Maxima by Example: Ch. 1, Getting Started *

Edwin L. Woollett

Contents

*Date: June 18, 2008. This version uses Maxima 5.15. Check `http://www.csulb.edu/~woollett/` for the latest version of these notes. Send comments and suggestions to `woollett@charter.net`

COPYING AND DISTRIBUTION POLICY

This document is part of a series of notes titled "Maxima by Example" and is made available via the author's webpage `http://www.csulb.edu/~woollett/` to aid new users of the Maxima computer algebra system.

These notes (with some modifications) will be published in book form eventually via Lulu.com in an arrangement which will continue to allow unlimited free download of the pdf files as well as the option of ordering a low cost paperbound version of these notes.

Acknowledgements

Many of the examples used in these notes are from the Maxima manual and the Maxima mailing list. I have organized the topics in an order which starts with the basic issues of using the software in a Window's XP environment, and then I focus on what seem to be the most pressing questions for a newcomer to Maxima. In addition to the organization of the examples presented, I have contributed a running commentary on what is important to take away from a given example. I have preferred more details rather than less, since experimenting with Maxima's behavior is my way of getting comfortable with this powerful software. The Maxima mailing list has been a great help to the author. He would like to thank the Maxima developers for their friendly help via this list. Readers should browse the superb Russian webpage "Maxima Beginner's FAQ" (in English!) authored by Alexey Beshenov, `http://beshenov.ru/maxima/faq.html`.

1 Getting Started

1.1 Introduction

1.1.1 What is Maxima?

Maxima is a powerful computer algebra system (CAS) which combines analytic (symbolic), graphical, and numerical abilities. See the Maxima sourceforge webpage `http://maxima.sourceforge.net/`

A cousin of the commercial Macsyma CAS (now available but without support), Maxima is a freely available and open source program which is being continuously improved by a team of volunteers. When compared with Mathematica or Maple, Maxima has a more basic interface, but has the advantage in price (!). Students, teachers, and researchers can "own" multiple copies for home, laptop, and desktop without the expense of buying licenses for each copy.

There are known "bugs" in the present version (a new version is available about three times each year), and the volunteer developers and programming experts are dealing with these known bugs as time permits.

Maxima is not only "free" and will always stay that way, but also comes with a copy of the underlying source code (in a dialect of the Lisp language), which a user can modify to suit his own research needs and then share with the Maxima community of users.

Of course, the physics student and researcher can usually use the existing Maxima program constructs to craft solutions for solving physics problems and can write programs directly in the Maxima language which is easy to learn.

For Windows users, a self-installing binary for Windows is available, making it easy for Windows users to get a fast start.

I recommend that you first familiarize yourself with the Maxima work environment by downloading and installing Maxima on your computer, and starting up the XMaxima interface.

1.2 Using the XMaxima Interface

To start up XMaxima and the Maxima computational engine, you can use the Start, All Programs, Maxima 5.14.0, xmaxima link. You can also put a quick start icon on your desktop to XMaxima. Go back to that same Start menu item: All Programs, Maxima 5.14.0, and right click xmaxima, and select "create icon". The created second entry "xmaxima(2)" can then be dragged to your desktop and renamed by again right clicking the (new) desktop icon and selecting "rename". The icon links to the file (on my computer):
`c:\Program Files\Maxima-5.14.0\bin\xmaxima.exe`.

If you are new to Maxima, go through the quick start tutorial in the bottom window of XMaxima. You can either click on the expression to have it evaluated, or you can enter the expression, followed by a semi-colon

(;) in the upper window and press enter to have Maxima carry out the operation. In the top XMaxima window, note that the two key command Alt+p will type in the previous entry, and you can keep entering Alt+p until you get the entry you want (to edit or simply rerun).

Once you know what you are doing, you can close the bottom window by using the Menu bar. On that menu bar, choose Options, Toggle Browser Visibility. To use only keyboard commands to close the bottom window, use `Alt+o`, `Esc`, `Enter`. To quit XMaxima, choose File, Exit on the menu bar (or `Alt+f`, `x`).

The second short intro can be found on the Start,All Programs, Maxima 5.14, Introduction link. Written by Cornell University (Dept. of Theoretical and Applied Mechanics) Professor Richard Rand (`http://tam.cornell.edu/`), this is an excellent short introduction to many basic features of Maxima. The advice in this introduction to exit Maxima by typing "quit();" is relevant if you are using the "command line maxima" version, aka "maxima console".

1.2.1 Using Console - Command Line Maxima

You can also start up this version from the Start menu. The shortcut on the start menu links to a file called "maxima.bat" in the same directory as "xmaxima.exe". If you want to use the command line maxima "console" interface, I recommend that you again create a desktop icon, and then right click that icon, and select "properties". On the Options tab, make sure Display Options is set to Window. On the Font tab, select Raster fonts, 8 x 12. On the Layout tab, Window size, I have set width = 90, height = 42. and make sure "let system position window" box is checked. On the Colors tab, click on "Screen Background" and then click on the white box (far right box). Then click on "Screen text", and then click on the black box (far left box). Leave the other tab settings alone. When you are satisfied, click "apply", and then "ok".

Now, when you start the console version of Maxima, with this icon, you will be using a decent size window, with black type on a white background, which will be easier to work with than the defaults.

You can retrieve previous lines of input by using the Up and Down arrow keys. You can edit a line using "end", "home", right arrow, left arrow, delete, and backspace keys, adding text with the right toggle of the insert key. You can copy portions of the maxima console screen by right clicking the screen, type "k" for "mark", then drag over the text you want to select, then press "Enter". (NOT ctrl+C !!) The resulting clipboard contents can then be pasted into any other document. You can paste clipboard contents into the Maxima console screen by right clicking the cursor, and typing "p" for "paste". Only small items should be pasted this way into the console window. The chief disadvantage of the console mode is the difficulty of retrieving multiple line commands and editing multiple line commands. (If you have a method, please let me know!). When I want to use console mode for serious work, I am loading a text file of maxima commands to see how they work, and then I edit those commands with a text editor and then reload the file into maxima.

1.2.2 Text Editors

I use the freely available Notepad2 or Notepad++, which makes it easier to look for and save files which do not have the extension ".txt". Notepad++ is freely available at `http://notepad-plus.sourceforge.net/uk/site.htm`. Notepad2 is freely available at `http://www.notepad2.com/`. Both Notepad2 and Notepad++ use color-coding to balance pairs of parentheses and pairs of brackets.

1.2.3 The Maxima Manual

The most important continuous source of information about Maxima syntax and reserved words is the Maxima Manual, which you should leave open in a separate window. To open a separate Maxima Manual window from inside the XMaxima interface, go the menu item: Help, Maxima Manual. Alternatively on the Windows Start Menu, All Programs, Maxima 5.14 panel, choose "Reference Manual". You can create a desktop icon link to that same Help Manual by following the steps described above for a XMaxima desktop link, but now using the "Reference Manual" icon link. That Help Manual file lives (on my computer) at:
`c:\Program Files\Maxima-5.14.0\share\maxima\5.14.0\doc\chm\maxima.chm`.

Play with moving around this reference manual via either "Contents" or "Index". For example, select "Index" and start typing "integrate". By the time you have typed in "inte", you are close to the "integrate" entry, and you can either continue to type the whole word, or use the down arrow key to select that entry. Then press "Enter". On the right side is the Manual entry for "integrate":

> Function: integrate (expr, x)
> Function: integrate (expr, x, a, b)
> Attempts to symbolically compute the integral of expr with respect to x. integrate (expr, x) is an indefinite integral, while integrate (expr, x, a, b) is a definite integral, with limits of integration a and b. The limits should not contain x, although integrate does not enforce this restriction. a need not be less than b. If b is equal to a, integrate returns zero.
> ...(continues).

To scroll the Manual page, double-click on the page, and then the Page Down and Up keys and the Up and Down Arrow keys should allow scrolling in that area of the Manual. To return to the index entry panel, left click that panel, and type "diff", which which take you to the section of the Manual which explains how to evaluate a derivative.

Take a look at some of the short tutorial notes which you can download from the Maxima docs page at sourceforge:
`http://maxima.sourceforge.net/docs.shtml`.

The Maxima community shares an active "mailing list" which includes questions and suggested solutions to specific problems of all types. You can browse the mailing list archive and/or subscribe to the mailing list at the webpage:
`http://maxima.sourceforge.net/maximalist.html`.

1.3 Expressions in Maxima

The basic unit of information in Maxima is the expression. An expression is made up of a combination of operators, numbers, variables, and constants.

