Introduction to Maxima

Richard H. Rand
Dept. of Theoretical and Applied Mechanics, Cornell University *

Copyright (c) 1988-2010 Richard H. Rand.
This document is free; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation. See the GNU General Public License for more details at http://www.gnu.org/copyleft/gpl.html

Contents

*Adapted from "Perturbation Methods, Bifurcation Theory and Computer Algebra" by Rand and Armbruster, Springer, 1987. Adapted to LaTeX and HTML by Nelson L. Dias (nldias@simepar.br), SIMEPAR Technological Institute and Federal University of Paraná, Brazil. Updated by Robert Dodier, August 2005.

1 Introduction

To invoke Maxima in a console, type

```
maxima <enter>
```

The computer will display a greeting of the sort:

```
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
This is a development version of Maxima. The function bug_report()
provides bug reporting information.
(%i1)
```

The `(%i1)` is a "label". Each input or output line is labelled and can be referred to by its own label for the rest of the session. `i` labels denote your commands and `o` labels denote displays of the machine's response. *Never use variable names like* `%i1` *or* `%o5`*, as these will be confused with the lines so labeled.*

Maxima distinguishes lower and upper case. All built-in functions have names which are lowercase only (`sin`, `cos`, `save`, `load`, etc). Built-in constants have lowercase names (`%e`, `%pi`, `inf`, etc). If you type `SIN(x)` or `Sin(x)`, Maxima assumes you mean something other than the built-in `sin` function. User-defined functions and variables can have names which are lower or upper case or both. `foo(XY)`, `Foo(Xy)`, `FOO(xy)` are all different.

2 Special keys and symbols

1. To end a Maxima session, type `quit();`.

2. To abort a computation without leaving Maxima, type `^C`. (Here `^` stands for the control key, so that `^C` means first press the key marked control and hold it down while pressing the C key.) It is important for you to know how to do this in case, for example, you begin a computation which is taking too long. For example:

   ```
   (%i1) sum (1/x^2, x, 1, 100000)$
   ^C
   Maxima encountered a Lisp error:

    Interactive interrupt at #x7FFFF74A43C3.

   Automatically continuing.
   To enable the Lisp debugger set *debugger-hook* to nil.
   (%i2)
   ```

3. In order to tell Maxima that you have finished your command, use the semicolon (`;`), followed by a return. Note that the return key alone does not signal that you are done with your input.

4. An alternative input terminator to the semicolon (`;`) is the dollar sign (`$`), which, however, supresses the display of Maxima's computation. This is useful if you are computing some long intermediate result, and you don't want to waste time having it displayed on the screen.

5. If you wish to repeat a command which you have already given, say on line `(%i5)`, you may do so without typing it over again by preceding its label with two single quotes (`''`), i.e., `''%i5`. (Note that simply inputing `%i5` will not do the job — try it.)

6. If you want to refer to the immediately preceding result computed by Maxima, you can either use its `o` label, or you can use the special symbol percent (`%`).

7. The standard quantities e (natural log base), i (square root of -1) and π (3.14159...) are respectively referred to as `%e`, `%i`, and `%pi`. Note that the use of `%` here as a prefix is completely unrelated to the use of `%` to refer to the preceding result computed.

8. In order to assign a value to a variable, Maxima uses the colon (`:`), not the equal sign. The equal sign is used for representing equations.

3 Arithmetic

The common arithmetic operations are

`+` addition

`-` subtraction

`*` scalar multiplication

`/` division

`^` or `**` exponentiation

`.` matrix multiplication

`sqrt(x)` square root of `x`.

Maxima's output is characterized by exact (rational) arithmetic. For example,

```
(%i1) 1/100 + 1/101;
                                201
(%o1)                          -----
                               10100
```

If irrational numbers are involved in a computation, they are kept in symbolic form:

```
(%i2) (1 + sqrt(2))^5;
                                        5
(%o2)                       (sqrt(2) + 1)
(%i3) expand (%);
                           7/2
(%o3)                 3 2    + 5 sqrt(2) + 41
```

However, it is often useful to express a result in decimal notation. This may be accomplished by following the expression you want expanded by "`,numer`":

```
(%i4) %, numer;
(%o4)                    82.01219330881977
```

Note the use here of `%` to refer to the previous result. In this version of Maxima, `numer` gives 16 significant figures, of which the last is often unreliable. However, Maxima can offer *arbitrarily high precision* by using the `bfloat` function:

```
(%i5) bfloat (%o3);
(%o5)                   8.201219330881976b1
```

