

# MAXIMA COMMAND TOUR

**警告** あなたが必要とする情報は掲載されていません。



# 目次

%	1
%	1
%c	1
%e	2
%gamma	2
%i	2
%phi	2
%pi	3
%th(自然数)	3
<b>A</b>	<b>5</b>
abs(数)	5
acos(数・式)	5
acosh(数・式)	6
acoth(数・式)	6
acsch(数・式)	6
addcol(行列, リスト・行列)	7
addrow(行列, リスト・行列)	7
adjoin(要素, 集合)	8
adjoint(行列)	8
allbut(番号, 番号, ...)	9
allroots(多項式)	9
asec(数・式)	9
asech(数・式)	10
asin(数・式)	10
asinh(数・式)	10
assume(等式・不等式)	11
atan(数・式)	11

atanh(数・式) . . . . .	12
augcoefmatrix([f, g, ...], [x, y, ...]) . . . . .	12
<b>B</b>	<b>13</b>
belln(数) . . . . .	13
bern(数) . . . . .	13
bfloat(数) . . . . .	13
bfloatp(数) . . . . .	14
binomial(n, r) . . . . .	14
block(処理1, 処理2, ..., 処理n) . . . . .	15
box(処理) または box(処理, タイトル) . . . . .	15
<b>C</b>	<b>17</b>
cardinality(集合) . . . . .	17
carg(複素数) . . . . .	17
ceiling(数) . . . . .	18
cf(数・リスト) . . . . .	18
cfdisrep(リスト) . . . . .	19
charat(文字, 自然数) . . . . .	19
charlist(文字) . . . . .	19
charpoly(行列, 変数) . . . . .	19
coeff(式, 変数, 次数) . . . . .	20
coefmatrix([f, g, ...], [x, y, ...]) . . . . .	20
col(行列, 自然数) . . . . .	21
concat(a, b, ...) . . . . .	21
conjugate(複素数) . . . . .	21
cons(元, リスト) . . . . .	22
copylist(リスト) . . . . .	22
copymatrix(行列) . . . . .	23
cos(数・式) . . . . .	23
cosh(数・式) . . . . .	24
cot(数・式) . . . . .	24
coth(数・式) . . . . .	24
csc(数・式) . . . . .	25
csch(数・式) . . . . .	25
cylindrical(r, z, z min, z max, $\varphi$ min, $\varphi$ max) . . . . .	25

D	27
define(関数, 式)	27
demo(ファイル名)	27
denom(式)	28
diagmatrix(次数, 成分)	29
diff(関数, 変数1, 階数1, 変数2, 階数2, ...)	29
disjoin(要素, 集合)	30
display(式)	30
display2d	30
divide(割られる多項式, 割る多項式)	31
divisors(整数)	31
divsum(整数, 次数)	32
dontfactor: リスト	32
E	35
ematrix(m, n, x, i, j)	35
endcons(要素, リスト)	35
enhanced3d = true	35
entier(数)	36
evenp(引数)	36
explicit(関数, 変数1, a1, b1, 変数2, a2, b2)	37
expand(式)	37
exponentialize(式) または exponentialize	38
ezgcd(多項式1, 多項式2, 多項式3, .....)	39
F	41
factcomb(階乗の式)	41
factor(数・式)	41
factorflag: false	42
facts()	42
fib(整数)	43
fibtophi(式)	43
filled_func = false	43
find_root(式, a, b)	44
fix(数)	45
flatten(集合・リスト)	45

floor(数)	45
for	46
forget(性質1, 性質2, ...)	46
fpprec: 16	47
freeof(x 1, x 2, ..., 式)	47
<b>G</b>	<b>49</b>
gamma(数)	49
gcd(数・式, 数・式)	49
gcdex(数・式, 数・式)	50
genmatrix(配列, i2, j2, i1, j2)	50
geometric_mean(リスト)	51
gfactor(多項式)	51
globalsolve: false	51
grind(式)	52
<b>H</b>	<b>55</b>
halfangles: false	55
hankel(リスト1, リスト2)	55
hermite(整数, 変数)	56
hessian(関数, 変数リスト)	57
hipow(式, 変数)	57
histogram(リスト, オプション1, ...)	58
<b>I</b>	<b>59</b>
ibase: 10	59
ident(自然数)	59
if 条件 then 式1 else 式2	60
ifactors(自然数)	60
ilt(関数, s, t)	60
imagpart(複素数)	61
implicit(方程式, x, x min, x max, y, y min, y max)	61
inpart(式, 番号1, 番号2, ...)	62
inprod(ベクトル1, ベクトル2)	63
integer_partitions(自然数, 長さ)	63
integrate(関数, 変数, 下端, 上端)	63
intersect(集合1, 集合2, ...)	64

intosum(式) . . . . .	64
inv_mod(整数, 法) . . . . .	65
is(等式・不等式) . . . . .	65
<b>J</b> . . . . .	<b>67</b>
jacobi(整数, 奇数) . . . . .	67
jacobian([関数1, 関数2, ...], [変数1, 変数2, ...]) . . . . .	68
jordan(行列) . . . . .	68



%

%

直前の結果を参照します。

```
(%i1) (x + y)^3;
```

```
(%o1) (y + x)3
```

```
(%i2) expand(%);
```

```
(%o2) y3 + 3 x y2 + 3 x2 y + x3
```

%c

積分定数です。

```
(%i1) diff(f(x), x) = x;
```

```
(%o1)  $\frac{d}{dx} (f(x)) = x$ 
```

```
(%i2) ode2(%, f(x), x);
```

```
(%o2)  $f(x) = \frac{x^2}{2} + \%c$ 
```

```
(%i3) diff(%, x);
```

```
(%o3)  $\frac{d}{dx} (f(x)) = x$ 
```

**%e**

自然対数の底 (Napier<sup>\*1</sup> の数) を表す定数です。定義は  $e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$  です。

```
(%i1) float(%e);
(%o1)                2.718281828459045
(%i2) log(%e);
(%o2)                1
```

**%gamma**

Euler–Mascheroni 定数  $\gamma = \lim_{n \rightarrow \infty} \left(\sum_{k=1}^n \frac{1}{k} - \log n\right)$  を表す定数です。

```
(%i1) float(%gamma);
(%o1)                .5772156649015329
```

**%i**

虚数単位です。

```
(%i1) %i^2;
(%o1)                - 1
```

**%phi**

黄金比  $\phi = \frac{1 + \sqrt{5}}{2}$  を表す定数です。

<sup>\*1</sup> John Napier (1550-1617) : スコットランドの数学者

```
(%i1) float(%phi);
(%o1) 1.618033988749895
(%i2) subst(%phi, x, x^2 - x - 1);
(%o2) %phi2 - %phi - 1
(%i3) ratsimp(%);
(%o3) %phi2 - %phi - 1
(%i4) ratsimp(%), algebraic: ture;
(%o4) 0
```

## %pi

円周率  $\pi$  を表す定数です。

```
(%i1) float(%pi);
(%o1) 3.141592653589793
(%i2) sin(%pi);
(%o2) 0
(%i3) cos(%pi);
(%o3) - 1
```

## %th(自然数)

自然数  $n$  を引数にとり、 $n$  個前の結果を参照します。

```
(%i1) 4!;
(%o1) 24
(%i2) 5!;
```

```
(%o2) 120
(%i3) 6!;
(%o3) 720
(%i4) display(%th(3));
      %th(3) = 24
(%o4) done
```

# A

## abs(数)

実数や複素数の絶対値を返す関数です。

```
(%i1) abs(-100);
(%o1)                                100
(%i2) solve(x^2 + 1, x);
(%o2)                                [x = - %i, x = %i]
(%i3) abs(%);
(%o3)                                [abs(x) = 1, abs(x) = 1]
```

## acos(数・式)

逆余弦関数、すなわち閉区間  $0 \leq x \leq \pi$  を定義域とした余弦関数  $f(x) = \cos x$  の逆関数  $f^{-1}(x) = \arccos x$  です。

```
(%i1) acos(1/sqrt(2));
(%o1)                                %pi
                                      ---
                                      4
(%i2) cos(acos(x));
(%o2)                                x
```

## acosh(数・式)

逆双曲線余弦関数、すなわち左閉無限区間  $0 \leq x < \infty$  を定義域としたときの双曲線余弦関数  $\cosh x = \frac{e^x + e^{-x}}{2}$  の逆関数です。  $\cosh^{-1} x = \log(x + \sqrt{x^2 - 1})$  と表せます。

```
(%i1) acosh(1);
(%o1) 0
(%i2) acosh(2), numer;
(%o2) 1.316957896924817
```

## acoth(数・式)

逆双曲線余接関数、すなわち双曲線余接関数  $\coth x = \frac{1}{\tanh x}$  の逆関数です。  $\coth^{-1} x = \frac{1}{2} \log \frac{x+1}{x-1}$  と表せます。

```
(%i1) coth(acoth(x));
(%o1) x
(%i2) acoth(coth(x));
(%o2) x
```

## acsch(数・式)

逆双曲線余割関数、すなわち双曲線余割関数  $\operatorname{csch} x = \frac{1}{\sinh x}$  の逆関数です。  $\operatorname{csch}^{-1} x = \operatorname{sgn} x \log \frac{1 + \sqrt{1 + x^2}}{|x|}$  と表せます。ここで、 $\operatorname{sgn} x$  は  $x$  の符号です。

```
(%i1) acsch(x) + acsch(-x);
(%o1) 0
```