1.3.1 Operators in Maxima

The table on the next page lists some Maxima operators in order of priority, from lowest to highest.
The input `x^2+3` means $x^2 + 3$, and not x^{2+3}. The exponentiation has higher precedence than addition. The input `2^3^4` stands for `2^(3^4)`. Parentheses can be used to force order of operations, or simply for clarity.

```
(%i1) x^2+3;
                                    2
(%o1)                              x  + 3
(%i2) 2^3^4;
(%o2)                  24178516392292583494123 52
(%i3) 2^(3^4);
(%o3)                  24178516392292583494123 52
(%i4) (2^3)^4;
(%o4)                               4096
```

Since the operator $*$ has precedence over $+$, $a + b * c$ means $a + (b * c)$, rather than $(a + b) * c$.

Operator	Description
+	addition
−	subtraction
*	multiplication
/	division
−	negation
^	exponentiation
.	non-commutative multiplication
^^	non-commutative exponentiation
!	factorial
!!	double factorial

Table 1: Operators in Order of Increasing Precedence

1.3.2 Numbers in Maxima

Maxima uses

- Integers, like 123456.
- Rational numbers, like $3/2$, are ratios of integers.
- Floats and bigfloats are "floating point numbers. The Maxima manual has the **float** description:

  > Function: **float** `(expr)`
  > Converts integers, rational numbers and bigfloats in `expr` to floating point numbers. It is also an evflag, `float` causes non-integral rational numbers and bigfloat numbers to be converted to floating point.

```
(%i1) [x,y,z,w] :[123.0, 1.23e2, 1.23d2, float(123) ];
(%o1)                     [123.0, 123.0, 123.0, 123.0]
(%i2) map( floatnump,[x,y,z,w] );
(%o2)                          [true, true, true, true]
(%i3) [sin(1),sin(1.0),float(sin(1)),ev(sin(1),float),ev(sin(1),numer)];
(%o3) [sin(1), 0.8414709848079, 0.8414709848079, sin(1), 0.8414709848079]
```

  The arithmetic precision for floating point numbers is 16 digits. See the manual **bfloat** description for "bigfloats".

- Complex numbers, like `4 + 2*%i` and `a + b*%i`. Maxima assumes the symbols `a` and `b` represent real numbers by default.

1.3.3 Variables in Maxima

Variables are named quantities. You can either bind a value to a variable, or you can leave the variable unbound and treat it formally. Binding a value to a variable is called assignment. This section shows you how you can use both bound and unbound variables in expressions. One of the features of Maxima which leads to ease of use is the fact that you do not need to declare symbols to be of any special type (like integer) or have special properties (like being non-zero) in order to use the symbol in expressions. The Maxima developer Stavros Macrakis has commented on this feature via the Mar. 23, 2008 Maxima mailing list:

> One of the things that makes Maxima (relatively) easy to use is precisely the lack of declarations. We don't require the user to declare `x` as a non-zero real to allow `1/x`, and he can even substitute `false` or `matrix([2,3],[4,6])` for `x` later.

To assign a value to a variable, type the name of the variable, followed by a colon, followed by the value.

```
(%i1) values;
(%o1)                                  []
(%i2) a : 3 + 7;
(%o2)                                  10
(%i3) a;
(%o3)                                  10
(%i4) values;
(%o4)                                  [a]
(%i5) c : a + b;
(%o5)                               b + 10
(%i6) b : 5;
(%o6)                                   5
(%i7) c;
(%o7)                               b + 10
(%i8) ''c;
(%o8)                                  15
(%i9) remvalue(a);
(%o9)                                  [a]
(%i10) values;
(%o10)                              [c, b]
(%i11) [b,c];
(%o11)                           [5, b + 10]
(%i12) remvalue(all);
(%o12)                              [c, b]
(%i13) values;
(%o13)                                 []
(%i14) [c,b];
(%o14)                              [c, b]
```

When the symbol `c` was defined in input `%i5`, `a` was already bound to the value 10, but the symbol `b` was unbound; the symbol `a` was replaced by 10 and `%o5` became the binding of `c`. Notice that once we gave `b` a value in input `%i6`, the variable `c` retains its "binding" to the value `b + 10` in output `%o7`. We used two single quotes in input `%i8` to force evaluation of the symbol `b`. We use **values** to see what symbols are bound to values, and **remvalue**(...) to break the binding.

There are a number of reserved words which cannot be used as variable names. Their use would cause a possibly cryptic syntax error.

```
integrate           next          from              diff
in                  at            limit             sum
for                 and           elseif            then
else                do            or                if
unless              product       while             thru
step
```

Of course there are many other Maxima function names, global option variables, and system option variables which you should also avoid when naming your own variables. One quick way to check on name conflicts is to keep the html Maxima help manual up in a separate window, and have the Index panel available to type in a name you want to use. The slow way to check on name conflicts is to wait for Maxima to give you a strange error message as to why what you are trying to do won't work.

1.3.4 Constants in Maxima

The following table summarizes predefined constants.

Constant	Description
%e	Base of the natural logarithms (e)
%i	The square root of (-1) (i)
%pi	The transcendental constant pi (π)
%phi	The golden mean $(1+\sqrt{5})/2$
%gamma	The Euler-Mascheroni constant
inf	Real positive infinity (∞)
minf	Real negative infinity $(-\infty)$

Table 2: Maxima Predefined Constants

Here are the numerical values to 16 digit precision.

```
(%i1) float( [%e,%pi,%phi,%gamma] );
(%o1) [2.718281828459045, 3.141592653589793, 1.618033988749895,
                                                        0.57721566490153]
```

We can use **bfloat** to get more digits. First we show the default result, and then the result of changing **fpprec** to a value larger than its default value of 16. Note that the value of **fpprec** does not affect the precision of floating point arithmetic or the number of digits returned by the function **float**.

```
(%i2) fpprec;
(%o2)                                16
(%i3) bfloat(%pi);
(%o3)                          3.141592653589793b0
(%i4) bfloat(%pi),fpprec:100;
(%o4) 3.1415926535897932384626433832795028841971693993751058209749445923078164\
0628620899862803482534211706 8b0
```

1.3.5 Rational Expressions in Maxima, rat(...)

Maxima has the **example** function which will present pre-arranged examples of the use of various Maxima functions. An example of using **example** for one of the most useful Maxima functions, **ratsimp**:

```
(%i5) example(ratsimp);
(%i6) sin(x/(x+x^2)) = %e^((1+log(x))^2-log(x)^2)
                                                    2      2
                           x          (log(x) + 1)  - log (x)
(%o6)                sin(------) = %e
                          2
                         x  + x
(%i7) ratsimp(%)
                                    1          2
(%o7)                          sin(-----) = %e x
                                   x + 1
    ( continues...  )
```

Notice that the output of the **example** function shows the input lines `%in` with no semicolon at the end. Of course, in an interactive session, you would end your input with a semicolon or a dollar sign.

example, if called with no argument as in `example();`, presents a list of available examples stored in a special text file called "manual.demo" which the **example** function reads.

```
(%i13) example();
Not Found.  You can look at:
(%o13) [additive, algsys, allroots, antisymmetric, append, arrayinfo, arrays,
at, atvalue, augcoefmatrix, bezout, block, bothcoeff, catch, cf, cfdisrep,
cfexpand, changevar, charpoly, coeff, combine, commutative, complex, content,
defmatch, deftaylor, delete, depends, derivdegree, desolve, diff, display,
divide, do, dotscrules, dpart, echelon, eliminate, entermatrix, equations, ev,
evaluation, evenfun, exp, expand, factcomb, factor, factorsum, featurep,
freeof, fullmap, fullmapl, funcsolve, functions, genmatrix, get, gfactor,
gradef, horner, if, ilt, inpart, integrate, is, isolate, lambda, laplace,
lassociative, let, letrules, limit, linear, linsolve, listofvars, lists,
logcontract, map, matchdeclare, matrices, minfactorial, multiplicative,
multthru, nary, nounify, nroots, numfactor, nusum, oddfun, ode2, optimize,
ordergreat, orderless, outative, part, partfrac, partition, pickapart,
poissimp, polarform, poly_discriminant, posfun, powerseries, printprops,
product, properties, propvars, qunit, radcan, rank, rassociative, rat,
ratcoeff, ratdiff, ratexpand, ratsimp, ratsubst, ratweight, realpart,
realroots, residue, resultant, reveal, reverse, risch, rootscontract, scanmap,
scsimp, solve, specint, sqfr, substinpart, subst, substpart, sum, symmetric,
syntax, taylor, taytorat, tellrat, tellsimp, triangularize, trig, trigexpand,
trigreduce, unorder, xthru, zeroequiv]
```

Many of these **example** examples are reproduced in the Maxima manual.

