Maxima by Example: Ch.3: Algebra, Part 2 *

Edwin L. Woollett

June 18, 2008

Contents

*This version uses Maxima 5.14. 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.

NON-PROFIT PRINTING AND DISTRIBUTION IS PERMITTED.
You may make copies of this document and distribute them to others as long as you charge no more than the costs of printing.

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.

3 Algebra, Part 2

We continue presenting algebra examples. In Ch. 2, we presented examples for expanding, factoring, and substituting. We will freely use techniques presented already in Ch. 2.

3.1 Extracting Parts of an Expression

Often one wants to apply some simplification function to only a part of a complicated expression. Maxima has many tools to select the correct part of an expression.

We will present examples of the use of the following Maxima functions:

- **part** returns the subexpression you specify, according to its position in the expression.
- **dpart** is similar to **part** except that it returns the entire expression, with the selected subexpression displayed inside a box.
- **substpart** substitutes the characters you specify for the indicated subexpression, then returns the new value of the expression.
- **pickapart** assigns intermediate display lines to all subexpressions of an expression, down to a specified depth.
- **reveal** displays an expression to the specified integer depth, indicating the length of each part.
- **lhs** and **rhs** return the left and right sides of the given equation, respectively.
- **first** and **last** return the first and last part of the specified expression, respectively.
- **rest** returns the expression with one or more of its leading elements removed.
- **coeff** and **ratcoef** return the coefficient of a given variable in the specified expression.

In addition, the system variable **piece** holds the last expression selected with one of the part extraction commands. The variable **piece** is set during the execution of the part extraction command and thus can be used within the command itself.

3.1.1 part, dpart, and piece

To extract part of an expression you can use **part**`(exp, n)`, where `exp` refers to an equation or expression, and `n` is an integer that represents the part you want.

Here **part** is used to extract the second part of eq1 (the right hand side) and then extract the first term of the right hand side:

```
(%i1) eq1 : x^2 + 2*x + 2 = y^2 + 1;
                                  2              2
(%o1)                            x  + 2 x + 2 = y  + 1
(%i2) part(eq1, 2);
                                        2
(%o2)                                  y  + 1
(%i3) part(%,1);
                                          2
(%o3)                                    y
```

Notice in the following table that part 0 is always the operator and the arguments of the operator are the successive parts. The equation `a = b` is interpreted by the **part** command as if it were in the functional notation `"="(a,b)` similar to `f(x,y,z)`.

Version	$expr : f(x, y, z);$	$expr : a = b;$
part(expr, 0)	f	$=$
part(expr, 1)	x	a
part(expr, 2)	y	b
part(expr, 3)	z	

Table 1: Using the **part** function

With additional arguments, **part**`(expr,num1, ...,numn)` allows you to obtain the part of the expression `expr` specified by part `num1` then find the `num2` part of the resulting expression, and so on.

The following is equivalent to the combined inputs `(%i2)` and `(%i3)` above:

```
(%i4) part(eq1,2,1);
                                            2
(%o4)                                      y
```

The command **dpart**`(expr, num1, ..., numn)` is similar to **part** except that instead of simply returning the specified subexpression, this command returns the entire expression with the selected subexpression displayed inside a box.

In addition, the system variable **piece** holds the last expression selected with one of the part selection commands, such as **part**, **dpart**, and **substpart** (see below).

Here we select part of the expression expr1 to be highlighted with a box in the output:

```
(%i5) expr1 : (x^3 + 3*x^2 + 3*x + 1)/(d^(x^3 + 3*x^2 + 3*x + 1) + n);
                             3      2
                            x  + 3 x  + 3 x + 1
(%o5)                     ------------------------
                                3      2
                               x  + 3 x  + 3 x + 1
                          n + d
(%i6) dpart(expr1,2,2,2);
                             3      2
                            x  + 3 x  + 3 x + 1
(%o6)                    --------------------------
                               """"""""""""""""""""""
                               " 3      2           "
                               "x  + 3 x  + 3 x + 1"
                               """"""""""""""""""""""
                         n + d
```

We next ask for the value of **piece**, whose value is the subexpression selected by **dpart** above.

```
(%i7) piece;
                                    3      2
(%o7)                              x  + 3 x  + 3 x + 1
```

3.1.2 substpart

The command **substpart** `(x,expr, num1, ...,numn)` substitutes `x` for the subexpression (of the given expr) picked out by the set of integers `(num1,num2,...,numn)`, then returns the resulting new value of the expression `expr`. You indicate the subexpression for **substpart** just as you do for **part** by specifying the arguments `expr, num1, ....,numn`.

Note that `x` can be some operator to be substituted for an operator of `expr`. In some cases, you might need to enclose `x` in double quotes; for example, the command **substpart** `("+", a*b,0)` returns `b + a`.

Here we factor the part of `expr1` selected by **dpart**, and then substitute back into the expression.

```
(%i8) substpart( factor(piece), expr1, 2, 2, 2);
                                   3      2
                                  x  + 3 x  + 3 x + 1
(%o8)                             -------------------
                                                3
                                         (x + 1)
                                     n + d
```

3.1.3 pickapart and reveal

The command **pickapart** `(expr, depth)` assigns intermediate expression labels `%t1, %t2, ...` to all subexpressions of the expression `expr` down to the specified integer depth. You will find this command useful for dealing with large expressions, and in order to assign parts of expressions to variables without having to use the **part** command.

Here we display subexpressions of (large) expr2 down to the second level, assigning each subexpression to an intermediate expression label `%tn`.

```
(%i9) expr2 : log(a*x^2 + b*x + c)^4 - 1/(1 + 1/y)^(1/2)
               + exp( -%i*cos(12*a - b + c) )
               + a0*sin(a*x^2 + b)^2  -  x*y;
                     1            4    2                    2    2
(%o9) - x y - ----------- + log (a x  + b x + c) + a0 sin (a x  + b)
                     1
              sqrt(- + 1)
                     y
                                                              - %i cos(c - b + 12 a)
                                                          + %e
(%i10) pickapart(expr2, 2);
(%t10)                                 x y


                                        1
(%t11)                              -----------
                                        1
                                    sqrt(- + 1)
                                        y


                                      2
(%t12)                           log(a x  + b x + c)
```

```
                                     2    2
(%t13)                           sin  (a x  + b)


(%t14)                       - %i cos(c - b + 12 a)

                                 %t14       4
(%o14)               %t13 a0 + %e     + %t12  - %t11 - %t10
```

The expressions identified by intermediate expression labels can then be used by referring to that label.

```
(%i15) ex22 : %t12$
(%i16) ex22;
                                       2
(%o16)                         log(a x  + b x + c)
```

The function **reveal** `(expr, depth)` displays the expression `expr` to the specified integer depth, indicating the length of each part.