## addcol(行列, リスト・行列)

行列の横にリストや行列を連結した行列を返す関数です。第 1 引数は行列でなければなりません。

```
(%i1) M: matrix([a, b], [c, d]);  
(%o1)          [ a b ]  
              [ c d ]  
  
(%i2) addcol(M, [1, 2]);  
(%o2)          [ a b 1 ]  
              [ c d 2 ]  
  
(%i3) N: matrix([1, 2], [3, 4]);  
(%o3)          [ 1 2 ]  
              [ 3 4 ]  
  
(%i4) addcol(M, N);  
(%o4)          [ a b 1 2 ]  
              [ c d 3 4 ]
```

## addrow(行列, リスト・行列)

行列の縦にリストや行列を連結した行列を返す関数です。第 1 引数は行列でなければなりません。

```
(%i1) M: matrix([a, b], [c, d]);  
(%o1)          [ a b ]  
              [ c d ]  
  
(%i2) addrow(M, [1, 2]);  
(%o2)          [ a b ]  
              [ c d ]  
              [ 1 2 ]
```

```
(%i3) N: matrix([1, 2], [3, 4]);

(%o3)          [ 1  2 ]
              [ 3  4 ]

(%i4) addrow(M, N);

(%o4)          [ a  b ]
              [ c  d ]
              [ 1  2 ]
              [ 3  4 ]
```

## adjoin(要素, 集合)

集合 (第 2 引数) に要素 (第 1 引数) を追加した集合を返す関数です。

```
(%i1) adjoin(x, {a, b, c, d, e});

(%o1)          {a, b, c, d, e, x}

(%i2) adjoin(a, {a, b, c, d, e});

(%o2)          {a, b, c, d, e}
```

## adjoint(行列)

余因子行列を返す関数です。

```
(%i1) A: matrix([a, b],[c, d]);

(%o1)          [ a  b ]
              [ c  d ]

(%i2) B: adjoint(%);

(%o2)          [ d  - b ]
              [ - c  a ]

(%i3) A . B;
```

```
(%o3)      [ a d - b c      0      ]
           [      0      a d - b c ]
```

## allbut(番号, 番号, ...)

part系コマンド (part, inpart, substpart, substinpart, dpart, lpart) や kill コマンドを実行する際、除外番号を指定する関数です。

```
(%i1) x[1] + x[2] + x[3] + x[4] + x[5] + x[6];
(%o1)      x   + x   + x   + x   + x   + x
           6   5   4   3   2   1

(%i2) part(% , allbut(1, 3, 4));
(%o2)      x   + x   + x
           5   2   1
```

## allroots(多項式)

一変数多項式  $f(x)$  を引数にとり、方程式  $f(x) = 0$  の全ての近似解を求める関数です。なお、変数は  $x$  である必要はありません。

```
(%i1) allroots(w^3 + 1);
(%o1) [w = .8660254037844386 %i + 0.5,
       w = 0.5 - .8660254037844386 %i, w = - 1.0]
```

多項式  $w^3+1$  は  $(w+1)(w^2-w+1)$  と因数分解でき、方程式  $w^2-w+1=0$  の解  $w = \frac{1 \pm \sqrt{-3}}{2}$  です。これを小数で表したものが、上の2つの複素数解です。

## asec(数・式)

逆正割関数、すなわち定義域を  $0 \leq x < \frac{\pi}{2}$  および  $\frac{\pi}{2} < x \leq \pi$  としたときの正割関数  $\sec x = \frac{1}{\cos x}$  の逆関数です。

```
(%i1) asec(x) + asec(-x);
```

```
(%o1) %pi
```

### asech(数・式)

逆双曲線正割関数、すなわち双曲線正割関数  $\operatorname{sech} x = \frac{1}{\cosh x}$  の逆関数です。  $\operatorname{sech}^{-1} x = \log \frac{1 + \sqrt{1-x^2}}{|x|}$  と表せます。

```
(%i1) asech(x) + asech(-x);
```

```
(%o1) 2 asech(x)
```

### asin(数・式)

逆正弦関数、すなわち閉区間  $-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}$  を定義域としたときの正弦関数  $\sin x$  の逆関数です。

```
(%i1) asin(-1/2);
```

```
(%o1) -  $\frac{\%pi}{6}$ 
```

```
(%i2) sin(asin(x));
```

```
(%o2) x
```

### asinh(数・式)

逆双曲線正弦関数、双曲線正弦関数  $\sinh x = \frac{e^x - e^{-x}}{2}$  の逆関数です。  $\sinh^{-1} x = \log(x + \sqrt{x^2 + 1})$  と表せます。

```
(%i1) asinh(1), numer;
(%o1)                                0.881373587019543
(%i2) asinh(x) + asinh(-x);
(%o2)                                0
```

## assume(等式・不等式)

大小関係の性質を割り当てる関数です。利用できる比較演算子は <, <=, equal, notequal, >, >= です。

```
(%i1) sqrt(a^2);
(%o1)                                abs(a)
(%i2) assume(a <= 0);
(%o2)                                [a <= 0]
(%i3) sqrt(a^2);
(%o3)                                - a
```

なお、現在割り当てられている性質を調べるには関数 `facts(式)` を、また、割り当てられている性質を解除するには関数 `forget(性質)` を用います。

## atan(数・式)

逆正接関数、すなわち開区間  $-\frac{\pi}{2} < x < \frac{\pi}{2}$  を定義域としたときの正接関数  $\tan x$  の逆関数です。

```
(%i1) atan(sqrt(3));
(%o1)                                %pi
                                      ---
                                      3
(%i2) sin(atan(x));
(%o2)                                x
                                      -----
```

$$\sqrt{x^2 + 1}$$

### atanh(数・式)

逆双曲線正接関数、双曲線正接関数  $\tanh x = \frac{\sinh x}{\cosh x}$  の逆関数です。  $\tanh^{-1} x = \frac{1}{2} \log \frac{1+x}{1-x}$  と表せます。

```
(%i1) atanh(0.5);
```

```
(%o1) .5493061443340549
```

### augcoefmatrix([f, g, ...], [x, y, ...])

方程式のリスト [f, g, ...] と変数のリスト [x, y, ...] を引数にとり、定数項を含めた係数行列を生成する関数。

```
(%i1) [4 * x - 3 * y = -1, 2 * x + y = 5];
```

```
(%o1) [4 x - 3 y = - 1, y + 2 x = 5]
```

```
(%i2) augcoefmatrix(%, [x, y]);
```

```
(%o2) [ 4  - 3   1 ]
      [ 2   1  - 5 ]
```

## B

### belln(数)

0以上の整数  $n$  を引数にとり、 $n$  番目の Bell<sup>\*1</sup> 数を返す関数です。

```
(%i1) makelist(belln(i), i, 0, 7);
(%o1) [1, 1, 2, 5, 15, 52, 203, 877]
```

なお、Bell 数  $B_n$  とは、 $n$  個の元からなる集合を部分集合の直和で表す方法の個数です。

### bern(数)

0以上の整数  $n$  を引数にとり、 $n$  番目の Bernoulli<sup>\*2</sup> 数<sup>\*3</sup>を返す関数です。

```
(%i1) makelist(bern(i), i, 0, 7);
(%o1) [1, - 1/2, 1/6, 0, - 1/30, 0, 1/42, 0]
```

なお、Bernoulli 数  $B_n$  とは、関数  $f(x) = \frac{x}{e^x - 1}$  の Maclaurin<sup>\*4</sup> 展開  $f(x) = \sum_{n=0}^{\infty} \frac{B_n}{n!} x^n$  の係数として定義される数です。

### bfloat(数)

変数 fpprec に設定された桁数（標準は 16）の小数に変換する関数です。

<sup>\*1</sup> Eric Temple Bell (1883-1960) : スコットランドの数学者であり、作家

<sup>\*2</sup> Jakob Bernoulli (1654-1705) : スイスの数学者

<sup>\*3</sup> 最初の発見者は關孝和

<sup>\*4</sup> Colin Maclaurin (1698-1746) : スコットランドの数学者

```
(%i1) bfloat(%pi);
(%o1) 3.141592653589793b0
(%i2) block([fpprec: 100], bfloat(%pi));
(%o2) 3.141592653589793238462643383279502884197169399375105820974944\
592307816406286208998628034825342117068b0
```

出力の末尾に表れる **b0** は  $\times 10^0$  を意味しています。

## **bfloatp(数)**

引数が多倍長小数 (bigfloat) なら **true** を、そうでなければ **false** を返す関数です。

```
(%i1) float(sqrt(2));
(%o1) 1.414213562373095
(%i2) bfloatp(%);
(%o2) false
(%i3) bfloat(sqrt(2));
(%o3) 1.414213562373095b0
(%i4) bfloatp(%);
(%o4) true
```

## **binomial(n, r)**

二項係数  $\frac{n!}{r!(n-r)!}$  を返す関数です。

```
(%i1) binomial(n, 5);
(%o1) 
$$\frac{(n-4)(n-3)(n-2)(n-1)n}{120}$$

