

# Index

## **abs**( $x$ )

Returns the absolute value or vector length of  $x$ .

```
abs(3 + 4i)
```

5

## **adj**( $m$ )

Returns the adjunct of matrix  $m$ . Adjunct is equal to determinant times inverse.

```
A = ((a,b),(c,d))  
adj(A) == det(A) inv(A)
```

1

## **and**( $a, b, \dots$ )

Returns 1 if all arguments are true (nonzero). Returns 0 otherwise.

```
and(1=1,2=2)
```

1

## **arccos**( $x$ )

Returns the arc cosine of  $x$ .

```
arccos(1/2)
```

$\frac{1}{3}\pi$

## **arccosh**( $x$ )

Returns the arc hyperbolic cosine of  $x$ .

## **arcsin**( $x$ )

Returns the arc sine of  $x$ .

```
arcsin(1/2)
```

$\frac{1}{6}\pi$

## **arcsinh(*x*)**

Returns the arc hyperbolic sine of *x*.

## **arctan(*y*, *x*)**

Returns the arc tangent of *y* over *x*. If *x* is omitted then *x* = 1 is used.

```
arctan(1,0)
```

$$\frac{1}{2}\pi$$

## **arctanh(*x*)**

Returns the arc hyperbolic tangent of *x*.

## **arg(*z*)**

Returns the angle of complex *z*.

```
arg(2 - 3i)
```

− arctan(3, 2)

## **binding(*s*)**

The result of evaluating a symbol can differ from the symbol's binding. For example, the result may be expanded. The **binding** function returns the actual binding of a symbol.

```
p = quote((x + 1)^2)
```

```
p
```

$$p = x^2 + 2x + 1$$

```
binding(p)
```

$$(x + 1)^2$$

## **break**

Break out of a **loop** or **for** function.

```
k = 0
```

```
loop(k = k + 1, test(k == 4, break), print(k))
```

$$k = 1$$
$$k = 2$$
$$k = 3$$

## **ceiling( $x$ )**

Returns the smallest integer greater than or equal to  $x$ .

```
ceiling(1/2)
```

1

## **check( $x$ )**

If  $x$  is true (nonzero) then continue, else stop. Expression  $x$  can include the relational operators =, ==, <, <=, >, >=. Use the **not** function to test for inequality.

```
A = exp(i pi)
B = -1
check(A == B) -- stop here if A not equal to B
```

## **choose( $n, k$ )**

Returns the binomial coefficient  $n$  choose  $k$ .

```
choose(52,5) -- number of poker hands
```

2598960

## **clear**

Clears all symbol definitions.

## **clock( $z$ )**

Returns complex  $z$  in polar form with base of negative 1 instead of  $e$ .

```
clock(2 - 3i)
```

$13^{1/2} (-1)^{-\arctan(3,2)/\pi}$

## **cofactor( $m, i, j$ )**

Returns the cofactor of matrix  $m$  for row  $i$  and column  $j$ .

```
A = ((a,b),(c,d))
cofactor(A,1,2) == adj(A)[2,1]
```

1

**conj**( $z$ )

Returns the complex conjugate of  $z$ .

`conj(2 - 3i)`

$2 + 3i$

**contract**( $a, i, j, \dots$ )

Returns the contraction of tensor  $a$  with respect to indices  $i, j$ , etc. If  $i$  and  $j$  are omitted then 1 and 2 are used. The argument list can be extended for multiple contract operations. The arguments are evaluated from left to right. For example, `contract(A,1,2,2,3)` is equivalent to `contract(contract(A,1,2),2,3)`.

`A = ((a,b),(c,d))`

`contract(A) -- trace of matrix A`

$a + d$

**cos**( $x$ )

Returns the cosine of  $x$ .

`cos(pi/4)`

$\frac{1}{2^{1/2}}$

**cosh**( $x$ )

Returns the hyperbolic cosine of  $x$ .

`expform(cosh(x))`

$\frac{1}{2} \exp(-x) + \frac{1}{2} \exp(x)$

**cross**( $u, v$ )

Returns the cross product of vectors  $u$  and  $v$ .

