# 1 Introduction

Eigenmath was created for doing physics problems, so here is an example from quantum mechanics.

Let

$$X = x, \quad P = -i\hbar\frac{\partial}{\partial x}$$

Show that

$$(XP - PX)\psi(x,t) = i\hbar\psi(x,t)$$

Eigenmath code:

```
X(f) = x f
P(f) = -i hbar d(f,x)
X(P(psi(x,t))) - P(X(psi(x,t)))
```

Result:

$i\hbar\psi(x,t)$

In three dimensions (symbol $\otimes$ is outer product, $\nabla$ is gradient)

$$X = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \otimes, \quad P = -i\hbar\nabla$$

Eigenmath code:

```
X(f) = outer((x,y,z),f)
P(f) = -i hbar d(f,(x,y,z))
X(P(psi(x,y,z,t))) - P(X(psi(x,y,z,t)))
```

Result:

$$\begin{bmatrix} i\hbar\psi(x,y,z,t) & 0 & 0 \\ 0 & i\hbar\psi(x,y,z,t) & 0 \\ 0 & 0 & i\hbar\psi(x,y,z,t) \end{bmatrix}$$

The main takeaway is that in Eigenmath code

$$\frac{\partial f(x)}{\partial x} = \texttt{d(f(x),x)}$$

and

$$\nabla f(x,y,z) = \texttt{d(f(x,y,z),(x,y,z))}$$

# 2 Syntax

Arithmetic operators have the expected precedence of multiplication and division before addition and subtraction. Subexpressions in parentheses have highest precedence.

| *Math* | *Eigenmath* | *Comment* |
|---|---|---|
| $a = b$ | `a == b` | *test for equality* |
| $-a$ | `-a` | *negation* |
| $a + b$ | `a+b` | *addition* |
| $a - b$ | `a-b` | *subtraction* |
| $ab$ | `a b` | *multiplication, also* `a*b` |
| $\frac{a}{b}$ | `a/b` | *division* |
| $\frac{a}{bc}$ | `a/b/c` | *division is left-associative* |
| $a^2$ | `a^2` | *power* |
| $\sqrt{a}$ | `sqrt(a)` | *square root, also* `a^(1/2)` |
| $a\,(b+c)$ | `a (b+c)` | *space is required* |
| $f(a)$ | `f(a)` | *function* |
| $\begin{pmatrix} a \\ b \\ c \end{pmatrix}$ | `(a,b,c)` | *vector* |
| $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ | `((a,b),(c,d))` | *matrix* |
| $F^1{}_2$ | `F[1,2]` | *tensor component access* |
| | `"hello, world"` | *string literal* |
| $\pi$ | `pi` | |
| $e$ | `exp(1)` | *base of natural logarithm* |

Parentheses are required around negative exponents. For example,

```
10^(-3)
```

instead of

```
10^-3
```

In general, parentheses are always required when the exponent is an expression. For example, `x^1/2` is evaluated as $(x^1)/2$ which is probably not the desired result.

```
x^1/2
```

$\frac{1}{2}x$

Using `x^(1/2)` yields the desired result.

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

$x^{1/2}$

# 3 Symbols

Symbols are defined with an equals sign.

```
N = 212^17
```

No result is printed when a symbol is defined. To see the value of a symbol, just evaluate it.

```
N
```

$N = 352947114576027513230189734205586617139 2$

Symbols can have more that one letter. Everything after the first letter is displayed as a subscript.

```
NA = 6.02214 10^23
NA
```

$N_A = 6.02214 \times 10^{23}$

A symbol can be the name of a Greek letter.

```
xi = 1/2
xi
```

$\xi = \frac{1}{2}$

Greek letters can appear in subscripts.

```
Amu = 2.0
Amu
```

$A_{\mu} = 2.0$

The following example shows how a symbol is scanned to find Greek letters.

```
alphamunu = 1
alphamunu
```

$\alpha_{\mu\nu} = 1$

Symbol definitions are evaluated serially until a terminal symbol is reached. The following example sets $A = B$ followed by $B = C$. Then when $A$ is evaluated, the result is $C$.

```
A = B
B = C
A
```

$A = C$

Although $A = C$ is printed, inside the program the binding of $A$ is still $B$, as can be seen with the `binding` function.

```
binding(A)
```

$B$

The `quote` function returns its argument unevaluated and can be used to clear a symbol. The following example clears $A$ so that its evaluation goes back to being $A$ instead of $C$.

```
A = quote(A)
A
```

$A$

# 4 Units of measure

Symbols and quoted strings can be used for units of measure.

```
v = 1.2 meter / second
v
```

$$\frac{1.2\, m_{eter}}{s_{econd}}$$

Assign strings to unit symbols for improved display appearance.

```
meter = "m"
second = "s"
v
```

$$\frac{1.2\,\mathrm{m}}{\mathrm{s}}$$

Derived units can be handled by converting to base units.

```
h = 6.626 10^(-34) joule second
joule = kilogram meter^2 / second^2
kilogram = "kg"
h
```

$$h = \frac{6.626 \times 10^{-34}\,\mathrm{kg\,m^2}}{\mathrm{s}}$$

Here is a trick for displaying derived units. In this example, convert joules to string "J".

```
h "J" / joule
```

$6.626 \times 10^{-34}\,\mathrm{J\,s}$

Eigenmath script for a calculation from "The Los Alamos Primer."

```
-- energy released per atom
E = 170 10^6 "eV" / "atom"
-- convert eV to ergs
E = E 4.8 10^(-10) / 300 "erg" / "eV"
-- convert atoms to grams
E = E / (3.88 10^(-22) "gram" / "atom")
-- convert ergs to tons of TNT
E = E / (3.6 10^16 "erg" / "tons of TNT")
-- energy per kg of U235
E = E 1000 "gram"
E
```