Here is the Maxima manual entry for this function:

Function: **reveal** `(expr, depth)`
Replaces parts of `expr` at the specified integer depth with descriptive summaries.

- Sums and differences are replaced by `Sum(n)` where `n` is the number of operands of the sum.
- Products are replaced by `Product(n)` where `n` is the number of operands of the product.
- Exponentials are replaced by `Expt`.
- Quotients are replaced by `Quotient`.
- Unary negation is replaced by `Negterm`.

When `depth` is greater than or equal to the maximum depth of `expr`, `reveal (expr, depth)` returns `expr` unmodified.

Here we use **reveal** to summarize the five term expression `expr2` to successively greater depths.

```
(%i17) reveal(expr2, 2);
(%o17)           Negterm + Negterm + Expt + Product(2) + Expt
(%i18) reveal(expr2,3);
                                            4              Negterm
(%o18)       - Product(2) - Quotient + log  + a0 Expt + %e
(%i19) reveal(expr2,4);
                     1        4                  2     - Product(2)
(%o19)      - x y - ---- + log (Sum(3)) + a0 sin  + %e
                    sqrt
```

```
(%i20)  reveal(expr2,5);
                    1            4
(%o20) - x y - ------------ + log (Product(2) + Product(2) + c)
              sqrt(Sum(2))
                                                      2             - %i cos
                                              + a0 sin (Sum(2)) + %e
(%i21)
```

At depth 2, both the `log^4` term and the `%e^(neg)` term are simply labeled `Expt`, while the `-1/sqrt` term is simply `Negterm`.

Looking at depth 3, the `a0*sin^2` term is labeled as `a0 Expt`, while the `1/sqrt` term becomes `Quotient`.

3.1.4 rhs and lhs

The function **lhs** extracts the left hand side of an equation, and the function **rhs** extracts the right hand side.

Here we use **rhs** to extract the right side of the equation below.

```
(%i1) eq1 : x^2 + 2*x + 1 = y^2$
(%i2) rhs(eq1);
                                     2
(%o2)                               y
(%i3) lhs(eq1);
                              2
(%o3)                        x  + 2 x + 1
```

If the argument is not an equation, but rather an expression, **lhs** returns the expression and **rhs** returns 0.

```
(%i4) lhs(%);
                              2
(%o4)                        x  + 2 x + 1
(%i5) rhs(%);
(%o5)                                0
```

The Maxima manual has the following entry for **lhs**.

> Function: **lhs** `(expr)`
> Returns the left-hand side (that is, the first argument) of the expression `expr`, when the operator of `expr` is one of the relational operators `< <= = # equal notequal >= >`, one of the assignment operators `:= ::= : ::`, or a user-defined binary infix operator, as declared by `infix`.
> When `expr` is an atom or its operator is something other than the ones listed above, `lhs` returns `expr`.

3.1.5 first, last, and rest

The syntax **first** `(expr)` returns the first part of the expression `expr` and the syntax **last** `(expr)` returns the last part of the expression `expr`.
The syntax **rest** `(expr)` returns the expression with its leading element removed.
With an additional integer argument, **rest** `(expr, num)` returns the expression `expr` with the first `num` terms removed.

Here we use **first** to display the first term in the expression `expr2`:

```
(%i7)  expr2 : log(a*x^2 + b*x + c)^4 - 1/(1 + 1/y)^(1/2)
                + exp( -%i*cos(12*a - b + c) )
                + a0*sin(a*x^2 + b)^2  -  x*y;
                   1            4    2                    2    2
(%o7) - x y - ----------- + log (a x  + b x + c) + a0 sin (a x  + b)
                   1
              sqrt(- + 1)
                   y
                                                         - %i cos(c - b + 12 a)
                                                     + %e
(%i8) first(expr2);
(%o8)                               - x y
(%i9) last(expr2);
                              - %i cos(c - b + 12 a)
(%o9)                       %e
```

Note that "first" and "last" refer to the internal canonical ordering which Maxima uses, for example, when Maxima displays the expression, and does not refer to the order in which the terms of the expression were entered on the input line.
Here we use **rest** to display all of `expr2` except the first term, and secondly, display all of `expr2` except the first two terms.

```
(%i10) rest(expr2);
              1            4    2                    2    2
(%o10) - ----------- + log (a x  + b x + c) + a0 sin (a x  + b)
              1
         sqrt(- + 1)
              y
                                                         - %i cos(c - b + 12 a)
                                                     + %e
(%i11) rest(expr2,2);
          4    2                    2    2           - %i cos(c - b + 12 a)
(%o11) log (a x  + b x + c) + a0 sin (a x  + b) + %e
```

The functions **first**, **last**, and **rest** have similar behavior when the argument is a list, rather than an expression.

```
(%i12) mylist : [a,b,c,d,e];
(%o12)                         [a, b, c, d, e]
(%i13) [first(mylist),last(mylist), rest(mylist, 2) ];
(%o13)                        [a, e, [c, d, e]]
```

3.1.6 coeff and ratcoef

The commands **coeff** and **ratcoef**, take as arguments an expression, a variable in the expression for which you want the coefficient, and optionally, a power to which the variable is raised. Both functions return the coefficient. The function **ratcoef** expands and rationally simplifies the expression before finding the coefficient, and thus can produce answers different from **coeff**, which is purely syntactic.

The Maxima manual has the following entry for **coeff**:

> Function: **coeff** `(expr, x, n)`
> Returns the coefficient of `x^n` in `expr`. The integer `n` may be omitted if it is 1. `x` may be an atom, or complete subexpression of `expr` e.g., `sin(x)`, `a[i+1]`, `x + y`, etc. (In the last case the expression

(x + y) should occur in expr). Sometimes it may be necessary to expand or factor expr in order to make x^n explicit. This is not done automatically by coeff.

The Maxima manual has two examples, the first showing use with an equation, and the second showing use with an expression:

```
(%i1) eq1 : 2*a*tan(x) + tan(x) + b = 5*tan(x) + 3;
(%o1)               2 a tan(x) + tan(x) + b = 5 tan(x) + 3
(%i2) coeff(eq1, tan(x) );
(%o2)                             2 a + 1 = 5
(%i3) e2 : y + x*%e^x + 1;
                                         x
(%o3)                            y + x %e  + 1
(%i4) coeff(e2, x, 0);
(%o4)                                y + 1
```

The Maxima manual entry for **ratcoef** is:

> Function: **ratcoef** (expr, x, n)
> Function: **ratcoef** (expr, x)
> Returns the coefficient of the expression x^n in the expression expr. If omitted, n is assumed to be 1.
> The return value is free (except possibly in a non-rational sense) of the variables in x.
> If no coefficient of this type exists, 0 is returned.
> ratcoef expands and rationally simplifies its first argument and thus it may produce answers different from those of coeff which is purely syntactic.
> Thus ratcoef ((x + 1)/y + x, x) returns (y + 1)/y whereas coeff returns 1.
> ratcoef (expr, x, 0), viewing expr as a sum, returns a sum of those terms which do not contain x.
> Therefore if x occurs to any negative powers, ratcoef should not be used.
> Since expr is rationally simplified before it is examined, coefficients may not appear quite the way they were envisioned.