The number of significant figures displayed is controlled by the Maxima variable `fpprec`, which has the default value of 16:

```
(%i6) fpprec;
(%o6)                             16
```

Here we reset `fpprec` to yield 100 digits:

```
(%i7) fpprec: 100;
(%o7)                             100
(%i8) ''%i5;
(%o8) 8.201219330881975641524897300208124427852048438593149412\
21237124017312418754011041266612384955016056 1b1
```

Note the use of two single quotes (`''`) in `(%i8)` to repeat command `(%i5)`. Maxima can handle very large numbers without approximation:

```
(%i9) 100!;
(%o9) 933262154439441526816992388562667004907159682643816214685\
929638952175999932299156089414639761565182862536979208272237582\
51185210916864000000000000000000000000
```

4 Algebra

Maxima's importance as a computer tool to facilitate analytical calculations becomes more evident when we see how easily it does algebra for us. Here's an example in which a polynomial is expanded:

```
(%i1) (x + 3*y + x^2*y)^3;
                           2            3
(%o1)                     (x  y + 3 y + x)
(%i2) expand (%);
       6  3      4  3       2  3       3      5  2       3  2
(%o2) x  y  + 9 x  y  + 27 x  y  + 27 y  + 3 x  y  + 18 x  y
                                      2      4        2      3
                             + 27 x y  + 3 x  y + 9 x  y + x
```

Now suppose we wanted to substitute `5/z` for `x` in the above expression:

```
(%i3) %o2, x=5/z;
           2        3                 2               3
      135 y    675 y    225 y   2250 y    125   5625 y    1875 y
(%o3) ------ + ------ + ----- + ------- + --- + ------- + ------
        z         2        2        3       3       4         4
                  z        z        z       z       z         z
                                                 2         3
                                           9375 y    15625 y        3
                                         + ------- + -------- + 27 y
                                             5          6
                                            z          z
```

The Maxima function `ratsimp` will place this over a common denominator:

```
(%i4) ratsimp (%);
           3  6        2  5         3           4
(%o4) (27 y  z  + 135 y  z  + (675 y  + 225 y) z
          2         3          3            2         2
 + (2250 y  + 125) z  + (5625 y  + 1875 y) z  + 9375 y  z
          3   6
 + 15625 y )/z
```

Expressions may also be `factor`ed:

```
(%i5) factor (%);
                           2             3
                    (3 y z  + 5 z + 25 y)
(%o5)               ---------------------
                              6
                             z
```

Maxima can obtain exact solutions to systems of nonlinear algebraic equations. In this example we `solve` three equations in the three unknowns `a`, `b`, `c`:

```
(%i6) a + b*c = 1;
(%o6)                       b c + a = 1
(%i7) b - a*c = 0;
(%o7)                       b - a c = 0
(%i8) a + b = 5;
(%o8)                        b + a = 5
(%i9) solve ([%o6, %o7, %o8], [a, b, c]);
            25 sqrt(79) %i + 25      5 sqrt(79) %i + 5
(%o9) [[a = -------------------, b = -----------------,
            6 sqrt(79) %i - 34       sqrt(79) %i + 11
    sqrt(79) %i + 1        25 sqrt(79) %i - 25
c = ---------------], [a = -------------------,
          10               6 sqrt(79) %i + 34
    5 sqrt(79) %i - 5        sqrt(79) %i - 1
b = -----------------, c = - ---------------]]
    sqrt(79) %i - 11              10
```

Note that the display consists of a "list", i.e., some expression contained between two brackets `[ ...]`, which itself contains two lists. Each of the latter contain a distinct solution to the simultaneous equations.

Trigonometric identities are easy to manipulate in Maxima. The function `trigexpand` uses the sum-of-angles formulas to make the argument inside each trig function as simple as possible:

```
(%i10) sin(u + v) * cos(u)^3;
                            3
(%o10)                  cos (u) sin(v + u)
(%i11) trigexpand (%);
                3
(%o11)       cos (u) (cos(u) sin(v) + sin(u) cos(v))
```

The function `trigreduce`, on the other hand, converts an expression into a form which is a sum of terms, each of which contains only a single `sin` or `cos`:

```
(%i12) trigreduce (%o10);
       sin(v + 4 u) + sin(v - 2 u)   3 sin(v + 2 u) + 3 sin(v)
(%o12) --------------------------- + ------------------------
                    8                             8
```