Result:

$E = 19473.1$ tons of TNT

# 5 Function definitions

The following example defines a sinc function and evaluates it at $\pi/2$.

```
f(x) = sin(x)/x
f(pi/2)
```

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

In a function definition, use `eval` to evaluate an argument with a substitution.

```
h(f,a,b) = eval(f,x,b) - eval(f,x,a)
h(x^2, 1, 2)
```

3

To define a local symbol in a function, extend the argument list. In the following example, argument `y` is used as a local symbol. Note that function `L` is called without supplying an argument for `y`.

```
L(f,n,y) = eval(exp(y) d(exp(-y) y^n, y, n) / n!, y, f)
L(cos(x),2)
```

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

Use `do` when multiple steps are needed in a function. The last `do` item is the return value. The following example defines function `I` for integrating hydrogen wavefunctions.

```
I(f) = do(
  f = expform(f r^2 sin(theta)),
  f = defint(f, theta, 0, pi, phi, 0, 2 pi),
  f = integral(f,r),
  -eval(f,r,0) -- return value
)
```

Notes:
1. Maximum number of arguments is nine.
2. Argument scope is restricted to just the function definition.
3. Function definitions cannot be nested.

# 6 Arithmetic

Big integer arithmetic is used so that numerical values can exceed machine size.

```
2^64
```

18446744073709551616

```
212^17
```

352947114576027513230189734205866171392

Rational number arithmetic is used by default.

```
1/2 + 1/3
```

$\frac{5}{6}$

Mixed mode arithmetic gives a floating point result.

```
1/2 + 1/3.0
```

0.833333

The `float` function converts integers and rationals to floating point values.

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

$3.52947 \times 10^{39}$

The following example shows how to enter a floating point value using scientific notation.

```
epsilon = 1.0 10^(-6)
epsilon
```

$\varepsilon = 1.0 \times 10^{-6}$

# 7 Complex numbers

Symbol `i` is initialized to $\sqrt{-1}$.

Complex quantities can be entered in either rectangular or polar form.

```
a + i b
```

$a + ib$

```
exp(1/3 i pi)
```

$\exp\left(\frac{1}{3}i\pi\right)$

Converting a complex number to rectangular or polar coordinates causes simplification of mixed forms.

```
A = 1 + i
B = sqrt(2) exp(1/4 i pi)
A - B
```

$1 + i - 2^{1/2} \exp\left(\frac{1}{4}i\pi\right)$

```
rect(last)
```

0

Rectangular complex quantities, when raised to a power, are multiplied out.

```
(a + i b)^2
```

$a^2 - b^2 + 2iab$

When $a$ and $b$ are numerical and the power is negative, the evaluation is done as follows.

$$(a + ib)^{-n} = \left(\frac{a - ib}{(a + ib)(a - ib)}\right)^n = \left(\frac{a - ib}{a^2 + b^2}\right)^n$$

Here are a few examples.

```
1/(2 - i)
```

$\frac{2}{5} + \frac{1}{5}i$

```
(-1 + 3 i)/(2 - i)
```

$-1 + i$

The absolute value of a complex number returns its magnitude.

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

5

The imaginary unit can be changed from $i$ to $j$ by defining $j = \sqrt{-1}$.

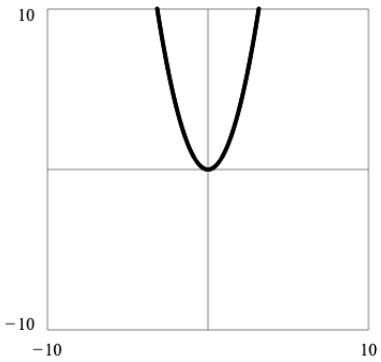
```
j = sqrt(-1)
sqrt(-4)
```

$2j$

# 8 Draw

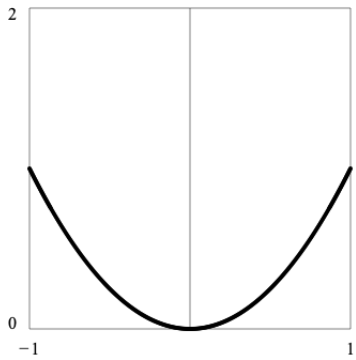
`draw(f,x)` draws a graph of function $f$ of $x$.

```
draw(x^2,x)
```



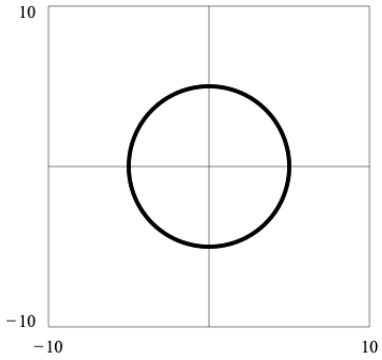

The vectors `xrange` and `yrange` control the scale of the graph.

```
xrange = (-1,1)
yrange = (0,2)
draw(x^2)
```



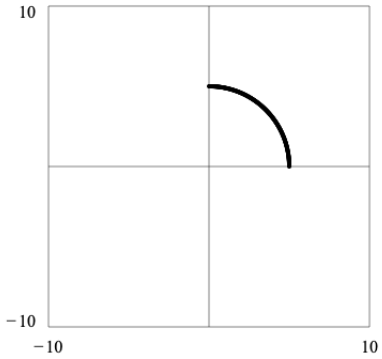

Parametric drawing occurs when a function returns a vector. The vector `trange` controls the parametric range. The default is `trange=(-pi,pi)`. In the following example, `draw` varies `theta` over the default range $-\pi$ to $+\pi$.

```
xrange = (-10,10)
yrange = (-10,10)
f = 5 (cos(theta),sin(theta))
draw(f,theta)
```