```

cf. makefact

## block(処理<sub>1</sub>, 処理<sub>2</sub>, ..., 処理<sub>n</sub>)

複数の処理を順に実行していく関数です。最後の処理の結果のみが出力されます。鍵括弧 [ ] で括られた処理は実行後に破棄されるため、局所変数を利用したプログラムの作成に利用されます。

```
(%i1) block(a: 1, b: 2, c: 3);
(%o1)                                     3
(%i2) display([a, b, c]);
                                     [a, b, c] = [1, 2, 3]
(%o2)                                     done
(%i3) block([x: 1], y: 2, z: y + 1);
(%o3)                                     3
(%i4) display([x, y, z]);
                                     [x, y, z] = [x, 2, 3]
(%o4)                                     done
```

なお、関数 block を用いずに複数の処理をカンマ (,) で区切って並べた場合、2 番目以降の処理を条件として、1 番目の処理が実行されます。

```
(%i5) [u, v, w], u:1, v:2, w:3;
(%o5)                                     [1, 2, 3]
(%i6) display([u, v, w]);
                                     [u, v, w] = [u, v, w]
(%o6)                                     done
```

## box(処理) または box(処理, タイトル)

出力結果全体を囲む関数です。タイトルをつけることも出来ます。

```
(%i1) box(expand((x - y)^4));
```

```
(%o1)      .....  
" 4      3      2 2      3      4"  
"y  - 4 x y  + 6 x  y  - 4 x  y  + x  "  
      .....
```

```
(%i2) box(expand((x - y)^4), "Example");
```

```
(%o2)      "Example".....  
" 4      3      2 2      3      4"  
"y  - 4 x y  + 6 x  y  - 4 x  y  + x  "  
      .....
```

# C

## cardinality(集合)

集合の元（要素）の個数を返す関数です。

```
(%i1) s: {1, 2, 2, 3, 3, 3};
(%o1)                                     {1, 2, 3}
(%i2) cardinality(s);
(%o2)                                     3
(%i3) simp: false;
(%o3)                                     false
(%i4) t: {1, 2, 2, 3, 3, 3};
(%o4)                                     {1, 2, 2, 3, 3, 3}
(%i5) cardinality(t);
(%o5)                                     3
```

## carg(複素数)

複素数の偏角を返す関数です。

```
(%i1) carg(1 + %i);
(%o1)                                      $\frac{\%pi}{4}$ 
(%i2) carg(1 + 2*%i);
```

```
(%o2) atan(2)
```

## ceiling(数)

引数以上の最小の整数を返す関数です。すなわち、引数が  $x$  のとき、 $n-1 < x \leq n$  を満たす整数  $n$  を返します。

```
(%i1) ceiling(%pi);
(%o1) 4
(%i2) ceiling(-sqrt(2));
(%o2) - 1
```

## cf(数・リスト)

有理数や平方根を含む式（実 2 次体の元）を引数にとり、正則連分数（分子が全て 1 の連分数）に展開する関数です。引数が  $a + \frac{1}{b + \frac{1}{c + \dots}}$  と連分数展開された場合、結果はリス

ト形式  $[a, b, c, \dots]$  で出力されます。リスト形式で表された正則連分数を引数とすることも出来ます。

```
(%i1) cf(30/13);
(%o1) [2, 3, 4]
(%i2) 2 + 1/(3 + 1/4);
(%o2) 30
-----
13
(%i3) cf([2, 3, 4] / 2);
(%o3) [1, 6, 2]
(%i4) 1 + 1/(6 + 1/2);
(%o4) 15
-----
13
```

## cfdisrep(リスト)

リスト  $[a, b, c, \dots]$  を引数にとり、連分数  $a + \frac{1}{b + \frac{1}{c + \dots}}$  に変換し、出力する関数です。

```
(%i1) cfdisrep([a, b, c, d, e]);
(%o1)      a + -----
              1
            b + -----
                  1
                c + -----
                      1
                    d + -----
                          e
```

## charat(文字, 自然数)

文字列から 1 字を取り出す関数です。

```
(%i1) charar("I love Maxima.", 5);
(%o1)      v
```

## charlist(文字)

文字列を 1 字ずつのリストに分解する関数です。日本語は使えません。

```
(%i1) charlist("I love Maxima.");
(%o1)      [I, , l, o, v, e, , M, a, x, i, m, a, .]
```

## charpoly(行列, 変数)

行列の固有多項式 (特性多項式) を返す関数です。

```
(%i1) M: matrix([a, b], [c, d]);
(%o1)          [ a  b ]
              [ c  d ]
(%i2) charpoly(M, x);
(%o2)          (a - x) (d - x) - b c
```

なお、行列  $M$  の固有多項式とは、行列式  $|M - xE|$  のことです。

### coeff(式, 変数, 次数)

式  $f$  および変数  $x$ 、次数  $n$  を引数にとり、 $f$  における  $x^n$  の係数を返す関数です。同様に係数を返す関数 `ratcoeff` と違い、展開や因数分解を行わず、あるがままの状態の係数を返します。

```
(%i1) (a*x + b)*(c*x + d) - x^2;
(%o1)          (a x + b) (c x + d) - x2
(%i2) coeff(%, x, 2);
(%o2)          - 1
(%i3) (x + 1)/a + x/b;
(%o3)          x + 1   x
              ----- + -
                 a     b
(%i4) coeff(%, x, 1);
(%o4)          1
              -
              b
```

なお、次数が 1 の場合は `coeff(式, 変数)` のように省略できます。

### coefmatrix([f, g, ...], [x, y, ...])

連立方程式  $[f, g, \dots]$  と変数  $[x, y, \dots]$  を引数にとり、係数行列を生成する関数です。

```
(%i1) coefmatrix([a*x + b*y = c, d*x + e*y = f], [x, y]);
```

```
(%o1)      [ a  b ]
           [ d  e ]
```

## col(行列, 自然数)

行列  $M$  と自然数  $i$  を引数にとり、行列  $M$  の第  $i$  列を返す関数です。

```
(%i1) M: matrix([1, 2, 3], [4, 5, 6], [7, 8, 9]);
```

```
(%o1)      [ 1  2  3 ]
           [ 4  5  6 ]
           [ 7  8  9 ]
```

```
(%i2) col(M, 2);
```

```
(%o2)      [ 2 ]
           [ 5 ]
           [ 8 ]
```

## concat(a, b, ...)

引数を連結する関数です。

```
(%i1) concat("M", "a", "x", "i", "m", "a");
```

```
(%o1)      Maxima
```

cf. sconcat

## conjugate(複素数)

共役複素数を返す関数です。

```
(%i1) conjugate(5 + 3 * %i);
(%o1)                    5 - 3 %i
```

## cons(元, リスト)

リスト  $L$  と要素  $x$  を引数にとり、 $L$  の先頭に  $x$  を追加したリストを生成する関数です。

```
(%i1) L: [1, 2, 3, 4, 5];
(%o1)                    [1, 2, 3, 4, 5]
(%i2) cons(0, L);
(%o2)                    [0, 1, 2, 3, 4, 5]
(%i3) cons(-1, %);
(%o3)                    [- 1, 0, 1, 2, 3, 4, 5]
```

## copylist(リスト)

リストを複製する関数です。通常の代入 ( $:$ ) は期待通りに機能しません。

```
(%i1) L: [1, 2, 3, 4, 5];
(%o1)                    [1, 2, 3, 4, 5]
(%i2) A: L;
(%o2)                    [1, 2, 3, 4, 5]
(%i3) B: copylist(L);
(%o3)                    [1, 2, 3, 4, 5]
(%i4) L[3]: x;
(%o4)                    x
(%i5) display([A, B]);
[A, B] = [[1, 2, x, 4, 5], [1, 2, 3, 4, 5]]
```

```
(%o5) done
```

## copymatrix(行列)

行列を複製する関数です。通常の代入 (:) は期待通りに機能しません。

```
(%i1) M: matrix([a, b], [c, d]);
(%o1)      [ a  b ]
          [ c  d ]

(%i2) A: M;
(%o2)      [ a  b ]
          [ c  d ]

(%i3) B: copymatrix(M);
(%o3)      [ a  b ]
          [ c  d ]
```

$A$  も  $B$  も見た目は同じですが、元の行列  $M$  に手を加えると、 $A$  の方はそれにつられて変更されてしまいます。

```
(%i4) M[1][1]: x;
(%o4)      x

(%i5) display([A, B]);
[A, B] = [[ [ x  b ] ], [ [ a  b ] ]
          [ [ c  d ] ], [ [ c  d ] ]]

(%o5) done
```

## cos(数・式)

余弦関数 (コサイン) です。引数の単位はラジアン [rad] です。

```
(%i1) cos(5*%pi/6);
```

```
(%o1)          sqrt(3)
          - ----
             2
```

### cosh(数・式)

双曲線余弦関数 (ハイパボリック・コサイン)  $\cosh x = \frac{e^x + e^{-1}}{2}$  です。

```
(%i1) cosh(1) + sinh(1), numer;
```

```
(%o1)          2.718281828459045
```

### cot(数・式)

余接関数 (コタンジェント)  $\cot x = \frac{1}{\tan x}$  です。引数の単位はラジアン [rad] です。

```
(%i1) cot(%pi/3);
```

```
(%o1)          1
          ----
          sqrt(3)
```

### coth(数・式)

双曲線余接関数 (ハイパボリック・コタンジェント)  $\operatorname{csch} x = \frac{1}{\tanh x} = \frac{\cosh x}{\sinh x}$  です。

```
(%i1) trigexpand(coth(2*x));
```

```
(%o1)          2
          coth (x) + 1
          ----
          2 coth(x)
```