The functions `realpart` and `imagpart` will return the real and imaginary parts of a complex expression:

```
(%i13) w: 3 + k*%i;
(%o13)                        %i k + 3
(%i14) w^2 * %e^w;
                                 2   %i k + 3
(%o14)                (%i k + 3)  %e
(%i15) realpart (%);
                3       2               3
(%o15)         %e  (9 - k ) cos(k) - 6 %e  k sin(k)
```

5 Calculus

Maxima can compute derivatives and integrals, expand in Taylor series, take limits, and obtain exact solutions to ordinary differential equations. We begin by defining the symbol `f` to be the following function of `x`:

```
(%i1) f: x^3 * %e^(k*x) * sin(w*x);
                          3   k x
(%o1)                    x  %e    sin(w x)
```

We compute the derivative of `f` with respect to `x`:

```
(%i2) diff (f, x);
         3   k x                2   k x
(%o2) k x  %e    sin(w x) + 3 x  %e    sin(w x)
                                                  3   k x
                                             + w x  %e    cos(w x)
```

Now we find the indefinite integral of `f` with respect to `x`:

```
(%i3) integrate (f, x);
           6      3  4      5  2    7   3
(%o3) (((k w  + 3 k  w  + 3 k  w  + k ) x
       6      2  4      4  2      6   2
 + (3 w  + 3 k  w  - 3 k  w  - 3 k ) x
            4       3  2      5         4       2  2      4
 + (- 18 k w  - 12 k  w  + 6 k ) x - 6 w  + 36 k  w  - 6 k )
   k x                  7      2  5      4  3    6     3
 %e    sin(w x) + ((- w  - 3 k  w  - 3 k  w  - k  w) x
         5       3  3      5     2
 + (6 k w  + 12 k  w  + 6 k  w) x
       5       2  3       4              3       3      k x
 + (6 w  - 12 k  w  - 18 k  w) x - 24 k w  + 24 k  w) %e
             8      2  6      4  4      6  2    8
 cos(w x))/(w  + 4 k  w  + 6 k  w  + 4 k  w  + k )
```

A slight change in syntax gives definite integrals:

```
(%i4) integrate (1/x^2, x, 1, inf);
(%o4)                            1
(%i5) integrate (1/x, x, 0, inf);

defint: integral is divergent.
 -- an error. To debug this try: debugmode(true);
```

Next we define the simbol `g` in terms of `f` (previously defined in `%i1`) and the hyperbolic sine function, and find its Taylor series expansion (up to, say, order 3 terms) about the point `x = 0`:

```
(%i6) g: f / sinh(k*x)^4;
                         3   k x
                        x  %e    sin(w x)
(%o6)                   -----------------
                               4
                          sinh (k x)
(%i7) taylor (g, x, 0, 3);
                        2    3   2         2    3   3
         w    w x   (w k  + w ) x    (3 w k  + w ) x
(%o7)/T/ -- + --- - -------------- - ---------------- + . . .
          4    3           4                3
         k    k          6 k              6 k
```

The limit of `g` as `x` goes to 0 is computed as follows:

```
(%i8) limit (g, x, 0);
                              w
(%o8)                         --
                               4
                              k
```

Maxima also permits derivatives to be represented in unevaluated form (note the quote):

```
(%i9) 'diff (y, x);
                                dy
(%o9)                           --
                                dx
```

The quote operator in `(%i9)` means "do not evaluate". Without it, Maxima would have obtained 0:

```
(%i10) diff (y, x);
(%o10)                           0
```

Using the quote operator we can write differential equations:

```
(%i11) 'diff (y, x, 2) + 'diff (y, x) + y;
                           2
                          d y   dy
(%o11)                    --- + -- + y
                            2   dx
                          dx
```

Maxima's `ode2` function can solve first and second order ODE's:

```
(%i12) ode2 (%o11, y, x);
             - x/2          sqrt(3) x            sqrt(3) x
(%o12) y = %e      (%k1 sin(---------) + %k2 cos(---------))
                               2                    2
```

