Eigenmath Manual

April 24, 2021

Contents

1 Introduction **2**
- 1.1 Syntax . . . 3
- 1.2 Testing for equality . . . 4
- 1.3 Arithmetic . . . 4
- 1.4 Exponents . . . 5
- 1.5 Symbols . . . 5
- 1.6 User defined functions . . . 7
- 1.7 Scripts . . . 7
- 1.8 Draw . . . 9
- 1.9 Complex numbers . . . 11
- 1.10 Linear algebra . . . 13

2 Calculus **15**
- 2.1 Derivative . . . 15
- 2.2 Gradient . . . 15
- 2.3 Template functions . . . 15
- 2.4 Integral . . . 16
- 2.5 Arc length . . . 18
- 2.6 Line integrals . . . 20
- 2.7 Surface area . . . 22
- 2.8 Surface integrals . . . 23
- 2.9 Green's theorem . . . 24
- 2.10 Stokes' theorem . . . 25

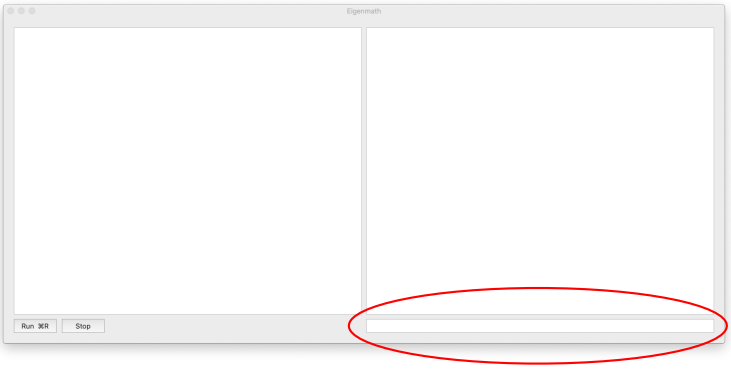
3 Quantum Computing **27**

4 Function Reference **31**

5 Tricks **48**

1 Introduction

The field at the bottom of the Eigenmath window is for entering calculations that get evaluated right away.



We can use Eigenmath to check the following arithmetic from Vladimir Nabokov's autobiography "Speak, Memory."

> A foolish tutor had explained logarithms to me much too early, and I had read (in a British publication, the *Boy's Own Paper*, I believe) about a certain Hindu calculator who in exactly two seconds could find the seventeenth root of, say, 352947114576027513 230189734205586617139 2 (I am not sure I have got this right; anyway the root was 212).

In the field at the bottom of the Eigenmath window, enter the following to compute 212^{17}.

```
212^17
```

After pressing the return key, Eigenmath displays the following result.

352947114576027513230189734205586617139 2

So Nabokov did get it right after all. Now let us see if Eigenmath can find the seventeenth root of this number, like the Hindu calculator could.

```
N = 212^17
N^(1/17)
```

Eigenmath displays the following result.

212

When a symbol is assigned a value, such as N above, no result is printed. To see the value of a symbol, just evaluate it.

```
N
```

$N = 352947114576027513230189734205586617139 2$

The previous example shows a convention that will be used throughout this manual. That is, the color blue indicates something that the user should type. The computer response is shown in black.

1.1 Syntax

The following table summarizes the various operators and expression syntax. The arithmetic operators have the expected precedence, that is, multiplication and division are evaluated before add and subtract. Subexpressions surrounded by 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, alternatively,* `a*b`
$\frac{a}{b}$	`a/b`	*division*
$\frac{a}{bc}$	`a/b/c`	*division operator is left-associative*
a^2	`a^2`	*power*
$\sqrt{a}$	`sqrt(a)`	*square root, alternatively,* `a^(1/2)`
$a(b + c)$	`a (b+c)`	*note the space in between, alternatively,* `a*(b+c)`
$f(a)$	`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*
π	`pi`	–
e	`exp(1)`	*natural number*

1.2 Testing for equality

The infix operator == is used to test for equality of operands. The operator evaluates to 1 if the operands are equal and 0 if the operands are not equal.

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

1

Note: Equality tests involving floating point numbers can be problematic due to roundoff error.

1.3 Arithmetic

Normally Eigenmath uses integer and rational number arithmetic.

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

$\frac{5}{6}$

A floating point value causes Eigenmath to switch to floating point arithmetic.