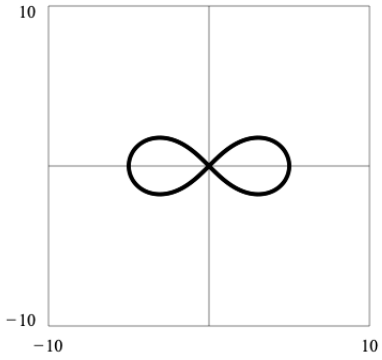

In the following example, `trange` is reduced to draw a quarter circle instead of a full circle.

```
trange = (0,pi/2)
f = 5 (cos(theta),sin(theta))
draw(f,theta)
```



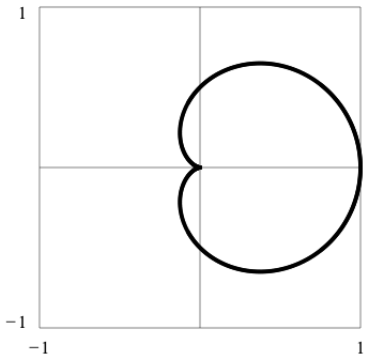

Draw a lemniscate.

```
trange = (-pi,pi)
X = cos(t) / (1 + sin(t)^2)
Y = sin(t) cos(t) / (1 + sin(t)^2)
f = 5 (X,Y)
draw(f,t)
```



Draw a cardioid.

```
r = (1 + cos(t)) / 2
u = (cos(t),sin(t))
f = r u
xrange = (-1,1)
yrange = (-1,1)
trange = (0,2pi)
draw(f,t)
```



# 9 Linear algebra

`dot(a,b,...)` returns the inner product of vectors, matrices, and higher rank tensors. Also known as the matrix product. Arguments are evaluated from right to left for maximum efficiency when the rightmost argument is a vector.

Example 1. Compute the product $AX$ for

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

```
A = ((a11,a12),(a21,a22))
X = (x1,x2)
dot(A,X)
```

$$\begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{bmatrix}$$

Example 2. Solve for vector $X$ in $AX = B$.

```
A = ((3,7),(1,-9))
B = (16,-22)
X = dot(inv(A),B)
X
```

$$X = \begin{bmatrix} -\frac{5}{17} \\ \frac{41}{17} \end{bmatrix}$$

Example 3. Show that

$$A^{-1} = \frac{\operatorname{adj} A}{\det A}$$

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

1

# 10 Component arithmetic

Tensor plus scalar adds scalar to each tensor component.

```
A = ((a,b),(c,d))
A + 10
```

$$\begin{bmatrix} a+10 & b+10 \\ c+10 & d+10 \end{bmatrix}$$

The product of two tensors is the Hadamard (element-wise) product.

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

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

Tensor raised to a power raises each component to the power.

```
A = ((a,b),(c,d))
A^2
```

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

# 11 Quantum computing

A quantum computer can be simulated by applying rotations to a unit vector $u \in \mathbb{C}^{2^n}$ where $\mathbb{C}$ is the set of complex numbers and $n$ is the number of qubits. The dimension is $2^n$ because a register with $n$ qubits has $2^n$ eigenstates. (Recall that an eigenstate is the output of a quantum computer.) Quantum operations are "rotations" because they preserve $|u| = 1$. Mathematically, a rotation of $u$ is equivalent to the product $Ru$ where $R$ is a $2^n \times 2^n$ matrix.

Eigenstates $|j\rangle$ are represented by the following vectors. (Each vector has $2^n$ elements.)

$$\begin{aligned}
|0\rangle &= (1, 0, 0, \ldots, 0) \\
|1\rangle &= (0, 1, 0, \ldots, 0) \\
|2\rangle &= (0, 0, 1, \ldots, 0) \\
&\vdots \\
|2^n - 1\rangle &= (0, 0, 0, \ldots, 1)
\end{aligned}$$

A quantum computer algorithm is a sequence of rotations applied to the initial state $|0\rangle$. (The sequence could be combined into a single rotation by associativity of matrix multiplication.) Let $\psi_f$ be the final state of the quantum computer after all the rotations have been applied. Like any other state, $\psi_f$ is a linear combination of eigenstates.

$$\psi_f = \sum_{j=0}^{2^n-1} c_j |j\rangle, \quad c_j \in \mathbb{C}, \quad |\psi_f| = 1$$

The last step is to measure $\psi_f$ and get a result. Measurement rotates $\psi_f$ to an eigenstate $|j\rangle$. The measurement result is $|j\rangle$. The probability $P_j$ of getting a specific result $|j\rangle$ is

$$P_j = |c_j|^2 = c_j c_j^*$$

Note that if $\psi_f$ is already an eigenstate then no rotation occurs. (The probability of observing a different eigenstate is zero.) Since the measurement result is always an eigenstate, the coefficients $c_j$ cannot be observed. However, the same calculation can be run multiple times to obtain a probability distribution of results. The probability distribution is an estimate of $|c_j|^2$ for each $|j\rangle$ in $\psi_f$.

Unlike a real quantum computer, in a simulation the final state $\psi_f$, or any other state, is available for inspection. Hence there is no need to simulate the measurement process. The probability distribution of the result can be computed directly as

$$P = \psi_f \, \psi_f^*$$

where $\psi_f \, \psi_f^*$ is the Hadamard (element-wise) product of vector $\psi_f$ and its complex conjugate. Result $P$ is a vector such that $P_j$ is the probability of eigenstate $|j\rangle$ and

$$\sum_{j=0}^{2^n-1} P_j = 1$$

Note: Eigenmath index numbering begins with 1 hence `P[1]` is the probability of $|0\rangle$, `P[2]` is the probability of $|1\rangle$, etc.