Some of the Maxima functions include the text "**rat**". These functions have nothing to do with rodents! Rather, these functions and flags are tools to manipulate rational expressions, or sums of rational expressions. A rational expression is the quotient of two polynomials. The Maxima function **num**`(expr)` returns the numerator of the rational expression `expr`. The function **denom**`(expr)` returns the denominator of the rational expression `expr`. For all rational expressions `expr`, **num**`(expr)`/**denom**`(expr)` is the same as `expr`. A polynomial is a special case of a rational expression.

A special internal representation for a rational expression is used by "rational functions" such as **ratsimp** and **factor**. This special representation is called Canonical Rational Expression (CRE), and internal use of these forms proves to be more efficient (faster with less storage requirements). The Maxima function **rat** converts an expression to CRE form. You can force your own ordering on the resulting expression by supplying arguments `arg1, arg2,...,argn` to **rat**, using the syntax **rat** `(expr,arg1,arg2,...,argn)`. The ordering of "variables" is then `argn` is the "main variable", and the other arg's (reading right to left) are assigned orders beneath the "main variable". Starting with the default ordering:

```
(%i1) e1: (x + y)^4;
                                        4
(%o1)                                (y + x)
(%i2) rat(e1);
                         4         3       2  2       3       4
(%o2)/R/                y  + 4 x y  + 6 x  y  + 4 x  y + x
(%i3) rat(e1,x);
                         4         3       2  2       3       4
(%o3)/R/                x  + 4 y x  + 6 y  x  + 4 y  x + y
```

the input `%i3` makes `x` the "main variable". Note that the symbol `/R/` which follows the line labels above indicates (in the general case) that either the expression is in CRE form, or some subexpression is in CRE form.

See the Maxima manual, ch. 12, Polynomials for more information on CRE's.

1.4 List Creation and Manipulation

A list in Maxima is an ordered set of elements, separated by commas and enclosed in square brackets. An element of a list can be any Maxima expression. You can assign a list as the value of a variable, and then refer to its individual elements as subscripted variables. Lists can be used as arguments to some commands, such as **solve** and **matrix**. Other commands return results in a list, including the Maxima function **solve**, and the system variables **functions** and **values**.

The Maxima manual, Sec. 37.1, introduces the list as the fundamental structure in the Lisp language which underlies the Maxima language (some editing added):

> Lists are the basic building blocks for Maxima and Lisp. All data types other than arrays, hash tables, and numbers are represented as Lisp lists. These Lisp lists have the form `((MPLUS) $A 2)`, to indicate the expression $a+2$. At the Maxima level one would see the "infix notation" a+2. Maxima also has lists which are printed as `[1, 2, 7, x+y]`, for a list with 4 elements. Internally this corresponds to a Lisp list of the form `((MLIST) 1 2 7 ((MPLUS) $X $Y ))`. The "flag" which denotes the "type field" of the Maxima expression is a list itself; after it has been through the simplifier, the Lisp list `(MPLUS)` having the single element `MPLUS` becomes the Lisp list `(MPLUS SIMP)` which has two elements.

> The Maxima list `[1, 2, 7, x+y]`, after going through the simplifier, is represented by the Lisp list: `((MLIST SIMP) 1 2 7 ((MPLUS SIMP) $X $Y))`. Note that a Lisp list consists of elements separated by spaces, and parentheses are used rather than square brackets. A Maxima variable `a` appears as `$A` in the Lisp list, and just as an element of a Maxima list can itself be a list, the Lisp list `(MPLUS SIMP)` can be an element of a Lisp list.

1.4.1 The Maxima List and the Lisp List

Once you have bound a Maxima list to a variable, say `mylist`, you can see the internal Lisp representation as follows:

```
(%i1) mylist:[a,a*b,3*%i,4*%pi,[1,2] ];
(%o1)                     [a, a b, 3 %i, 4 %pi, [1, 2]]
(%i2) :lisp $mylist
((MLIST SIMP) $A ((MTIMES SIMP) $A $B) ((MTIMES SIMP) 3 $%I)
 ((MTIMES SIMP) 4 $%PI) ((MLIST SIMP) 1 2))
(%i2) a1 : mylist[1];
(%o2)                                      a
(%i3) :lisp $a1
                                           $A
```

Notice that you end the `:lisp` input with just `Enter`, not including a semicolon, and you use the Lisp name of the variable which starts with the dollar sign `$`.

1.4.2 cons, endcons, append

There are some Maxima functions designed to create, get properties of, and manipulate lists, for example **append**, **cons**, **copylist**, **endcons**, **first**, **last**, **length**, **listp**, **makelist**, **rest**, **reverse**, **second**.

The manual entry for **cons** (think: "construct", as in "to put together the parts of something; to build") is

> Function: **cons** `(expr, alist)`
> Returns a new list constructed of the element `expr` as its first element, followed by the elements of `alist`.
> `cons` also works on other expressions, e.g. `cons(x, f(a,b,c)); -> f(x,a,b,c).`

The manual entry for **endcons** is

> Function: **endcons** `(expr, alist)`
> Returns a new list consisting of the elements of `alist` followed by `expr`.
> `endcons` also works on general expressions, e.g. `endcons(x, f(a,b,c)); -> f(a,b,c,x).`

Given two lists `v1` and `v2`, which we can think of as explicit representations of vectors, Maxima lets us add, subtract, multiply, divide, find the inner product, and even raise one of the lists to a "power" given by the other list (dubious usefulness!).

```
(%i1) v1 : [a,b,c]$
(%i2) v2 : [1, 2, 3]$
(%i3) v1 + v2;
(%o3)                          [a + 1, b + 2, c + 3]
(%i4) v1 - v2;
(%o4)                          [a - 1, b - 2, c - 3]
(%i5) v1*v2;
(%o5)                              [a, 2 b, 3 c]
(%i6) v1.v2;
(%o6)                              3 c + 2 b + a
(%i7) v1 / v2;
                                        b  c
(%o7)                                [a, -, -]
                                        2  3
```

```
(%i8) v2 / v1;
                                    1  2  3
(%o8)                              [-, -, -]
                                    a  b  c
(%i9) v1^v2;
                                       [1, 2, 3]
(%o9)                          [a, b, c]
```

All of these operations, except for the "inner product" and the strange last result, return lists.

To join lists, rather than adding an element at the beginning or end, use **append**, which has the manual description:

> Function: **append**`(list\_1, ..., list\_n)`
> Returns a single list of the elements of `list\_1` followed by the elements of `list\_2`, ...
> `append` also works on general expressions, e.g.
> `append (f(a,b), f(c,d,e)); yields f(a,b,c,d,e).`

Some examples of list functions at work:

```
(%i10) length(v1);
(%o10)                                  3
(%i11) first(v1);
(%o11)                                  a
(%i12) rest(v1);
(%o12)                                [b, c]
(%i13) last(v1);
(%o13)                                  c
(%i14) cons(a0,v1);
(%o14)                           [a0, a, b, c]
(%i15) endcons(zz,v1);
(%o15)                           [a, b, c, zz]
(%i16) append(v1,v2);
(%o16)                         [a, b, c, 1, 2, 3]
```

1.4.3 makelist and map

The Maxima manual entry for **makelist** is

> Function: **makelist**`(expr, i, i_0, i_1)`
> Function: **makelist**`(expr, x, list)`
> Constructs and returns a list, each element of which is generated from `expr`.
>
> `makelist (expr, i, i_0, i_1)` returns a list, the j'th element of which is equal to `ev (expr, i=j)` for `j` equal to `i_0` through `i_1`.
>
> `makelist (expr, x, list)` returns a list, the j'th element of which is equal to `ev (expr, x=list[j])` for `j` equal to `1` through `length (list)`.

Examples:

```
(%i1) makelist(concat(x,i),i,1,6);
(%o1)                [x1, x2, x3, x4, x5, x6]
(%i2) makelist(x=y,y,[a,b,c]);
(%o2)                  [x = a, x = b, x = c]
```

The manual examples display the use of **concat** and also a method of constructing a list of solutions. Here are two more examples of the use of **makelist**, which continue the session numbering above the manual quote for **makelist**:

```
(%i3) makelist(x^3,x, v1);
                                    3   3   3
(%o3)                             [a , b , c ]
(%i4) makelist(sin(j*%pi/3), j, 1, 6 );
                  sqrt(3)  sqrt(3)        sqrt(3)     sqrt(3)
(%o4)           [-------, -------, 0, - -------, - -------, 0]
                     2        2              2           2
```

The first example of **makelist** gets the values of "x" from the list `v1`. The second example of **makelist** uses a dummy index "j" with explicit beginning and end values. There is no option for a "step size" other than the value 1.

