

Basic Operations

<code>expr.subs([(x, 2), (y, 4), (z, 0)])</code>	substitute x with 2 etc.
<code>sympify(str_expr)</code>	convert strings into SymPy expressions
<code>expr.evalf(15, chop=True)</code>	evaluate a numerical expression into a floating point number
<code>lambdify(x, expr, "numpy")</code>	converts the SymPy names to the names of the given numerical library
<code>init_printing()</code>	This will automatically enable the best printer available in your environment.
<code>simplify(expr)</code>	simplify mathematical expressions
<code>expand(expr)</code>	expand polynomial expressions
<code>factor(expr)</code>	takes a polynomial and factors it into irreducible factors over the rational numbers
<code>factor_list(expr)</code>	returns a list with the factors. More structured.
<code>collect(expr, x)</code>	collects common powers of a term in an expression
<code>cancel(expr)</code>	take any rational function and put it into the standard canonical form
<code>apart(expr)</code>	performs a partial fraction decomposition on a rational function

Matrices

<code>Matrix([1, 2, 3])</code>	matrix constructor(mutable matrix)
<code>shape(expr)</code>	shape of matrix
<code>M.row(0)</code>	get the first row
<code>M.col(-1)</code>	get the last column
<code>M.col_del(0)</code>	delete first column
<code>M.row_del(1)</code>	delete second row
<code>M.row_insert(1, Matrix([[0, 4]]))</code>	insert a row
<code>M.col_insert(0, Matrix([1, -2]))</code>	insert a column
<code>M**-1</code>	inverse of M
<code>M.T</code>	transpose of M
<code>eye(n)</code>	create a nxn identity matrix
<code>zeros(n,m)</code>	creates a nxm matrix of zeroes
<code>ones(n,m)</code>	creates a nxm matrix of ones
<code>diag(expr)</code>	creates a matrix with expr in the diagonal
<code>M.det()</code>	computes the determinant of M
<code>M.rref()</code>	put a matrix into reduced row echelon form
<code>M.nullspace()</code>	returns a list of column vectors that span the nullspace of the matrix
<code>M.columnspace()</code>	returns a list of column vectors that span the column-space of the matrix
<code>M.eigenvals()</code>	eigenvals returns a dictionary of eigenvalue: algebraic_multiplicity pairs

Matrices (cont)

<code>M.eigenvals()</code>	returns a list of tuples of the form (eigenvalue, algebraic_multiplicity, [eigenvectors])
<code>M.diag(ona- lize())</code>	returns a tuple (P, D), where D is diagonal and $M = P D P^{-1}$
<code>M.charpoly(lambda)</code>	return the characteristic polynomial

Trigonometric Simplification

<code>trigsimp(expr)</code>	simplify expressions using trigonometric identities
<code>expand_trig(expr)</code>	expand trigonometric functions

Powers

<code>powsimp(expr)</code>	use power identities
<code>expand_power_exp(x**(a + b))</code>	$x^a * x^b$
<code>expand_power_base((xy)**a)</code>	$x^a * y^a$
<code>powdenest((x**a)**b)</code>	x^{a*b}

Exponentials and logarithms

<code>expand_log(expr)</code>
<code>logcombine(expr)</code>

Special Functions

<code>factorial(n)</code>	return the factorial of n
<code>binomial(n, k)</code>	return the binomial coefficient of n and k
<code>gamma(z)</code>	return the gamma function
<code>expr.rewrite(function)</code>	rewrite expr in terms of function
<code>expand_func(expr)</code>	expand special functions



By gloo13

cheatography.com/gloo13/

Not published yet.

Last updated 13th August, 2023.

Page 1 of 2.

Sponsored by **Readable.com**

Measure your website readability!

<https://readable.com>

Special Functions (cont)

hypere- rewrite hyper in terms of more
x-
pand(- standard functions
expr)

combsi- simplify combinatorial expres-
mp(- sions
expr)

gammas simplify expressions with
imp- gamma functions or combin-
(expr) atorial functions

Assumptions

positive negative

real complex

integer

expr.a- The full set of known predicates
ssu- for a symbol

mpt-
ions0

posify- replace all symbols in an
(expr) expression with symbols that
have the assumption positi-
ve=True

Calculus

diff(expr, x, n) nth order derivative of
expr in terms of x

Derivative(expr, create an unevaluated
x, n) derivative

deriv.doit() evaluate an unevaluated
derivative

integrate(expr, x, integrate expr from a to
a, b) b

Integral(expr, x, create an unevaluated
n) integral

limit(expr, x, xo) limit of expr to xo

Limit(expr, x, xo) create an unevaluated
limit

expr.series(x, nth order series
x0, n) expansion of expr
around x0

expr.series(x, remove O notation
x0, n).rem-
oveO()

Calculus (cont)

differentiat- differentiate using finite
e_finite(- differences
expr)

expr.as_f- generate approximations of
inite_differ- the derivative to arbitrary
ence() order

Solvers

solveset(expr, x, solve expr=0
domain=S.Com-
plexes, dict=False)

linsolve([expr1, solve a linear system
expr2, ...], (x, y, ...)) of equations

nonlinsolve([expr1, solve a non linear
expr2, ...], [x, y, ...]) system of equations

dsolve(diffeq, f(x)) solves differential
equation diffeq

roots(expr, x) o get the solutions of
a polynomial
including multiplicity