The Eigenmath function `rotate`$(u, s, k, \ldots)$ 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 $s, k, \ldots$ 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. The available rotation codes are

| | |
|---|---|
| $C, k$ | Control prefix |
| $H, k$ | Hadamard |
| $P, k, \phi$ | Phase modifier (use $\phi = \frac{1}{4}\pi$ for $T$ rotation) |
| $Q, k$ | Quantum Fourier transform |
| $V, k$ | Inverse quantum Fourier transform |
| $W, k, j$ | Swap bits |
| $X, k$ | Pauli X |
| $Y, k$ | Pauli Y |
| $Z, k$ | Pauli Z |

Control prefix $C, k$ modifies the next rotation code so that it is a controlled rotation with $k$ as the control qubit. Use two or more prefixes to specify multiple control qubits. For example, $C, k, C, j, X, m$ is a Toffoli rotation. Fourier rotations $Q, k$ and $V, k$ are applied to qubits 0 through $k$. ($Q$ and $V$ ignore any control prefix.)

List of `rotate`$(u, s, k, \ldots)$ error codes:

1 Argument $u$ is not a vector or does not have $2^n$ elements where $n = 1, 2, \ldots, 15$.

2 Unexpected end of argument list (i.e., missing argument).

3 Bit number format error or range error.

4 Unknown rotation code.

Example: Verify the following truth table for quantum operator CNOT where qubit 0 is the control and qubit 1 is the target. (Target is inverted when control is set.)

| Target | Control | Output |
|---|---|---|
| 0 | 0 | 00 |
| 0 | 1 | 11 |
| 1 | 0 | 10 |
| 1 | 1 | 01 |

```
U(psi) = rotate(psi,C,0,X,1) -- CNOT, control 0, target 1

ket00 = (1,0,0,0)
ket01 = (0,1,0,0)
ket10 = (0,0,1,0)
```

```
ket11 = (0,0,0,1)

U(ket00) == ket00
U(ket01) == ket11
U(ket10) == ket10
U(ket11) == ket01
```

Here are some useful Eigenmath code snippets for setting up a simulation and computing the result.

1. Initialize $\psi = |0\rangle$.

```
n = 4            -- number of qubits (example)
N = 2^n          -- number of eigenstates
psi = zero(N)
psi[1] = 1
```

2. Compute the probability distribution for state $\psi$.

```
P = psi conj(psi)
```

Hence

$$\begin{aligned}
\texttt{P[1]} &= \text{probability that } |0\rangle \text{ will be the result} \\
\texttt{P[2]} &= \text{probability that } |1\rangle \text{ will be the result} \\
\texttt{P[3]} &= \text{probability that } |2\rangle \text{ will be the result} \\
&\vdots \\
\texttt{P[N]} &= \text{probability that } |N-1\rangle \text{ will be the result}
\end{aligned}$$

3. (Only for macOS) Draw a probability distribution.

```
xrange = (0,N)
yrange = (0,1)
draw(P[ceiling(x)],x)
```

4. Compute an expectation value.

```
sum(k,1,N, (k - 1) P[k])
```

5. Make the high order qubit "don't care."

```
for(k,1,N/2, P[k] = P[k] + P[k + N/2])
```

Hence for $N = 16$

$$\begin{aligned}
\texttt{P[1]} &= \text{probability that the result will be } |0\rangle \text{ or } |8\rangle \\
\texttt{P[2]} &= \text{probability that the result will be } |1\rangle \text{ or } |9\rangle \\
\texttt{P[3]} &= \text{probability that the result will be } |2\rangle \text{ or } |10\rangle \\
&\vdots \\
\texttt{P[8]} &= \text{probability that the result will be } |7\rangle \text{ or } |15\rangle
\end{aligned}$$

# 12 Derivative

`d(f,x)` returns the derivative of $f$ with respect to $x$.

```
d(x^2,x)
```

$2x$

Extend the argument list for multiderivatives.

```
f = 1 / (x + y)
d(f,x,y)
```

$$\frac{2}{(x+y)^3}$$

```
d(sin(x),x,x)
```

$-\sin(x)$

Another syntax for $n$th derivative.

```
d(sin(x),x,2)
```

$-\sin(x)$

The gradient of $f$ is returned for vector $x$ in `d(f,x)`.

```
r = sqrt(x^2 + y^2)
d(r,(x,y))
```

$$\begin{bmatrix} \frac{x}{(x^2+y^2)^{1/2}} \\ \frac{y}{(x^2+y^2)^{1/2}} \end{bmatrix}$$

The $f$ in `d(f,x)` can be a vector or higher rank function. Gradient increases rank by one.

```
F = (x^2,y^2)
X = (x,y)
d(F,X)
```

$$\begin{bmatrix} 2x & 0 \\ 0 & 2y \end{bmatrix}$$

# 13 Template functions

Function $f$ in `d(f,x)` does not have to be defined, it can be a template function with just a name and an argument list. The argument list determines the result. For example, `d(f(x),x)` evaluates to itself because $f$ depends on $x$. However, `d(f(x),y)` evaluates to zero because $f$ does not depend on $y$.