6 Matrix calculations

Maxima can compute the determinant, inverse and eigenvalues and eigenvectors of matrices which have symbolic elements (i.e., elements which involve algebraic variables.) We begin by entering a matrix `m` element by element:

```
(%i1) m: entermatrix (3, 3);

Is the matrix  1. Diagonal  2. Symmetric  3. Antisymmetric  4. General
Answer 1, 2, 3 or 4 :
4;
Row 1 Column 1:
0;
Row 1 Column 2:
1;
Row 1 Column 3:
a;
Row 2 Column 1:
1;
Row 2 Column 2:
0;
Row 2 Column 3:
1;
Row 3 Column 1:
1;
Row 3 Column 2:
1;
Row 3 Column 3:
0;

Matrix entered.
                            [ 0  1  a ]
                            [         ]
(%o1)                       [ 1  0  1 ]
                            [         ]
                            [ 1  1  0 ]
```

Next we find its transpose, determinant and inverse:

```
(%i2) transpose (m);
                            [ 0  1  1 ]
                            [         ]
(%o2)                       [ 1  0  1 ]
                            [         ]
                            [ a  1  0 ]
(%i3) determinant (m);
(%o3)                          a + 1
(%i4) invert (m), detout;
```

```
                          [ - 1   a    1  ]
                          [               ]
                          [  1   - a   a  ]
                          [               ]
                          [  1    1   - 1 ]
(%o4)                     -----------------
                                a + 1
```

In (%i4), the modifier `detout` keeps the determinant outside the inverse. As a check, we multiply `m` by its inverse (note the use of the period to represent matrix multiplication):

```
(%i5) m . %o4;
                                 [ - 1   a    1  ]
                                 [               ]
                                 [  1   - a   a  ]
                 [ 0  1  a ]     [               ]
                 [         ]     [  1    1   - 1 ]
(%o5)            [ 1  0  1 ] . -----------------
                 [         ]           a + 1
                 [ 1  1  0 ]
(%i6) expand (%);
         [   a       1                                  ]
         [ ----- + -----         0              0       ]
         [ a + 1   a + 1                                ]
         [                                              ]
         [                   a       1                  ]
(%o6)    [       0        ----- + -----        0       ]
         [                a + 1   a + 1                 ]
         [                                              ]
         [                                    a       1 ]
         [       0              0        ----- + ----- ]
         [                                a + 1   a + 1 ]
(%i7) factor (%);
                          [ 1  0  0 ]
                          [         ]
(%o7)                     [ 0  1  0 ]
                          [         ]
                          [ 0  0  1 ]
```

In order to find the eigenvalues and eigenvectors of `m`, we use the function `eigenvectors`:

```
(%i8) eigenvectors (m);
           sqrt(4 a + 5) - 1  sqrt(4 a + 5) + 1
(%o8) [[[- ----------------, ----------------, - 1],
                  2                  2
                   sqrt(4 a + 5) - 1    sqrt(4 a + 5) - 1
[1, 1, 1]], [[[1, - ----------------, - ----------------]],
                        2 a + 2              2 a + 2
     sqrt(4 a + 5) + 1  sqrt(4 a + 5) + 1
[[1, ----------------, ----------------]], [[1, - 1, 0]]]]
          2 a + 2            2 a + 2
```

In %o8, the first triple gives the eigenvalues of m and the next gives their respective multiplicities (here each is unrepeated). The next three triples give the corresponding eigenvectors of m. In order to extract from this expression one of these eigenvectors, we may use the part function:

```
(%i9) part (%o23, 2, 1, 1);
                 sqrt(4 a + 5) - 1    sqrt(4 a + 5) - 1
(%o9)     [1, - ----------------, - ----------------]
                      2 a + 2              2 a + 2
```

7 Programming in Maxima

So far, we have used Maxima in the interactive mode, rather like a calculator. However, for computations which involve a repetitive sequence of commands, it is better to execute a program. Here we present a short sample program to calculate the critical points of a function f of two variables x and y. The program cues the user to enter the function f, then it computes the partial derivatives f_x and f_y, and then it uses the Maxima command solve to obtain solutions to $f_x = f_y = 0$. The program is written outside of Maxima with a text editor, and then loaded into Maxima with the batch command. Here is the program listing:

```
/* ------------------------------------------------------------------------
   this is file critpts.max:
   as you can see, comments in maxima are like comments in C

   Nelson Luis Dias, nldias@simepar.br
   created 20000707
   updated 20000707
   ------------------------------------------------------------------------ */
critpts():=(
   print("program to find critical points"),
/* ------------------------------------------------------------------------
   asks for a function
   ------------------------------------------------------------------------ */
   f:read("enter f(x,y)"),
/* ------------------------------------------------------------------------
```

```
   echoes it, to make sure
   ------------------------------------------------------------------------ */
   print("f = ",f),
/* ------------------------------------------------------------------------
   produces a list with the two partial derivatives of f
   ------------------------------------------------------------------------ */
   eqs:[diff(f,x),diff(f,y)],
/* ------------------------------------------------------------------------
   produces a list of unknowns
   ------------------------------------------------------------------------ */
   unk:[x,y],
/* ------------------------------------------------------------------------
   solves the system
   ------------------------------------------------------------------------ */
   solve(eqs,unk)
)$
```