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

0.833333

An integer or rational number result can be converted to a floating point value by entering *float*.

```
212^17
```

352947114576027513230189734205866171392

```
float
```

3.52947×10^{39}

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

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

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

1.4 Exponents

Eigenmath requires parentheses around negative exponents. For example,

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

instead of

```
10^-3
```

The reason for this is that the binding of the negative sign is not always obvious. For example, consider

```
x^-1/2
```

It is not clear whether the exponent should be -1 or $-1/2$. So Eigenmath requires

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

which is unambiguous.

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)
```

$x^{1/2}$

1.5 Symbols

As we saw earlier, symbols are defined using 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 Eigenmath scans the entire symbol to find Greek letters.

```
alphamunu = 1
alphamunu
```

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

When a symbolic chain is defined, Eigenmath follows the chain as far as possible. 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

1.6 User defined functions

Most of the functions commonly used in math and physics are included in Eigenmath. See the Reference section at the end of the manual for a complete list. There is also a facility for the user to define additional functions.

User functions are defined using the syntax *function-name* (*arg-list*) = *expr* where *arg-list* is a comma separated list of zero to nine symbols that receive arguments. Unlike symbol definitions, *expr* is not evaluated when *function-name* is defined. Instead, *expr* is evaluated when *function-name* is used in a subsequent computation.

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

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

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

After a user function is defined, *expr* can be recalled using the *binding* function.

```
binding(f)
```

$$\frac{\sin(x)}{x}$$

If local symbols are needed in a function, they can be appended to *arg-list*. (The caller does not have to supply all the arguments.) The following example uses Rodrigues's formula to compute an associated Legendre function of $\cos\theta$.

$$P_n^m(x) = \frac{1}{2^n\,n!}(1-x^2)^{m/2}\frac{d^{n+m}}{dx^{n+m}}(x^2-1)^n$$

Function P below first computes $P_n^m(x)$ for local variable x and then uses *eval* to replace x with f. In this case, $f = \cos\theta$.

```
x = 123 -- global x in use, need local x in P
P(f,n,m,x) = eval(1/(2^n n!) (1 - x^2)^(m/2) d((x^2 - 1)^n,x,n + m),x,f)
P(cos(theta),2,0) -- arguments f, n, m, but not x
```

$\frac{3}{2}\cos(\theta)^2 - \frac{1}{2}$

The scope of function arguments is limited to the function definition.

1.7 Scripts

Scripting is a way of automatically running a sequence of calculations. A script is entered in the left-hand field of the Eigenmath window.

To create a script, enter one calculation per line in the script field. Nothing happens until the Run button is clicked. When the Run button is clicked, Eigenmath evaluates the script line by line. After a script runs, all of its symbols are available for immediate mode calculation. Scripts can be saved and loaded using the File menu.

Here is an example script that can be pasted into the script field and then run by clicking the Run button.

```
"Solve for vector X in AX = B"
A = ((1,2),(3,4))
B = (5,6)
X = dot(inv(A),B)
X
```

After clicking the Run button, the following result is displayed.

```
Solve for vector X in AX = B
```

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

A handy debugging aid is to include the line $trace = 1$ in the script. When $trace = 1$ each line of the script is displayed as it is evaluated. For example, here is the previous script with the addition of $trace = 1$.

