Funded by
the European Union

Innovative Open Source courses for Computer Science

Part I

Mathematical Analysis supported by wxMaxima

Rudolf Blaško

2022

Contents

Chapter 1

Introduction to wxMaxima

Basic terms

wxMaxima is a document based interface for the computer algebra system Maxima. wxMaxima provides menus and dialogs for many common maxima commands, autocompletion, inline plots and simple animations. wxMaxima is distributed under the GPL license.

xMaxima is a graphical interface for Maxima, written in Tcl/Tk. It also provides the Open Source plotting program for Maxima, which can do some of the plots done by Maxima's default plotter (gnuplot) and a few more that gnuplot cannot do.

Maxima is one of the Open Source programs with open source code. The program can be compiled in various OS, including Windows, GNU/Linux and MacOS X. A precompiled program for GNU/Linux and Windows is available free of charge on the SourceForge website `https://sourceforge.net/projects/maxima/files/`.

After starting the wxMaxima environment, a menu window will appear on the screen at the top. Below the menu is a space where we can enter commands and where outputs appear.

```
(%i1)  First input line.
(%o1)  First output line.
(%i2)  Second input line.
(%o2)  Second output line.
```

We enter commands on separate lines (input lines), their execution is ensured by simultaneously pressing the Shift keys and Enter or by clicking on in the menu icon (Send the current cell to maxima). Input lines are listed with (%i1) and output lines are listed with (%o1). The numbers for the input line and the corresponding output line are identical and based on this number, we can refer to the content of these lines.

```
(%i1)  solve(0=x+2,x);
(%o1)  [x = -2]
```

```
(%i2)  %i1;
(%o2)  solve(0 = x + 2, x)
(%i3)  %o1;
(%o3)  [x = -2]
```

The commands are executed on new separate lines (output lines). Commands on input lines can be terminated with the symbol `;` (which the system will automatically complete) or the `$` symbol, which suppresses the display of the corresponding output.

We can enter more commands on the input line, but we must separate them symbols `;` or `$`. We can also structure the command on multiple input lines.

```
(%i1)  a:2;b:3;solve(a*x+b*x^2=0,x)
(a)    2
(b)    3
(%o1)  [x = -2/3, x = 0]
(%i2)  a:2$ b:3$ solve(a*x+b*x^2=0,x);
(%o2)  [x = -2/3, x = 0]
(%i3)  a:2$
       b:3$
       solve(a*x+b*x^2=0,x);
(%o3)  [x = -2/3, x = 0]
```

We can save the output in various shapes and then use it in other programs (LaTeX, MSWord equation editor, ...). Output (%o3) from the previous window we can:

- copy `Ctrl C` and `Ctrl V`, respectively copy as text (can be used eg. for MSWord equation editor): `x=-2/3,x=0`,
- copy as LaTeX `\[x=-\frac{2}{3}\operatorname{,}x=0\]`,
- copy as MathML, image, RTF, SVG...

The wxMaxima environment has a well-designed user help, which can be found in the `Help` menu. You can also open Help by pressing the `F1` key. The manual can be also find on the website `https://maxima.sourceforge.io/docs/manual/maxima_369.html`.

Basic Commands

The command `apropos` we can find out the exact name of the command using part of its name.

```
(%i1)  apropos("plot")
(%o1)  [barsplot,boxplot,contour_plot,get_plot_option,gnuplot,...
```

Command `describe` prints a description of the entered command.

```
(%i1)  describe(plot2d);
       -- Function: plot2d
       plot2d (<expr><,<range_x><,<options><)
       plot2d (<expr_<>=<expr_<>,<range_x><,<range_y><,<options><)
       plot2d ([parametric,<expr_x><,<expr><_y,<range><],<options><)
       plot2d ([discrete,<points><],<options><)
       plot2d ([contour,<expr><],<range_x><,<range_y><,<options><)
       plot2d ([<type_<>,...,<type_n><],<options><)
       There are 5 types of plots that can be plotted by 'plot2d':
            1. Explicit functions. 'plot2d' ...
       . . .
(%o1)  true
```

Expressions are entered using the usual characters of operations, sessions and functions. Arguments of functions and commands are in parentheses, multiplication symbol `*` must be entered! The exponentiation is specified by the character `^` or the pair `**`.