Instead of using **makelist**, we can **map** either a named function (created with the delayed assignment symbol :=, an anonymous **lambda** function, a builtin Maxima core function, or an operator on a pre-existing list. Let's first illustrate with an undefined symbol `f` how the Maxima function **map** can be used with one, two, or more lists at a time.

```
(%i1) f;
(%o1)                                  f
(%i2) map(f,[a,b,c]);
(%o2)                        [f(a), f(b), f(c)]
(%i3) map(f,[a,b,c], [aa,bb,cc] );
(%o3)                  [f(a, aa), f(b, bb), f(c, cc)]
```

We can reproduce results obtained above with two lists, and also create nested lists using **map**.

```
(%i1) v1 : [a,b,c]$
(%i2) v2 : [1,2,3]$
(%i3) v3:[ aa,bb,cc]$
(%i4) map("*",v1,v2);
(%o4)                           [a, 2 b, 3 c]
(%i5) map("/",v1,v2);
                                       b  c
(%o5)                                [a, -, -]
                                       2  3
(%i6) map("+",v1,v2);
(%o6)                        [a + 1, b + 2, c + 3]
(%i7) map("=",v1,v2);
(%o7)                        [a = 1, b = 2, c = 3]
(%i8) map("[",v1,v2);
(%o8)                     [[a, 1], [b, 2], [c, 3]]
(%i9) map("[",v1,v2,v3);
(%o9)              [[a, 1, aa], [b, 2, bb], [c, 3, cc]]
```

Example %i7 shows how you can construct a list of equations.

A Maxima function like **solve** can return a list of solutions, in the form of sublists of equations, such as [[a = 1,b = 2],[a = 1, b = 2]]. This returned list of solutions does not bind the parameters a and b, but we can use the Maxima **map** and **rhs** functions to make this assignment of solution values for later work.

```
(%i1) solns : [ [a = 1, b = 2], [a = 2, b = 1 ] ];
(%o1)                  [[a = 1, b = 2], [a = 2, b = 1]]
(%i2) solns[1];
(%o2)                           [a = 1, b = 2]
(%i3) [a,b] : map( rhs, solns[1] )$
(%i4) [a,b];
(%o4)                                [1, 2]
```

We can map core Maxima functions onto a list to generate lists of function values.

```
(%i1) fpprintprec:5$
(%i2) xlist : makelist(j,j,0,20)/20.0;
(%o2) [0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6,
                                    0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0]
(%i3) map( sin, xlist);
(%o3) [0, 0.05, 0.0998, 0.149, 0.199, 0.247, 0.296, 0.343, 0.389, 0.435,
   0.479, 0.523, 0.565, 0.605, 0.644, 0.682, 0.717, 0.751, 0.783, 0.813, 0.841]
```

The same goes for user defined functions:

```
(%i4) f(x) := x^3$
(%i5) map( f, xlist );
(%o5) [0, 1.25E-4, 0.001, 0.00338, 0.008, 0.0156, 0.027, 0.0429, 0.064,
0.0911, 0.125, 0.166, 0.216, 0.275, 0.343, 0.422, 0.512, 0.614, 0.729, 0.857,
1.0]
```

Likewise, another way to generate output %o18 above (near the beginning of this section) is to again use **map** as follows:

```
(%i6) g(x) := sin(x*%pi/3);
                                          x %pi
(%o6)                          g(x) := sin(-----)
                                           3
(%i7) jlist : makelist(j,j,1,6);
(%o7)                          [1, 2, 3, 4, 5, 6]
(%i8) map(g,jlist);
                 sqrt(3)  sqrt(3)        sqrt(3)     sqrt(3)
(%o8)           [-------, -------, 0, - -------, - -------, 0]
                    2        2             2           2
(%i9) functions;
(%o9)                             [f(x), g(x)]
```

1.4.4 Anonymous Unnamed Function lamda

An alternative to a separate definition of a named function like `g(x)` above, is to use the anonymous (or un-named) lamda function syntax as follows.

```
(%i10) jlist;
(%o10)                          [1, 2, 3, 4, 5, 6]
(%i11) map(lambda([x],sin(x*%pi/3) ), jlist);
                 sqrt(3)  sqrt(3)        sqrt(3)    sqrt(3)
(%o11)           [-------, -------, 0, - -------, - -------, 0]
                    2        2             2          2
(%i12) functions;
(%o12)                            [f(x), g(x)]
```

This method avoids adding named functions to your work session. In the input `%i9`, `x` is a dummy variable just as in the function definition for `g(x)`, and the square bracket `[x]` entry indicates there is only one input argument, and the next entry shows, given `x`, what the function output should be. Experienced programmers avoid introducing needless names of things like lists and functions by enclosing one function by another, and then perhaps by another, ...etc. As in:

```
(%i13) map(lambda([x],sin(x*%pi/3) ), makelist(j,j,1,6) );
                 sqrt(3)  sqrt(3)        sqrt(3)    sqrt(3)
(%o13)          [-------, -------, 0, - -------, - -------, 0]
                    2        2             2          2
```

Before we leave this discussion of the important Maxima function **makelist**, we want to exhibit a difference one must confront when using named functions vs. named expressions with **makelist**. In the following session, we define a named function `f(x)` and the expression `g` which depends on a parameter `z`. We then try to use each with the **makelist** function.

```
(%i1) v1 : [a,b,c]$
(%i2) f(x) := x^3$
(%i3) g : z^4$
(%i4) makelist(f(y),y,v1);
                                  3   3   3
(%o4)                            [a , b , c ]
(%i5) makelist(g,z,v1);
                                  4   4   4
(%o5)                            [z , z , z ]
(%i6) makelist(''g,z,v1);
                                  4   4   4
(%o6)                            [a , b , c ]
(%i7) functions;
(%o7)                               [f(x)]
```

Note the use of two single quotes before the symbol `g` inside the **makelist** function, which force an extra evaluation and which then gives the desired result. For most work in Maxima, it is usually easier to use named or un-named expressions rather than named functions, but not with **makelist**.

1.4.5 Accessing List Elements Using part(...) or Bracket Pairs

We can access individual elements of a list using either the **part** function (which we discuss in more detail in Ch. 3 (Algebra, Part 2) or (more simply) by using the syntax `mylist[j]`. We can obtain a new list with an element replaced by a different element by using the **substitute** function, or the more convenient alias **subst** name.

```
(%i1) v1 : [a,b,c];
(%o1)                                [a, b, c]
(%i2) v1[2];
(%o2)                                   b
(%i3) part(v1,2);
(%o3)                                   b
(%i4) subst(bb,v1[2], v1);
(%o4)                               [a, bb, c]
(%i5) v1;
(%o5)                                [a, b, c]
```

Let's show how to use **makelist** to create a list of (x,y) pairs from two separate lists of x values and y values of the same length.

```
(%i6) xlist : [1,2,3,4];
(%o6)                             [1, 2, 3, 4]
(%i7) ylist : [1,4,9,16];
(%o7)                             [1, 4, 9, 16]
(%i8) xylist : makelist([xlist[i],ylist[i]],i,1,length(xlist) );
(%o8)                  [[1, 1], [2, 4], [3, 9], [4, 16]]
```

1.4.6 Creating a List of Fractional Values

Here we present a small function which takes an integer `n` which is the number of divisions we want to make for a unit increase in `x`. We can divide (or multiply) the output of `xfrac(n)` by some power of ten to achieve the desired sizes.

```
(%i1) fpprintprec:5$
(%i2) xfrac(n) := makelist(j,j,0,10*n)/float(n)$
(%i3) xfrac(2);
(%o3) [0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5,
                                                 7.0, 7.5, 8.0, 8.5, 9.0, 9.5, 10.0]
(%i4) xfrac(3);
(%o4) [0, 0.333, 0.667, 1.0, 1.3333, 1.6667, 2.0, 2.3333, 2.6667, 3.0, 3.3333,
3.6667, 4.0, 4.3333, 4.6667, 5.0, 5.3333, 5.6667, 6.0, 6.3333, 6.6667, 7.0,
7.3333, 7.6667, 8.0, 8.3333, 8.6667, 9.0, 9.3333, 9.6667, 10.0]
(%i5) xfrac(4)/10.0;
(%o5) [0, 0.025, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175, 0.2, 0.225, 0.25,
0.275, 0.3, 0.325, 0.35, 0.375, 0.4, 0.425, 0.45, 0.475, 0.5, 0.525, 0.55,
0.575, 0.6, 0.625, 0.65, 0.675, 0.7, 0.725, 0.75, 0.775, 0.8, 0.825, 0.85,
0.875, 0.9, 0.925, 0.95, 0.975, 1.0]
```