Here we compare **coeff** with **ratcoef**.

```
(%i5) e3 : (a*x + b)^2;
                                             2
(%o5)                             (a x + b)
(%i6) coeff(e3,x);
(%o6)                                 0
(%i7) ratcoef(e3, x);
(%o7)                               2 a b
```

The function **coeff** is acting on the expression (a*x + b)^2, where (a*x + b) is seen as a single entity. On the other hand, **ratcoef** first expands the expression and then looks for coefficients of x in the expanded expression.

We can, of course, first do the expansion by hand, and then use **coeff** to get the same result:

```
(%i8) expand(e3);
                            2  2              2
(%o8)                      a  x  + 2 a b x + b
(%i9) coeff(%,x);
(%o9)                               2 a b
```

3.2 Extracting Parts of a Complex Expression

3.2.1 realpart, imagpart, rectform, polarform

The following functions allow you to manipulate expressions containing complex numbers.

- **realpart** and **imagpart** return the real and imaginary parts of an expression, respectively.
- **rectform** returns an expression in the form `a + %i*b`, where `a` and `b` are purely real.
- **polarform** returns an expression in the form `r*%e^(%i*theta)` where `r` and `theta` are purely real, and `%e` is the base of the natural logarithms.

The Maxima manual has the following entry for **realpart**:

> Function: **realpart** `(expr)`
> Returns the real part of `expr`. The functions `realpart` and `imagpart` will work on expressions involving trigonometic and hyperbolic functions, as well as square root, logarithm, and exponentiation.

Here is an example of the use of all four of the above functions (note x and y are assumed to be real).

```
(%i1) e1 : sin( exp(%i*y + x) );
                                            %i y + x
(%o1)                                sin(%e        )
(%i2) realpart(e1);
                                 x                x
(%o2)                      sin(%e  cos(y)) cosh(%e  sin(y))
(%i3) imagpart(e1);
                                 x                x
(%o3)                      cos(%e  cos(y)) sinh(%e  sin(y))
(%i4) rectform(e1);
               x                x                   x                x
(%o4) %i cos(%e  cos(y)) sinh(%e  sin(y)) + sin(%e  cos(y)) cosh(%e  sin(y))
(%i5) polarform(e1);
              2   x               2   x
(%o5) sqrt(cos (%e  cos(y)) sinh (%e  sin(y))
      2   x               2   x
 + sin (%e  cos(y)) cosh (%e  sin(y))) expt(%e,
               x                x                 x                x
%i atan2(cos(%e  cos(y)) sinh(%e  sin(y)), sin(%e  cos(y)) cosh(%e  sin(y))))
```

The **polarform** result makes use of **atan2** `(y,x)` which returns the arctangent of y/x in the interval $(-\pi, \pi)$.

3.3 Evaluating Trigonometric Functions

The following table provide the names of the trig functions in Maxima. Maxima can compute the derivatives of all these functions.

Circular	Inverse Circular	Hyperbolic	Inverse Hyperbolic
sin	asin	sinh	asinh
cos	acos	cosh	acosh
tan	atan	tanh	atanh
cot	acot	coth	acoth
sec	asec	sech	asech
csc	acsc	csch	acsch

Table 2: Trig Functions and Inverses

Maxima always numerically evaluates trigonometric functions (such as **sin**, **asin**, **sinh**, and **asinh**) that have floating point arguments. To avoid introducing approximations prematurely, Maxima does not do so automatically for trigonometric functions that have integer arguments. In cases where Maxima can return an exact value, however, a number can result. This section presents the following commands:

- **exponentialize** converts the given expression containing trigonometric functions to an exponential with complex variables
- **demoivre** converts the given exponential with complex variables to a trigonometric function.

Several option variables control the evaluation of trigonometric functions. The options **numer**, **exponentialize**, and **%emode**. are introduced in this section.

You can evaluate trigonometric functions with integer arguments numerically by setting the option variable **numer** to true.

```
(%i1) sin(0);
(%o1)                                0
(%i2) sin(1);
(%o2)                              sin(1)
(%i3) sin(1),numer;
(%o3)                          0.8414709848079
(%i4) numer;
(%o4)                               false
(%i5) sin(1),numer:true;
(%o5)                          0.8414709848079
(%i6) numer;
(%o6)                               false
```

Inputs `%i3` and `%i5` are equivalent and have the effect of setting **numer** temporarily true.

The Maxima computation engine is aware of simple values attained by trig functions if the argument has the special form $n\pi/m$, where n is an integer, and m has one of the values $[1, 2, 3, 4, 5, 12]$.

```
(%i7) [sin(%pi/2),sin(%pi/3),sin(%pi/4)];
                                sqrt(3)      1
(%o7)                       [1, -------, -------]
                                   2     sqrt(2)
(%i8) %,numer;
(%o8)                 [1, 0.86602540378444, 0.70710678118655]
```

Evaluating functions or expressions which include trig functions is simple. Here is an example of a Maxima function $f(x)$ and a Maxima expression g. First, define the function f in terms of the trig function sin.

```
(%i1) f(z) := sin(z)^2 + 1;
                                                 2
(%o1)                             f(z) := sin (z) + 1
```

Now evaluate f at $z = x + 1$.

```
(%i2) f(x+1);
                                          2
(%o2)                                  sin (x + 1) + 1
```

Define the expression g in terms of trig functions and then evaluate the expression at the angle $\pi/3$,:

```
(%i3) g : cos(x)^2 - sin(x)^2;
                                         2          2
(%o3)                                 cos (x) - sin (x)
(%i4) g, x = %pi/3;
                                               1
(%o4)                                        - -
                                               2
```

To differentiate an expression or function we use:

```
(%i5) diff(g,x);
(%o5)                                 - 4 cos(x) sin(x)
(%i6) diff(f(z),z);
(%o6)                                  2 cos(z) sin(z)
```

To get the indefinite integral we use:

```
(%i7) integrate(g,x);
                              sin(2 x)            sin(2 x)
                              -------- + x    x - --------
                                 2                   2
(%o7)                         ------------ - ------------
                                   2              2
(%i8) integrate(f(z),z);
                                        sin(2 z)
                                    z - --------
                                           2
(%o8)                               ------------ + z
                                         2
```

