Maxima

©2010-2016 Mitch Richling https://www.mitchr.me
Last Updated 2016-07-09

Interaction & Help

- End a command ; (Use $ to suppress output)
- Previous or n-th result % %th(n)
- numeric evaluateion *expr*,numer
- Quit Maxima quit()
- Command info & Examples describe(*command*) example(*command*)
- Help search (exact or fuzzy) ? *command* ?? *command*

Input and Output

- loading library load(<lib-name>)
- saving expressions to a file save(*string*,$expr_1$,...)
- loading saved maxima session loadfile(*string*)
- Print expression print
- TEX and FORTRAN output tex fortran
- executing a system command system(*string*)

Variables for Customization

- Setting bfloat precision fpprec fpprintprec
- Interactive printing display2d linel stardisp

Constants

- real positive/negative infinity inf minf
- Pi, e, and imaginary unit %pi %e %i
- boolean constants true false

Operators

- Basic arithmatic + - * / ^
- Noncommutative Product/Exponent . ^^ &: A^^-1 . A = I
- factorial !
- equality & not equal = #
- inequalities < > >= <=
- logical operators and or no

Language Constructs

- block block($expr_1$,..,return(*value*))
- for-loop for *var*:$value_1$ thru $value_2$ ⟦step $value_3$⟧ do *block*
- while-loop for *var*:*value* while *expr* do *block*
- if-then-else if *expr* then $block_1$ else $block_2$
- assignment : &: a : 10;
- define function := &: f(x) := x * x;
- define function f(x):=block($expr_1$,..,return(*expr*))
- lambda expr lambda([var_1,...], *expr*)

Series

- Compute powerseries powerseries(*expr*,*var*,<point>)
- Compute truncated taylor series taylor(*expr*,*var*,<point>,<pow>)

Basic functions

- absolute value abs
- Check for even or odd evenp, oddp
- Check for numeric type integerp bfloatp
- Normal & integer square root sqrt isqrt
- max & min (2 or more args) max, min
- GCD (2 or more args) gcd
- convert to float/bigfloat float bfloat
- rounding numbers round truncate floor ceiling
- numerator and denominator num denom
- real and imaginary parts realpart imagpart
- signum signum
- complex natural logarithm plog
- natural log and exponential log exp
- Trigononumetric sin asin cos acos tan atan atan2
- Hypertrigononumetric sinh asinh cosh acosh tanh atanh
- division remainder mod($expr_1$,$expr_2$)
- binomial coefficient binomial($expr_1$,$expr_2$)
- Product product(*expr*,*var*,$value_{\text{from}}$,$value_{\text{to}}$)
- Sum sum(*expr*,*var*,$value_{\text{from}}$,$value_{\text{to}}$)

Working with expressions

- Add assumption assume(*pred*1,...)
- Add functional dependency depends(var_y, var_x)
- Declare variables $\mathbb{R}$ or $\mathbb{C}$ declare([var_1,...],complex,*var*,real)
- substitute $expr_{\text{sub}}$ for *var* in *expr* subst($expr_{\text{sub}}$,*var*,*expr*)
  subst(*expr*, *var*=$expr_{\text{sub}}$)
  list looks like [var_1=$expr_1$,...] subst(*list*,*expr*)
- combine terms over common denominator xthru(*expr*)
- Factor variables out of *expr* factorout (*expr*, var_1, ...))
- expand expressions expand(*expr*) trigexpand(*expr*)
- factor an expression or integer factor(*expr*)
- Convert to canonical rational form rat(*expr*) trigrat(*expr*)
- Trigonumeric expressions trigreduce(*expr*) trigsimp(*expr*)
- Right & Left Hand Side rhs(*expr*) & lhs(*expr*)
- EXPLICIT coefficient of var^n coeff(*expr*, *var*, *n*)
- Highest EXPLICIT exponent of *var* hipow(*expr*, *var*)

Lists

- create list [$expr_1$,..]
- create copy of list copylist(*list*)
- gets the ith element *list*[i]
- get named elements first second ... ninth tenth last
- checks membership member(*expr*, *list*)
- concatenates lists append($list_1$, $list_2$)
- Length of list length(*list*)
- sort list sort(*list*), sort(*list*,*pred*)
- reverse list reverse(*list*)
- sublist (LISP find_if) sublist(*list*, *pred*)
- Operate element wise + - * / :
- Map func to elements map(*func*, $list_1$,...)
- Reduce via binary op xreduce(*string*, $list_1$) &: xreduce("*", [1,2]);
- Sum over list lsum(*expr*,*var*,*list*)
- Generate list expr makelist(*expr*, *var*≪,n_{beg},n_{end}‖,*list*≫)

Strings And Characters

- create string "..."
- create character "." -- i.e. like a 1 element string
- list of characters in string charlist(*string*)
- substring substring(*string*, n_{beg}⟦,n_{end}⟧)
- String by repeating char smake(*n*, *char*)
- Convert expression to string string(*expr*)
- Convert string to expression eval_string(*string*)
- Display strings with quotes stringdisp

