%i8) ev(determinant(H), cp[2]);
(%o8) −6
(%i9) cp[1];
(%o9) [L = 1, K = 2, μ = 4]
```

Hence we have only one local maximum given by $L = 1$, $K = 2$ and $\mu = 4$.

Important
Recall that the *Maxima* code for second order mixed partial derivatives is
`diff(F(K,L), L,1, K,1)`.

Answers to the Exercises

Important
This chapter just lists *Maxima* code that helps to solve the given problems. By no means these are complete answers to all of the exercises. In particular in many cases the *Maxima* output needs further interpretation.

Also notice that there is usually more than one approach to solve a problem with *Maxima*. There may be much more ways to get a sensible result to the exercises than those listed in this chapter.

1.
```
gr: (1+sqrt(5))/2;
float(gr);
```
2.
```
gr: (1+sqrt(5))/2; ev(x^2-x-1,x=gr); %, numer;
```
3.
```
solve(x^2-x-1=0,x);
```
4.
```
(-1)^(1/3);
solve(x^3=-1,x);
```
5.
```
ev(bfloat(%pi), fpprec:100);
```
6.
```
ln: log;
```
7.
```
log10(x):= float(log(x)/log(10));
log10(1000);
```
8.
```
heron(a,b,c):= 0.25*sqrt((a^2+b^2+c^2)^2-2*(a^4+b^4+c^4));
```
9.
```
? solve;
```
10.
```
apropos("solve");
```
11.
```
reset(); kill(all);
```
12.
```
expand((a+b)^10);
```
13.
```
factor(a^10-b^10);
factor(a^10+b^10);
```
14.
```
ratsimp((a^10-b^10)/(a^2-b^2));
```
15.
```
solve(x^2+2*(sin(alpha)+cos(alpha))*x+sin(2*alpha)+1=0, x);
trigexpand(%); trigsimp(%);
unique(%);
```
16.
```
sgn(x):= if (x<0) then -1 else if (x=0) then 0 else 1;
sgn(-1); sgn(0); sgn(%pi);
```
17.
```
assume(alpha>0,beta>0,alpha<1,beta<1,alpha+beta<1);
is(alpha>-1/2);
is(alpha> 1/2);
is(alpha> 1);
forget(alpha>0,beta>0,alpha<1,beta<1,alpha+beta<1);
```

18.
```
numeric(x,prec):= ev(bfloat(x), fpprec:prec);
numeric(1/%e,30);
```

19.
```
plot2d(1/sqrt(2*%pi)*exp(-x^2/2),[x,-3,3],[ylabel,"density"]);
```

20.
```
plot2d([1/sqrt(2*%pi)*exp(-x^2/2), 1/(%pi*(1+x^2))], [x,-3,3],
[legend, "Gaussian distribution", "Cauchy distribution"], [style,
lines, [lines, 5]], [color, black, red], [ylabel,"density"]);
```

21.
```
plot2d([1/x,1/x^2,1,x,x^2],[x,0,2],[y,0,2]);
```

22.
```
datalist: [[.25,-1.03], [.5,-0.63], [.75,-0.28], [1.,0.],
[1.25,0.22], [1.5,0.38], [1.75,0.47]];
plot2d([log(x), [discrete, datalist]], [x,0,2], [y,-2,1], [style,
lines, points], [point_type, diamond]);
```

23.
```
plot2d([parametric, sin(4*%pi*t), cos(4*%pi*t)],
[t,0,4],[nticks,401]);
```

24.
```
/*a*/ plot3d(1/(2*%pi)*exp(-0.5*(x^2+y^2)), [x,-3,3], [y,-3,3]);
/*b*/ plot3d(1/(2*%pi)*exp(-0.5*(x^2+y^2)), [x,-3,3], [y,-3,3],
[palette,false], [color, blue]);
/*c*/ plot3d(1/(2*%pi)*exp(-0.5*(x^2+y^2)), [x,-3,3], [y,-3,3],
[palette,false], [color, cyan, blue]);
/*d*/ plot3d(1/(2*%pi)*exp(-0.5*(x^2+y^2)), [x,-3,3], [y,-3,3],
[mesh_lines_color,false]);
```

25.
```
load(draw)$
draw2d(ylabel="density", explicit(1/sqrt(2*%pi)*exp(-x^2/2),
x,-3,3));
```

26.
```
load(draw)$
draw2d(ylabel="density", key="Gaussian distribution", color=black,
explicit(1/sqrt(2*%pi)*exp(-x^2/2), x,-3,3), key="Cauchy
distribution", color=red, line_width=5, explicit(1/(%pi*(1+x^2)),
x,-3,3));
```

27.
```
load(draw)$
draw2d(yrange=[0,2], key="1/x", color=blue, explicit(1/x,
x,0,2), key="1/x^2", color=red, explicit(1/x^2, x,0,2), key="1",
color=green, explicit(1, x,0,2), key="x", color=magenta, explicit(x,
x,0,2), key="x^2", color=black, explicit(x^2, x,0,2));
```

28.