1.5 Data Files: Read, Write, Fit, and Plot

1.5.1 file_search

When reading and writing data files, we can use a Maxima function to check on the existence of a file. The function **file_search** has the manual entry (in part):

Function: **file_search** (filename)
Function: **file_search** (filename, pathlist)
file_search searches for the file filename and returns the path to the file (as a string) if it can be found; otherwise file_search returns false.
file_search (filename) searches in the default search directories, which are specified by the global variables file_search_maxima, file_search_lisp, and file_search_demo.
file_search first checks if the actual name passed exists, before attempting to match it to "wildcard" file search patterns.
See file_search_maxima concerning file search patterns.
The argument filename can be a path and file name, or just a file name, or, if a file search directory includes a file search pattern, just the base of the file name (without an extension).
For example,

```
file_search ("c:/home/wfs/special/zeta.mac");
file_search ("zeta.mac");
file_search ("zeta");
```

all find the same file, assuming the file exists and c:/home/wfs/special/###.mac is in the global list file_search_maxima.
file_search (filename, pathlist) searches only in the directories specified by pathlist, which is a list of strings.
The argument pathlist supersedes the default search directories, so if the path list is given, file_search searches only the ones specified, and not any of the default search directories.
Even if there is only one directory in pathlist, it must still be given as a one-element list.
The user may modify the default search directories.
See file_search_maxima.

1.5.2 file_search_maxima

Among the global lists of search paths used by some Maxima functions is the option variable: **file_search_maxima**. Here is part of the manual description (edited for a windows user):

Option variable: **file_search_maxima**
Option variable: **file_search_lisp**
Option variable: **file_search_demo**
These variables specify lists of directories to be searched by load, demo, and some other Maxima functions.
The default values of these variables name various directories in the Maxima installation.
The user can modify these variables, either to replace the default values or to append additional directories.
For example,

```
file_search_maxima: ["c:/usr/local/foo/###.mac",  "c:/usr/local/bar/###.mac"]$
```

replaces the default value of file_search_maxima, while

```
file_search_maxima: append (file_search_maxima,
                    ["c:/usr/local/foo/###.mac", "c:/usr/local/bar/###.mac"])$
```

appends two additional directories.
It may be convenient to put such an expression in the file maxima-init.mac, so that the file search path is assigned automatically when Maxima starts.

1.5.3 maxima-init.mac and maxima_userdir

If you have not redefined the Maxima string **maxima_userdir** in your startup file or work session, you can find where Maxima looks for the startup file as follows:

```
(%i1) maxima_userdir;
(%o1)           C:/Documents and Settings/Edwin Woollett/maxima
```

In my startup file, placed where Maxima expects to find it, I have redefined the value of this Maxima string, in the hope that this will simplify my interaction with Maxima files. Output files for plotting are supposed to be written to the directory named by `maxima_tempdir`.

Here is my current startup file, `maxima-init.mac`, a text file written with notepad2, and left with dos-windows line endings (CR LF) (note that it is easy to change a dos file to a unix file with line endings (LF) or a mac file with line endings (CR), using the menu selection: File, Line Endings):

```
/* this is c:\Documents and Settings\Edwin Woollett\maxima\maxima-init.mac  */

maxima_userdir: "c:/work2" $
maxima_tempdir : "c:/work2"$
file_search_maxima : append(["c:/work2/###.{mac,mc,out}"],file_search_maxima )$
file_search_usage :  append(["c:/work2/###.{mac,mc,out}"],file_search_usage )$
disp("hello  ted")$
```

Note that I must use the unix like forward slash `/`, even though the dos -Windows convention is the backward slash `\`. With this startup file in place, I get the following response from XMaxima:

```
(%i1) maxima_userdir;
(%o1)                              c:/work2
(%i2) file_search_maxima;
(%o2) [c:/work2/###.{mac,mc,out},
C:/Documents and Settings/Edwin Woollett/maxima/###.{mac,mc},
C:\PROGRA~1\MAXIMA~3.0/share/maxima/5.14.0/share/###.{mac,mc}, C:\PROGRA~1\MAX\
IMA~3.0/share/maxima/5.14.0/share/{affine,algebra,algebra/charsets,algebra/sol\
ver,calculus,combinatorics,contrib,contrib/boolsimp,contrib/descriptive,contri\
b/diffequations,contrib/diffequations/tests,contrib/distrib,contrib/ezunits,co\
ntrib/format,contrib/fractals,contrib/fresnel,contrib/gentran,contrib/gentran/\
test,contrib/gf,contrib/graphs,contrib/Grobner,contrib/levin,contrib/lurkmathm\
l,contrib/maximaMathML,contrib/mcclim,contrib/numericalio,contrib/pdiff,contri\
b/prim,contrib/rand,contrib/sarag,contrib/simplex,contrib/simplex/Tests,contri\
b/solve_rec,contrib/state,contrib/stats,contrib/stringproc,contrib/vector3d,co\
ntrib/unit,contrib/Zeilberger,diff_form,diffequations,dynamics,draw,lapack,lbf\
gs,linearalgebra,integequations,integration,macro,matrix,misc,numeric,orthopol\
y,physics,simplification,sym,tensor,tensor/tests,trigonometry,utils,vector}/##\
#.{mac,mc}]
```

1.5.4 Viewing a File with printfile

In addition to the startup file contents just discussed, I also have shortcut links to the XMaxima binary (`xmaxima.exe`) and the Maxima console batch file (`maxima.bat`) located in my work folder `c:\work2\`. By starting up Maxima using these shortcut links from my work folder, I again simplify locating Maxima related files.

With the above described "setup", we can practice looking for and viewing a simple dos formatted text file called `windows-user.txt`, which is stored in the `c:\work2\` folder, and is a memo recalling a comment made on the Maxima mailing list.

```
(%i1) file_search("windows-user.txt");
(%o1)                              windows-user.txt
(%i2) printfile("windows-user.txt")$
As a dedicated windows xp user who is delighted to have windows binaries
do the install work and worry about subtleties, I would like to assure
all who are considering windows use that there is no problem with keeping previous
versions of Maxima around.
```

The manual entry for **file_search** `(filename)` asserts that this syntax returns the path to the requested file if the file is found. We see that if the file is found in the working directory as I have set it up (`c:\work2\`), all that is returned is the name of the file.

1.5.5 Creating a Data File using with_stdout

Let's use the file name "tmp.dat", first checking whether such a file already exists:

```
(%i2) file_search("tmp.dat");
(%o2)                                   false
(%i3) with_stdout( "tmp.dat", for i:1 thru 4 do print(i, i^2, i^3) )$
(%i4) file_search("tmp.dat");
(%o4)                                  tmp.dat
(%i5) printfile("tmp.dat")$
1 1 1
2 4 8
3 9 27
4 16 64
```

Opening the file tmp.dat with notepad2, we see that **with_stdout** creates a unix formatted file, with (LF) line endings.

1.5.6 Creating a List from a Data File: read_nested_list

Given a data file, like tmp.dat, we can create a Maxima nested list containing the data items on each line of the file as a separate sublist. For this we use **read_nested_list**.

```
(%i6) datalist1 : read_nested_list("tmp.dat");
(%o6)               [[1, 1, 1], [2, 4, 8], [3, 9, 27], [4, 16, 64]]
```

The function **read_nested_list** can read a text file also, first for a case where there are no punctuation marks:

```
(%i7) printfile("text.dat")$
this is line one
this is line two
this is line three
(%i8) textdata : read_nested_list("text.dat");
(%o8)   [[this, is, line, one], [this, is, line, two], [this, is, line, three]]
```

Here is the same text but now each line ends with a period, and the file is called text2.dat:

```
(%i9) printfile("text2.dat")$
this is line one.
this is line two.
this is line three.
(%i10) textdata2 : read_nested_list("text2.dat");
(%o10) [[this, is, line, one, .], [this, is, line, two, .], [this, is, line, three, .]]
```

We see that the period is treated like a separate "atom".

```
(%i11) pp : part(textdata2,1,5);
(%o11)                                          .
(%i12) atom(pp);
(%o12)                                         true
(%i13) :lisp $pp
                                                $.
```

1.5.7 Write List to Data File with write_data

We use the syntax **write_data** `(list,filename)`.

```
(%i14) datalist1;
(%o14)                 [[1, 1, 1], [2, 4, 8], [3, 9, 27], [4, 16, 64]]
(%i15) write_data(datalist1,"tmp2.dat")$
(%i16) printfile("tmp2.dat")$
1 1 1
2 4 8
3 9 27
4 16 64
```

We find again that **write_data** creates a unix formatted data file.