**curl**( $v$ )

Returns the curl of vector  $v$  with respect to symbols  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{z}$ .

**d**( $f, x, \dots$ )

Returns the partial derivative of  $f$  with respect to  $x$  and any additional arguments.

**d**(sin( $x$ ),  $x$ )

$\cos(x)$

Multiderivatives are computed by extending the argument list.

**d**(sin( $x$ ),  $x, x$ )

$-\sin(x)$

A numeric argument  $n$  computes the  $n$ th derivative with respect to the previous symbol.

**d**(sin( $x\ y$ ),  $x, 2, y, 2$ )

$x^2y^2 \sin(xy) - 4xy \cos(xy) - 2 \sin(xy)$

Argument  $f$  can be a tensor of any rank. Argument  $x$  can be a vector. When  $x$  is a vector the result is the gradient of  $f$ .

**F** = (**f**(), **g**(), **h**())

**X** = ( $x, y, z$ )

**d**(**F**, **X**)

$$\begin{bmatrix} d(f(), x) & d(f(), y) & d(f(), z) \\ d(g(), x) & d(g(), y) & d(g(), z) \\ d(h(), x) & d(h(), y) & d(h(), z) \end{bmatrix}$$

Symbol **d** can be used as a variable name. Doing so does not conflict with function **d**.

Symbol **d** can be redefined as a different function. The function **derivative**, a synonym for **d**, can be used to obtain a partial derivative.

**defint**( $f, x, a, b, \dots$ )

Returns the definite integral of  $f$  with respect to  $x$  evaluated from  $a$  to  $b$ . The argument list can be extended for multiple integrals. The following example integrates over theta then over phi.

**defint**(sin(theta), theta, 0, pi, phi, 0, 2 pi)

$4\pi$

## **denominator**( $x$ )

Returns the denominator of expression  $x$ .

```
denominator(a/b)
```

$b$

## **det**( $m$ )

Returns the determinant of matrix  $m$ .

```
A = ((a,b),(c,d))  
det(A)
```

$ad - bc$

## **dim**( $a, n$ )

Returns the dimension of the  $n$ th index of tensor  $a$ . Index numbering starts with 1.

```
A = ((1,2),(3,4),(5,6))  
dim(A,1)
```

3

## **div**( $v$ )

Returns the divergence of vector  $v$  with respect to symbols  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{z}$ .

## **do**( $a, b, \dots$ )

Evaluates each argument from left to right. Returns the result of the final argument.

```
do(A=1,B=2,A+B)
```

3

## **dot**( $a, b, \dots$ )

Returns the dot product of vectors, matrices, and tensors. Also known as the matrix product. Arguments are evaluated from right to left. The following example solves for  $X$  in  $AX = B$ .

```
A = ((1,2),(3,4))  
B = (5,6)  
X = dot(inv(A),B)  
X
```

$$\begin{bmatrix} -4 \\ \frac{9}{2} \end{bmatrix}$$

## **eigenvec(*m*)**

Returns eigenvectors for matrix *m*. Matrix *m* is required to be numerical, real, and symmetric. The return value is a matrix with each column an eigenvector. Eigenvalues are obtained as shown.

```
A = ((3,5),(5,3))
Q = eigenvec(A)
D = dot(transpose(Q),A,Q) -- eigenvalues on diagonal of D
D
```

$$D = \begin{bmatrix} 8 & 0 \\ 0 & -2 \end{bmatrix}$$

## **erf(*x*)**

Error function of *x*. Returns a numerical value if *x* is a real number.

```
erf(1.0)
```

0.842701

```
d(erf(x),x)
```

$$\frac{2 \exp(-x^2)}{\pi^{1/2}}$$

## **erfc(*x*)**

Complementary error function of *x*. Returns a numerical value if *x* is a real number.

```
erfc(1.0)
```