Symbol `:` is used to assign a value to the right of the expression to the left. The following commands solve the equation $2x + 3x^2 = 0$ with unknown variable x.

```
(%i1)  a:2$ b:3$ solve(a*x+b*x^2=0,x);
(%o1)  [x = -2/3, x = 0]
```

In the menu `View` and submenu `Display equations` we can change display output lines for shapes `in 2D`, `as 1D ASCII` or `as ASCII Art`.

The default display is `in 2D`. You can also change the output settings with the command `set_display`. Setting to shape `in 2D` has argument `none`.

```
(%i1)  x/sqrt(x^2+1);set_display('none)$
(%o1)  x/sqrt(x^2+1)              /* in 2D */
```

Using the `ascii` argument command `set_display` change the display output to the form `as 1D ASCII` and using the `xml` argument to form `as ASCII Art`.

```
(%i1)  x/sqrt(x^2+1);set_display('ascii)$
(%o1)  x/sqrt(x^2+1)      /* as 1D ASCII */
```

```
(%i2)  x/sqrt(x^2+1);set_display('xml)$
                x
(%o2)  ---------        /* as ASCII Art */
                 2
               sqrt(x + 1)
```

With the `kill` command we can remove variables with all their assignments and properties from memory.

```
(%i1)  kill(a,b)    /* removes all bindings from the arguments a,b */
(%i2)  kill(all)    /* removes all items on all infolists */
```

Working with Numbers and Basic Constants

Maxima can work with real numbers written in numerical or symbolic form. The way of writing real numbers can be set in the menu `Numeric` using the switch `Numeric Output` between numeric and symbolic display. Here we can also choose the method and accuracy of numerical display. The setting of the variable `numer` determines the method of displaying.

By default, 16 digits (including the decimal point) are displayed. The display accuracy is defined by the variable `fpproc` and affects the display using `bfloat`. Output `float` always displays the same. We can increase or decrease the accuracy practically indefinitely. We can change it globally and locally for only one variable or command.

```
(%i1)  log(2);
(%o1)  log(2)
(%i2)  log(2),numer;
(%o2)  0.6931471805599453
(%i3)  float(log(2));
(%o3)  0.6931471805599453
(%i4)  bfloat(log(2));
(%o4)  6.931471805599453b-1
(%i5)  log(2),bfloat;
(%o5)  6.931471805599453b-1
(%i6)  bfloat(log(2)),fpprec=34;
(%o6)  6.9314718055994530941723212145817766b-1
(%i6)  bfloat(log(2)),fpprec=134;
```

(%o6) $6.9314718055994530941723212145\,[78digits]\,10205706857336855202357581 3b-1$

Numeric constants e, π, i (imaginary unit) have the prefix `%`, i.e. `%e`, `%pi`, `%i`. This also applies to constants that are part of or the result of calculations. They also have the prefix `%`.

Maxima has predefined constants `inf`, `minf` for real infinite ∞, $-\infty$ and `infinity` for complex infinity.

Logical constants `true` and `false` they represent truth and untruth.

```
(%i1) %pi+%i+%e;
(%o1) %pi+%i+%e
(%i2) [minf,inf];
(%o2) [minf,inf]
(%i3) infinity;
(%o3) infinity
```

We do not deal with complex numbers in this course, so we will only mention how they are displayed. By default, complex numbers are entered in algebraic form (`rectform`). They can be converted to trigonometric (exponential) form using the command `polarform`.

```
(%i1) z:1+%i;
(z)   %i+1
(%i2) polarform(z)+rectform(z);
(%o2) sqrt(2)*%e^(%i*%pi/4)+%i+1
```

Assignments and Functions

The `:` operator we use to assign values or expressions to variables. We define functions using the assignment `:=`.

```
(%i1) f(x):=x^2+2*x+3;
(%o1) f(x):=x^2+2*x+3
(%i6) f(x);f(y);f(x+1);f(-2);f(1);
(%o2) x^2+2*x+3
(%o3) y^2+2*y+3
(%o4) (x+1)^2+2*(x+1)+3
(%o5) 3
(%o6) 6
```