1.5.8 Using Random Numbers to Create Noisy Data

Here we start with a clean signal and add a small random value to create a noisy data file. The Maxima function **random** `(n)`, for `n` an integer, produces a psuedo-random number in the range `[0,n-1]`. With the call **random** `(x)`, for `x` a floating point number, we get a psuedo-random number y in the range $0 < y < x$. You can get the same set of random numbers as shown in these notes provided you begin with the same initial Maxima input `%i1`, which starts off the random number generator with a "seed" related to the input integer. (We are following the manual's example of seeding the generator).

```
(%i1) set_random_state ( make_random_state (654321) )$
```

We can then generate 20 random numbers in the range $0 \leq n \leq 10$.

```
(%i2) makelist(random(11),j,1,20 );
(%o2)     [5, 1, 10, 8, 7, 2, 4, 8, 1, 1, 6, 5, 2, 0, 7, 6, 3, 3, 1, 8]
```

and another 20 random numbers in the range $0 < x < 1.0$ (to avoid looking all the digits being used, we used the global floating point option variable **fpprintprec**, which by default (val = 0) shows all the digits:

```
(%i3) fpprintprec : 5$
(%i4) makelist(random(1.0),j,1,20 );
(%o4) [0.478, 0.714, 0.255, 0.962, 0.839, 0.849, 0.212, 0.915, 0.956, 0.23,
          0.715, 0.207, 0.631, 0.0784, 0.436, 0.405, 0.759, 0.697, 0.93, 0.563]
```

and, finally, 20 random numbers in the range $-0.2 < x < 0.2$:

```
(%i5) makelist( -0.2 + random(0.4), j, 1, 20 );
(%o5) [- 0.14, 0.0716, - 0.146, - 0.101, - 0.0415, 0.167, 0.133, - 0.173,
0.0156, 0.00882, - 0.0301, - 0.0171, 0.00596, 0.133, - 0.0332, - 0.0884,
- 0.186, - 0.0337, - 0.0721, - 0.0756]
```

Let's practice adding noise to a straight line of the form $y = x + 1$. We will create the data file fit1.dat containing ten pairs (x, y), where $y = x+1+r$, where $r = -0.2+random(0.4)$ will be a number in the range $-0.2 < r < 0.2$. We need to seed the random number generator (as above) before writing the (x, y) pairs to our data file.

Let's do both jobs with a user defined function that uses the **block** function.

```
(%i6) doline():=block([x,y,r],
           set_random_state (make_random_state (654321)),
           with_stdout( "fit1.dat" , for i:1 thru 10 do (
      x : i,
              r: -0.2 + random(0.4),
              y: 1 + x + r,
              print(x,y)
           )    /* end do  */
           )    /* end with_stdout */
        )$
```

We need the parentheses at the end of the text "doline", even though we are not providing any input parameters to the function. Now we "run" the function `doline()` to create the data file with the noisy data.

```
(%i7) doline()$
```

Now check the file contents:

```
(%i8) printfile("fit1.dat")$
1 1.8904
2 3.0708
3 3.9215
4 5.1813
5 5.9443
6 7.0156
7 7.8441
8 8.8806
9 9.8132
10 11.129
```

The square bracket `[]` which is the first argument to **block** holds variables which are "local" in the sense that their values are not known globally. Of course we can check that assertion:

```
(%i9) [x,y,r];
(%o9)                                [x, y, r]
```

We see that the three variables used are not globally bound.

1.5.9 Using lsquares_estimates for a Fit

We now use the created data file, fit1.dat, to practice reading a data file, and finding a straight line "fit" to the data, and then construct a plot which includes the data points and the best fit line generated by the least squares routine.

The Maxima function **lsquares_estimates** `(datamat,varlist,eqn,paramlist)` can be used if we first use **load** `(lsquares)`. This is the simplest version with the least number of arguments: other optional arguments are available, which we discuss later. `datamat` is a **matrix** in which each row is one (x, y) pair(for the case we are interested in here), so our matrix will have ten rows and two columns. `varlist` will be the variables we are trying to relate with the equation we provide, and `paramlist` is a list of the equation parameters to be estimated from the provided data. Normally, the larger the data set, (in our case, the more samples from the noisy straight line), the more accurate the determination of the equation parameters. Since we need to convert our data file to a Maxima matrix, we can use the function **read_matrix** `(filename)`. In the default display, the output looks like a usual matrix. We can get the one dimensional output using the **grind** function. Notice that the **grind** function puts a dollar sign $ at the end of its output.

```
(%i10) datamatrix: read_matrix("fit1.dat");
                                  [ 1   1.8904 ]
                                  [            ]
                                  [ 2   3.0708 ]
                                  [            ]
                                  [ 3   3.9215 ]
                                  [            ]
                                  [ 4   5.1813 ]
                                  [            ]
                                  [ 5   5.9443 ]
(%o10)                            [            ]
                                  [ 6   7.0156 ]
                                  [            ]
                                  [ 7   7.8441 ]
                                  [            ]
                                  [ 8   8.8806 ]
                                  [            ]
                                  [ 9   9.8132 ]
                                  [            ]
                                  [ 10  11.129 ]
(%i11) grind(%)$
matrix([1,1.8904],[2,3.0708],[3,3.9215],[4,5.1813],[5,5.9443],[6,7.0156],
       [7,7.8441],[8,8.8806],[9,9.8132],[10,11.129])$
```

We now use the **lsquares** Maxima package to let the function **lsquares_estimates** find the best fit parameters `(a,b)` which our noisy data imply for the data model $y(x) = a\,x + b$.

```
(%i12) load(lsquares);
(%o12) C:/PROGRA~1/MAXIMA~3.0/share/maxima/5.14.0/share/contrib/lsquares.mac
```

```
(%i13) lsquares_estimates( datamatrix, [x, y], y = a*x + b, [a, b] );
                          5203976222276543      15290146019
(%o13)             [[a = ----------------, b = -----------]]
                          5229349671150000      15355149375
(%i14) soln : float(%)[1];
(%o14)                       [a = 0.995, b = 0.996]
(%i15) [a,b] : map( rhs, soln)$
(%i16) [a,b];
(%o16)                            [0.995, 0.996]
```

We see that with ten (noisy) data points, the least squares algorithm has come close to our initial parameters of the noiseless function $y(x) = x + 1$.

We now use **plot2d** to show the ten points of data along with the least squares fit line. First we convert the data matrix to a nested data list (see the next section for more discussion).

```
(%i17) datalist : substpart("[", datamatrix, 0 );
(%o17) [[1, 1.8904], [2, 3.0708], [3, 3.9215], [4, 5.1813], [5, 5.9443],
             [6, 7.0156], [7, 7.8441], [8, 8.8806], [9, 9.8132], [10, 11.129]]
(%i18) plot2d( [ a*x+b, [discrete,datalist] ], [ x, 0, 10] ,
             [style, [lines], [points] ],
            [legend, "fit", "data"] )$
```

Your plot should look roughly like:

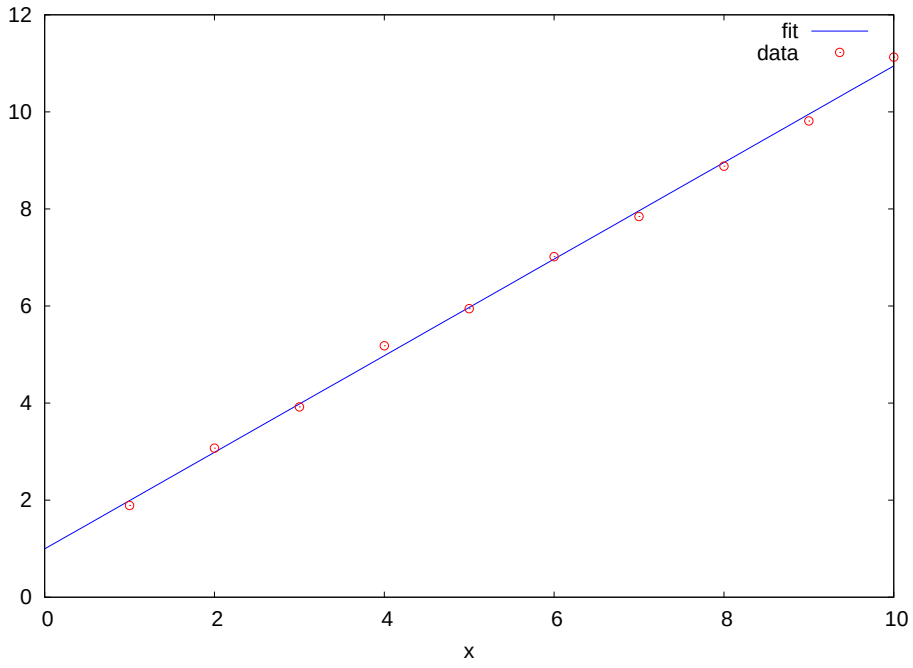

Figure 1: leastsquares fit no 1

1.5.10 Nested List ↔ Matrix using apply and substpart

Since the function **lsquares_estimates** requires a **matrix** object as the first argument, we present two methods of converting a nested data list to a **matrix** object.