## csc(数・式)

余割関数（コセカント） $\csc x = \frac{1}{\sin x}$  です。引数の単位はラジアン [rad] です。

```
(%i1) csc(%pi/4);
```

```
(%o1) sqrt(2)
```

## csch(数・式)

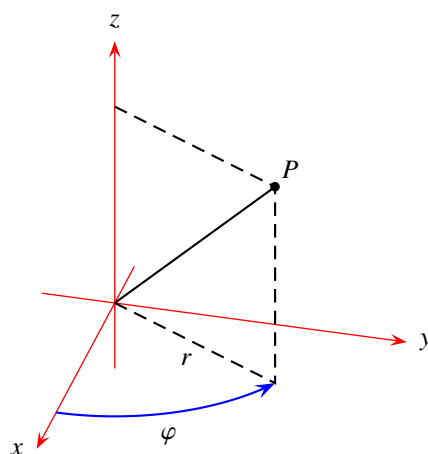
双曲線余割関数（ハイパボリック・コセカント） $\operatorname{csch} x = \frac{1}{\sinh x}$  です。

```
(%i1) diff(csch(x), x);
```

```
(%o1) - coth(x) csch(x)
```

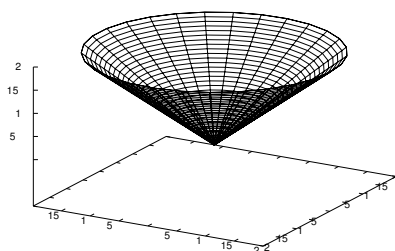
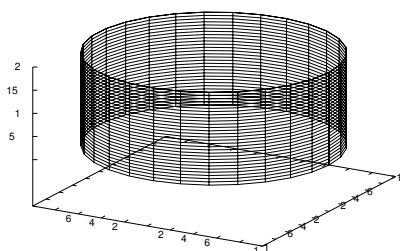
## cylindrical(r, z, Zmin, Zmax, φmin, φmax)

円柱座標系によるグラフを書く関数です。draw パッケージが必要です。



```
(%i2) load(draw)$
```

```
(%i3) draw3d(cylindrical(1, z, 0, 2, phi, 0, 2*%pi));  
(%o3) [gr3d(cylindrical)]  
(%i4) draw3d(cylindrical(z, z, 0, 2, phi, 0, 2*%pi));  
(%o4) [gr3d(cylindrical)]
```



## D

### define(関数, 式)

関数を定義する関数です。

```
(%i1) x^2 - y^2;
(%o1)          2      2
          x  - y
(%i2) define(f(x, y), %);
(%o2)          2      2
          f(x, y) := x  - y
(%i3) f(x, 2);
(%o3)          2
          x  - 4
(%i4) define(g[x], %);
(%o4)          2
          g  := x  - 4
          x
(%i5) makelist(g[i], i, 1, 5);
(%o5)          [- 3, 0, 5, 12, 21]
```

### demo(ファイル名)

デモンストレーションを実行する関数です\*1。

\*1 デモファイルは Maxima の命令を列挙したごく普通のバッチファイルです。

```
(%i1) demo(trgsmp)$
batching /usr/local/share/maxima/5.14.0cvs/demo/trgsmp.dem
At the _ prompt, type ';' followed by enter to get next demo

(%i2)          2      2      (1 - sin (x)) cos(x)
tan(x) sec (x) + -----
                    2
                    cos (x)

(%o2)          2      2      1 - sin (x)
sec (x) tan(x) + -----
                    2
                    cos(x)

-
```

プロンプト `_` に続けて [Enter] キーを押すと、次のデモが実行されます\*2。途中で終了するには `q[Enter]` 実行します。

## denom(式)

あるがままの状態での分母を返す関数です。内部で通分や因数分解等の計算を実行しないことに注意が必要です。

```
(%i1) (x + y)/(a - b);

(%o1)          y + x
          -----
          a - b

(%i2) denom(%);

(%o2)          a - b

(%i3) x/2 + y/2;

(%o3)          y   x
          --- + ---
           2   2

(%i4) denom(%);

(%o4)          1
```

\*2 画面には「;[Enter] をタイプせよ」と出力されていますが、Xmaxima 以外ではセミコロンは不要です。

cf. ratdenom

## diagmatrix(次数, 成分)

次数  $n$  と成分  $x$  を引数にとり、対角成分の全てが  $x$  であるような  $n$  次正方行列を返す関数です。

```
(%i1) diagmatrix(3, sin(x));
```

```
(%o1)      [ sin(x)  0  0 ]
           [  0  sin(x)  0 ]
           [  0  0  sin(x) ]
```

## diff(関数, 変数<sub>1</sub>, 階数<sub>1</sub>, 変数<sub>2</sub>, 階数<sub>2</sub>, ...)

関数  $f$ 、変数  $x$ 、階数  $n$  を引数にとり、 $n$  階偏導関数  $\frac{\partial^n f}{\partial x^n}$  を求める関数です。第 1 引数の関数が 1 変数関数の場合が通常の導関数です。階数が 1、すなわち、 $f'(x)$  を求める場合は、階数を省略することが出来ます。

```
(%i1) diff(x^3, x);
```

```
(%o1)      2
           3 x
```

```
(%i2) diff(x^3, x, 2);
```

```
(%o2)      6 x
```

```
(%i3) diff(x^3*y^3, x, 1, y, 1);
```

```
(%o3)      2 2
           9 x y
```

なお、変数 (第 2 引数) 以降を省略した場合は、全微分を返します。

```
(%i4) diff(x^3*y^3);
```

```
(%o4)      3 2      2 3
           3 x y del(y) + 3 x y del(x)
```

ここで、`del(x)`、`del(y)` はそれぞれ  $dx$ 、 $dy$  を表します。

## `disjoin(要素, 集合)`

指定した要素を取り除いた集合を返す関数です。

```
(%i1) S: {1, 2, 3, 4, 5};
(%o1) {1, 2, 3, 4, 5}
(%i2) disjoin(1 + 1, S);
(%o2) {1, 3, 4, 5}
(%i3) disjoin(6, S);
(%o3) {1, 2, 3, 4, 5}
```

## `display(式)`

方程式「引数=戻り値」の形に出力する関数です。

```
(%i1) f(x) := tan(x);
(%o1) f(x) := tan(x)
(%i2) display(trigexpand(tan(x + y)));
trigexpand(tan(y + x)) =  $\frac{\tan(y) + \tan(x)}{1 - \tan(x) \tan(y)}$ 
(%o2) done
```

## `display2d`

`true` (デフォルト) または `false` を設定し、出力方法を制御する変数です。 `true` の場合は 2 次元表示、`false` の場合は 1 次元表示になります。

```
(%i1) a/b + c/d;
c a
```

```
(%o1) 
$$\frac{-}{d} + \frac{-}{b}$$

(%i2) integrate(sin(x)/x, x);
(%o2) 
$$\int \frac{\sin(x)}{x} dx$$

(%i3) display2d: false;
(%o3) false
(%i4) a/b + c/d;
(%o4) c/d+a/b
(%i5) integrate(sin(x)/x, x);
(%o5) 'integrate(sin(x)/x,x)
```

## divide(割られる多項式, 割る多項式)

多項式同士の割り算を実行し、商  $Q$  と余り  $R$  を求め、リスト形式  $[Q, R]$  で出力する関数です。

```
(%i1) divide(x^5 - x^4, x^2 + 1);
(%o1) 
$$[x^3 - x^2 - x + 1, x - 1]$$

```

## divisors(整数)

整数  $n$  を引数にとり、 $n$  の正の約数全体の集合を返す関数です。

```
(%i1) divisors(100);
(%o1) {1, 2, 4, 5, 10, 20, 25, 50, 100}
```

## divsum(整数, 次数)

整数  $n$  と次数  $k$  を引数にとり、 $n$  の正の約数の  $k$  乗和  $\sum_{d|n} d^k$  を返す関数です。次数が 1 の場合は、省略することが出来ます。

```
(%i1) divsum(20);
(%o1) 42
(%i2) divsum(20, 3);
(%o2) 9198
(%i3) S: divisors(20);
(%o3) {1, 2, 4, 5, 10, 20}
(%i4) tree_reduce("+", args(S));
(%o4) 42
(%i5) tree_reduce("+", args(S)^3);
(%o5) 9198
```

## dontfactor: リスト

因数分解しない変数を設定する変数です。デフォルトは空 []、つまり、全ての変数が因数分解の対象となります。

```
(%i1) f: expand((x^2 - 1)*(y^2 - 1));
(%o1) x^2 y^2 - y^2 - x^2 + 1
(%i2) dontfactor;
(%o2) []
(%i3) factor(f);
(%o3) (x - 1) (x + 1) (y - 1) (y + 1)
(%i4) dontfactor: [x];
(%o4) [x]
```

```
(%i5) factor(f);
```

```
(%o5)          2  
(x  - 1) (y - 1) (y + 1)
```



## E

### `ematrix(m, n, x, i, j)`

$(i, j)$  成分が  $x$  で、それ以外が全て 0 であるような  $m \times n$  行列を生成する関数です。

```
(%i1) ematrix(2, 5, %pi, 2, 4);
(%o1)          [ 0  0  0  0  0 ]
              [ 0  0  0 %pi 0 ]
```