```
"Solve for vector X in AX = B"
trace = 1
A = ((1,2),(3,4))
B = (5,6)
X = dot(inv(A),B)
X
```

The result is

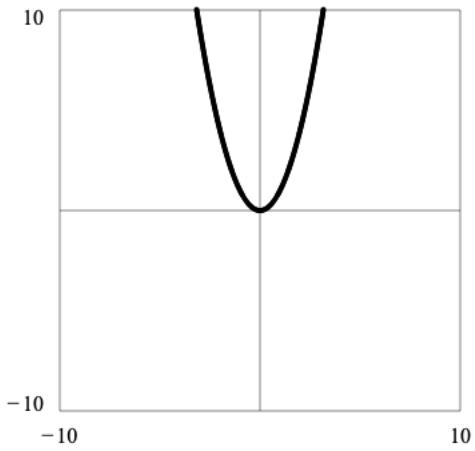
```
Solve for vector X in AX = B
A = ((1,2),(3,4))
B = (5,6)
X = dot(inv(A),B)
X
```

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

1.8 Draw

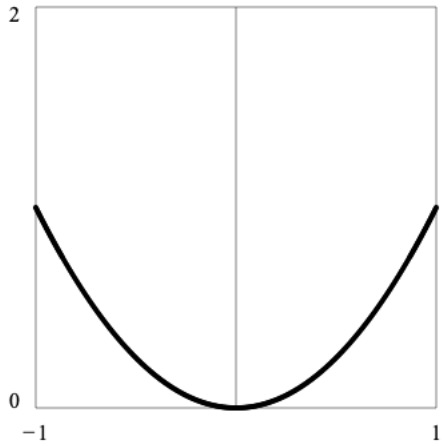
$draw(f, x)$ draws a graph of function f of x. (The default second argument is x.)

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



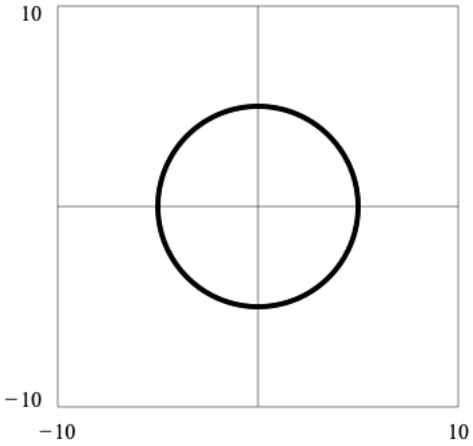
The vectors *xrange* and *yrange* control the scale of the graph.

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



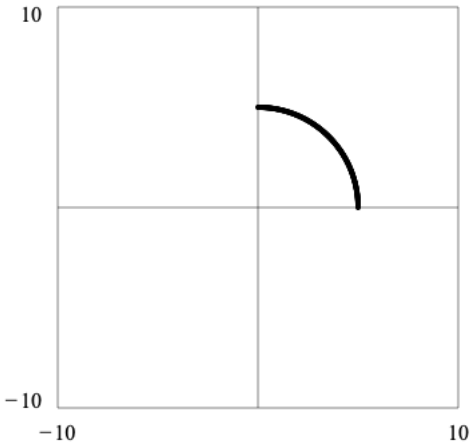
Parametric drawing occurs when a function returns a vector. The vector *trange* controls the parametric range. The default is $trange = (-\pi, \pi)$. In the following example, *draw* varies *theta* over the default range $-\pi$ to $+\pi$.

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



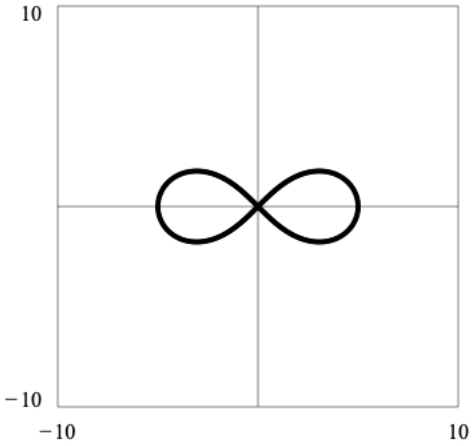
In the following example, *trange* is reduced to draw a quarter circle instead of a full circle.

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



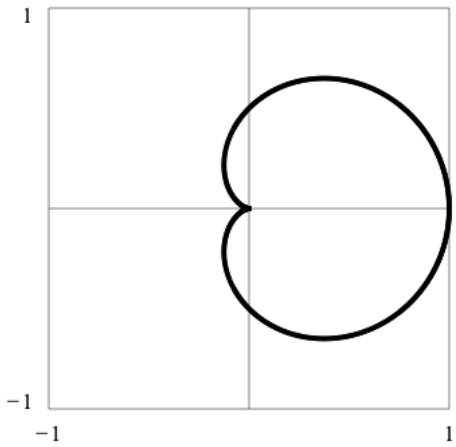
Lemniscate.

```
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)
```



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)
```



1.9 Complex numbers

When Eigenmath starts up, it defines symbol i as $i = \sqrt{-1}$. Symbol i can be redefined and used for some other purpose if need be.

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)
```

$2j$

1.10 Linear algebra

The *dot* function is used to multiply tensors. For example, let

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \text{and} \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

The product Ax is computed as follows.