The first and easiest method uses **apply**. Since we are only interested here in the basic ideas, we define a "toy" data list with two (x, y) pairs called "mylist".

```
(%i1) mylist : [[a,b],[c,d]];
(%o1)                          [[a, b], [c, d]]
(%i2) mymatrix : apply('matrix, mylist);
                                    [ a  b ]
(%o2)                               [      ]
                                    [ c  d ]
(%i3) grind(mymatrix)$
matrix([a,b],[c,d])$
```

The second method is based on the Maxima function **substpart**, which we discuss in more detail in Ch. 3 (Algebra, Part 2), along with the related Maxima function **part**. We first explore the naming of the parts of mylist using **part**, and then use the function **substpart** to get the desired **matrix** object.

```
(%i4) length(mylist);
(%o4)                                  2
(%i5) makelist(part(mylist,j),j,0,2 );
(%o5)                        [[, [a, b], [c, d]]
(%i6) part(mylist,0);
(%o6)                                  [
(%i7) mymatrix : substpart(matrix,mylist,0);
                                    [ a  b ]
(%o7)                               [      ]
                                    [ c  d ]
(%i8) grind(mymatrix)$
                           matrix([a,b],[c,d])$
(%i9) part(mymatrix,0);
(%o9)                               matrix
```

We can transform a **matrix** object into a nested list, again using **substpart**:

```
(%i10) substpart("[", mymatrix, 0 );
(%o10)                         [[a, b], [c, d]]
```

1.5.11 The Syntax of lsquares_estimates

If you look at the Maxima manual entry for **lsquares_estimates**, you find the syntax presented as:

> Function: **lsquares_estimates**`(D, x, e, a)`
> Function: **lsquares_estimates**`(D, x, e, a, initial = L, tol = t)`
> Estimate parameters `a` to best fit the equation `e` in the variables `x` and `a` to the data `D`, as determined by the method of least squares. `lsquares_estimates` first seeks an exact solution, and if that fails, then seeks an approximate solution.
>
> The return value is a list of lists of equations of the form `[a = ..., b = ..., c = ...]`. Each element of the list is a distinct, equivalent minimum of the mean square error. The data `D` must be a matrix. Each row is one datum (which may be called a 'record' or 'case' in some contexts), and each column contains the values of one variable across all data. The list of variables `x` gives a name for each column of `D`,

even the columns which do not enter the analysis. The list of parameters `a` gives the names of the parameters for which estimates are sought. The equation `e` is an expression or equation in the variables `x` and `a`; if `e` is not an equation, it is treated the same as `e = 0`.

Additional arguments to `lsquares_estimates` are specified as equations and passed on verbatim to the function `lbfgs` which is called to find estimates by a numerical method when an exact result is not found.

If some exact solution can be found (via solve), the data `D` may contain non-numeric values. However, if no exact solution is found, each element of `D` must have a numeric value. This includes numeric constants such as `%pi` and `%e` as well as literal numbers (integers, rationals, ordinary floats, and bigfloats). Numerical calculations are carried out with ordinary floating-point arithmetic, so all other kinds of numbers are converted to ordinary floats for calculations.

The second example shown for **lsquares_estimates** in the manual uses an optional argument not mentioned in the syntax listing: `iprint = [i, j]`. The form `iprint = [-1, 0]`, which we use below, surpresses progress messages as the numerical program **lbfgs** looks for a solution. You can see a complete description of this option list in the manual entry for **lbfgs**. The acronym **lbfgs** comes from the names: Broyden, Fletcher, Goldfarb, and Shanno, and the initial "l" comes from the "limited memory" version by J. Nocedal of the Department of Electrical Engineering and Computer Science at Norwestern University.

Many examples of the use of the **lsquares** package are found in the file lsquares.mac, which is found in the ...share/contrib folder. You can also see great examples of efficient programming in the Maxima language in that file.

1.5.12 Fit to a More Complicated Equation

Here is a record of creating a noisy data set from the function $r + 2\,e^{-3x}$, where r is a small random number, and finding a least squares fit. We create the data set and use it without writing and reading a data file. For the creation of datalist in `%i6` we could have used **makelist** as previously described.

```
(%i1) fpprintprec : 5 $
(%i2) xlist : float( makelist(j/10,j,0,10 ) );
(%o2)       [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
(%i3) set_random_state ( make_random_state (654321) )$
(%i4) y(z) := -0.2 + random(0.4) + 2*exp(-3*z)$
(%i5) ylist : map(y,xlist);
(%o5) [1.8904, 1.5524, 1.0192, 0.994, 0.547, 0.462, 0.175, 0.125, - 0.00536,
                                                                  0.264, 0.0908]
(%i6) datalist : map( "[", xlist, ylist );
(%o6) [[0.0, 1.8904], [0.1, 1.5524], [0.2, 1.0192], [0.3, 0.994],
[0.4, 0.547], [0.5, 0.462], [0.6, 0.175], [0.7, 0.125], [0.8, - 0.00536],
[0.9, 0.264], [1.0, 0.0908]]
(%i7) datamatrix : apply('matrix, datalist )$
(%i8) grind(datamatrix)$
matrix([0.0,1.8904],[0.1,1.5524],[0.2,1.0192],[0.3,0.994],[0.4,0.547],
       [0.5,0.462],[0.6,0.175],[0.7,0.125],[0.8,-0.00536],[0.9,0.264],
       [1.0,0.0908])$
(%i9) load(lsquares);
(%o9) C:/PROGRA~1/MAXIMA~3.0/share/maxima/5.14.0/share/contrib/lsquares.mac
(%i10) solnlist: lsquares_estimates(datamatrix,[x,y],y=a*exp(-b*x),[a,b],iprint=[-1,0]
(%o10)                    [[a = 1.9638, b = 3.0296]]
(%i11) solnlist : solnlist[1];
```

```
(%o11)                      [a = 1.9638, b = 3.0296]
(%i12) [a,b] : map( rhs, solnlist )$
(%i13) [a,b];
(%o13)                          [1.9638, 3.0296]
(%i14) plot2d( [ a*exp(-b*x), [discrete,datalist] ], [ x, 0, 1] ,
              [style, [lines], [points] ],
             [legend, "fit", "data"]  )$
```

Maxima draws a plot close to the following figure.

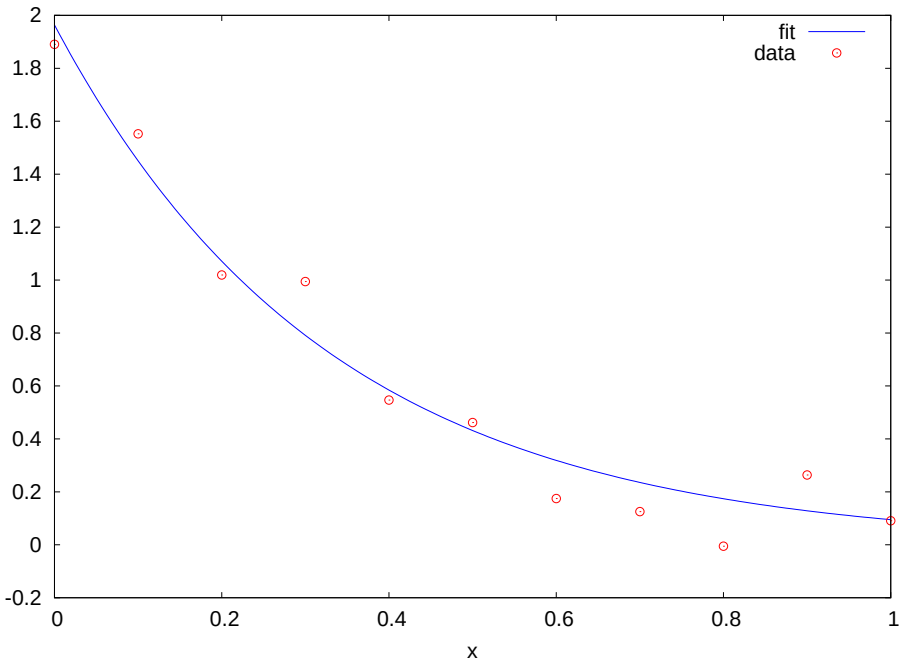

Figure 2: leastsquares fit no 2

1.5.13 plot2d: Squaring the Circle

There are times when you need to plot a circle or a square, and you may have to fiddle with the `x` and `y` ranges to get the circle to be "round". In the Maxima manual, under plot2d, there is an example of such a "geometric plot" with the following code:

```
(%i1) plot2d ([parametric, cos(t), sin(t), [t,-%pi,%pi],
      [nticks,80]], [x, -4/3, 4/3])$
```

Since $\cos(t)$ and $\sin(t)$ each have values in the range $[-1, 1]$, this parametric plot produces a circle of radius 1, and the default `y-axis` range is the same. The `x-axis` range is forced to be $[-4/3, 4/3]$ to get a round (?) circle. To experiment with getting a circle, we draw a unit circle enclosed by a "square" box, and define a function which allows the user to input a ratio "r" which will be the ratio of the `x-range` to the `y-range`. To have a record of this experiment and allow for more plot options, we created a file `start.mac`, with the following contents, which includes the results of our experiment expanding the plot window to full screen. We then used the **load** function to load the definitions into our work session.