### `endcons(要素, リスト)`

リスト  $L$  と要素  $x$  を引数にとり、 $L$  の末尾に  $x$  を追加したリストを生成する関数です。

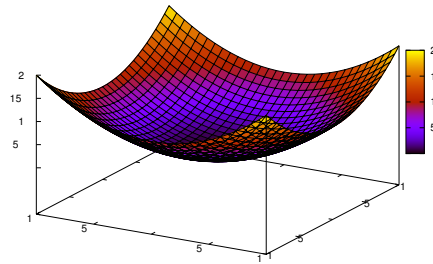
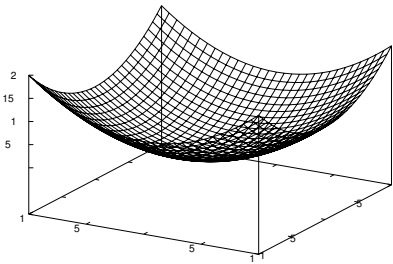
```
(%i1) L: [a, b, c, d, e];
(%o1)          [a, b, c, d, e]
(%i2) endcons(f, L);
(%o2)          [a, b, c, d, e, f]
```

### `enhanced3d = true`

3次元曲面の表面に色をつけます。gnuplot の pm3d です。draw パッケージが必要です。

```
(%i1) load(draw)$
(%i2) draw3d(explicit(x^2 + y^2, x, -1, 1, y, -1, 1));
(%o2)          [gr3d(explicit)]
```

```
(%i2) draw3d(enhanced3d = true,
  explicit(x^2 + y^2, x, -1, 1, y, -1, 1));
(%o2) [gr3d(explicit)]
```



## entier(数)

引数を超えない最大の整数を返す関数です。すなわち、引数  $x$  に対して、不等式  $n \leq x < n + 1$  を満たす整数  $n$  を返します。同じ働きをする関数 `fix` も用意されています。

```
(%i1) entier(sqrt(2));
(%o1) 1
(%i2) entier(-sqrt(2));
(%o2) - 2
```

## evenp(引数)

引数が偶数なら `true` を、それ以外なら `false` を返す関数です。

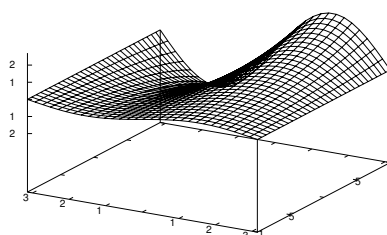
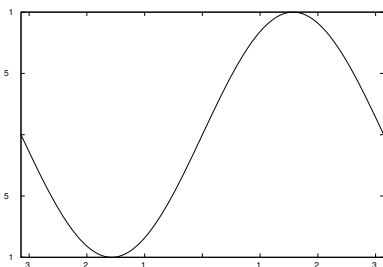
```
(%i1) evenp(28);
(%o1) true
(%i2) evenp(28.0);
```

```
(%o2) false
```

## explicit(関数, 変数1, a1, b1, 変数2, a2, b2)

陽関数（つまり普通の関数）のグラフを書く関数です。draw パッケージが必要です。

```
(%i1) load(draw)$
(%i2) draw2d(explicit(sin(x), x, -%pi, %pi));
(%o2) [gr2d(explicit)]
(%i3) draw3d(explicit(sin(x)*%e^y, x, -%pi, %pi, y, -1, 1));
(%o3) [gr3d(explicit)]
```



## expand(式)

引数を展開する関数です。なお、展開する指数の上限と下限に制限を加えることも出来ます。expand(式, p, n) のように実行すると  $-n$  乗から  $p$  乗までが展開され、それ以外の次数は展開されません。

```
(%i1) f(n) := (x + 1)^n;
(%o1) f(n) := (x + 1)^n
(%i2) expand(f(3));
3 2
```

```
(%o2)          x  + 3 x  + 3 x + 1
(%i3) expand(f(-4));
(%o3)          1
          -----
          4      3      2
         x  + 4 x  + 6 x  + 4 x + 1
(%i4) expand(f(-4) + f(-3) + f(2) + f(3), 2, 3);
(%o4)          1          3      1      2
          ----- + (x + 1)  + ----- + x  + 2 x + 1
          3      2          4
         x  + 3 x  + 3 x + 1          (x + 1)
```

## exponentialize(式) または exponentialize

関数として用いた場合、引数に含まれる双曲線関数を指数表現に変換します。変数として用いた場合、true を代入しておく、関数としての exponentialize を適用しなくても自動的に双曲線関数が指数の形に変換されます。

```
(%i1) exponentialize(tanh(x));
(%o1)          x      - x
          %e  - %e
          -----
          x      - x
          %e  + %e
(%i2) exponentialize;
(%o2)          false
(%i3) exponentialize: true;
(%o3)          true
(%i4) display([sinh(x), cosh(x)]);
(%o4)          done
          x      - x      x      - x
          %e  - %e      %e  + %e
[sinh(x), cosh(x)] = [-----, -----]
                      2          2
```

**ezgcd(多項式<sub>1</sub>, 多項式<sub>2</sub>, 多項式<sub>3</sub>, .....**

複数の整数や多項式  $P_1, P_2, P_3, \dots$  を引数にとり、それらの最大公約数  $g$  及び引数の  $g$  による商をリスト形式  $[g, P_1/g, P_2/g, P_3/g, \dots]$  で返す関数です。

```
(%i1) ezgcd(54, 90, 126);
```

```
(%o1) [18, 3, 5, 7]
```

```
(%i2) ezgcd(x^2 - x, -x + 1, x^2 - 1);
```

```
(%o2) [x - 1, x, - 1, x + 1]
```

ちなみに、ezgcd は Extended Zassenhaus Greatest Common Divisor の略語です。



## F

### factcomb(階乗の式)

階乗 (!) の係数が定数になるように変形する関数です。

```
(%i1) n * (n - 1)!;
(%o1)          n (n - 1)!
(%i2) factcomb(%);
(%o2)          n!
(%i3) n^2 * n!;
(%o3)          2
              n n!
(%i4) factcomb(%);
(%o4)          (n + 2)! - 3 (n + 1)! + n!
```

### factor(数・式)

因数分解を実行する関数です。

```
(%i1) factor(1234567890);
(%o1)          2 3 5 3607 3803
(%i2) factor(x^4 - y^4);
(%o2)          - (y - x) (y + x) (y^2 + x^2)
```

## factorflag: false

因数分解の際に係数を因数分解するか否かを決定する変数です。デフォルトは false (因数分解しない) です。

```
(%i1) f: 100*x^2 - 100;
(%o1)          2
          100 x  - 100
(%i2) factorflag;
(%o2)          false
(%i3) factor(f);
(%o3)          100 (x - 1) (x + 1)
(%i4) factorflag: true;
(%o4)          true
(%i5) factor(f);
(%o5)          2 2
          2 5 (x - 1) (x + 1)
```

## facts()

関数 `assume` によって割り当てられている性質を一覧表示する関数です。

```
(%i1) assume(a < b, b <= c);
(%o1)          [b > a, c >= b]
(%i2) assume(equal(x, y), notequal(y, z));
(%o2)          [equal(x, y), notequal(y, z)]
(%i3) facts();
(%o3)          [b > a, c >= b, equal(x, y), notequal(y, z)]
```

## fib(整数)

整数  $n$  を引数にとり、 $n$  番目の Fibonacci<sup>\*1</sup> (フィボナッチ) 数を返す関数です。

```
(%i1) fib(100);
(%o1) 354224848179261915075
(%i2) makelist(fib(n), n, -3, 7);
(%o2) [2, -1, 1, 0, 1, 1, 2, 3, 5, 8, 13]
```

なお、Fibonacci 数とは、漸化式

$$F_0 = 0, \quad F_1 = 1, \quad F_{n+2} = F_{n+1} + F_n$$

で定義された数列 (Fibonacci 数列)  $\{F_n\}$  に現れる数のことです。

## fibtophi(式)

Fibonacci 数  $\text{fib}(n)$  を黄金比  $\phi = \frac{1 + \sqrt{5}}{2}$  を用いた一般項  $\frac{\phi^n - (1 - \phi)^n}{2\phi - 1}$  の形に変換する関数です。

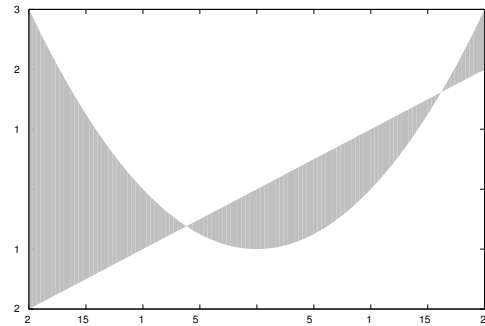
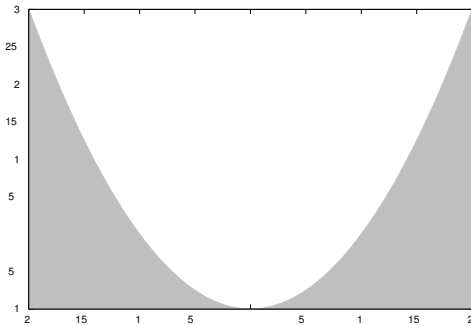
```
(%i1) fibtophi(fib(n));
(%o1) \frac{\phi^n - (1 - \phi)^n}{2\phi - 1}
```