Example 1. $f(x)$ depends on $x$.

```
d(f(x),x)
```

$\mathrm{d}(f(x),x)$

Example 2. $f(x)$ does not depend on $y$.

```
d(f(x),y)
```

0

Example 3. $f(x,y)$ depends on both $x$ and $y$.

```
d(f(x,y),y)
```

$\mathrm{d}(f(x,y),y)$

Example 4. $f()$ is a wildcard that matches any symbol.

```
d(f(),t)
```

$\mathrm{d}(f(),t)$

Template functions are useful for working with differential forms. For example, show that

$$\nabla \cdot (\nabla \times \mathbf{F}) = 0$$

```
F = (Fx(),Fy(),Fz())
div(curl(F))
```

0

# 14 Laplacian

The Laplacian $\nabla^2$ is the divergence of the gradient of scalar function $f$.

$$\nabla^2 f = \nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}$$

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

$\mathrm{d}(\mathrm{d}(f(), x), x) + \mathrm{d}(\mathrm{d}(f(), y), y) + \mathrm{d}(\mathrm{d}(f(), z), z)$

This is the vector Laplacian.

$$\nabla^2 \mathbf{A} = \nabla \cdot \nabla \mathbf{A} - \nabla \times (\nabla \times \mathbf{A}) = \begin{pmatrix} \nabla^2 A_x \\ \nabla^2 A_y \\ \nabla^2 A_z \end{pmatrix} = \frac{\partial^2 \mathbf{A}}{\partial x^2} + \frac{\partial^2 \mathbf{A}}{\partial y^2} + \frac{\partial^2 \mathbf{A}}{\partial z^2}$$

```
A = (Ax(),Ay(),Az())
div(grad(A)) - curl(curl(A)) == d(A,x,x) + d(A,y,y) + d(A,z,z)
```

1

Show that

$$\nabla \cdot \nabla \mathbf{A} = \nabla(\nabla \cdot \mathbf{A})$$

```
div(grad(A)) == grad(div(A))
```

1

# 15 Integral

`integral(f,x)` returns the integral of $f$ with respect to $x$.

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

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

Extend the argument list for multiple integrals.

```
f = x y
integral(f,x,y)
```

$\frac{1}{4}x^2y^2$

`defint(f,x,a,b)` computes 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 computes the integral of $f = x^2$ over the domain of a semicircle. For each $x$ along the abscissa, $y$ ranges from 0 to $\sqrt{1-x^2}$.

```
defint(x^2, y, 0, sqrt(1 - x^2), x, -1, 1)
```

$\frac{1}{8}\pi$

Alternatively, `eval` can be used to compute a definite integral step by step.

```
I = integral(x^2,y)
I = eval(I,y,sqrt(1 - x^2)) - eval(I,y,0)
I = integral(I,x)
eval(I,x,1) - eval(I,x,-1)
```

$\frac{1}{8}\pi$

Here is a useful trick. Integrals involving sine and cosine can often be solved using exponentials. For example, the definite integral

$$\int_0^{2\pi} \left(\sin^4 t - 2\cos^3(t/2)\sin t\right) dt$$

can be solved as follows.

```
f = sin(t)^4 - 2 cos(t/2)^3 sin(t)
f = expform(f)
defint(f, t, 0, 2 pi)
```

$\frac{3}{4}\pi - \frac{16}{5}$

# 16 Arc length

Let $g(t)$ be a parametric function that draws a curve in $\mathbb{R}^n$. The arc length from $g(a)$ to $g(b)$ is given by

$$\int_a^b |g'(t)|\, dt$$

where $|g'(t)|$ is the length of the tangent vector at $g(t)$.

Example 1. Find the length of the curve $y = x^2$ from $x = 0$ to $x = 1$.

```
g = (t,t^2)
defint(abs(d(g,t)),t,0,1)
```

$\frac{1}{2}\, 5^{1/2} - \frac{1}{4}\log(2) + \frac{1}{4}\log(2\, 5^{1/2} + 4)$

```
float
```

1.47894

As expected, the result is greater than $\sqrt{2} \approx 1.414$, the length of a straight line from $(0, 0)$ to $(1, 1)$.

The following script does a discrete computation of the arc length by dividing the curve into 100 pieces.

```
g(t) = (t,t^2)
h(k) = abs(g(k/100.0) - g((k-1)/100.0))
sum(k,1,100,h(k))
```

1.47894

As expected, the discrete result matches the analytic result.

Example 2. Find the length of the curve $y = x^{3/2}$ from the origin to $x = \frac{4}{3}$.

```
g = (t,t^(3/2))
defint(abs(d(g,t)),t,0,4/3)
```

$\frac{56}{27}$

# 17 Line integral

There are two kinds of line integrals, one for scalar fields and one for vector fields. The following table shows how both are based on the calculation of arc length.

| | Abstract form | Computable form |
|---|---|---|
| Arc length | $\int_C ds$ | $\int_a^b \lvert g'(t)\rvert \, dt$ |
| Line integral, scalar field | $\int_C f \, ds$ | $\int_a^b f(g(t)) \, \lvert g'(t)\rvert \, dt$ |
| Line integral, vector field | $\int_C (F \cdot u) \, ds$ | $\int_a^b F(g(t)) \cdot g'(t) \, dt$ |

Note that for the measure $ds$ we have

$$ds = |g'(t)| \, dt$$

For vector fields, symbol $u$ is the unit tangent vector

$$u = \frac{g'(t)}{|g'(t)|}$$

Note that $u$ cancels with $ds$ as follows.

$$\int_C (F \cdot u) \, ds = \int_a^b \left( F(g(t)) \cdot \frac{g'(t)}{|g'(t)|} \right) |g'(t)| \, dt = \int_a^b F(g(t)) \cdot g'(t) \, dt$$

Example 1. Evaluate $\int_C x \, ds$ where $C$ is a straight line from $(0, 0)$ to $(1, 1)$.

```
x = t
y = t
g = (x,y)
defint(x abs(d(g,t)), t, 0, 1)
```

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

Example 2. Evaluate $\int_C x \, dx$ where $C$ is a straight line from $(0, 0)$ to $(1, 1)$.