```
load(draw)$
datalist: [[.25,-1.03], [.5,-0.63], [.75,-0.28], [1.,0.],
[1.25,0.22], [1.5,0.38], [1.75,0.47]];
draw2d(yrange=[-2,1], color=blue, explicit(log(x), x,0,2),
color=red, point_type=filled_diamant, point_size=2,
points(datalist));
```

29.
```
load(draw)$
draw2d(nticks=29, parametric(sin(4*%pi*t),cos(4*%pi*t), t,0,4));
```

30.
```
load(draw)$
draw3d(explicit(1/(2*%pi)*exp(-0.5*(x^2+y^2)), x,-3,3, y,-3,3));
```

31.
```
x: [1,2,3]; y: [4,5,6];
x+y; x-y; x.y; 3*x;
```

32.
```
A: matrix([1,2,3],[4,5,6],[7,8,9]);
B: matrix([-1,0,2],[-2,1,3],[2,4,-1]);
A+B; A-B; 3*A; A.B; A^^2; B^^(-1);
determinant(A); transpose(A);
A^^(-1);
```
33.
```
A: matrix([1,2,3],[4,5,6],[7,8,9]);
B: matrix([-1,0,2],[-2,1,3],[2,4,-1]);
rank(A); rank(B);
nullity(A); nullity(B);
nullspace(A); nullspace(B);
columnspace(A); columnspace(B);
CS: columnspace(A); a1*first(CS)+a2*second(CS);
```
34.
```
"(a)"
eq1: 2*x[1]+3*x[2]+4*x[3]= 2;
eq2: 4*x[1]+3*x[2]+  x[3]=10;
eq3:   x[1]+2*x[2]+4*x[3]= 5;
solve([eq1,eq2,eq3], [x[1],x[2],x[3]]);
"(b)"
eq1: 2*x[1]+2*x[2]+  x[3]+3*x[4]=10;
eq2: 3*x[1]+5*x[2]+2*x[3]-  x[4]=30;
eq3:   x[1]+2*x[2]+  x[3]-  x[4]=12;
solve([eq1,eq2,eq3], [x[1],x[2],x[3],x[4]]);
"(c)"
eq1: 2*x[1]+10*x[2]+4*x[3]+ 9*x[4]= 1;
eq2:   x[1]+ 6*x[2]+5*x[3]+ 3*x[4]= 1;
eq3: 2*x[1]+16*x[2]+9*x[3]+11*x[4]=-1;
eq4:   x[1]+ 5*x[2]+2*x[3]+ 5*x[4]= 2;
eq5:            x[2]+3*x[3]        = 4;
solve([eq1,eq2,eq3,eq4,eq5], [x[1],x[2],x[3],x[4]]);
```
35.
```
A: matrix([1,2],[3,4]);
es: eigenvectors(A);
eigval: es[1][1];
v[1]: es[2][1];  v[2]: es[2][2];
```
36.
```
A: matrix([1,2,1],[2,1,3],[1,3,1]);
es: eigenvectors(A);
eigvals: float(realpart(es[1][1]));
eigvects: float(realpart(es[2]));
```

37.
```
A: diagmatrix(3,1);
esA: eigenvectors(A);
numberofeigenvaluesA:   length(esA[1][1]);
algebraicmultiplicityA: esA[1][2];
geometricmultiplicityA: length(esA[2][1]);
B: matrix([1,1,1],[0,1,1],[0,0,1]);
esB: eigenvectors(B);
numberofeigenvaluesB:   length(esB[1][1]);
algebraicmultiplicityB: esB[1][2];
geometricmultiplicityB: length(esB[2][1]);
```

38.
```
limit((n+1)/(n-1),n,inf);
limit((1+1/n)^n,n,inf);
limit(sin(x)/x,x,0);
limit(sin(1/x)/x,x,0);
limit(sin(1/x),x,0);
limit(sin(1/x)*x,x,0);
limit(exp(1/x),x,0);
limit(exp(1/x),x,0,plus);
limit(exp(1/x),x,0,minus);
```

39. `limit(((x+h)^5-x^5)/h,h,0);`

40. `sum(binomial(n,k),k,0,n), simpsum;`

41. `sum(a[i]*b[n-i+1]-a[n-i+1]*b[i],i,1,n), n:10;`

42. `product((x-y),y,1,1000);`

43.
```
diff(x^alpha*y^beta,x); diff(x^alpha*y^beta,y);
diff(x^alpha*y^beta,x,2); diff(x^alpha*y^beta,y,2);
diff(x^alpha*y^beta,x,1,y,1);
```

44. `diff(log(f(x)));`

45.