## filled\_func = false

関数 `explicit` とともに用い、領域の塗りつぶしを制御する変数です。true、false、関数のいずれかを設定します。true の場合は、描画領域の下辺との間を塗りつぶし、関数が設定されている場合は、その関数と、`explicit` で指定された関数の間を塗りつぶします。draw パッケージが必要です。

<sup>\*1</sup> Leonardo Fibonacci (1170? — 1250?): イタリアの数学者

```
(%i1) load(draw)$
(%i2) draw2d(fill_color = grey, filled_func = true,
            explicit(x^2-1, x, -2, 2));
(%o2)                                     [gr2d(explicit)]
(%i3) draw2d(fill_color = grey, filled_func = x,
            explicit(x^2-1, x, -2, 2));
(%o3)                                     [gr2d(explicit)]
```



## find\_root(式, a, b)

指定した閉区間  $[a, b]$  (または  $[b, a]$ ) 内で、方程式の近似解 (もしくは関数の零点の近似値) の 1 つを求める関数です。なお、区間の端点における符号が一致する場合はエラーになります。

```
(%i1) f(x) := (x + 1) * x * (x - 1);
(%o1)                                     f(x) := (x + 1) x (x - 1)
(%i2) find_root(f(x), -2, 2);
(%o2)                                     0.0
(%i3) find_root(f(x), -2, 10);
(%o3)                                     1.0
(%i4) find_root(f(x), -1/2, 10);
function has same sign at endpoints
```

```
[f(- 0.5) = 0.375, f(10.0) = 990.0]
-- an error. To debug this try debugmode(true);
```

## fix(数)

引数を超えない最大の整数を返す関数です。すなわち、引数  $x$  に対して、不等式  $n \leq x < n + 1$  を満たす整数  $n$  を返します。同じ働きをする関数 `entier` も用意されています。

```
(%i1) fix(%e);
(%o1) 2
(%i2) fix(-%e);
(%o2) - 3
```

## flatten(集合・リスト)

集合やリストの入れ子状態を解消する関数です。

```
(%o1) {1, {1, 2, 4}, {1, 5}, {3, {1, 2}}}
```

```
(%i2) flatten(%);
(%o2) {1, 2, 3, 4, 5}
```

```
(%i3) [1, [1], [2], [1, 2]];
(%o3) [1, [1], [2], [1, 2]]
```

```
(%i4) flatten(%);
(%o4) [1, 1, 2, 1, 2]
```

## floor(数)

引数以上の最大の整数を返す関数です。すなわち、引数が  $x$  のとき、 $n \leq x < n + 1$  を満たす整数  $n$  を返します。

```
(%i1) floor(%pi);
(%o1) 3
(%i2) floor(-sqrt(2));
(%o2) - 2
```

cf. ceiling

## for

for ループを実行するための演算子です。

```
(%i1) s: 0;
(%o1) 0
(%i2) for i: 1 thru 10 do s: s + i;
(%o2) done
(%i3) s;
(%o3) 55
```

## forget(性質<sub>1</sub>, 性質<sub>2</sub>, ...)

関数 `assume` によって割り当てられている性質を解除する関数です。

```
(%i1) assume(x + y <= 0, x - y > 0);
(%o1) [y + x <= 0, x > y]
(%i2) forget(x > y);
(%o2) [x > y]
(%i3) facts();
(%o3) [0 >= y + x]
```

全ての性質をまとめて解除したい場合は、`forget(facts())` や `kill(all)` を実行し

ます。

## fpprec: 16

多倍長小数の桁数を設定する変数です。

```
(%i1) bfloat(sqrt(3));
(%o1) 1.732050807568877b0
(%i2) fpprec: 100;
(%o2) 100
(%i3) bfloat(sqrt(3));
(%o3) 1.732050807568877293527446341505872366942805253810380628055806\
979451933016908800037081146186757248576b0
```

## freeof(x<sub>1</sub>, x<sub>2</sub>, ..., 式)

複数の式や文字  $x_1, x_2, \dots$  と式  $A$  を引数にとり、 $x_1, x_2, \dots$  の全てが  $A$  の内部表現に含まれていれば `false` を、含まれていなければ `true` を返す関数です。

```
(%i1) hoge: sin(x) * cos(y - z);
(%o1) sin(x) cos(z - y)
(%i2) freeof(sin(x), hoge);
(%o2) false
(%i3) freeof(sin(y), hoge);
(%o3) true
(%i4) freeof(z - y, hoge);
(%o4) false
(%i5) freeof(y - z, hoge);
(%o5) true
(%i6) freeof("*", hoge);
```

```
(%o6)                                     false
(%i7) freeof("-", hoge);
(%o7)                                     true
(%i8) freeof("+", hoge);
(%o8)                                     false
```

# G

## gamma(数)

ガンマ関数  $\Gamma(z)$  です。自然数  $n$  に対しては、 $\Gamma(n) = (n-1)!$  が成り立つことが知られています。

```
(%i1) gamma(6);
(%o1) 120
(%i2) 5!;
(%o2) 120
(%i3) gamma(1/2);
(%o3) sqrt(%pi)
```

なお、ガンマ関数  $\Gamma(z)$  は無限乗積  $\lim_{n \rightarrow \infty} \frac{n^z n!}{\prod_{k=0}^n (z+k)}$  により定義され、特に、実部が正の複素数  $z$  に対しては、 $\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt$  が成り立ちます。

## gcd(数・式, 数・式)

2つの整数や多項式の最大公約数を計算する関数です。

```
(%i1) gcd(2000, 1988);
(%o1) 4
(%i2) gcd(x^2 - 4, x^2 + x - 6);
(%o2) x - 2
```

なお、関数 `gcd` と同じ文字列 `gcd` の変数も用意されており、最大公約数を求める際のアルゴリズムを選択することができます。

### `gcdex(数・式, 数・式)`

2つの整数や多項式  $p$ ,  $q$  を引数にとり、最大公約数  $g = \gcd(p, q)$  および一次不定方程式  $ap + bq = g$  の解  $a$ ,  $b$  を計算し、リスト形式  $[a, b, g]$  で出力する関数です。

```
(%i1) gcdex(330, 374);
(%o1) [8, - 7, 22]
(%i2) 8 * 330 + (-7) * 374;
(%o2) 22
```

### `genmatrix(配列, i2, j2, i1, j1)`

2次元の配列関数（ラムダ表現） $a[i, j]$  を用いて行列

$$\begin{pmatrix} a[i_1, j_1] & a[i_1, j_1 + 1] & \cdots & a[i_1, j_2] \\ a[i_1 + 1, j_1] & a[i_1 + 1, j_1 + 1] & \cdots & a[i_1 + 1, j_2] \\ \vdots & \vdots & \ddots & \vdots \\ a[i_2, j_1] & a[i_2, j_1 + 1] & \cdots & a[i_2, j_2] \end{pmatrix}$$

を生成する関数です。

```
(%i1) a: lambda([i, j], (x + i)^j);
(%o1) lambda([i, j], (x + i)^j)
(%i2) genmatrix(a, 3, 4, 1, 1);
(%o2) [ [ x + 1 (x + 1)^2 (x + 1)^3 (x + 1)^4 ]
        [ x + 2 (x + 2)^2 (x + 2)^3 (x + 2)^4 ]
        [ x + 3 (x + 3)^2 (x + 3)^3 (x + 3)^4 ] ]
```

## geometric\_mean(リスト)

リスト  $[a_1, a_2, \dots, a_n]$  を引数にとり、相乗平均  $\sqrt[n]{a_1 a_2 \cdots a_n}$  を返す関数です。descriptive パッケージが必要です。

```
(%i1) load(descriptive)$
(%i2) geometric_mean([a, b]);
(%o2)                                sqrt(a b)
(%i3) geometric_mean([a, b, c]);
(%o3)                                a1/3 b1/3 c1/3
(%i4) geometric_mean([a, b, c, d]);
(%o4)                                (a b c d)1/4
```

## gfactor(多項式)

1 変数多項式を引数にとり、Gauss の整数環  $\mathbb{Z}[i] = \{a + bi \mid a, b \in \mathbb{Z}\}$  における因数分解を求める関数です。

```
(%i1) factor(x^4 - 1);
(%o1)                                (x - 1) (x + 1) (x2 + 1)
(%i2) gfactor(x^4 - 1);
(%o2)                                (x - 1) (x + 1) (x - %i) (x + %i)
```

## globalsolve: false

変数 `globalsolve` に `true` を代入しておく、関数 `solve` (や `linsolve`) で連立一次方程式を解いた際、変数に解が代入されます。

```
(%i1) solve([3*x - 2*y = -5, 2*x + y = 6], [x, y]);
(%o1) [[x = 1, y = 4]]
(%i2) x + y;
(%o2) y + x
(%i3) globalsolve: true;
(%o3) true
(%i4) solve([3*x - 2*y = -5, 2*x + y = 6], [x, y]);
(%o4) [[x : 1, y : 4]]
(%i5) x + y;
(%o5) 5
```