0.157299

```
d(erfc(x),x)
```

$$-\frac{2 \exp(-x^2)}{\pi^{1/2}}$$

**eval**( $f, x, a, y, b, \dots$ )

Returns  $f$  evaluated with  $x$  replaced by  $a$ ,  $y$  replaced by  $b$ , etc. All arguments can be expressions.

```
f = sqrt(x^2 + y^2)
eval(f,x,3,y,4)
```

5

In the following example, `eval` is used to replace `x` with `cos(theta)`.

```
-- associated legendre of cos theta
P(1,m,x) = test(m < 0, (-1)^m (1 + m)! / (1 - m)! P(1,-m),
              1 / (2^1 1!) sin(theta)^m *
              eval(d((x^2 - 1)^1, x, 1 + m), x, cos(theta)))
```

`P(2,-1)`

$-\frac{1}{2} \cos(\theta) \sin(\theta)$

**exp**( $x$ )

Returns the exponential of  $x$ .

```
exp(i pi)
```

-1

**expcos**( $z$ )

Returns the cosine of  $z$  in exponential form.

```
expcos(z)
```

$\frac{1}{2} \exp(iz) + \frac{1}{2} \exp(-iz)$

**expcosh**( $z$ )

Returns the hyperbolic cosine of  $z$  in exponential form.

```
expcosh(z)
```

$\frac{1}{2} \exp(-z) + \frac{1}{2} \exp(z)$



## **expform(*x*)**

Returns expression  $x$  with trigonometric and hyperbolic functions converted to exponentials.

`expform(cos(x) + i sin(x))`

$\exp(ix)$

## **expsin(*z*)**

Returns the sine of  $z$  in exponential form.

`expsin(z)`

$-\frac{1}{2}i \exp(iz) + \frac{1}{2}i \exp(-iz)$

## **expsinh(*z*)**

Returns the hyperbolic sine of  $z$  in exponential form.

`expsinh(z)`

$-\frac{1}{2} \exp(-z) + \frac{1}{2} \exp(z)$

## **exptan(*z*)**

Returns the tangent of  $z$  in exponential form.

`exptan(z)`

$\frac{i}{\exp(2iz) + 1} - \frac{i \exp(2iz)}{\exp(2iz) + 1}$

## **exptanh(*z*)**

Returns the hyperbolic tangent of  $z$  in exponential form.

`exptanh(z)`

$-\frac{1}{\exp(2z) + 1} + \frac{\exp(2z)}{\exp(2z) + 1}$

## **factorial(*n*)**

Returns the factorial of  $n$ . The expression `n!` can also be used.

`20!`

2432902008176640000

## **float**( $x$ )

Returns expression  $x$  with rational numbers and integers converted to floating point values. The symbol `pi` and the natural number are also converted.

```
float(212^17)
```

$$3.52947 \times 10^{39}$$

## **floor**( $x$ )

Returns the largest integer less than or equal to  $x$ .

```
floor(1/2)
```

0

## **for**( $a, b, c, d, e, f, \dots$ )

For  $a$  equals  $b$  through  $c$  inclusive, evaluate the remaining arguments in a loop. Arguments  $b$  and  $c$  are integers. Symbol  $a$  is advanced by plus or minus 1 in the direction of  $c$  each time through the loop. Use `break` to break out of the loop early. The original value of  $a$  is restored after `for` completes. Note that if symbol `i` is used for  $a$  then the imaginary unit is overridden in the scope of `for`.

```
for(k,1,3,print(k))
```

$$k = 1$$
$$k = 2$$
$$k = 3$$

## **grad**( $f$ )

Returns the gradient  $d(f, (x, y, z))$ .

```
grad(f())
```

$$\begin{bmatrix} d(f(), x) \\ d(f(), y) \\ d(f(), z) \end{bmatrix}$$

## **hadamard**( $a, b, \dots$ )

Returns the Hadamard (element-wise) product.

```
X = (a,b,c)
```

```
hadamard(X,X)
```

$$\begin{bmatrix} a^2 \\ b^2 \\ c^2 \end{bmatrix}$$

**i**

Symbol **i** is initialized to the imaginary unit  $\sqrt{-1}$ .

```
exp(i pi)
```

$-1$

Note: It is ok to clear or redefine **i** and use the symbol for something else.