Maxima contains many more functions than standard programming languages. These are not only the real functions themselves, but also various functions for their support. The basic functions include `sign(x)`, `abs(x)`, `floor(x)` (bottom whole of x) `round(x)` (rounded x to the nearest whole number), `truncate(x)` (removes all digits after the decimal point), `ceiling(x)` (upper integer x).

```
(%i2)  f(x):=sign(x)$
            print(f(-3.6),f(-3.2),f(-3),f(0),f(3),f(3.2),f(3.6))$
       neg neg neg zero pos pos pos
(%i4)  f(x):=abs(x)$
            print(f(-3.6),f(-3.2),f(-3),f(0),f(3),f(3.2),f(3.6))$
       3.6 3.2 3 0 3 3.2 3.6
(%i6)  f(x):=floor(x)$
            print(f(-3.6),f(-3.2),f(-3),f(0),f(3),f(3.2),f(3.6))$
       −4 −4 −3 0 3 3 3
(%i8)  f(x):=round(x)$
            print(f(-3.6),f(-3.2),f(-3),f(0),f(3),f(3.2),f(3.6))$
       −4 −3 −3 0 3 3 4
(%i10) f(x):=truncate(x)$
            print(f(-3.6),f(-3.2),f(-3),f(0),f(3),f(3.2),f(3.6))$
       −3 −3 −3 0 3 3 3
(%i12) f(x):=ceiling(x)$
            print(f(-3.6),f(-3.2),f(-3),f(0),f(3),f(3.2),f(3.6))$
       −3 −3 −3 0 3 4 4
```

We used the command `print` to format the report.

```
(%i3)  a:2$ b:log(2),numer$
            print("Logarithm of a number",a," is ",log(a),"=",b)$
       Logarithm of a number 2 is log (2) = 0.6931471805599453
```

Maxima contains many elementary functions. They are, for example `exp(x)=%e^x`, `log(x)`, trigonometric functions, their inverse functions `sin(x)` and `asin(x)`, `cos(x)` and `acos(x)`, `tan(x)` and `atan(x)`, `cot(x)` and `acot(x)`, hyperbolic functions and their inverse functions `sinh(x)` and `asinh(x)`, `cosh(x)` and `acosh(x)`, `tanh(x)` and `atanh(x)`, `coth(x)` and `acoth(x)` etc.

Maxima also includes many features to support them. Some of them are not implemented

directly in the wxMaxima environment, but in external libraries called packages. These packages are loaded into the system using the `load` command. We will show the `spangl` package for an example to support work with trigonometric functions.

```
(%i2) print(tan(%pi/8), ratsimp(tan(%pi/8)), trigsimp(tan(%pi/8)))$
```
$\tan\left(\frac{\pi}{8}\right)\ \tan\left(\frac{\pi}{8}\right)\ \frac{\sin\left(\frac{\pi}{8}\right)}{\cos\left(\frac{\pi}{8}\right)}$
```
(%i3) load(spangl);
(%o3) ../share/trigonometry/spangl.mac
(%i4) tan(%pi/8);
```
(%o4) $\sqrt{2}-1$

Working with Expressions

Maxima operations and calculations take place in an environment, in which the system presupposes the validity of certain conditions. We may change these terms. Many times we need to change the conditions only locally for a particular calculation without to change global settings. For this purpose, Maxima has a very efficient `ev` command that allows define a specific environment within a single command.

After entering the command `ev(a,b1,b2,..., bn)` the expression `a` is evaluated if the conditions `b1`, `b2`, ..., `bn` are met. These conditions can be equations, assignments, functions, switches (logical settings). The example shows an example of solving a quadratic equation using the command `solve`. Variables `a`, `b`, `c` after executing the command `ev` they do not have values assigned.

```
(%i1) ev(solve(a*x^2+b*x+c=0,x),a:2,b:-1,c=-3);
```
(%o1) $[x=\frac{3}{2}, x=-1]$
```
(%i2) solve(a*x^2+b*x+c=0,x);
```
(%o2) $\left[x=-\frac{\sqrt{b^2-4ac}+b}{2a}, x=\frac{\sqrt{b^2-4ac}-b}{2a}\right]$

Maxima offers several commands for simplifying and editing various expressions. The basic functions can be found in the `Simplify` menu. With the `ratsimp` commands and `trigsimp` we have already met and when adjusting the value of `tan(%pi/8)` they did not have the desired effect.