Matrices and Linear Algebra

- Create matrix (args become rows) matrix($list_1$,...)
- Create n_rxn_c zero matrix zeromatrix (n_r, n_c)
- Create *n*x*n* diagonal matrix diagmatrix(*n*, *expr*)
- Create an *n*x*n* identity matrix ident(*n*)
- Create an n_1xn_2 constant matrix *value*+zeromatrix(n_r, n_c)
- Create n_1xn_2 matrix from func genmatrix(*func*, n_r, n_c)
- Create a copy copymatrix(*matrix*)
- map *func* onto each matrix elem matrixmap(*func*,*matrix*)
- Append columns addcol(*matrix*, $list_1$, ...)
- Append rows addrow(*matrix*, $list_1$, ...)
- Get column and return as matrix col(*matrix*,*n*)
- Get row and return as matrix row(*matrix*,*n*)
- Element at (n_r, n_c) *matrix*[n_r, n_c]
- Dimensions as list: [n_r, n_c] matrix_size(*matrix*)
- Compute adjoint of a matrix adjoint(*matrix*)
- Compute determinant determinant(*matrix*)
- Compute charactstic polynomial charpoly(*matrix*,*var*)
- Compute eigenvalues eigenvalues(*matrix*)
- Compute eigenvectors eigenvectors*matrix*)
- Compute inverse invert(*matrix*)
- Compute the rank rank(*matrix*)
- transpose transpose(*matrix*)
- Compute upper triangular form triangularize(*matrix*)
- Compute echelon form echelon(*matrix*)
- Compute Cholesky factorization cholesky(*matrix*)

Prime Numbers

- Prime factorization of *n* ifactors(*n*) factor(*n*)
- Check primality primep
- Prime number enumeration next_prime prev_prime
- Eulers totient-function totient

Random Numbers

- A random number, or *n* in a list, from U[0,*value*) random(*value*⟦,*n*⟧)
- Generators in 'load(distrib)'
  - random_cauchy
  - random_chi2
  - random_exp
  - random_f
  - random_normal
  - random_pareto
  - random_geometric
  - random_poisson
  - random_bernoulli
  - random_pareto
  - random_beta
  - random_weibull
  - random_rayleigh
  - random_gamma
  - random_gumbel
  - random_student_t
  - random_lognormal
  - random_laplace
  - random_binomial
  - random_continuous_uniform
  - random_discrete_uniform
  - random_negative_binomial
  - random_noncentral_chi2
  - random_noncentral_student_t
  - random_hypergeometric
  - random_logistic

Polynomials

- Coefficient of var^n ratcoef(*expr*, *var*, *n*)
- Convert to horners form horner(*poly*,*var*)
- List with quotient and remainder divide($poly_1$,$poly_2$)
- Just the quotient quotient($poly_1$,$poly_2$)
- Just the remainder remainder($poly_1$,$poly_2$)
- List [*a*, *b*, GCD] such that a $poly_1$ + b $poly_2$ = GCD gcdex($poly_1$, $poly_2$)
- Greatest common divisor gcd($poly_1$,...)
- Eliminate vars from equations eliminate([$expr_1$,...], [var_1,...])
- Factor polynomial over $\mathbb{Q}$ or $\mathbb{Z}[i]$ factor(*poly*) gfactor(*poly*)
- Product of polynomials fasttimes($poly_1$, $poly_2$)

Calculus

- limit *expr* as *var* → *value* limit(*expr*,*var*,*value*⟦,PLUS ‖ MINUS⟧)
- derivative of order *n* diff(*expr*,*var*⟦,*n*⟧)
- antiderivative integrate(*expr*,*var*)
- Definate integral integrate(*expr*,*var*,$value_{\text{low}}$,$value_{\text{high}}$)

Equations

- Roots of a real polynomial allroots(*poly*)
- Rational approx of poly roots realroots(*poly*, $value_{\text{tolerance}}$)
- Solve system of equations solve([$expr_1$,..],[var_1,...])
- finds zero in [$value_{\text{low}}$,$value_{\text{high}}$] find_root(*expr*,*var*,$value_{\text{low}}$,$value_{\text{high}}$, [$value_{\text{abserr}}$, $value_{\text{relerr}}$])
- Use bisection (see find_root) bf_find_root(...)
- Search for zero via Newton 'load(newton1)' loads this function newton(*expr*,*var*,$value_{\text{guess}}$,$value_{\text{eps}}$)
- Solve 2nd order ODE BVP bc2(<solution>, $value_{x1}$, $value_{y1}$, $value_{x2}$, $value_{y2}$)
- Solve system of linear ODEs desolve([$expr_1$,...], [var_1,...])
- Solve 1st order ODE IVP ic1(<solution>, $value_x$, $value_y$)
- Solve 2nd order ODE IVP ic2(<solution>, $value_x$, $value_y$, $value_d$)
- Solve 1st or 2nd order ODE ode2(*expr*, var_y, var_x)
  - Use depends or ' depends(y,x); ode2(diff(y,x)=(1-2*x)/(3*y^2-4),y,x);
  - Use depends or ' ode2('diff(y,x)=(1-2*x)/(3*y^2-4),y,x);

Plotting

- 2D plot2d(*expr*,[*var*,$value_{\text{low}}$,$value_{\text{high}}$])
- 3D plot3d(*expr*,[var_1,$value_{\text{low}}$,$value_{\text{high}}$],[var_2,$value_{\text{low}}$,$value_{\text{high}}$])
- ODE plotdf(*expr*) Ex: plotdf((1-2*x)/(3*y^2-4));

Numerical Integration

- Integrate *expr* over [$value_{\text{low}}$,$value_{\text{high}}$] quad_qag(*expr*,var_1,$value_{\text{low}}$,$value_{\text{high}}$,n_{key}, [$value_{\text{epsrel}}$,$value_{\text{epsabs}}$,$value_{\text{limit}}$]) $value_{\text{key}}$ is Gauss-Kronrod order
- Integrate *expr* over [$value_{\text{low}}$,$value_{\text{high}}$] quad_qags(*expr*,var_1,$value_{\text{low}}$,$value_{\text{high}}$, [$value_{\text{epsrel}}$,$value_{\text{epsabs}}$,$value_{\text{limit}}$])