We have $x \, dx = (F \cdot u) \, ds$ hence

```
x = t
y = t
g = (x,y)
F = (x,0)
defint(dot(F,d(g,t)), t, 0, 1)
```

$\frac{1}{2}$

The following line integral problems are from *Advanced Calculus, Fifth Edition* by Wilfred Kaplan.

Example 3. Evaluate $\int y^2\,dx$ along the straight line from $(0,0)$ to $(2,2)$.

The following solution parametrizes $x$ and $y$ so that the endpoint $(2,2)$ corresponds to $t = 1$.

```
x = 2 t
y = 2 t
g = (x,y)
F = (y^2,0)
defint(dot(F,d(g,t)), t, 0, 1)
```

$\frac{8}{3}$

Example 4. Evaluate $\int z\,dx + x\,dy + y\,dz$ along the path $x = 2t + 1$, $y = t^2$, $z = 1 + t^3$, $0 \le t \le 1$.

```
x = 2 t + 1
y = t^2
z = 1 + t^3
g = (x,y,z)
F = (z,x,y)
defint(dot(F,d(g,t)), t, 0, 1)
```

$\frac{163}{30}$

# 18 Surface area

Let $S$ be a surface parameterized by $x$ and $y$. That is, let $S = (x, y, z)$ where $z = f(x, y)$. The tangent lines at a point on $S$ form a tiny parallelogram. The area $a$ of the parallelogram is given by the magnitude of the cross product.

$$a = \left| \frac{\partial S}{\partial x} \times \frac{\partial S}{\partial y} \right|$$

By summing over all the parallelograms we obtain the total surface area $A$. Hence

$$A = \int \int dA = \int \int a \, dx \, dy$$

The following example computes the surface area of a unit disk parallel to the $xy$ plane.

```
z = 2
S = (x,y,z)
a = abs(cross(d(S,x),d(S,y)))
defint(a,y,-sqrt(1 - x^2),sqrt(1 - x^2),x,-1,1)
```

$\pi$

The result is $\pi$, the area of a unit circle, which is what we expect. The following example computes the surface area of $z = x^2 + 2y$ over a unit square.

```
z = x^2 + 2y
S = (x,y,z)
a = abs(cross(d(S,x),d(S,y)))
defint(a,x,0,1,y,0,1)
```

$\frac{5}{8}\log(5) + \frac{3}{2}$

The following exercise is from *Multivariable Mathematics* by Williamson and Trotter, p. 598. Find the area of the spiral ramp defined by

$$S = \begin{pmatrix} u\cos v \\ u\sin v \\ v \end{pmatrix}, \quad 0 \le u \le 1, \quad 0 \le v \le 3\pi$$

```
x = u cos(v)
y = u sin(v)
z = v
S = (x,y,z)
a = expform(abs(cross(d(S,u),d(S,v))))
defint(a,u,0,1,v,0,3pi)
```

$\frac{3\pi}{2^{1/2}} + \frac{3}{2}\pi \log\left(2^{1/2} + 1\right)$

```
float
```

10.8177

# 19 Surface integral

A surface integral is like adding up all the wind on a sail. In other words, we want to compute

$$\iint \mathbf{F} \cdot \mathbf{n}\, dA$$

where $\mathbf{F} \cdot \mathbf{n}$ is the amount of wind normal to a tiny parallelogram $dA$. The integral sums over the entire area of the sail. Let $S$ be the surface of the sail parameterized by $x$ and $y$. (In this model, the $z$ direction points downwind.) By the properties of the cross product we have the following for the unit normal $\mathbf{n}$ and for $dA$.

$$\mathbf{n} = \frac{\frac{\partial S}{\partial x} \times \frac{\partial S}{\partial y}}{\left|\frac{\partial S}{\partial x} \times \frac{\partial S}{\partial y}\right|} \qquad dA = \left|\frac{\partial S}{\partial x} \times \frac{\partial S}{\partial y}\right| dx\, dy$$

Hence

$$\iint \mathbf{F} \cdot \mathbf{n}\, dA = \iint \mathbf{F} \cdot \left(\frac{\partial S}{\partial x} \times \frac{\partial S}{\partial y}\right) dx\, dy$$

The following exercise is from *Advanced Calculus* by Wilfred Kaplan, p. 313. Evaluate the surface integral

$$\iint_S \mathbf{F} \cdot \mathbf{n}\, d\sigma$$

where $\mathbf{F} = xy^2z\mathbf{i} - 2x^3\mathbf{j} + yz^2\mathbf{k}$, $S$ is the surface $z = 1 - x^2 - y^2$, $x^2 + y^2 \leq 1$ and $\mathbf{n}$ is upper.

Note that the surface intersects the $xy$ plane in a circle. By the right hand rule, crossing $x$ into $y$ yields $\mathbf{n}$ pointing upwards hence

$$\mathbf{n}\, d\sigma = \left(\frac{\partial S}{\partial x} \times \frac{\partial S}{\partial y}\right) dx\, dy$$

The following code computes the surface integral. The symbols $f$ and $h$ are used as temporary variables.

```
z = 1 - x^2 - y^2
F = (x y^2 z, -2 x^3, y z^2)
S = (x,y,z)
f = dot(F,cross(d(S,x),d(S,y)))
h = sqrt(1 - x^2)
defint(f, y, -h, h, x, -1, 1)
```

$\frac{1}{48}\pi$

# 20 Index

## abs($x$)

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

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

5

## adj($m$)

Returns the adjunct of matrix $m$. The inverse of $m$ equals adjunct divided by determinant.