**imag**( $z$ )

Returns the imaginary part of complex  $z$ .

```
imag(2 - 3i)
```

$-3$

**infixform**( $x$ )

Converts expression  $x$  to a string and returns the result.

```
p = (x + 1)^2  
infixform(p)
```

$x^2 + 2x + 1$

**inner**( $a, b, \dots$ )

Returns the inner product of vectors, matrices, and tensors. Also known as the matrix product.

```
A = ((a,b),(c,d))  
B = (x,y)  
inner(A,B)
```

$$\begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

Note: **inner** and **dot** are the same function.

**integral**( $f, x$ )

Returns the integral of  $f$  with respect to  $x$ .

```
integral(x^2,x)
```

$\frac{1}{3}x^3$

**inv**(*m*)

Returns the inverse of matrix *m*.

```
A = ((1,2),(3,4))  
inv(A)
```

$$\begin{bmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{bmatrix}$$

**j**

Set `j=sqrt(-1)` to use `j` for the imaginary unit instead of `i`.

```
j = sqrt(-1)  
1/sqrt(-1)
```

$$-j$$

**kronecker**(*a*, *b*, ...)

Returns the Kronecker product of *a*, *b*, etc.

```
I = ((1,0),(0,1))  
A = ((a,b),(c,d))  
kronecker(I,A)
```

$$\begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$$

See also

Anticommutator example

**last**

The result of the previous calculation is stored in `last`.

```
212^17
```

```
3529471145760275132301897342055866171392
```

```
last^(1/17)
```

212

Symbol `last` is an implied argument when a function has no argument list.

`212^17`

3529471145760275132301897342055866171392

`float`

$3.52947 \times 10^{39}$

**lgamma**( $x$ )

Returns the log of the absolute value of the Gamma function of  $x$ .

`lgamma(0.5)`

0.572365

**log**( $x$ )

Returns the natural logarithm of  $x$ .

`log(x^y)`

$y \log(x)$

**loop**( $a, b, c, \dots$ )

Evaluate arguments in a loop. Use `break` to break out of the loop.

`k = 0`

`loop(k = k + 1, test(k == 4, break), print(k))`

$k = 1$

$k = 2$

$k = 3$

**mag**( $z$ )

Returns the magnitude of complex  $z$ . Function `mag` treats undefined symbols as real while `abs` does not.

`mag(x + i y)`

$(x^2 + y^2)^{1/2}$

## **minor**( $m, i, j$ )

Returns the minor of matrix  $m$  for row  $i$  and column  $j$ .

```
A = ((1,2,3),(4,5,6),(7,8,9))  
minor(A,1,1) == det(minormatrix(A,1,1))
```

1

## **minormatrix**( $m, i, j$ )

Returns a copy of matrix  $m$  with row  $i$  and column  $j$  removed.

```
A = ((1,2,3),(4,5,6),(7,8,9))  
minormatrix(A,1,1)
```

$$\begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$$

## **noexpand**( $x$ )

Evaluates expression  $x$  without expanding products of sums.

```
noexpand((x + 1)^2 / (x + 1))
```

$x + 1$

## **not**( $x$ )

Returns 0 if  $x$  is true (nonzero). Returns 1 otherwise.

```
not(1=1)
```

0

## **nroots**( $p, x$ )

Returns the approximate roots of polynomials with real or complex coefficients. Multiple roots are returned as a vector.

```
p = x^5 - 1  
nroots(p,x)
```

$$\begin{bmatrix} 1 \\ -0.809017 + 0.587785i \\ -0.809017 - 0.587785i \\ 0.309017 + 0.951057i \\ 0.309017 - 0.951057i \end{bmatrix}$$

**number( $x$ )**

Returns 1 if  $x$  is a real number. Returns 0 otherwise.

`number(1/2)`

1

`number(x)`

0

**numerator( $x$ )**

Returns the numerator of expression  $x$ .