```
A = ((1,2),(3,4))
x = (x1,x2)
dot(A,x)
```

$$\begin{bmatrix} x_1 + 2x_2 \\ 3x_1 + 4x_2 \end{bmatrix}$$

The following example shows how to use *dot* and *inv* to solve for the vector X in $AX = B$.

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

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

The *dot* function can have more than two arguments. For example, $dot(A, B, C)$ can be used for the dot product of three tensors.

Square brackets are used for component access. Index numbering starts with 1.

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

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

The following example demonstrates the relation $A^{-1} = \frac{\text{adj}\, A}{\det A}$.

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

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

```
adj(A)
```

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

```
det(A)
```

$ad - bc$

```
inv(A) - adj(A)/det(A)
```

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

Sometimes a calculation will be simpler if it can be reorganized to use *adj* instead of *inv*. The main idea is to try to prevent the determinant from appearing as a divisor. For example, suppose for matrices A and B you want to check that

$$A - B^{-1} = 0$$

Depending on the complexity of $\det B$, the software may not be able to find a simplification that yields zero. Should that occur, the following alternative formulation can be tried.

$$A \det B - \operatorname{adj} B = 0$$

2 Calculus

2.1 Derivative

$d(f,x)$ returns the derivative of f with respect to x. The x can be omitted for expressions in x.

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

$2x$

The following table summarizes the various ways to obtain multi-derivatives.

$\frac{\partial^2 f}{\partial x^2}$	`d(f,x,x)`	`d(f,x,2)`
$\frac{\partial^2 f}{\partial x\,\partial y}$	`d(f,x,y)`	
$\frac{\partial^{m+n+\cdots} f}{\partial x^m\,\partial y^n\cdots}$	`d(f,x,...,y,...)`	`d(f,x,m,y,n,...)`

2.2 Gradient

The gradient of f is obtained by using a vector for 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 tensor function. Gradient raises the rank by one.

```
F = (x + 2 y,3 x + 4 y)
X = (x,y)
d(F,X)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

2.3 Template functions

The function f in $d(f)$ does not have to be defined. It can be a template function with just a name and an argument list. Eigenmath checks the argument list to figure out what to do. 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.

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

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

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

0

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

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

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

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

As the final example shows, an empty argument list causes $d(f)$ to always evaluate to itself, regardless of the second argument.

Template functions are useful for experimenting with differential forms. For example, let us check the identity

$$\mathrm{div}(\mathrm{curl}\, F) = 0$$

for an arbitrary vector function F.

```
F = (F1(x,y,z),F2(x,y,z),F3(x,y,z))
curl(U) = (d(U[3],y) - d(U[2],z),d(U[1],z) - d(U[3],x),d(U[2],x) - d(U[1],y))
div(U) = d(U[1],x) + d(U[2],y) + d(U[3],z)
div(curl(F))
```

0

2.4 Integral

$integral(f, x)$ returns the integral of f with respect to x. The x can be omitted for expressions in x. The argument list can be extended for multiple integrals.

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

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

```
integral(x y,x,y)
```

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

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

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

As an alternative, the *eval* function 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. Difficult integrals involving sine and cosine can often be solved by using exponentials. Trigonometric simplifications involving powers and multiple angles turn into simple algebra in the exponential domain. 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 = circexp(f)
defint(f,t,0,2pi)
```

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

Here is a check of the result.

```
g = integral(f,t)
f - d(g,t)
```

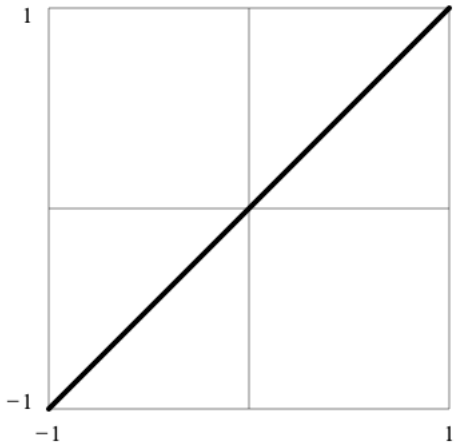
0

The fundamental theorem of calculus is a formal expression of the inverse relation between integrals and derivatives.

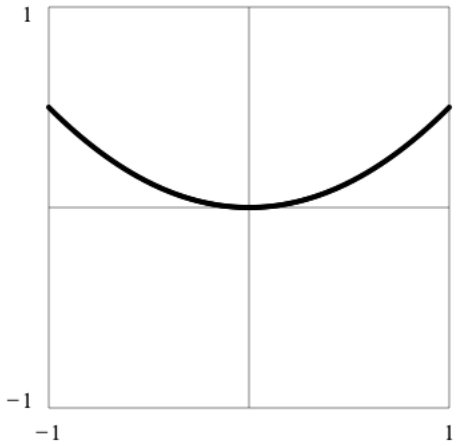
$$\int_a^b f'(x)\,dx = f(b) - f(a)$$

Here is an Eigenmath demonstration of the fundamental theorem of calculus.