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

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

## and($a, b, \ldots$)

Returns 1 if all arguments have nonzero values, returns 0 otherwise. Arguments can use the relational operators `==` `<` `<=` `>` `>=`

```
and(a,b)
```

1

## arg($z$)

Returns the angle of complex $z$. Symbols are treated as representing real numbers. If $z$ is a vector, matrix, or higher order tensor then `arg` is applied to each component.

```
arg(x + i y)
```

$\arctan(y, x)$

## 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.

```
check(exp(i pi) == -1)
```

## 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 $e$. Symbols are treated as representing real numbers. If $z$ is a vector, matrix, or higher order tensor then `clock` is applied to each component.

```
clock(x + i y)
```

$(-1)^{\frac{\arctan(y,x)}{\pi}} \left[x^2 + y^2\right]^{1/2}$

## conj($z$)

Returns the complex conjugate of $z$. Symbols are treated as representing real numbers. If $z$ is a vector, matrix, or higher order tensor then `conj` is applied to each component.

```
conj(x + i y)
```

$x - iy$

## contract($a, i, j, \ldots$)

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 `x`, `y`, and `z`.

## d($f, x, \ldots$)

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),x,2)
```

$-\sin(x)$

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)
```

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

Symbol `d` can be used as a variable name. Doing so does not conflict with function `d`. Function `derivative` is a synonym for `d`. Symbol `d` can be redefined as a different function in which case `derivative` is still available.

## defint($f, x, a, b, \ldots$)

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 `x`, `y`, and `z`.

## do($a, b, \ldots$)

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

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

3

## dot($a, b, \ldots$)

Returns the inner product of arguments $a$, $b$, etc. Arguments can have any rank and are evaluated from right to left.

```
dot((a,b),(c,d))
```

$ac + bd$

## 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($x$)

Returns the error function of $x$. Returns a numerical value if $x$ is a real number.

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

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

## erfc($x$)

Returns the complementary error function of $x$. Returns a numerical value if $x$ is a real number.

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

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

## eval($f, a, b, c, d, \ldots$)

Returns $f$ evaluated with expression $a$ replaced by expression $b$, $c$ by $d$, etc.

```
f = exp(i x)
eval(f,x,pi)
```

$-1$

## exit

Terminate and return to the shell (only for shell-mode Eigenmath).

## exp($x$)

Returns the exponential of $x$.

```
exp(i pi)
```

$-1$

## expform($x$)

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

```
expform(cos(x))
```

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

## factorial($n$)

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

```
20!
```

2432902008176640000

## fdist($x, df_1, df_2$)

Returns the probability that a random sample from an $F$-distribution is less than or equal to $x$.

## 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($k, a, b, c, \ldots$)

For $k$ equals $a$ to $b$ inclusive, evaluate the remaining arguments in a loop. Symbol $k$ is advanced by plus or minus 1 in the direction of $b$ each time through the loop. Use `break` to exit the loop immediately. The original value of $k$ is restored after `for` completes. If symbol `i` is used for $k$ 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())
```

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

## hadamard($a, b, \ldots$)

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 the imaginary unit.

```
i^2
```

$-1$

## imag($z$)

Returns the imaginary part of complex $z$. Symbols are treated as representing real numbers. If $z$ is a vector, matrix, or higher order tensor then `imag` is applied to each component.

```
imag(x + i y)
```

$y$

## incbeta($x, a, b$)

Returns the incomplete beta function of $x$.

## infixform($x$)

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

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

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

## inner($a, b, \ldots$)

Returns the inner product of arguments $a$, $b$, etc. Arguments can have any rank and are evaluated from right to left.

```
inner((a,b),(c,d))
```

$ac + bd$

## integral($f, x, \ldots$)

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

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

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

## inv($m$)

Returns the inverse of matrix $m$.

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

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

## kronecker($a, b, \ldots$)

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

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

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

## last

Symbol `last` holds the previous result.

```
212^17
```

352947114576027513230189734205586617139 2

```
last
```

352947114576027513230189734205586617139 2

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

```
212^17
```

352947114576027513230189734205586617139 2

```
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)
```

$y \log(x)$

## logform($x$)

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

```
logform(arccos(x))
```

$$-i \log \left[x + i \left[-\operatorname{abs}(x)^2 + 1\right]^{1/2}\right]$$

## loop($a, b, c, \ldots$)

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 $z$. Symbols are treated as representing real numbers. If $z$ is a vector, matrix, or higher order tensor then `mag` is applied to each component.

```
mag(x + i y)
```

$$\left[x^2 + y^2\right]^{1/2}$$

## 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 1 if $x$ has a value of zero, returns 0 otherwise.

```
not(a == b)
```

1

## 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)
```

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

## 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$(a, b, \ldots)$

Returns 1 if any argument has a nonzero value, returns 0 otherwise. Arguments can use the relational operators `==` `<` `<=` `>` `>=`

```
or(a,b)
```

1

## outer$(a, b, \ldots)$

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. Symbols are treated as representing real numbers. If $z$ is a vector, matrix, or higher order tensor then `polar` is applied to each component.

```
polar(x + i y)
```

$$\left[x^2 + y^2\right]^{1/2} \exp(i \arctan(y, x))$$

## power

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

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

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

## print$(a, b, \ldots)$

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$(k, a, b, f)$

For $k$ equals $a$ to $b$ evaluate $f$. Returns the product of all $f$. The original value of $k$ is restored after `product` completes. If `i` is used for $k$ then the imaginary unit is overridden in the scope of `product`.

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

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

## product$(y)$

Returns the product of components of $y$.