`numerator(a/b)`

$a$

**or( $a, b, \dots$ )**

Returns 1 if at least one argument is true (nonzero). Returns 0 otherwise.

`or(1=1,2=2)`

1

**outer( $a, b, \dots$ )**

Returns the outer product of vectors, matrices, and tensors.

`A = (a,b,c)`

`B = (x,y,z)`

`outer(A,B)`

$$\begin{bmatrix} ax & ay & az \\ bx & by & bz \\ cx & cy & cz \end{bmatrix}$$

**pi**

Symbol for  $\pi$ .

`exp(i pi)`

-1

## **polar(*z*)**

Returns complex  $z$  in polar form.

```
polar(x - i y)
```

$$(x^2 + y^2)^{1/2} \exp(-i \arctan(y, x))$$

## **power**

Use `^` to raise something to a power. Use parentheses for negative powers.

```
x^(-2)
```

$$\frac{1}{x^2}$$

## **print(*a, b, ...*)**

Evaluate arguments and print the results. Useful for printing from inside a `for` loop.

```
for(j,1,3,print(j))
```

```
j = 1  
j = 2  
j = 3
```

## **product(*i, j, k, f*)**

For  $i$  equals  $j$  through  $k$  evaluate  $f$ . Returns the product of all  $f$ .

```
product(j,1,3,x + j)
```

$$x^3 + 6x^2 + 11x + 6$$

The original value of  $i$  is restored after `product` completes. If symbol `i` is used for index variable  $i$  then the imaginary unit is overridden in the scope of `product`.

## **product(*y*)**

Returns the product of components of  $y$ .

```
y = (1,2,3,4)  
product(y)
```



## **quote( $x$ )**

Returns expression  $x$  without evaluating it first.

```
quote((x + 1)^2)
```

$$(x + 1)^2$$

## **rand()**

Returns a random floating point value from the interval  $[0, 1)$ .

```
rand()
```

0.655424

## **rank( $a$ )**

Returns the number of indices that tensor  $a$  has.

```
A = ((a,b),(c,d))
```

```
rank(A)
```

2

## **rationalize( $x$ )**

Returns expression  $x$  with everything over a common denominator.

```
rationalize(1/a + 1/b + 1/2)
```

$$\frac{2a + ab + 2b}{2ab}$$

Note: **rationalize** returns an unexpanded expression. If the result is assigned to a symbol, evaluating the symbol will expand the result. Use **binding** to retrieve the unexpanded expression.

```
f = rationalize(1/a + 1/b + 1/2)
```

```
binding(f)
```

$$\frac{2a + ab + 2b}{2ab}$$

## **real( $z$ )**

Returns the real part of complex  $z$ .

```
real(2 - 3i)
```

2

## **rect( $z$ )**

Returns complex  $z$  in rectangular form.

```
rect(exp(i x))
```

$$\cos(x) + i \sin(x)$$

## **roots( $p, x$ )**

Returns the rational roots of a polynomial. Multiple roots are returned as a vector.

```
p = (x + 1) (x - 2)
roots(p,x)
```

$$\begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

If no roots are found then `nil` is returned. A `nil` result is not printed so the following example uses `infixform` to print `nil` as a string.

```
p = x^2 + 1
infixform(roots(p,x))
```

`nil`

## **rotate( $u, s, k, \dots$ )**

Rotates vector  $u$  and returns the result. Vector  $u$  is required to have  $2^n$  elements where  $n$  is an integer from 1 to 15. Arguments  $s, k, \dots$  are a sequence of rotation codes where  $s$  is an upper case letter and  $k$  is a qubit number from 0 to  $n - 1$ . Rotations are evaluated from left to right. See the section on quantum computing for a list of rotation codes.

```
psi = (1,0,0,0)
rotate(psi,H,0)
```

$$\begin{bmatrix} \frac{1}{2^{1/2}} \\ \frac{1}{2^{1/2}} \\ 0 \\ 0 \end{bmatrix}$$

## **run( $x$ )**

Run script  $x$  where  $x$  evaluates to a filename string. Useful for importing function libraries.

```
run("/Users/heisenberg/EVA2.txt")
```