## grind(式)

引数を「Maxima への入力に適した形」に出力する関数です。引数に自作の関数名を与えると、その関数の内容を出力してくれます。

```
(%i1) factor(x^4 - 16);
(%o1) (x - 2) (x + 2) (x^2 + 4)
(%i2) grind(%);
(x-2)*(x+2)*(x^2+4)$
(%o2) done
(%i3) rho(n) := block([g: 1, x: 0, y: 1],
  for i: 1 while g = 1 or g = n do (
    x: mod(x^2 + 1, n),
    y: mod((y^2 + 1)^2 + 1, n),
    g: gcd(y - x, n)
  ),
  g
)$
(%i4) rho(77);
(%o4) 7
(%i5) grind(rho);
```

```
rho(n):=block([g:1,x:0,y:1],
  for i while g = 1 or g = n do
    (x:mod(x^2+1,n),y:mod((y^2+1)^2+1,n),g:gcd(y-x,n)),g)$
(%o5) done
```

なお、上記の関数 rho は、 $\rho$  法を用いた素因数分解法の「不完全な」実装例です。



# H

## halfangles: false

三角関数において半角の公式を適用するか否かを設定する変数です。

```
(%i1) [sin(x/2), cos(x/2)];
(%o1) [sin( $\frac{x}{2}$ ), cos( $\frac{x}{2}$ )]

(%i2) halfangles: true;
(%o2) true

(%i3) [sin(x/2), cos(x/2)];
(%o3) [ $\frac{\sqrt{1 - \cos(x)}}{\sqrt{2}}$ ,  $\frac{\sqrt{\cos(x) + 1}}{\sqrt{2}}$ ]
```

## hankel(リスト<sub>1</sub>, リスト<sub>2</sub>)

Hankel 行列を生成する関数です。第 2 引数を省略すると、第 1 引数と同じ長さで成分が全て 0 のリストが指定されてものと見なされます。

```
(%i1) hankel([1, 2, 3, 4]);
(%o1) [ 1  2  3  4 ]
      [ 2  3  4  0 ]
      [ 3  4  0  0 ]
      [ 4  0  0  0 ]

(%i2) hankel([1, 2, 3, 4], [a, b, c, d, e, f, g]);
```

```
(%o2)      [ 1  2  3  4  b  c  d ]
           [ 2  3  4  b  c  d  e ]
           [ 3  4  b  c  d  e  f ]
           [ 4  b  c  d  e  f  g ]
```

### hermite(整数, 変数)

整数  $n$  と変数  $x$  を引数にとり、Hermite 多項式  $H_n(x)$  を返す関数です。orthopoly パッケージが必要です。

```
(%i1) load(orthopoly)$
(%i2) hermite(2, x);
(%o2)      - 2 (1 - 2 x )^2
(%i3) define(f(x), hermite(3, x));
(%o3)      f(x) := - 12 x (1 - 2 x )^2 / 3
(%i4) diff(f(x), x, 2) - 2*x*diff(f(x), x) + 2*3*f(x);
(%o4)      - 2 x (16 x^2 - 12 (1 - 2 x )^2 / 3) - 72 x (1 - 2 x )^2 / 3 + 48 x
(%i5) expand(%);
(%o5)      0
```

なお、Hermite 多項式とは、常微分方程式

$$\left( \frac{d^2}{dx^2} - 2x \frac{d}{dx} + 2n \right) H_n(x) = 0$$

を満たす多項式  $H_n(x)$  のことです。

## hessian(関数, 変数リスト)

関数  $f(x_1, x_2, \dots, x_n)$  と変数のリスト  $[x_1, x_2, \dots, x_n]$  を引数にとり、Hesse 行列 (Hessian matrix) を返す関数です。

```
(%i1) hessian(x^3*y^3, [x, y]);
(%o1)

$$\begin{bmatrix} 6x^2y^3 & 9x^2y^2 \\ 9x^2y^2 & 6xy^3 \end{bmatrix}$$

(%i2) depends(f, [x, y]);
(%o2) [f(x, y)]
(%i3) hessian(f, [x, y]);
(%o3)

$$\begin{bmatrix} \frac{d^2 f}{dx^2} & \frac{d^2 f}{dx dy} \\ \frac{d^2 f}{dx dy} & \frac{d^2 f}{dy^2} \end{bmatrix}$$

```

なお、Hesse 行列  $H(f)$  とは、 $(i, j)$  成分  $H(f)_{ij}$  が  $\frac{\partial^2 f}{\partial x_i \partial x_j}$  で与えられる行列のことです。

## hipow(式, 変数)

式  $f(x)$  の変数  $x$  についての (内部表現における) 次数を返す関数です。

```
(%i1) sin(x) + tan(x)*sin(x)^2 + 1;
(%o1)

$$\sin^2(x) \tan(x) + \sin(x) + 1$$

(%i2) hipow(%, sin(x));
(%o2) 2
(%i3) expand((x^2 + 1)^4);
(%o3)

$$x^8 + 4x^6 + 6x^4 + 4x^2 + 1$$

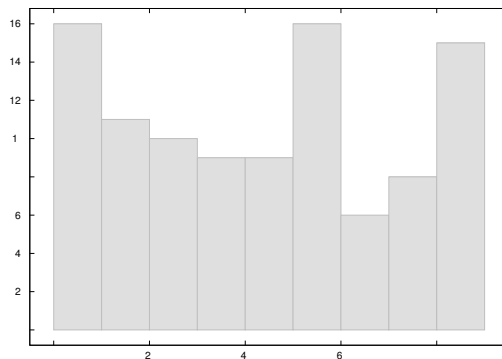
```

```
(%i4) hipow(% , x);
(%o4)                                     8
(%i5) hipow((x^2 + 1)^4, x);
(%o5)                                     2
```

## histogram(リスト, オプション<sub>1</sub>, ...)

1次元のリストまたは行列を引数にとり、ヒストグラムを描く関数です。descriptive パッケージが必要です。

```
(%i1) load(descriptive)$
(%i2) s: makelist(random(10), i, 1, 100);
(%o2) [2, 2, 4, 5, 4, 1, 9, 5, 8, 3, 5, 5, 0, 6, 9, 0, 9, 4, 7, 6, 9,
9, 3, 9, 6, 6, 6, 3, 0, 1, 2, 9, 9, 8, 6, 5, 3, 3, 7, 8, 4, 6, 5, 0,
7, 6, 5, 4, 7, 2, 8, 3, 0, 6, 2, 6, 6, 7, 4, 6, 9, 1, 6, 8, 2, 0, 2,
2, 1, 3, 5, 2, 9, 9, 1, 6, 9, 0, 8, 8, 8, 3, 9, 0, 6, 3, 2, 3, 0, 9,
9, 4, 4, 5, 7, 6, 2, 1, 0, 4]
(%i3) histogram(s, nclasses=9, fill_color=gray, fill_density=0.5);
(%o3) [gr2d(bars)]
```



上の実行例は、関数 `random` を用いて、0 から 9 までの整数を 100 個発生させ、その出現回数をグラフ化したものです。



## if 条件 then 式<sub>1</sub> else 式<sub>2</sub>

if 文です。

```
(%i1) define(f(x), if x <= 0 then "A" else "B");
(%o1)          f(x) := if x <= 0 then "A" else "B"
(%i2) f(0);
(%o2)          A
(%i3) f(1);
(%o3)          B
```

## ifactors(自然数)

自然数  $n$  を引数にとり、 $n$  の素因数分解  $n = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$  をリスト形式  $[[p_1, e_1], [p_2, e_2], \dots, [p_k, e_k]]$  で出力する関数です。

```
(%i1) ifactors(60);
(%o1)          [[2, 2], [3, 1], [5, 1]]
```

## ilt(関数, s, t)

関数  $F(s)$  に対して、逆 Laplace 変換  $f(t) = \mathcal{L}^{-1}[F(s)]$  を計算する関数です。関数  $F(s)$  は、分母が 1 次または 2 次の因子のみを持つ有理式でなければなりません。

```
(%i1) ilt(s^2/(s^3 + 1), s, t);
(%o1)          
$$\frac{2 e^{t/2} \cos\left(\frac{\sqrt{3} t}{2}\right)}{3} + \frac{e^{-t}}{3}$$

(%i2) laplace(%, t, s);
(%o2)          
$$2 (2 s - 1) \quad 1$$

```

```
(%o2)          ----- + -----
              2              2
              3 (2 s  - 2 s + 2)  3 (s + 1)

(%i3) ratsimp(%);

(%o3)          2
              s
              ---
              3
              s + 1
```

なお、逆 Laplace 変換  $\mathcal{L}^{-1}[F(s)]$  とは、関数  $F(s)$  から関数  $f(t) = \lim_{p \rightarrow \infty} \frac{1}{2\pi i} \int_{c-ip}^{c+ip} F(s) e^{st} ds$  を求める計算のことです。