Maxima offers using the `example` command examples of individual commands. Let's take a look at some of the examples offered by `example(ratsimp)`.

```
(%i2) f(x):=b*(a/b-x)+b*x+a$ print(f(x),"?",ratsimp(f(x)))$
```
$bx+b(\frac{a}{b}-x)+a\ ?\ 2a$
```
(%i3) ratsimp(a+1/a);
```

```
(%o3)  (a^2+1)/a
(%i4)  ev(x^(a+1/a),ratsimp);
(%o4)  x^(a+1/a)
(%i5)  ev(x^(a+1/a),ratsimpexpons);
(%o5)  x^((a^2+1)/a)
```

Function `expand` multiplies the relevant members in the expression.

Function `factor` on the contrary, it decomposes the expression.

Function `gfactor` it does so over a field of complex numbers.

```
(%i1)  f(x):=(x+1)*(x^2-4)*(x^2+4)$
(%i3)  ratsimp(f(x));expand(f(x));
(%o2)  x^5+x^4-16*x-16
(%o3)  x^5+x^4-16*x-16
(%i6)  factor(f(x));gfactor(f(x));factor(100);
(%o4)  (x-2)*(x+1)*(x+2)*(x^2+4)
(%o5)  (x-2)*(x+1)*(x+2)*(x-2*%i)*(x+2*%i)
(%o6)  2^2*5^2
```

We decompose a rational polynomial function into partial fractions using the command `partfrac`.

```
(%i1)  partfrac((x+1)/(x^2-2*x+1),x);
(%o1)  1/(x-1)+2/(x-1)^2
```

We can substitute expressions using the commands `subst(a,b,c)` and `ratsubst(a,b,c)`. The expression `a` will be replaced by `b` and subsequently substituted into the expression `c`. When using the `subst` command must be `b` the simplest part (atom) or a complete subexpression of the expression `c`. In the example, the subexpression is not `x+y` complete (missing `z`). The `ratsubst` command it also modifies the resulting expression.

```
(%i2)  subst(x+y,a,a^2+b^2); ratsubst(x+y,a,a^2+b^2);
(%o1)  (y+x)^2+b^2
(%o2)  y^2+2*x*y+x^2+b^2
(%i4)  subst(a,x+y,x+y+z); ratsubst(a,x+y,x+y+z);
(%o3)  z+y+x
(%o4)  z+a
```

Limits and Derivatives

In the `Calculus` menu we find functions for solving basic problems of mathematical analysis (limits, derivation, integration, sums of series, decomposition of a function into a Taylor polynomial, ...).

We calculate the limits using the command `limit`. The last parameter determines the direction of unilateral limits, has the values `plus` or `minus` and is optional. If not specified, Maxima calculates the limit as complex.

With the commands `limit(f(x),x,a)` and `limit(f(x),x,a,plus)` we calculate the limits $\lim_{x\to a} f(x)$ and $\lim_{x\to a^+} f(x)$.

```
(%i4) limit(1/x,x,0);
          limit(1/x,x,0,plus); limit(1/x,x,0,minus); limit(1/x,t,0);
(%o1) infinity
(%o2) ∞
(%o3) -∞
(%o4) 1/x
```

If we use apostrophe `'` before the command, the command will not be executed, it will only be displayed.

```
(%i2) limit(((1-n)/(1+3*n))^(1+4*n),n,inf);
          'limit(((1-n)/(1+3*n))^(1+4*n),n,inf);
(%o1) 0
(%o2) lim_{n→∞} ((1-n)/(3n+1))^(4n+1)
```