For Eigenmath installed from the Mac App Store, run files need to be put in the directory `~/Library/Containers/com.gweigt.eigenmath/Data/` and the filename does not require a path.

```
run("EVA2.txt")
```

## **sgn( $x$ )**

Returns the sign of  $x$  if  $x$  is a real number.

```
sgn(0)
```

0

```
sgn(1/2)
```

1

```
sgn(-1/2)
```

-1

```
sgn(-x)
```

$\text{sgn}(-x)$

## **simplify( $x$ )**

Returns expression  $x$  in a simpler form.

```
simplify(sin(x)^2 + cos(x)^2)
```

1

The equality operator simplifies automatically.

```
sin(x)^2 + cos(x)^2 == 1
```

1

**sin**( $x$ )

Returns the sine of  $x$ .

`sin(pi/4)`

$$\frac{1}{2^{1/2}}$$

**sinh**( $x$ )

Returns the hyperbolic sine of  $x$ .

`expform(sinh(x))`

$$-\frac{1}{2}\exp(-x) + \frac{1}{2}\exp(x)$$

**sqrt**( $x$ )

Returns the square root of  $x$ .

`sqrt(10!)`

$$720 \cdot 7^{1/2}$$

**stop**

In a script, it does what it says.

**sum**( $i, j, k, f$ )

For  $i$  equals  $j$  through  $k$  evaluate  $f$ . Returns the sum of all  $f$ .

`sum(j,1,5,x^j)`

$$x^5 + x^4 + x^3 + x^2 + x$$

The original value of  $i$  is restored after **sum** completes. If symbol **i** is used for index variable  $i$  then the imaginary unit is overridden in the scope of **sum**.

**sum**( $y$ )

Returns the sum of components of  $y$ .

`y = (1,2,3,4)`  
`sum(y)`

$$10$$

## **tan( $x$ )**

Returns the tangent of  $x$ .

```
simplify(tan(x) - sin(x)/cos(x))
```

0

## **tanh( $x$ )**

Returns the hyperbolic tangent of  $x$ .

```
expform(tanh(x))
```

$$-\frac{1}{\exp(2x) + 1} + \frac{\exp(2x)}{\exp(2x) + 1}$$

## **test( $a, b, c, d, \dots$ )**

If argument  $a$  is true (nonzero) then  $b$  is returned, else if  $c$  is true then  $d$  is returned, etc. If the number of arguments is odd then the final argument is returned if all else fails. Expressions can include the relational operators =, ==, <, <=, >, >=. Use the **not** function to test for inequality. (The equality operator == is available for contexts in which = is the assignment operator.)

```
A = 1
```

```
B = 1
```

```
test(A=B, "yes", "no")
```

yes

## **tgamma( $x$ )**

Returns the Gamma function of  $x$  if  $x$  is a real number.

```
tgamma(4)
```

6

## **trace**

Set **trace=1** in a script to print the script as it is evaluated. Useful for debugging. (To obtain the trace of a matrix, use **contract**.)

## **transpose**( $a, i, j, \dots$ )

Returns the transpose of tensor  $a$  with respect to indices  $i, j$ , etc. If  $i$  and  $j$  are omitted then 1 and 2 are used, hence a matrix can be transposed with a single argument. The argument list can be extended for multiple transpose operations. Arguments are evaluated from left to right. For example, `transpose(A,1,2,2,3)` is equivalent to `transpose(transpose(A,1,2),2,3)`

```
A = ((a,b),(c,d))  
transpose(A)
```

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

## **tty**

Set `tty=1` to show results in string format. Set `tty=0` to turn off. Can be useful when displayed results exceed window size.

```
tty = 1  
(x + 1)^2
```

```
x^2 + 2 x + 1
```

## **unit**( $n$ )

Returns an  $n$  by  $n$  identity matrix.

```
unit(3)
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## **zero**( $a, b, \dots$ )

Returns a null tensor with dimensions  $a, b$ , etc.

```
zero(2,3,3)
```

$$\begin{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{bmatrix}$$