## imagpart(複素数)

複素数  $z$  の虚部  $\Im(z)$  を返す関数です。

```
(%i1) z: (1 + %i)*(2 - 3*i);
(%o1)          (2 - 3 %i) (%i + 1)
(%i2) imagpart(z);
(%o2)          - 1
(%i3) expand(z);
(%o3)          5 - %i
```

cf. realpart

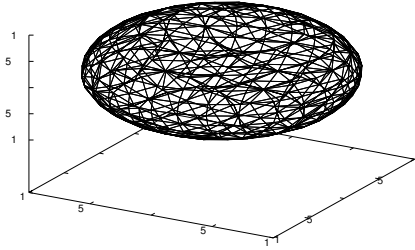
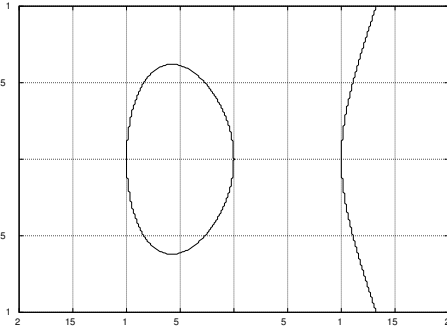
## implicit(方程式, x, xmin, xmax, y, ymin, ymax)

陰関数のグラフを描く関数です。draw パッケージが必要です。

```
(%i1) load(draw)$
(%i2) draw2d(implicit(y^2 = x^3 - x, x,-2,2, y,-1,1), grid = true);
(%o2)          [gr2d(implicit)]
(%i3) draw3d(implicit(x^2 + y^2 + z^2 = 1, x,-1,1, y,-1,1, z,-1,1));
```

(%o3)

[gr3d(implicit)]



なお、陰関数とは、「 $y = x$  の式」や「 $z = x$  と  $y$  の式」の形に表されていない関数のことです。

### **inpart(式, 番号<sub>1</sub>, 番号<sub>2</sub>, ...)**

式の一部を抽出する関数です。関数 `part` と似ていますが、`part` が表示された状態における番号を指定するのに対して、`inpart` で指定するのは内部表現における番号です。一般に、`part` より高速です。

(%i1) `t: x*a + y*b + z*c;`(%o1) `c z + b y + a x`(%i2) `inpart(t, 1);`(%o2) `a x`(%i3) `inpart(t, 1, 2);`(%o3) `x`(%i4) `part(t, 1);`(%o4) `c z`(%i5) `part(t, 1, 2);`(%o5) `z`

## inprod(ベクトル<sub>1</sub>, ベクトル<sub>2</sub>)

同じ長さの1次元ベクトル、またはリストを2つ引数にとり、それらの内積を計算する関数です。eigenパッケージが必要です。関数名 inprod の代わりに innerproduct を用いることも出来ます (全く同じ関数です)。

```
(%i1) load(eigen)$
(%i2) inprod([a, b, c], [d, e, f]);
(%o2)          c f + b e + a d
```

なお、内積  $\vec{x} \cdot \vec{y}$  とは、ベクトル  $\vec{x}$  と  $\vec{y}$  の成す角を  $\theta$  とするとき、 $|\vec{x}||\vec{y}|\cos\theta$  で与えられる値のことです。ベクトルが成分表示  $\vec{x} = (x_1, x_2, \dots, x_n)$ 、 $\vec{y} = (y_1, y_2, \dots, y_n)$  されている場合は、 $\vec{x} \cdot \vec{y} = \sum_{i=1}^n x_i y_i$  が成り立ちます。

## integer\_partitions(自然数, 長さ)

自然数  $n$  を引数にとり、総和が  $n$  になるようなリスト全体の集合を返す関数です。

```
(%i1) integer_partitions(4);
(%o1)          {[1, 1, 1, 1], [2, 1, 1], [2, 2], [3, 1], [4]}
(%i2) integer_partitions(4, 2);
(%o2)          {[2, 2], [3, 1], [4, 0]}
```

## integrate(関数, 変数, 下端, 上端)

関数の積分を計算する関数です。積分区間を省略すると不定積分を計算します。

```
(%i1) integrate(x^2, x, 0, 1);
(%o1)          1
              3
(%i2) integrate(sin(x)*e^x, x);
```

```
(%o2) 
$$\frac{e^x (\sin(x) - \cos(x))}{2}$$

```

2 番目の例のように、不定積分における積分定数  $C$  は省略されます。

## intersect(集合<sub>1</sub>, 集合<sub>2</sub>, ...)

複数の集合  $S_1, S_2, \dots$  を引数にとり、それらの共通部分（積集合） $S_1 \cap S_2 \cap \dots$  を返す関数です。関数名 `intersect` の代わりに `intersection` を用いることも出来ます（全く同じ関数です）。

```
(%i1) S: {1, 3, 5, 6, 7, 9};
(%o1) {1, 3, 5, 6, 7, 9}
(%i2) T: {2, 3, 4, 6, 7, 8};
(%o2) {2, 3, 4, 6, 7, 8}
(%i3) intersect(S, T);
(%o3) {3, 6, 7}
```

## intosum(式)

倍数因子を和記号の中に入れる関数です。

```
(%i1) sum(2 * i^3, i, 1, n);
(%o1) 
$$2 \sum_{i=1}^n i^3$$

(%i2) intosum(%);
(%o2) 
$$\sum_{i=1}^n 2 i^3$$

```

```

/
====
i = 1

```

## inv\_mod(整数, 法)

2つの整数  $n$ 、 $m$  を引数にとり、整数  $n$  の法  $m$  における逆元、すなわち、合同式  $nx \equiv 1 \pmod{m}$  を満たす整数  $x$  を求める関数です。逆元が存在しない場合は、`false` を返します。

```

(%i1) inv_mod(5, 16);
(%o1)                                     13
(%i2) remainder(5*%, 16);
(%o2)                                     1
(%i3) inv_mod(2, 16);
(%o3)                                     false

```

## is(等式・不等式)

等式・不等式に対して、成立すれば `true`、不成立なら `false`、不明なら `unknown` を返す関数です。

```

(%i1) is(3.1 > %pi);
(%o1)                                     false
(%i2) is(3.2 > %pi);
(%o2)                                     true
(%i3) is(a <= b);
(%o3)                                     unknown
(%i4) assume(a < b, b <= 0);
(%o4)                                     [b > a, b <= 0]
(%i5) is(a <= b);

```

---

(%o5)	true
(%i6) is(a < 0);	
(%o6)	true
(%i7) is(b < 0);	
(%o7)	unknown

# J

## jacobi(整数, 奇数)

Jacobi 記号  $\left(\frac{a}{n}\right)$  の値を返す関数です。

```
(%i1) jacobi(11, 21);
```

```
(%o1) - 1
```

なお、Jacobi 記号とは、平方剰余記号  $\left(\frac{a}{p}\right)$  の拡張として定義される関数です。平方剰余記号  $\left(\frac{a}{p}\right)$  とは、奇素数  $p$  と整数  $a$  に対して、合同式  $x^2 \equiv a \pmod{p}$  が解を持てば 1、解を持たなければ  $-1$ 、 $a$  が  $p$  の倍数なら 0 をとる関数です。これを、法（分母）が 3 以上の整数  $n$  に対して拡張します。 $n$  の素因数分解を  $n = p_1^{e_1} \times p_2^{e_2} \times \cdots \times p_k^{e_k}$  とするとき、

$$\left(\frac{a}{p_1}\right)^{e_1} \times \left(\frac{a}{p_2}\right)^{e_2} \times \cdots \times \left(\frac{a}{p_k}\right)^{e_k}$$

を Jacobi 記号と呼び、平方剰余記号の表記を流用し、 $\left(\frac{a}{n}\right)$  と表します。上の実行例  $\left(\frac{11}{21}\right) = -1$  を検算してみると、

```
(%i2) [mod(11, n), makelist(mod(i^2, n), i, 1, n)], n: 3;
```

```
(%o2) [2, [1, 1, 0]]
```

```
(%i3) [mod(11, n), makelist(mod(i^2, n), i, 1, n)], n: 7;
```

```
(%o3) [4, [1, 4, 2, 2, 4, 1, 0]]
```

より、 $\left(\frac{11}{21}\right) = \left(\frac{11}{3}\right) \times \left(\frac{11}{7}\right) = \left(\frac{2}{3}\right) \times \left(\frac{4}{7}\right) = (-1) \times 1 = -1$  が得られます。

**jacobian**([関数<sub>1</sub>, 関数<sub>2</sub>, ...], [変数<sub>1</sub>, 変数<sub>2</sub>, ...])

Jacobi 行列 (関数行列) を返す関数です。

```
(%i1) jacobian([x^2*y^2, log(x*y)], [x, y]);
```

```
(%o1)      [ 2 x y  2 x y ]
           [ 2 x y  2 x y ]
           [ 1      1 ]
           [ -      - ]
           [ x      y ]
```

なお、Jacobi 行列とは、座標変換 ( $n$  個の  $n$  変数関数の組)

$$f : (x_1, x_2, \dots, x_n) \mapsto (y_1, y_2, \dots, y_n)$$

に対して定義される行列

$$J(f) = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \dots & \frac{\partial y_n}{\partial x_n} \end{pmatrix}$$

のことです。通常 Jacobi 行列の行列式をヤコビアン (Jacobian) と呼びます。

**jordan**(行列)

行列の Jordan 標準形を返す関数です。diag パッケージが必要です。

```
(%i1) load(diag)$
```

```
(%i2) M: matrix([1,-1,1],[0,2,0],[1,0,1]);
```

```
(%o2)      [ 1 - 1  1 ]
           [ 0  2  0 ]
           [ 1  0  1 ]
```

```
(%i3) jordan(M);  
(%o3)          [[2, 2], [0, 1]]  
(%i4) dispJordan(%);  
(%o4)          [ 2  1  0 ]  
                [ 0  2  0 ]  
                [ 0  0  0 ]
```