The above indefinite integrals cry out for simplification.

```
(%i9) ratsimp(%o7);
                                        sin(2 x)
(%o9)                                   --------
                                           2
(%i10) ratsimp(%o8);
                                      sin(2 z) - 6 z
(%o10)                              - --------------
                                            4
```

3.3.1 exponentialize

When you set the option variable **exponentialize** to **true**, subsequent computations convert trigonometric functions to exponentials with complex variables. You can also use the command **exponentialize** `(expr)` which performs the same transformation on a given expression.

```
(%i11) exponentialize;
(%o11)                               false
(%i12) exponentialize(sin(x));
                                   %i x      - %i x
                              %i (%e     - %e      )
(%o12)                      - ----------------------
                                         2
(%i13) exponentialize:true$
(%i14) cos(x);
                                 %i x      - %i x
                               %e     + %e
(%o14)                         -----------------
                                         2
```

With x, y, and z all real (by default), the real and imaginary parts of the following expression `e1` are obvious from its definition. As an exercise, we convert the trig functions to their complex exponential equivalents and then find the real and imaginary parts of the resulting expression.

```
(%i15) exponentialize:false$
(%i16) e1 : tan(x) + %i*cos(y) - sin(z);
(%o16)                    - sin(z) + %i cos(y) + tan(x)
(%i17) e11 : exponentialize(e1);
             %i z      - %i z           %i y      - %i y           %i x      - %i x
       %i (%e     - %e      )    %i (%e     + %e      )    %i (%e     - %e      )
(%o17) ---------------------- + ---------------------- - ----------------------
                  2                        2                  %i x      - %i x
                                                            %e     + %e
(%i18) imagpart(e11);
(%o18)                              cos(y)
(%i19) realpart(e11);
                                sin(x)
(%o19)                          ------ - sin(z)
                                cos(x)
```

3.3.2 demoivre

To convert an expression containing exponentials whose arguments involve factors of `%i` to trig function equivalents, we use the function **demoivre** `(expr)`.

```
(%i20) demoivre(e11);
                                                      sin(x)
(%o20)                    - sin(z) + %i cos(y) + ------
                                                      cos(x)
```

Abraham De Moivre (1667 - 1754) was a French-born mathematician who pioneered the development of analytic geometry and the theory of probability. His name is associated with what Richard Feynman called ”the most remarkable formula in mathematics”: $e^{i\,\theta} = cos(\theta) + i\,sin(\theta)$.

3.3.3 %emode

The Maxima manual description of `%emode` is

> Option variable: **%emode**
> Default value: **true**
> When `%emode` is `true`, `%e^(%pi %i x)` is simplified as follows.
> `%e^(%pi %i x)` simplifies to `cos (%pi x) + %i sin (%pi x)` if `x` is an integer or a multiple of `1/2, 1/3, 1/4, or 1/6`, and then further simplified.
> For other numerical `x`, `%e^(%pi %i x)` simplifies to `%e^(%pi %i y)` where `y` is `x - 2 k` for some integer `k` such that `abs(y) < 1`.
> When `%emode` is `false`, no special simplification of `%e^(%pi %i x)` is carried out.

Here are some simple examples:

```
(%i1) %emode;
(%o1)                              true
(%i2) f(x) := %e^(%i*%pi*x);
                                      %i %pi x
(%o2)                        f(x) := %e
(%i3) f(1);
(%o3)                              - 1
(%i4) f(1/2);
(%o4)                              %i
(%i5) f(2/3);
                              sqrt(3) %i   1
(%o5)                         ---------- - -
                                  2        2
(%i6) f(1/4);
                           sqrt(2) %i   sqrt(2)
(%o6)                      ---------- + -------
                               2           2
(%i7) f(2);
(%o7)                               1
(%i8) f(3);
(%o8)                              - 1
(%i9) f(1.4);
                                  3 %i %pi
                                - --------
                                     5
(%o9)                         %e
(%i10) %emode: false$
(%i11) f(2);
                                2 %i %pi
(%o11)                        %e
```

3.4 Expanding and Simplifying Trig Expressions

You can expand expressions involving trigonometric functions. This section presents the following commands:

- **trigexpand** expands expressions that contain trigonometric and hyperbolic functions of sums of angles and of multiple angles.
- **trigreduce** combines products and powers of the trigonometric and hyperbolic functions for a specified variable and tries to eliminate these functions when they occur in the denominator.
- **trigsimp** converts expressions containing functions such as **tan** and **sec** to contain **sin**, **cos**, **sinh**, and **cosh** instead, so that **trigreduce** can further simplify the expressions.

3.4.1 trigexpand

The option variables **trigexpand**, **trigexpandplus**, **trigexpandtimes**, and **halfangles** are also described in this section.

The full Maxima manual description of **trigexpand** is

> Function: **trigexpand** `(expr)`
> Expands trigonometric and hyperbolic functions of sums of angles and of multiple angles occurring in `expr`. For best results, `expr` should be expanded.
> To enhance user control of simplification, this function expands only one level at a time, expanding sums of angles or multiple angles. To obtain full expansion into sines and cosines immediately, set the switch `trigexpand: true`.
> `trigexpand` is governed by the following global flags:
> `trigexpand` : (Default: **false**). If `true` causes expansion of all expressions containing sin's and cos's occurring subsequently.
> `halfangles` : (Default: **false**). If `true` causes half-angles to be simplified away.
> `trigexpandplus` : (Default: **true**). Controls the "sum" rule for trigexpand, expansion of sums (e.g. `sin(x + y)`) will take place only if `trigexpandplus` is `true`.
> `trigexpandtimes` : (Default: **true**). Controls the "product" rule for `trigexpand`, expansion of products (e.g. `sin(2 x)`) will take place only if `trigexpandtimes` is `true`.

By default, **trigexpandplus** is **true**, so Maxima expands a trig function of a sum of angles, like **sin(x + y)**. The option variable **trigexpandtimes** is **true** by default, so Maxima's default behavior is to expand a trig function whose argument is a multiple of some angle, hence expanding **sin(2*y)**.

Let's illustrate the **trigexpand** function and the **trigexpand** option with the simplest cases. The expression `e1` is a trig function of a multiple angle. The expression `e2` is a trig function of a sum of angles. The expression `e3` is a trig function of a sum of angles, one of which is a multiple of another angle.

```
(%i1) e1 : sin(2*x)$
(%i2) e2 : sin(x + y)$
(%i3) e3 : sin(2*x + y)$
(%i4) [trigexpand,trigexpandplus,trigexpandtimes];
(%o4)                        [false, true, true]
(%i5) trigexpand(e1);
(%o5)                          2 cos(x) sin(x)
(%i6) trigexpand(e2);
(%o6)                   cos(x) sin(y) + sin(x) cos(y)
(%i7) e4 : trigexpand(e3);
(%o7)                 cos(2 x) sin(y) + sin(2 x) cos(y)
```

```
(%i8) e5 : trigexpand(e4);
                  2         2
(%o8)          (cos (x) - sin (x)) sin(y) + 2 cos(x) sin(x) cos(y)
(%i9) e6 : trigexpand(e3),trigexpand;
                  2         2
(%o9)          (cos (x) - sin (x)) sin(y) + 2 cos(x) sin(x) cos(y)
```

We see that **trigexpand** (`sin( 2*x + y )` expands the sum of angles (top level) without expanding the multiple angle argument. A second use of **trigexpand** expands the multiple angle argument. Instead of two applications of the function **trigexpand**, we could have used one application, but with the **trigexpand** option set to **true** (as we did in input `%i9`). Remember that `f(expr),option` is equivalent to `f(expr),option:true`. The default value of the option **trigexpand** is **false**.