The program (which is actually a function with no argument) is called `critpts`. Each line is a valid Maxima command which could be executed from the keyboard, and which is separated by the next command by a comma. The partial derivatives are stored in a variable named `eqs`, and the unknowns are stored in `unk`. Here is a sample run:

```
(%i1) batch ("critpts.max");

batching #p/home/robert/tmp/maxima-clean/maxima/critpts.max
(%i2) critpts() := (print("program to find critical points"),
f : read("enter f(x,y)"), print("f = ", f),
eqs : [diff(f, x), diff(f, y)], unk : [x, y], solve(eqs, unk))
(%i3) critpts ();
program to find critical points
enter f(x,y)
%e^(x^3 + y^2)*(x + y);
                2    3
               y  + x
f =  (y + x) %e
(%o3) [[x = .4588955685487001 %i + .3589790871086935,
y = .4942017368275118 %i - .1225787367783657],
[x = .3589790871086935 - .4588955685487001 %i,
y = - .4942017368275118 %i - .1225787367783657],
[x = .4187542327234816 %i - .6923124204420268,
y = 0.455912070111699 - .8697262692814121 %i],
[x = - .4187542327234816 %i - .6923124204420268,
y = .8697262692814121 %i + 0.455912070111699]]
```

8 A partial list of Maxima functions

See the Maxima reference manual `doc/html/maxima_toc.html` (under the main Maxima installation directory). From Maxima itself, you can use `describe(`*`function name`*`)`.

`allroots(a)` Finds all the (generally complex) roots of the polynomial equation `A`, and lists them in `numer`ical format (i.e. to 16 significant figures).

`append(a,b)` Appends the list `b` to the list `a`, resulting in a single list.

`batch(a)` Loads and runs a program with filename `a`.

`coeff(a,b,c)` Gives the coefficient of `b` raised to the power `c` in expression `a`.

`concat(a,b)` Creates the symbol `ab`.

`cons(a,b)` Adds `a` to the list `b` as its first element.

`demoivre(a)` Transforms all complex exponentials in `a` to their trigonometric equivalents.

`denom(a)` Gives the denominator of `a`.

`depends(a,b)` Declares `a` to be a function of `b`. This is useful for writing unevaluated derivatives, as in specifying differential equations.

`desolve(a,b)` Attempts to solve a linear system `a` of ODE's for unknowns `b` using Laplace transforms.

`determinant(a)` Returns the determinant of the square matrix `a`.

`diff(a,b1,c1,b2,c2,...,bn,cn)` Gives the mixed partial derivative of `a` with respect to each `bi`, `ci` times. For brevity, `diff(a,b,1)` may be represented by `diff(a,b)`. `'diff(...)` represents the unevaluated derivative, useful in specifying a differential equation.

`eigenvalues(a)` Returns two lists, the first being the eigenvalues of the square matrix `a`, and the second being their respective multiplicities.

`eigenvectors(a)` Does everything that `eigenvalues` does, and adds a list of the eigenvectors of `a`.

`entermatrix(a,b)` Cues the user to enter an `a` × `b` matrix, element by element.

`ev(a,b1,b2,...,bn)` Evaluates `a` subject to the conditions `bi`. In particular the `bi` may be equations, lists of equations (such as that returned by `solve`), or assignments, in which cases `ev` "plugs" the `bi` into `a`. The `Bi` may also be words such as `numer` (in which case the result is returned in numerical format), `detout` (in which case any matrix inverses in `a` are performed with the determinant factored out), or `diff` (in which case all differentiations in `a` are evaluated, i.e., `'diff` in `a` is replaced by `diff`). For brevity in a manual command (i.e., not inside a user-defined function), the `ev` may be dropped, shortening the syntax to `a,b1,b2,...,bn`.

`expand(a)` Algebraically expands `a`. In particular multiplication is distributed over addition.

`exponentialize(a)` Transforms all trigonometric functions in `a` to their complex exponential equivalents.

`factor(a)` Factors `a`.