```
xrange = (-1,1)
yrange = (-1,1)
f = d(x^2/2)
draw(f,x)
```



```
xrange = (-1,1)
yrange = (-1,1)
f = integral(d(x^2/2))
draw(f,x)
```



The first graph shows that $f'(x)$ is antisymmetric, therefore the total area under the curve from -1 to 1 sums to zero. The second graph shows that $f(1) = f(-1)$. Hence for $f(x) = \frac{1}{2}x^2$ we have

$$\int_{-1}^{1} f'(x)\,dx = f(1) - f(-1) = 0$$

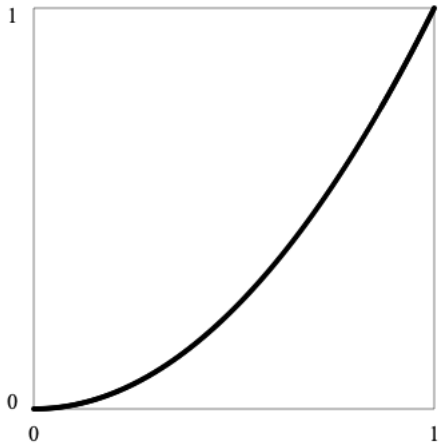
2.5 Arc length

Let $g(t)$ be a function that draws a curve. 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 $g(t)$. The integral sums over all of the tangent lengths to arrive at the total length from a to b. For example, let us measure the length of the following curve.

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



A suitable $g(t)$ for the arc is

$$g(t) = (t, t^2), \quad 0 \le t \le 1$$

Hence one Eigenmath solution for computing the arc length is

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

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

```
float
```

1.47894

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

The result seems rather complicated given that we started with a simple parabola. Let us inspect $|g'(t)|$ to see why.

```
g
```

$$g = \begin{bmatrix} t \\ t^2 \end{bmatrix}$$

```
d(g,t)
```

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

```
abs(d(g,t))
```

$(4t^2 + 1)^{1/2}$

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.

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

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

$\frac{56}{27}$

Because of the way t is substituted for x, the following code yields the same result.

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

$\frac{56}{27}$

2.6 Line integrals

There are two different 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	$\int_C (F \cdot u) \, ds$	$\int_a^b F(g(t)) \cdot g'(t) \, dt$

For the vector field form, the symbol u is the unit tangent vector

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

The length of the tangent vector cancels with ds as follows.

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

Evaluate

$$\int_C x\, ds \quad \text{and} \quad \int_C x\, dx$$

where C is a straight line from $(0,0)$ to $(1,1)$.

What a difference the measure makes. The first integral is over a scalar field and the second is over a vector field. This can be understood when we recall that

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

Hence for $\int_C x\, ds$ we have

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

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

For $\int_C x\, dx$ we have

```
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.

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

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

$\frac{8}{3}$

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 = 2t+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)
```

$\frac{163}{30}$

2.7 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

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

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

```
M1 = ((0,0,0),(0,0,-1),(0,1,0))
M2 = ((0,0,1),(0,0,0),(-1,0,0))
M3 = ((0,-1,0),(1,0,0),(0,0,0))
M = (M1,M2,M3)
cross(u,v) = dot(u,M,v)
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)
```

π

The result is π, 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.