Let's illustrate dealing with half-angles. The default value of **halfangles** is **false**.

```
(%i1) halfangles;
(%o1)                               false
(%i2) e1 : sin(x/2);
                                       x
(%o2)                               sin(-)
                                       2
(%i3) e1,halfangles;
                               sqrt(1 - cos(x))
(%o3)                          ----------------
                                   sqrt(2)
```

In input `%i3` we evaluated $sin(x/2)$ with **halfangles** locally set to **true**. This does not change the binding of the symbol `e1` nor the global value of **halfangles**:

```
(%i4) [e1,halfangles];
                                     x
(%o4)                          [sin(-), false]
                                     2
```

An example which contains a sum of angles, a multiple of an angle, and also a half angle:

```
(%i1) e1 : sin(2*x) + cosh(y - z) + tan(a/2)$
(%i2) [trigexpandtimes,trigexpandplus,halfangles];
(%o2)                         [true, true, false]
(%i3) trigexpand(e1);
                                                                    a
(%o3)   - sinh(y) sinh(z) + cosh(y) cosh(z) + 2 cos(x) sin(x) + tan(-)
                                                                    2
```

Here we bind **trigexpandtimes** to **false** locally to prevent expansion of trig functions of multiple angles:

```
(%i4) trigexpand(e1),trigexpandtimes:false;
                                                                 a
(%o4)       - sinh(y) sinh(z) + cosh(y) cosh(z) + sin(2 x) + tan(-)
                                                                 2
```

Here we locally bind **trigexpandplus** to **false** to prevent the expansion of trig functions of sums of angles:

```
(%i5) trigexpand(e1),trigexpandplus:false;
                                                         a
(%o5)                cosh(z - y) + 2 cos(x) sin(x) + tan(-)
                                                         2
```

Here we perform all possible expansions for the expression `e1`:

```
(%i6) trigexpand(e1),halfangles;
                                                             1 - cos(a)
(%o6) - sinh(y) sinh(z) + cosh(y) cosh(z) + 2 cos(x) sin(x) + ----------
                                                               sin(a)
(%i7) [trigexpandtimes,trigexpandplus,halfangles];
(%o7)                        [true, true, false]
```

Finally, we allow only expansion of trig functions of half angles:

```
(%i8) trigexpand(e1),trigexpandtimes:false,trigexpandplus:false,halfangles;
                                                1 - cos(a)
(%o8)                cosh(z - y) + sin(2 x) + ----------
                                                  sin(a)
```

3.4.2 trigreduce

The Maxima function **trigreduce** `(expr,var)` carries out a procedure which is inverse to the effect of **trigexpand**. Products and powers of trig functions are written as expressions involving trig functions whose arguments are multiples of angles. If you do not include the optional argument `var`, **trigreduce** uses all the variables in the expression. The Maxima manual description is:

> Function: **trigreduce** `(expr, x)`
> Function: **trigreduce** `(expr)`
> Combines products and powers of trigonometric and hyperbolic sin's and cos's of `x` into those of multiples of `x`. It also tries to eliminate these functions when they occur in denominators. If `x` is omitted then all variables in `expr` are used.

Here is a simple example of using **trigreduce**:

```
(%i1) e1 : sin(2*z) + sin(2*y)$
(%i2) e2 : trigexpand(e1);
(%o2)                 2 cos(z) sin(z) + 2 cos(y) sin(y)
(%i3) trigreduce(e2);
(%o3)                        sin(2 z) + sin(2 y)
(%i4) trigreduce(e2,z);
(%o4)                    sin(2 z) + 2 cos(y) sin(y)
(%i5) trigreduce(e2,y);
(%o5)                    2 cos(z) sin(z) + sin(2 y)
```

3.4.3 trigsimp

The Maxima function **trigsimp** uses the identities: $\sin^2 x + \cos^2 x = 1$ and $\cosh^2 x - \sinh^2 x = 1$ to convert expressions containing functions such as tan and sec to contain sin, cos, sinh, and cosh instead, so that **trigreduce** can further simplify the expression.

The functions **trigreduce**, **ratsimp**, and **radcan** may be able to further simplify the result. There is a **demo** file which shows the results of **trigsimp** on some quite long expressions. Use `demo("trgsmp.dem");` to run this **demo** file.

Consider the following expression:

```
(%i6) e3 : ( 1 - sin(x) )*( sec(x) + tan(x) );
(%o6)                   (1 - sin(x)) (tan(x) + sec(x))
(%i7) e4 : e3 - cos(x) -1 + (cosh(x)^2 - sinh(x)^2)^3 ;
                                                2          2    3
(%o7) (1 - sin(x)) (tan(x) + sec(x)) + (cosh (x) - sinh (x))  - cos(x) - 1
```

The function **trigsimp** can simplify this expression with one call:

```
(%i8) trigsimp(e4);
(%o8)                                 0
```

or you can simplify by using trig identities "by hand" as follows:

```
(%i9) ratsubst(cosh(x)^2 -1,sinh(x)^2,e4);
(%o9)        (1 - sin(x)) tan(x) - sec(x) sin(x) + sec(x) - cos(x)
(%i10) subst( sin(x)/cos(x), tan(x), %);
             (1 - sin(x)) sin(x)
(%o10)       ------------------- - sec(x) sin(x) + sec(x) - cos(x)
                   cos(x)
(%i11) ratsimp(%);
          2                                                        2
       sin (x) + (cos(x) sec(x) - 1) sin(x) - cos(x) sec(x) + cos (x)
(%o11) - --------------------------------------------------------------
                                  cos(x)
(%i12) trigreduce(%);
(%o12)                                0
```

3.5 Evaluating Summations

Maxima functions which help evaluate sums include

- **sum** is used to obtain the sum of values of an expression which depends on a "summation index" when the summation index varies over a given range. **sum** is a "verb" or "procedure" which mostly expands, as in

```
(%i1) sum( f(i), i, 2, 3 );
(%o1)                              f(3) + f(2)
```

- **nusum** is a procedure which uses "Gosper's algorithm"for "indefinite hypergeometric series summation". See the web page `http://en.wikipedia.org/wiki/Bill_Gosper`.