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

24

## 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 $z$. Symbols are treated as representing real numbers. If $z$ is a vector, matrix, or higher order tensor then `real` is applied to each component.

```
real(x + i y)
```

$x$

## rect($z$)

Returns complex $z$ in rectangular form. Symbols are treated as representing real numbers. If $z$ is a vector, matrix, or higher order tensor then `rect` is applied to each component.

```
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($u, s, k, \ldots$)

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 $s, k, \ldots$ 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)
```

$\mathrm{sgn}(-x)$

## 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 $7^{1/2}$

## stop

In a script, it does what it says.

## sum($k, a, b, f$)

For $k$ equals $a$ to $b$ evaluate $f$. Returns the sum of all $f$. The original value of $k$ is restored after `sum` completes. If `i` is used for $k$ then the imaginary unit is overridden in the scope of `sum`.

```
sum(k,1,3,x^k)
```

$x^3 + x^2 + x$

## 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))
```

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

## taylor($f, x, n, a$)

Returns the $n$th order Taylor series expansion of $f(x)$ at $a$. If argument $a$ is omitted then zero is used for the expansion point.

```
taylor(1/(1-x),x,5)
```

$x^5 + x^4 + x^3 + x^2 + x + 1$

## tdist($x, df$)

Returns the probability that a random sample from a $t$-distribution is less than or equal to $x$. The inverse is `tdistinv(x,df)`.

## test($a, b, c, d, \ldots$)

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. Arguments can use the relational operators `==` `<` `<=` `>` `>=`

```
test(a == b, "yes", "no")
```

no

## 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($a, i, j, \ldots$)

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($a, b, \ldots$)

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

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

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

# 21 Tricks

1. Use `==` to test for equality. In effect, `A==B` is equivalent to `simplify(A-B)==0`.

2. In a script, line breaking is allowed where the scanner needs something to complete an expression. For example, the scanner will automatically go to the next line after an operator.

3. Setting `trace=1` in a script causes each line to be printed just before it is evaluated. Useful for debugging.

4. The last result is stored in symbol `last`.

5. Use `contract(A)` to get the mathematical trace of matrix $A$.

6. Use `binding(s)` to get the unevaluated binding of symbol $s$.

7. Use `s=quote(s)` to clear symbol $s$.

8. Use `float(pi)` to get the floating point value of $\pi$. Set `pi=float(pi)` to evaluate expressions with a numerical value for $\pi$. Set `pi=quote(pi)` to make $\pi$ symbolic again.

9. Use `e=exp(1)` to assign the natural number $e$ to symbol `e`.

10. Assign strings to unit names so they are printed normally. For example, setting `meter="meter"` causes symbol `meter` to be printed as meter instead of $m_{eter}$.

11. Use `expsin` and `expcos` instead of `sin` and `cos`. Trigonometric simplifications occur automatically when exponentials are used. See also `expform` for converting an expression to exponential form.

12. Use `rect(expform(f))` to maybe find a new form of trigonometric expression `f`.

```
f = cos(theta/2)^2
rect(expform(f))
```

$\frac{1}{2}\cos(\theta)+\frac{1}{2}$

13. Complex number functions `conj`, `mag`, etc. treat undefined symbols as representing real numbers. To define symbols that represent complex numbers, use separate symbols for the real and imaginary parts.

```
z = x + i y
conj(z) z
```

$x^2+y^2$

```
z = A exp(i theta)
conj(z) z
```

$A^2$

14. Use `mag` for component magnitude, `abs` for vector magnitude.

```
y = (a, -b)
mag(y)
```

$$\begin{bmatrix} a \\ b \end{bmatrix}$$

```
abs(y)
```

$[a^2 + b^2]^{1/2}$

15. Use `draw(y[floor(x)],x)` to plot the values of vector `y`.

```
y = (1,2,3,4)
draw(y[floor(x)],x)
```

16. The following example demonstrates some `eval` tricks. (See exercise 4-10 of *Quantum Mechanics* by Richard Fitzpatrick.)

Let

$$\psi = \frac{\phi_1 + \phi_2}{2} \exp\left(-\frac{iE_1 t}{\hbar}\right) + \frac{\phi_1 - \phi_2}{2} \exp\left(-\frac{iE_2 t}{\hbar}\right)$$

where $\phi_1$ and $\phi_2$ are orthogonal. Let operator $A$ have the following eigenvalues.

$$A\phi_1 = a_1\phi_1$$
$$A\phi_2 = a_2\phi_2$$

Verify that

$$\langle A \rangle = \int \psi^* A \psi \, dx = \frac{a_1 + a_2}{2} + \frac{a_1 - a_2}{2} \cos\left(\frac{(E_1 - E_2)t}{\hbar}\right)$$

Because $\phi_1$ and $\phi_2$ are normalized we have $\int \phi_1^* \phi_1 \, dx = 1$ and $\int \phi_2^* \phi_2 \, dx = 1$. By orthogonality we have $\int \phi_1^* \phi_2 \, dx = 0$. Hence the integral can be accomplished with `eval`.

```
phi1 = r1(x) exp(i theta1(x)) -- note that conj(phi1) phi1 == r1(x)^2
phi2 = r2(x) exp(i theta2(x)) -- note that conj(phi2) phi2 == r2(x)^2

psi = 1/2 (phi1 + phi2) exp(-i E1 t / hbar) +
      1/2 (phi1 - phi2) exp(-i E2 t / hbar)

A(f) = eval(f, phi1, a1 phi1, phi2, a2 phi2) -- eigenvalues

f = conj(psi) A(psi)

Abar = eval(f, r1(x)^2, 1, r2(x)^2, 1, r1(x) r2(x), 0) -- integral

check(Abar == (a1 + a2) / 2 + (a1 - a2) / 2 cos((E1 - E2) t / hbar))

"ok"
```