Derivatives are calculated using the command `diff`. The parameter that determines the order of derivation is optional.

```
(%i4) f(x):=2*x^4-3*x+sin(x);
      print("f'=",diff(f(x),x),"=",diff(f(x),x,1))$
      print("f''=",diff(diff(f(x),x),x),"=",diff(f(x),x,2),
            "=",diff(f(x),x,1,x,1))$
      print("f^(10)=",diff(f(x),x,10),"=",diff(f(x),x,1,x,9))$
(%o1) f(x) := 2x^4 - 3x + sin(x)
      f' = cos(x) + 8x^3 - 3 = cos(x) + 8x^3 - 3
      f'' = 24x^2 - sin(x) = 24x^2 - sin(x) = 24x^2 - sin(x)
      f^(10) = - sin(x) = - sin(x)
```

We calculate partial derivatives using the same command.

```
(%i3)  g(x,y):=x^3*y^2-1;
       print("g'_x=",diff(g(x,y),x),", respectively g'_y=",
             diff(g(x,y),y,1))$
       print("g''_(xx)=",diff(g(x,y),x,2),", respectively g''_(yx)=",
             diff(g(x,y),y,1,x,1))$
```

(%o1) $g(x,y) := x^3y^2 - 1$

$g'\_x = 3x^2y^2$, respectively $g'\_y = 2x^3y$

$g''\_(xx) = 6xy^2$, respectively $g''\_(yx) = 6x^2y$

We calculate the Taylor polynomial nth degree using the command `taylor`. You can find this command in the `Calculus` menu and the `Get Series...` submenu. We calculate Taylor series of functions f degree n in the middle c with the command `taylor(f(x),x,c,n)`. Its coefficients are obtained using the command `coeff`. The use of this command depends on the `taylor` command.

```
(%i1)  t1:taylor(sin(x),x,0,5); t2:taylor(sin(x),x,-1,4);
```

(t1) $x - \frac{x^3}{6} + \frac{x^5}{120} + \cdots$

(t2) $-\sin(1) + \cos(1)(x+1) + \frac{\sin(1)(x+1)^2}{2} - \frac{\cos(1)(x+1)^3}{6} - \frac{\sin(1)(x+1)^4}{24} + \cdots$

```
(%i3)  print(coeff(sin(x),x,5)," and ",coeff(t1,x,5)," ",coeff(t2,x,5))$
```

0 and $\frac{1}{120}$ $\frac{\cos(1)}{120}$

Taylor polynomial of polynomial is again a polynomial, only it is expressed in another form. Practically, only the coordinate system will change in which we express the polynomial. The beginning of the system moves from point 0 to point -1.

In the following example, the Taylor polynomial of a given polynomial is calculated in another way. Command `taylor` gives three points at the end, even if development is closed.

```
(%i1)  f(x):=2*x^5-x^4-3*x^3-x+1;
```

(%o1) $f(x) := 2x^5 - x^4 + (-3)x^3 - x + 1$

```
(%i2)  tp1:taylor(f(x),x,-1,5);
```

(tp1) $2 + 4(x+1) - 17(x+1)^2 + 21(x+1)^3 - 11(x+1)^4 + 2(x+1)^5 + \cdots$

```
(%i4)  ratsimp(tp1);expand(tp1);
```

(%o3) $2x^5 - x^4 - 3x^3 - x + 1$

(%o4) $2x^5 - x^4 - 3x^3 - x + 1$

```
(%i6)  tpx:ratsubst(t,x+1,f(x));subst(x+1,t,tpx);
```

(tpx) $2t^5 - 11t^4 + 21t^3 - 17t^2 + 4t + 2$

(tp2) $2(x+1)^5 - 11(x+1)^4 + 21(x+1)^3 - 17(x+1)^2 + 4(x+1) + 2$

```
(%i7)  tp1-tp2;
```

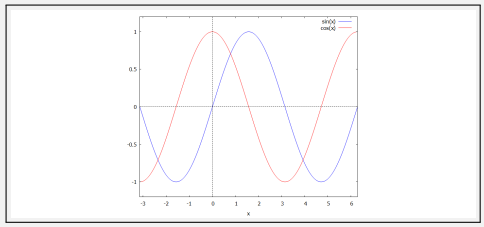
(%o7) $0 + \cdots$

Function Graphs

We can plot the function graph in several ways. The easiest way is to choose `Plot` in the menu submenu `Plot 2d ...`.