  > Ralph William Gosper, Jr., (b. 1943), known as Bill Gosper, is an American mathematician and programmer... Along with Richard Greenblatt, he may be considered to have founded the hacker community, and holds a place of pride in the Lisp community. He is also noted for his work on continued fractional representations of real numbers, and for suggesting the algorithm (which bears his name) for finding closed form hypergeometric identities.

  The **nusum** procedure is not useful with infinite summation limits.

- **unsum** is a backward difference function with the effect **unsum**$(f(n), n); \rightarrow f(n) - f(n-1)$

- **sumcontract** will combine sums whose upper and lower index values differ by constants.

- **intosum** places external factors inside the summation.

- **simplify_sum**, a package by Maxima developer Andrej Vodopivec, which tries both the Gosper algorithm, and, if needed, Zeilberger's algorithm for definite hypergeometric summation (see web page `http://en.wikipedia.org/wiki/Doron_Zeilberger`. You can download the book $A = B$, coauthored by Doron Zeilberger, at the web page `http://www.math.upenn.edu/~wilf/AeqB.html` This book was an outcome of one of the Knuth exercises: "Develop computer programs for simplifying sums that involve binomial coecients." In this book, chapter five discusses Gosper's algorithm, and chapter six discusses Zeilberger's algorithm.

Also discussed is the global flag **simpsum**, which extends the abilities of **sum** to handle cases like **sum**`(a^i,i,i0,i1)` and **sum**`(i^N,i,i0,i1)`.

3.5.1 sum, simpsum, binomial

The Maxima manual entry for **sum** is:

> Function: **sum**`(expr, i, i_0, i_1)`
> Represents a summation of the values of `expr` as the index `i` varies from `i_0` to `i_1`. The noun form **'sum** is displayed as an uppercase letter sigma.
> `sum` evaluates its summand `expr` and lower and upper limits `i_0` and `i_1`, `sum` quotes (does not evaluate) the index `i`.
> If the upper and lower limits differ by an integer, the summand `expr` is evaluated for each value of the summation index `i`, and the result is an explicit sum.
> Otherwise, the range of the index is indefinite. Some rules are applied to simplify the summation. When the global variable `simpsum` is `true`, additional rules are applied. In some cases, simplification yields a result which is not a summation; otherwise, the result is a noun form `'sum`.
> When the evflag (evaluation flag) `cauchysum` is true, a product of summations is expressed as a Cauchy product, in which the index of the inner summation is a function of the index of the outer one, rather than varying independently.
> The global variable `genindex` is the alphabetic prefix used to generate the next index of summation, when an automatically generated index is needed.
> `gensumnum` is the numeric suffix used to generate the next index of summation, when an automatically generated index is needed. When `gensumnum` is `false`, an automatically-generated index is only `genindex` with no numeric suffix.
> See also `sumcontract`, `intosum`, `bashindices`, `niceindices`, `nouns`, `evflag`, and `zeilberger`.

What is the sum of i^2 for $i = 1, 2, ..., 5$?

```
(%i1) sum(i^2, i, 1, 5);
(%o1)                                55
```

You could have used one version of the **do** statement to get the same answer as follows (an example similar to one in the Maxima manual):

```
(%i2) s:0$
(%i3) for i thru 5 do (s : s + i^2);
(%o3)                               done
(%i4) s;
(%o4)                                55
```

(Note that the lower limit is understood to be 1 in the above version.)

A version in which the initial value of the sum is left unspecified is:

```
(%i5) for i thru 5 do (ss : ss + i^2 );
(%o5)                              done
(%i6) ss;
(%o6)                             ss + 55
```

You can evaluate the previous output with $ss = 0$ as follows:

```
(%i7) ev(%, ss = 0 );
(%o7)                               55
```

If you apply the **ev** function to `%o6` again without specifying that $ss = 0$, Maxima evaluates all the variables in `%o6` and re-executes all function calls (here the function call is "plus"):

```
(%i8) ev(%o6);
(%o8)                            ss + 110
```

You can define an equation or expression, etc., containing a sum or sums that are not evaluated by using the noun form **'sum** employing a single quote. Here we write an equation relating y to a sum of powers of x with coefficients a_i using the noun form:

```
(%i9) y = 'sum( a[i]*x^i, i, 0, 6);
                                        6
                                       ====
                                       \         i
(%o9)                              y =  >     a  x
                                       /       i
                                       ====
                                       i = 0
(%i10) eq1 : ev(%, sum);
                         6       5       4       3       2
(%o10)         y = a  x  + a  x  + a  x  + a  x  + a  x  + a  x + a
                    6       5       4       3       2       1      0
```

The explicit equation was generated using the **ev** function with the **sum** function name included. The Maxima manual entry for **ev** begins as:

> **ev**`(expr, arg_1, ..., arg_n)`
> Evaluates the expression `expr` in the environment specified by the arguments `arg_1, ..., arg_n`. The arguments are switches (Boolean flags), assignments, equations, and functions. `ev` returns the result (another expression) of the evaluation.

and then, within a long list of possible arguments, one finds:

> Any other function names (e.g., `sum`) cause evaluation of occurrences of those names in `expr` as though they were verbs.

Maxima is able to simplify some sums defined with symbolic index limits if the global parameter **simpsum** is set **true** (either locally or globally). Here are two examples from the Maxima manual:

```
(%i1) simpsum;
(%o1)                              false
(%i2) sum (2^i + i^2, i, 0, n);
                               n
                              ====
                              \       i    2
(%o2)                          >     (2  + i )
                              /
                              ====
                              i = 0
(%i3) sum (2^i + i^2, i, 0, n), simpsum;
                                  3      2
                         n + 1   2 n  + 3 n  + n
(%o3)                   2      + --------------- - 1
                                       6
(%i4) sum(1/3^i, i, 1, inf);
                                 inf
                                 ====
                                 \     1
(%o4)                             >    --
                                 /      i
                                 ====  3
                                 i = 1
(%i5) sum(1/3^i, i, 1, inf),simpsum;
                                     1
(%o5)                                -
                                     2
```

And an example from the mailing list:

```
(%i1) sum( binomial(s, k), k, 1, s), simpsum:true;
                                   s
(%o1)                             2  - 1
```

The binomial coefficients are provided by the Maxima function **binomial**`(x,y)`, which has the manual description:

> Function: **binomial**`(x, y)`
> The binomial coefficient `x!/(y! (x - y)!)`. If `x` and `y` are integers, then the numerical value of the binomial coefficient is computed. If `y`, or `x - y`, is an integer, the binomial coefficient is expressed as a polynomial.