```
M1 = ((0,0,0),(0,0,-1),(0,1,0))
M2 = ((0,0,1),(0,0,0),(-1,0,0))
M3 = ((0,-1,0),(1,0,0),(0,0,0))
M = (M1,M2,M3)
cross(u,v) = dot(u,M,v)
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

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

```
M1 = ((0,0,0),(0,0,-1),(0,1,0))
M2 = ((0,0,1),(0,0,0),(-1,0,0))
M3 = ((0,-1,0),(1,0,0),(0,0,0))
M = (M1,M2,M3)
cross(u,v) = dot(u,M,v)
x = u cos(v)
y = u sin(v)
z = v
S = (x,y,z)
a = circexp(abs(cross(d(S,u),d(S,v))))
defint(a,u,0,1,v,0,3pi)
```

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

```
float
```

10.8177

2.8 Surface integrals

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

```
M1 = ((0,0,0),(0,0,-1),(0,1,0))
M2 = ((0,0,1),(0,0,0),(-1,0,0))
M3 = ((0,-1,0),(1,0,0),(0,0,0))
M = (M1,M2,M3)
cross(u,v) = dot(u,M,v)
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)
```

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

2.9 Green's theorem

Green's theorem tells us that

$$\oint P\,dx + Q\,dy = \iint \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dx\,dy$$

In other words, a line integral and a surface integral can yield the same result.

Example 1. The following exercise is from *Advanced Calculus* by Wilfred Kaplan, p. 287. Evaluate $\oint (2x^3 - y^3)\,dx + (x^3 + y^3)\,dy$ around the circle $x^2 + y^2 = 1$ using Green's theorem.

It turns out that Eigenmath cannot solve the double integral over x and y directly. Polar coordinates are used instead.

```
P = 2x^3 - y^3
Q = x^3 + y^3
f = d(Q,x) - d(P,y)
x = r cos(theta)
y = r sin(theta)
defint(f r,r,0,1,theta,0,2pi)
```

$\frac{3}{2}\pi$

The *defint* integrand is $f\,r$ because $r\,dr\,d\theta = dx\,dy$.

Now let us try computing the line integral side of Green's theorem and see if we get the same result. We need to use the trick of converting sine and cosine to exponentials so that Eigenmath can find a solution.

```
x = cos(t)
y = sin(t)
P = 2x^3 - y^3
Q = x^3 + y^3
f = P d(x,t) + Q d(y,t)
f = circexp(f)
defint(f,t,0,2pi)
```

$\frac{3}{2}\pi$

Example 2. Compute both sides of Green's theorem for $F = (1 - y, x)$ over the disk $x^2 + y^2 \leq 4$.

First compute the line integral along the boundary of the disk. Note that the radius of the disk is 2.

```
-- Line integral
P = 1 - y
Q = x
x = 2 cos(t)
y = 2 sin(t)
defint(P d(x,t) + Q d(y,t),t,0,2pi)
```

8π

```
-- Surface integral
x = quote(x) --clear x
y = quote(y) --clear y
h = sqrt(4-x^2)
defint(d(Q,x) - d(P,y),y,-h,h,x,-2,2)
```

8π

```
-- Try computing the surface integral using polar coordinates.
f = d(Q,x) - d(P,y) -- do before change of coordinates
x = r cos(theta)
y = r sin(theta)
defint(f r,r,0,2,theta,0,2pi)
```

8π

```
defint(f r,theta,0,2pi,r,0,2) -- try integrating over theta first
```

8π

In this case, Eigenmath solved both forms of the polar integral. However, in cases where Eigenmath fails to solve a double integral, try changing the order of integration.

2.10 Stokes' theorem

Stokes' theorem says that in typical problems a surface integral can be computed using a line integral. (There is some fine print regarding continuity and boundary conditions.) This is a useful theorem because usually the line integral is easier to compute. In rectangular coordinates the equivalence between a line integral on the left and a surface integral on the right is

$$\oint P\,dx + Q\,dy + R\,dz = \iint_S (\operatorname{curl} \mathbf{F}) \cdot \mathbf{n}\,d\sigma$$

where $\mathbf{F} = (P, Q, R)$. For S parametrized by x and y we have

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

Example: Let $\mathbf{F} = (y, z, x)$ and let S be the part of the paraboloid $z = 4 - x^2 - y^2$ that is above the xy plane. The perimeter of the paraboloid is the circle $x^2 + y^2 = 2$. The following script computes both the line and surface integrals. It turns out that we need to use polar coordinates for the line integral so that *defint* can succeed.

```
-- eigenmath.org/stokes-theorem.txt
"Surface integral"
z = 4 - x^2 - y^2
F = (y,z,x)
S = (x,y,z)
f = dot(curl(F),cross(d(S,x),d(S,y)))
x = r cos(theta)
y = r sin(theta)
defint(f r,r,0,2,theta,0,2pi)
"Line integral"
x = 2 cos(t)
y = 2 sin(t)
z = 4 - x^2 - y^2
P = y
Q = z
R = x
f = P d(x,t) + Q d(y,t) + R d(z,t)
f = circexp(f)
defint(f,t,0,2pi)
```

This is the result when the script runs. Both the surface integral and the line integral yield the same result.