`freeof(a,b)` Is true if the variable `a` is not part of the expression `b`.

`grind(a)` Displays a variable or function `a` in a compact format. When used with `writefile` and an editor outside of Maxima, it offers a scheme for producing `batch` files which include Maxima-generated expressions.

`ident(a)` Returns an `a` × `a` identity matrix.

`imagpart(a)` Returns the imaginary part of `a`.

`integrate(a,b)` Attempts to find the indefinite integral of `a` with respect to `b`.

`integrate(a,b,c,d)` Attempts to find the indefinite integral of `a` with respect to `b`. taken from `b` = `c` to `b` = `d`. The limits of integration `c` and `d` may be taken is `inf` (positive infinity) of `minf` (negative infinity).

`invert(a)` Computes the inverse of the square matrix `a`.

`kill(a)` Removes the variable `a` with all its assignments and properties from the current Maxima environment.

`limit(a,b,c)` Gives the limit of expression `a` as variable `b` approaches the value `c`. The latter may be taken as `inf` of `minf` as in `integrate`.

`lhs(a)` Gives the left-hand side of the equation `a`.

`loadfile(a)` Loads a disk file with filename `a` from the current default directory. The disk file must be in the proper format (i.e. created by a `save` command).

`makelist(a,b,c,d)` Creates a list of `a`'s (each of which presumably depends on `b`), concatenated from `b` = `c` to `b` = `d`

`map(a,b)` Maps the function `a` onto the subexpressions of `b`.

`matrix(a1,a2,...,an)` Creates a matrix consisting of the rows `ai`, where each row `ai` is a list of `m` elements, `[b1, b2, ..., bm]`.

`num(a)` Gives the numerator of `a`.

`ode2(a,b,c)` Attempts to solve the first- or second-order ordinary differential equation `a` for `b` as a function of `c`.

`part(a,b1,b2,...,bn)` First takes the `b1`th part of `a`, then the `b2`th part of that, and so on.

`playback(a)` Displays the last `a` (an integer) labels and their associated expressions. If `a` is omitted, all lines are played back. See the Manual for other options.

`ratsimp(a)` Simplifies `a` and returns a quotient of two polynomials.

`realpart(a)` Returns the real part of `a`.

`rhs(a)` Gives the right-hand side of the equation `a`.

`save(a,b1,b2,..., bn)` Creates a disk file with filename `a` in the current default directory, of variables, functions, or arrays `bi`. The format of the file permits it to be reloaded into Maxima using the `loadfile` command. Everything (including labels) may be `save`d by taking `b1` equal to `all`.

`solve(a,b)` Attempts to solve the algebraic equation `a` for the unknown `b`. A list of solution equations is returned. For brevity, if `a` is an equation of the form `c` = `0`, it may be abbreviated simply by the expression `c`.

`string(a)` Converts `a` to Maxima's linear notation (similar to Fortran's) just as if it had been typed in and puts `a` into the buffer for possible editing. The `string`'ed expression should not be used in a computation.

`stringout(a,b1,b2,...,bn)` Creates a disk file with filename `a` in the current default directory, of variables (e.g. labels) `bi`. The file is in a text format and is not reloadable into Maxima. However the strungout expressions can be incorporated into a Fortran, Basic or C program with a minimum of editing.

`subst(a,b,c)` Substitutes `a` for `b` in `c`.

`taylor(a,b,c,d)` Expands `a` in a Taylor series in `b` about `b` = `c`, up to and including the term $(\mathtt{b}-\mathtt{c})^{\mathtt{d}}$. Maxima also supports Taylor expansions in more than one independent variable; see the Manual for details.

`transpose(a)` Gives the transpose of the matrix `a`.

`trigexpand(a)` Is a trig simplification function which uses the sum-of-angles formulas to simplify the arguments of individual `sin`'s or `cos`'s. For example, `trigexpand(sin(x+y))` gives `cos(x) sin(y) + sin(x) cos(y)`.

`trigreduce(a)` Is a trig simplification function which uses trig identities to convert products and powers of `sin` and `cos` into a sum of terms, each of which contains only a single `sin` or `cos`. For example, `trigreduce(sin(x)^2)` gives `(1 - cos(2x))/2`.

`trigsimp(a)` Is a trig simplification function which replaces `tan`, `sec`, etc., by their `sin` and `cos` equivalents. It also uses the identity $\mathtt{sin}()^2 + \mathtt{cos}()^2 = 1$.