An alias for **binomial** is **binom**.

```
(%i2) binomial (11, 7);
(%o2)                               330
(%i3) binom(11, 7);
(%o3)                               330
(%i4) 11! / 7! / (11 - 7)!;
(%o4)                               330
```

3.5.2 nusum and unsum

Here is an example of an expression `expr` which Maxima cannot reduce to closed form, even with **simpsum** set equal to **true**.

```
(%i6) e1 : i/(4*i^2 - 1)^2;
                                        i
(%o6)                              -----------
                                       2     2
                                   (4 i  - 1)
(%i7) sum( e1, i, 1, n), simpsum;
                                 n
                                ====
                                \           i
(%o7)                            >     -----------
                                /          2     2
                                ====   (4 i  - 1)
                                i = 1
```

However, **nusum** is successful:

```
(%i8) nusum( e1, i, 1, n);
Dependent equations eliminated:  (3)
                                  n (n + 1)
(%o8)                            ------------
                                            2
                                 2 (2 n + 1)
```

The Maxima manual has the following entry for **nusum**:

> Function: **nusum** `(expr, x, i_0, i_1)`
> Carries out indefinite hypergeometric summation of `expr` with respect to `x` using a decision procedure due to R.W. Gosper. Both `expr` and the result must be expressible as products of integer powers, factorials, binomials, and rational functions.
> The terms ”definite” and ”indefinite summation” are used analogously to ”definite” and ”indefinite integration”. To sum indefinitely means to give a symbolic result for the sum over intervals of variable length, not just e.g. `0` to `inf`. Thus, since there is no formula for the general partial sum of the binomial series, `nusum` can’t do it.
> `nusum` and `unsum` know a little about sums and differences of finite products. See also `unsum`.

The Maxima manual entry for **unsum** is;

> Function: **unsum** `(f, n)`
> Returns the first backward difference `f(n) - f(n - 1)`. Thus `unsum` in a sense is the inverse of `sum`.

Simple examples of the use of **unsum**:

```
(%i9) [unsum(n^2,n), unsum(n^3,n) ];
                                             2
(%o9)                       [2 n - 1, 3 n  - 3 n + 1]
(%i10) unsum( f(n),n );
(%o10)                           f(n) - f(n - 1)
(%i11) sum( f(i), i, n-1,n );
(%o11)                           f(n) + f(n - 1)
```

The simplest example of the use of **nusum** is:

```
(%i12) nusum(j,j,0,2);
(%o12)                                  3
(%i13) nusum(j,j,0,n);
                                   n (n + 1)
(%o13)                             ---------
                                       2
(%i14) nusum(j,j,0,j);
                                   j (j + 1)
(%o14)                             ---------
                                       2
```

If we compare outputs `%o13` and `%o14`, we can interpret the strange syntax **nusum(j, j, 0, j)** as the two step process, `val` : nusum(j,j,0,n) where `val` is not a function of the dummy index `j`, followed by replacing `n` by `j`. Because the `j` that appears in the expression in slot one and slot two is a dummy index, (and could have been called anything), the only permanent variables for the answer are the contents of slot three and four.

3.5.3 intosum and sumcontract

The Maxima manual has the following **intosum** entry

> Function: **intosum** `(expr)`
> Moves multiplicative factors outside a summation to inside. If the index is used in the outside expression, then the function tries to find a reasonable index, the same as it does for sumcontract. This is essentially the reverse idea of the outative property of summations, but note that it does not remove this property, it only bypasses it.
> In some cases, a **scanmap**(**multthru**, `expr`) may be necessary before the **intosum**.

The manual entry for **sumcontract** is:

> Function: **sumcontract** `(expr)`
> Combines all sums of an addition that have upper and lower bounds that differ by constants.
> The result is an expression containing one summation for each set of such summations added to all appropriate extra terms that had to be extracted to form this sum.
> `sumcontract` combines all compatible sums and uses one of the indices from one of the sums if it can, and then trys to form a reasonable index if it cannot use any supplied.
> It may be necessary to do an `intosum(expr)` before the `sumcontract`.

Here is a simple example taken from the mailing list.

```
(%i3) s:sum(log(i),i,1,n+1)-sum(log(i),i,1,n);
                         n + 1            n
                         ====            ====
                         \               \
(%o3)                     >     log(i) -  >     log(i)
                         /               /
                         ====            ====
                         i = 1           i = 1
```

```
(%i4) intosum(%);
                           n + 1            n
                           ====            ====
                           \               \
(%o4)                       >     log(i) +  >     (- log(i))
                           /               /
                           ====            ====
                           i = 1           i = 1
(%i5) sumcontract(%);
(%o5)                               log(n + 1)
```

The following summations are automatically simplified into one sum:

```
(%i6) s2:sum(log(i),i,1,n)+sum(log(i),i,n+1,2*n);
                                     2 n
                                     ====
                                     \
(%o6)                                 >    log(i)
                                     /
                                     ====
                                     i = 1
```

3.5.4 simplify_sum, assume, forget, facts

The powerful simplification function **simplify_sum** has the manual description:

> Function: **simplify_sum** `(expr)`
> Tries to simplify all sums appearing in `expr` to a closed form.
> `simplify_sum` uses Gosper and Zeilberger algorithms to simplify sums.
> To use this function first load the `simplify_sum` package with `load("simplify_sum")`.

This package is the file `simplify_sum.mac` and is located in the `share/contrib/solve_rec/` folder.

In some of the following examples, the functions **assume**, **forget**, and **facts** are used to (in order) add new assumptions about variables, forget certain assumptions, and inquire as to what are the active assumptions in play.