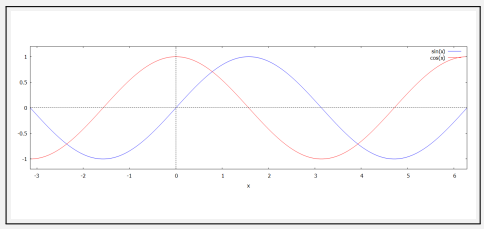
If we choose `Format=gnuplot`, the function is rendered by the command `plot2d` using the Open Source program `Gnuplot` to a new window. `Gnuplot` is automatically installed together with Maxima.

```
(%i1)  plot2d([sin(x),cos(x)],[x,-%pi,2*%pi],[y,-1.2,1.2],
       [plot_format, gnuplot])$
```



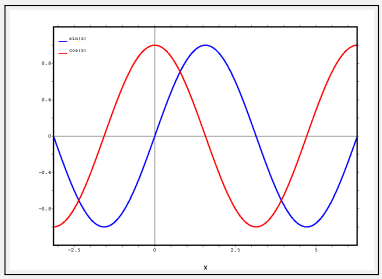
The graphs of the sine and cosine functions were not displayed in the real ratio of the x and y axes, but were optimized for the screen. We have to use e.g. `same_xy` parameter for proper display.

```
(%i1)  plot2d([sin(x),cos(x)],[x,-%pi,2*%pi],[y,-1.2,1.2],
       [plot_format, gnuplot],[same_xy])$
```



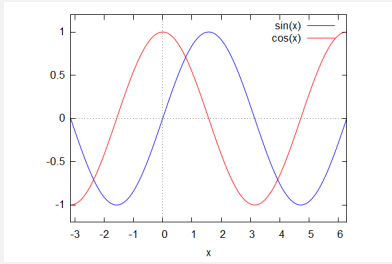
If we choose `Format=wxmaxima`, Maxima will plot the graph using the command `plot2d` to a new window. We can only save the image in postscript.

```
(%i1) plot2d([sin(x),cos(x)],[x,-%pi,2*%pi],[y,-1.2,1.2],
       [plot_format, xmaxima])$
```



If we choose `Format=inline`, Maxima draws a graph using the command `wxplot2d` into your environment.

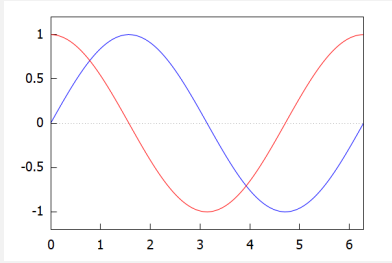
```
(%i1) wxplot2d([sin(x),cos(x)],[x,-%pi,2*%pi],[y,-1.2,1.2])$
```



(%o1)

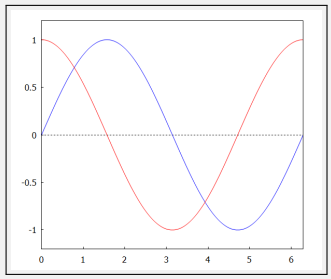
Commands `plot2d` and `wxplot2d` they have the same syntax and have many more parameters. Parameters can be found, for example, with the command `describe(plot2d)`.

```
(%i1) wxdraw2d(xaxis=true,yaxis=true,xrange=[0,2*%pi],yrange=[-1.2,1.2],
      color=blue,explicit((sin(x)),x,0,2*%pi),
      color=red,explicit((cos(x)),x,0,2*%pi))$
```



(%o1)

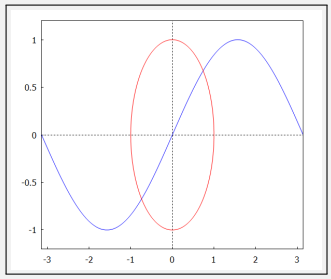
```
(%i1) draw2d(xaxis=true,yaxis=true,xrange=[0,2*%pi],yrange=[-1.2,1.2],
      color=blue,explicit((sin(x)),x,0,2*%pi),
      color=red,explicit((cos(x)),x,0,2*%pi))$
```



It is better to use the `wxdraw2d` command to print function graphs or `draw2d`, which should be routed to the output of `Gnuplot`. These commands have a slightly different syntax than the `wxplot2d`, `plot2d`. The print parameters are simpler and clearer. The plotted function must be located in the command `explicit`, `parametric` or `implicit`.