Here are the contents of my file "start.mac".

```
/* file start.mac */
print("start.mac: try to  adjust parameter r to get round circle of radius 1")$
```

```
print("squarelist")$
squarelist : [ [-1,1],[1,1],[1,-1],[-1,-1], [-1,1] ]$
print("docircle(r)")$
docircle(r) :=
    block([dy:1.5,dx ],
     dx : r*dy,display([dy,dx]),
        plot2d ( [
      [discrete,squarelist],
            [parametric, cos(t), sin(t), [t,-%pi,%pi],[nticks,80]]  ],
[style, [lines,2,0], [lines,2,0] ],
[x, -dx,dx ],
[y, -dy,dy],
            [gnuplot_preamble,
   " set xzeroaxis lw 2; set yzeroaxis lw 2; set nokey "]
        )
)$

print("starting value for r is 4/3 = 1.333  ")$

/* search for full screen  round circle with r:
 delx, dely in centimeters

r delx dely delx/dely (delx/dely -1)

4/3 13.1 12.3 1.065 0.065
1.4 12.5 12.3 1.02 0.02
1.43 12.24 12.4 0.99 -0.01

record of use
(%i1) load("start.mac");
start.mac: try to  adjust parameter r to get round circle of radius 1
squarelist
docircle(r)
starting value for r is 4/3 = 1.333
(%o1)                                start.mac
(%i2) docircle(1.333)$
                           [dy, dx] = [1.5, 1.9995]

(%o2)
(%i3) docircle(1.43)$
                            [dy, dx] = [1.5, 2.145]

*/
```

We conclude that the value `1.43` gives a rounder circle for our hardware.

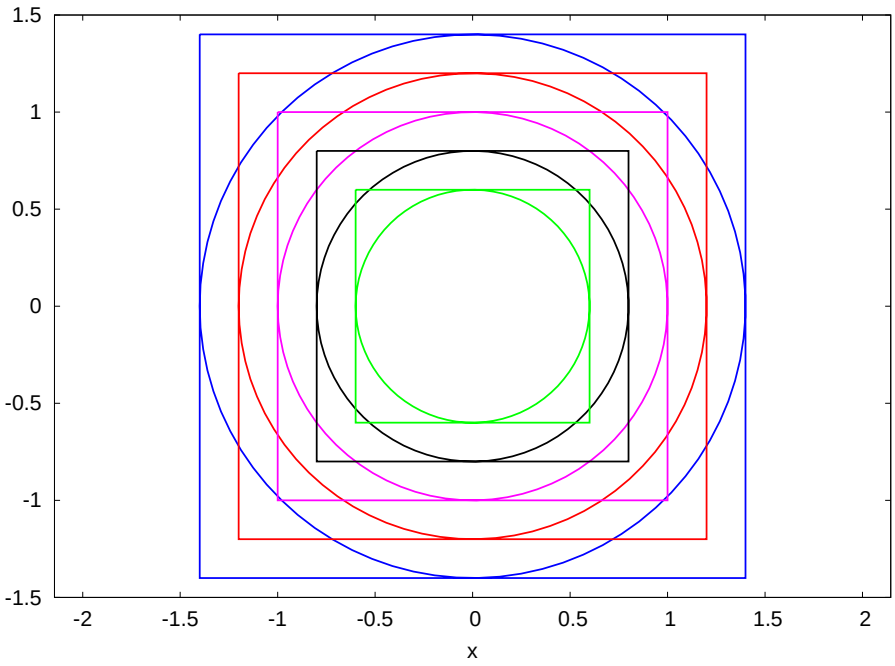

Figure 3: boxed circles

Using that value, here is a more colorful set of boxed circles. The element `[line,nw,nc]` which should be included for each plot element allows thicker lines (nw = 1, the default, is thin), and color choice nc with different colors for nc = 0,1,2,3,4,5,6,7=0, 8 = 1, etc.

Here is the code for the boxed circles in color (written in a work file start.mac and loaded into XMaxima:

```
/* console version  */
print("console version squarelist(r),dsq(r),circle(r), docircles()")$

squarelist(r) := [ [-r,r],[r,r],[r,-r],[-r,-r], [-r,r] ]$

dsq(r) := [discrete, squarelist(r) ]$

circle(r) := [parametric, r*cos(t), r*sin(t), [t,-%pi,%pi],[nticks,80]]$

docircles() :=
   block([dy:1.5,dx ],
      dx : 1.43*dy, display([dy,dx]),
      plot2d ( [  dsq(1.4), circle(1.4),dsq(1.2), circle(1.2),dsq(1.0),circle(1.0),
         dsq(0.8),circle(0.8),dsq(0.6),circle(0.6) ],
         [style, [lines,4,1], [lines,4,1], [lines,4,2],[lines,4,2],[lines,4,3],
            [lines,4,3],[lines,4,4],[lines,4,4], [lines,4,6],[lines,4,6] ],
         [x, -dx,dx ],
         [y, -dy,dy],
         [gnuplot_preamble, " set nokey "]
        )
)$
```

And record of use:

```
(%i2) load("start.mac");
console version squarelist(r),dsq(r),circle(r), docircles()
```

```
(%o2)                                start.mac
(%i3) docircles()$
                              [dy, dx] = [1.5, 2.145]
```

1.6 Latex Figures from Maxima plot2d

If you want to include figures (in a Tex file) related to your Maxima work, you can add two more elements to the **plot2d** arguments which will create an "eps" file. Here we illustrate using the first least square fit we did.

```
(%i21) plot2d( [ a*x+b, [discrete,datalist] ], [ x, 0, 10] ,
              [style, [lines], [points] ],
             [legend, "fit", "data"],
             [gnuplot_term,ps], [gnuplot_out_file,"c:/work2/fit1.eps"])$
```

You should get a silent response, and can then check your work directory for the presence of the new file "fit1.eps". You can then include that eps file as a figure in your Tex file with the lines:

```
\smallskip
%%\begin{figure}[!ht]
\begin{figure} [h]
   \centerline{\includegraphics[width=5in]{fit1.eps} }
\caption{leastsquares fit no 1}
\end{figure}
```

This works for me, with the Tex file context established by the following lines at the top of the Tex file `mbe1-start.tex`:

```
\documentclass[12pt]{article}
\usepackage[dvips,top=1.5cm,left=1.5cm,right=1.5cm,foot=1cm,bottom=1.5cm]{geometry}
\usepackage{times}
\usepackage{amsmath}
\usepackage{amsbsy}
\usepackage{graphicx}
\usepackage{url}
\urldef\tedhome\url{ http://www.csulb.edu/~woollett/  }
\urldef\tedmail\url{ woollett@charter.net}

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   title page
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\title{Maxima by Example: Ch. 1, Getting Started \thanks{Date: June 18, 2008.
   This version uses Maxima 5.15. Check \;  \tedhome \; for the latest
   version of these notes. Send comments and suggestions to \tedmail } }

%\author{Edwin L. Woollett \thanks{Dept. of Physics and Astron, Calif.
                     State University at Long Beach,\\
%                     author's email: \tedmail} }
\author{Edwin L. Woollett}
\date{}
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%          document
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
\begin{document}
%\SMALL
\small
\maketitle
\tableofcontents
\numberwithin{equation}{section}
\newpage
%\setcounter{section}{3}
COPYING AND DISTRIBUTION POLICY\\
\newpage
\normalsize
\section{Introduction}
\subsection{What is Maxima?}
```