```
depends(f,[x,y,t]); depends(x,y); depends(y,t);
diff(f,t);
diff(f,t,2);
derivabbrev: true$
diff(f,t);
diff(f,t,2);
```

46. `jacobian([exp(-x^2-y^2), x^2+y^2],[x,y]);`

47. `powerseries(sqrt(1+x^2),x,0);`

48.
```
f(x):= sqrt(1+x^2);
t(x):= ''(taylor(f(x),x,0,4));
plot2d([f(x),t(x)], [x,-1.5,1.5]);
```

49. `f(x):= (x^2+1)/x;`
```
fx: diff(f(x),x);
fxx: diff(f(x),x,2);
crit: solve(fx,x);
ev(fxx, crit[1]);
ev(fxx, crit[2]);
plot2d(f(x), [x,-2,2],[y,-5,5]);
```
50. `f(x,y):= -x^2+x*y+y^2;`
```
fx: diff(f(x,y), x);
fy: diff(f(x,y), y);
crit: solve([fx,fy], [x,y]);
H: hessian(f(x,y), [x,y]);
eigenvalues(ev(H,crit[1]));
plot3d(f(x,y), [x,-1,1],[y,-1,1]);
```
51. `g(x,y):= 2*(y-x^2)^2+(1-x)^2;`
```
gx: diff(g(x,y), x);
gy: diff(g(x,y), y);
crit: solve([gx,gy], [x,y]);
H: hessian(g(x,y), [x,y]);
eigenvalues(ev(H,crit)), numer;
plot3d(g(x,y), [x,0.5,1.5],[y,0.5,1.5]);
```
52. `h(x,y):= 1/x*log(x)-y^2+1;`
```
hx: diff(h(x,y), x);
hy: diff(h(x,y), y);
crit: [solve(hx,x), solve(hy,y)];
H: hessian(h(x,y),[x,y]);
eigenvalues(ev(H,crit));
plot3d(h(x,y), [x,1,5],[y,-1,1]);
```
53. `integrate(x^3*sin(-x),x);`

54. `integrate(x^3*sin(-x),x,0,%pi);`

55. `qres: quad_qags(tan(log(x)),x,1,2);`
```
a: qres[1];
```
56. `sola: ode2('diff(y,x)-k*y/x=0, y, x); ic1(sola, x=1, y=1); method;`
```
solb: ode2(x*'diff(y,x)-(1+y)=0, y, x); ic1(solb, x=1, y=1); method;
solc: ode2('diff(y,x)=x*y, y, x); ic1(solc, x=1, y=1); method;
sold: ode2('diff(y,x)+%e^y=0, y, x); ic1(sold, x=1, y=1); method;
sole: ode2('diff(y,x)=y^2, y, x); ic1(sole, x=1, y=1); method;
solf: ode2('diff(y,x)=sqrt(x^3*y), y, x); ic1(solf, x=1, y=1);
method;
```

57.
```
sola: ode2('diff(y,x,2)+'diff(y,x)-2*y=3, y, x); method;
ic2(sola, x=0,y=1,'diff(y,x)=1);
solb: ode2('diff(y,x,2)-6*'diff(y,x)+9*y=0, y, x); method;
ic2(solb, x=0,y=2,'diff(y,x)=0);
solc: ode2('diff(y,x,2)+2*'diff(y,x)+17*y=0, y, x); method;
ic2(solc, x=0,y=0,'diff(y,x)=1);
```

58.
```
sola: desolve(diff(y(x),x)-(1+y(x))=0, y(x));
atvalue(y(x), x=0, 1); desolve(diff(y(x),x)-(1+y(x))=0, y(x));
kill(y)$
solb: desolve(diff(y(x),x,2)+diff(y(x),x)-2*y(x)=3, y(x));
atvalue(y(x), x=0, 1); atvalue(diff(y(x), x), x=0, 1);
desolve(diff(y(x),x,2)+diff(y(x),x)-2*y(x)=3, y(x));
kill(y)$
solc: desolve(diff(y(x),x,2)-6*diff(y(x),x)+9*y(x)=3, y(x));
atvalue(y(x), x=0, 2); atvalue(diff(y(x),x), x=0, 0);
desolve(diff(y(x),x,2)-6*diff(y(x),x)+9*y(x)=3, y(x));
kill(y)$
sold: desolve(diff(y(x),x,2)+2*diff(y(x),x)+17*y(x)=3, y(x));
atvalue(y(x), x=0, 0); atvalue(diff(y(x),x), x=0, 1);
desolve(diff(y(x),x,2)+2*diff(y(x),x)+17*y(x)=3, y(x));
kill(y)$
```

59.
```
/*a*/ plotdf(y/x);
/*b*/ plotdf((1+y)/x);
/*c*/ plotdf(y*x);
/*d*/ plotdf(-%e^y);
/*e*/ plotdf(y^2);
/*f*/ plotdf(sqrt(x^3*y),[x,0.001,10],[y,0.001,10]);
```

60.
```
plotdf([x*(1-y),-y*(1-x)],[x,0,3],[y,0,3]);
```

Index