We plot a parametric curve or function in a similar way.

```
(%i1) draw2d(xaxis=true,yaxis=true,xrange=[-%pi,%pi],yrange=[-1.2,1.2],
      color=blue,explicit((sin(x)),x,-%pi,%pi),
      color=red,nticks=300,parametric(cos(t),sin(t),t,0,2*%pi))$
```



Sequences and Series

Sequences can be created in Maxima, for example, using the command `makelist` or with the statements of the cycle `for..do`.

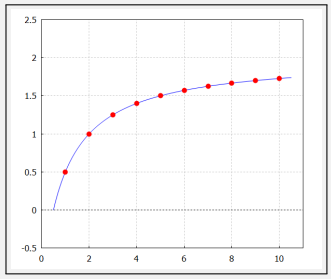
Command `makelist` creates a list that we can display as a whole and by members.

```
(%i2) S1:makelist(2*n^2-1,n,1,10);S2:makelist(2*n^2-1,n,2,10,2);
```

```
(S1)  [1, 7, 17, 31, 49, 71, 97, 127, 161, 199]
(S2)  [7, 31, 71, 127, 199]
(%i4) S1[1];S1[10];
(%o3) 1
(%o4) 199
```

Arranged pairs are enclosed in square brackets and can be displayed as points in a plane. In the following example, a sequence is also generated with its patterns and then plotted with a command `draw2d`.

```
(%i1) S1:makelist([n,(2*n-1)/(n+1)],n,1,10);
(S1)  [[1, 1/2], [2, 1], [3, 5/4], [4, 7/5], [5, 3/2], [6, 11/7], [7, 13/8], [8, 5/3], [9, 17/10], [10, 19/11]]
(%i2) draw2d(grid=true,xaxis=true,yaxis=true,xrange=[0,11],
      yrange=[-0.5,2.5],color=blue,explicit((2*n-1)/(n+1),n,0.5,10.5),
      point_type=7,color=red,points(S1))$
```



Using the command `for..do` we will list several members of the sequence $\left\{2n^2-1\right\}_{n=1}^{\infty}$.

```
(%i1) (for n:1 thru 12 do (a_n: 2*n^2-1, print(a_n)) )$
      1
      7
      17
      31
      49
      71
      97
      127
      161
```

```
199
241
287
```

A nice example of using the command `for..do` is a Fibonacci sequence.

```
(%i3) a0:0$ a1:1$
      (for i:1 thru 12 do (an:a1+a0, print(an), a1:a0, a0:an))$
1
1
2
3
5
8
13
21
34
55
89
144
```

We calculate the finite and infinite sum using the command `sum`.

```
(%i1) sum(2*n^2-1,n,1,8);
(%o1) 400
```

With this command, Maxima can calculate the exact sum of some infinite series. The sum of the series can be entered in the menu `Calculus` and the `Calculate Sum...` submenu.

```
(%i2) sum(1/k^2,k,1,inf);
      sum(1/k^2,k,1,inf),simpsum;
(%o1) ∑_{k=1}^{∞} (1/k^2)
(%o2) π^2/6
```

The number series from the previous example can be graphically represented as follows.

```
(%i1) a(n):=1/n^2$ rec:makelist(rectangle([i-1,0],[i,a(i)]),i,1,10)$
      draw2d(grid=true,xaxis=true,yaxis=true,xrange=[-1,11],
      yrange=[-0.2,1.2],border=true,color=black,fill_color=red,rec,
      color=blue,explicit(a(n),n,0,11))$
```