Here are a few examples of the use of **simplify_sum**:

```
(%i1) load("simplify_sum");
(%o1) C:/PROGRA~1/MAXIMA~3.0/share/maxima/5.14.0/share/contrib/solve_rec/simpl\
ify_sum.mac
(%i2) simplify_sum( sum( 1/k, k, 1, inf) );
(%o2)                                 inf
(%i3) simplify_sum( sum( 1/(k+1) , k, 0, inf) );
(%o3)                                 inf
(%i4) simplify_sum( sum( 1/(k+6) - 1/(k+8) , k, 0, inf) );
                                      13
(%o4)                                 --
                                      42
```

```
(%i5) s : sum( binomial(n,k)/(k+1), k, 0, n) ;
                             n
                            ====
                            \      binomial(n, k)
(%o5)                        >     --------------
                            /          k + 1
                            ====
                            k = 0
(%i6) simplify_sum(s);
                                  n + 1
                                 2      - 1
(%o6)                            ----------
                                   n + 1
(%i7) facts();
(%o7)                                []
(%i8) assume(n>k,m > k - 1 );
(%o8)                       [n > k, m - k + 1 > 0]
(%i9) facts();
(%o9)                       [n > k, m - k + 1 > 0]
(%i10) s : sum( (r*binomial(n,r)*binomial(m,k-r))/(k*binomial(n+m,k)),r,0,k);
                   k
                  ====
                  \
                   >    binomial(m, k - r) binomial(n, r) r
                  /
                  ====
                  r = 0
(%o10)            -----------------------------------------
                            k binomial(n + m, k)
(%i11) simplify_sum(s);
                               n (n + m - 1)!
(%o11)           ------------------------------------------
                 k (k - 1)! binomial(n + m, k) (n + m - k)!
(%i12) makefact(%);
                               k! n (n + m - 1)!
(%o12)                        -------------------
                              k (k - 1)! (n + m)!
(%i13) minfactorial(%);
                                      n
(%o13)                              -----
                                    n + m
(%i14) forget(n>k,m > k - 1 );
(%o14)                      [n > k, m - k + 1 > 0]
(%i15) facts();
(%o15)                               []
```

```
(%i16) s:sum((-1)^(k+1)/(2*k-1)^3,k,1,inf);
                                        inf
                                        ====        k + 1
                                        \      (- 1)
(%o16)                                   >     ----------
                                        /              3
                                        ====   (2 k - 1)
                                        k = 1
(%i17) simplify_sum(s);
                                        1         3         1
                                   psi  (-) + 4 %pi    psi  (-)
                                      2 4                 2 4
(%o17)                             ---------------- - -------
                                         128             128
(%i18) ratsimp(%);
                                                3
                                              %pi
(%o18)                                        ----
                                               32
```

A number of Maxima functions have appeared in these examples. We have already discussed the **binomial** function when giving examples of the use of the global flag **simpsum** above.

3.5.5 makefact and minfactorial

The function **makefact** has the description:

> Function: **makefact** `(expr)`
> Transforms instances of `binomial`, `gamma`, and `beta` functions in `expr` into factorials.
> See also `makegamma`.

The function **minfactorial** has the description:

> Function: **minfactorial** `(expr)`
> Examines `expr` for occurrences of two factorials which differ by an integer.
> `minfactorial` then turns one into a polynomial times the other.

with the example:

```
(%i19)  n!/(n+2)!;

                                               n!
(%o19)                                      --------
                                            (n + 2)!
(%i20) minfactorial(%);
                                                1
(%o20)                                   ---------------
                                         (n + 1) (n + 2)
```

3.5.6 psi, bfpsi, bfpsi0, gamma, and %gamma

Finally the **psi** function was generated by the action of **simplify_sum**, and has the manual description:

> Function: **psi**`[n] (x)`
> The derivative of `log (gamma (x))` of order `n+1`. Thus, `psi[0](x)` is the first derivative, `psi[1](x)` is the second derivative, etc.
> Maxima does not know how, in general, to compute a numerical value of `psi`, but it can compute some exact values for rational args. Several variables control what range of rational args `psi` will return an exact value, if possible. See `maxpsiposint`, `maxpsinegint`, `maxpsifracnum`, and `maxpsifracdenom`. That is, `x` must lie between `maxpsinegint` and `maxpsiposint`. If the absolute value of the fractional part of `x` is rational and has a numerator less than `maxpsifracnum` and has a denominator less than `maxpsifracdenom`, `psi` will return an exact value.
> The function `bfpsi` in the `bffac` package can compute numerical values.

The function **psi**`[n] (x)` is the polygamma function of order `n` evaluated at point `x`, and is defined as the `(n+1)'th` derivative of the log of the Gamma function **gamma(x)**. The particular case `n = 0` is the digamma function, which is the first derivative of the log of **gamma(x)**.

For positive integer n, the Gamma function $\Gamma(n) = (n-1)!$. Note that Maxima accepts either **factorial(m)** or `m!`.

The global variable **maxpsiposint** has the description:

> Option variable: **maxpsiposint**
> Default value: 20
> `maxpsiposint` is the largest positive value for which `psi[n](x)` will try to compute an exact value.

The global variable **maxpsinegint** has the description:

> Option variable: **maxpsinegint**
> Default value: -10
> `maxpsinegint` is the most negative value for which `psi[n](x)` will try to compute an exact value. That is if `x` is less than `maxnegint`, `psi[n](x)` will not return simplified answer, even if it could.

The global variable **maxpsifracnum** has the description:

> Option variable: **maxpsifracnum**
> Default value: 6
> Let `x` be a rational number less than one of the form `p/q`. If `p` is greater than `maxpsifracnum`, then `psi[n](x)` will not try to return a simplified value.

The global variable **maxpsifracdenom** has the description

> Option variable: **maxpsifracdenom**
> Default value: 6
> Let `x` be a rational number less than one of the form `p/q`. If `q` is greater than `maxpsifracdenom`, then `psi[n](x)` will not try to return a simplified value.

The **bfpsi** and the **bfpsi0** functions have the description

> Function: **bfpsi**`(n, z, fpprec)`
> Function: **bfpsi0**`(z, fpprec)`
> `bfpsi` is the polygamma function of real argument `z` and integer order `n`.
> `bfpsi0` is the digamma function. `bfpsi0 (z, fpprec)` is equivalent to `bfpsi (0, z, fpprec)`.
> These functions return bigfloat values. `fpprec` is the bigfloat precision of the return value.
> `load ("bffac")` loads these functions.

Here we have a brief look at getting exact and floating point values of the polygamma and digamma functions.

```
(%i1) [maxpsinegint,maxpsiposint,maxpsifracnum,maxpsifracdenom];
(%o1)                          [- 10, 20, 6, 6]
(%i2) load("bffac");
(%o2) C:/PROGRA~1/MAXIMA~3.0/share/maxima/5.14.0/share/numeric/bffac.mac
(%i3) psi[0](6);
                                 137
(%o3)                            --- - %gamma
                                 60
(%i4) float(%);
(%o4)                          1.706117668431801
(%i5) bfpsi0(6,8);
(%o5)                             1.7061177b0
(%i6) float(%);
(%o6)                          1.706117670983076
(%i7) float(%gamma);
(%o7)                          0.57721566490153
(%i8) psi[0](3/4);
                                        %pi
(%o8)                      - 3 log(2) + --- - %gamma
                                         2
(%i9) float(%);
(%o9)                         - 1.085860879786472
(%i10) float(bfpsi0(3/4,8));
(%o10)                        - 1.085860878229141
(%i11) psi[1](3);
                                      2
                                   %pi    5
(%o11)                             ---- - -
                                    6     4
(%i12) float(%);
(%o12)                         0.39493406684823
(%i13) float( bfpsi(1,3,8) );
(%o13)                         0.39493400231004
```

Output `%o7` finds the value of **%gamma**, the Euler-Mascheroni constant.