Surface integral
-4π
Line integral
-4π

3 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. 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 ψ_f be the final state of the quantum computer after all the rotations have been applied. Like any other state, ψ_f is a linear combination of eigenstates.

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

The last step is to measure ψ_f and get a result. Measurement rotates ψ_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 ψ_f is already an eigenstate then no rotation occurs. (The probability of rotating to 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 ψ_f.

Unlike a real quantum computer, in a simulation the final state ψ_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

$$P = \psi_f \odot \psi_f^*$$

where operator $\odot$ indicates a Hadamard (element-wise) product. The result P is a vector with component P_j 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 eigenstate $|0\rangle$, 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, ϕ	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.)

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.

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 ψ.

```
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. 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 $N = 16$

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

Example.

```
-- Verify the following truth table for cnot

-- Input Output
--    00     00
--    01     11
--    10     10
--    11     01

U(psi) = rotate(psi,C,0,X,1) -- cnot

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
```

See “Quantum Computing” at eigenmath.org for more examples.

4 Function Reference

abs(x)

Returns the absolute value or vector length of x.

```
X = (x,y,z)
abs(X)
```

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

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($a, b, \ldots$)

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)
```

$(x + 1)^2$

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

circexp(x)

Returns expression x with circular and hyperbolic functions converted to exponentials.

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

$\exp(ix)$

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)

Returns tensor a summed over indices i and j. If i and j are omitted then 1 and 2 are used. The expression `contract(m)` computes the trace of matrix m.

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

```
circexp(cosh(x))
```

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

d(f, x)

Returns the partial derivative of f with respect to x.

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

$2x$

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)
```

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

It is OK to use `d` as a variable name. It will not conflict with function `d`.

It is OK to redefine `d` as a different function. The function `derivative`, a synonym for `d`, can still be used to obtain a partial derivative.

defint(f, x, a, b)

Returns the definite integral of f with respect to x evaluated from a to b. The argument list can be extended for multiple integrals as shown in the following example.

```
f = (1 + cos(theta)^2) sin(theta)
defint(f, theta, 0, pi, phi, 0, 2pi) -- integrate over theta then over phi
```

$\frac{16}{3}\pi$

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 nth index of tensor a. Index numbering starts with 1.

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

3

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

```
-- solve for X in AX=B
A = ((1,2),(3,4))
B = (5,6)
X = dot(inv(A),B)
X
```

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

draw(f, x)

Draws a graph of $f(x)$. Drawing ranges can be set with `xrange` and `yrange`.

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

eval(f, x, a)

Returns expression f evaluated at x equals a. The argument list can be extended for multivariate expressions. For example, `eval(f,x,a,y,b)` is equivalent to `eval(eval(f,x,a),y,b)`.

```
eval(x + y,x,a,y,b)
```

$a + b$

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)
```

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

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)
```

3.52947×10^{39}

floor(x)

Returns the largest integer less than or equal to x.

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

0

for($i, j, k, a, b, \ldots$)

For i equals j through k evaluate a, b, etc.

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

$A = 1$
$A = 2$
$A = 3$

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

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 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

inner($a, b, \ldots$)

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$(a, b, \ldots)$

Returns the Kronecker product of vectors and matrices.

```
A = ((1,2),(3,4))
B = ((a,b),(c,d))
kronecker(A,B)
```

$$\begin{bmatrix} a & b & 2a & 2b \\ c & d & 2c & 2d \\ 3a & 3b & 4a & 4b \\ 3c & 3d & 3c & 4d \end{bmatrix}$$

last

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

```
212^17
```

352947114576027513230189734205866171392

```
last
```

$last = 352947114576027513230189734205866171392$

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

```
float
```

3.52947×10^{39}

log(x)

Returns the natural logarithm of x.

```
log(x^y)
```

$y \log(x)$

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

numerator(x)

Returns the numerator of expression x.

```
numerator(a/b)
```

a

or$(a, b, \ldots)$

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

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

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 π.

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

Evaluate expressions 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)
```

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

Note: 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`.

quote(x)

Returns expression x without evaluating it first.

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

$(x + 1)^2$

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))
```

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

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 section 3 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(*file*)

Run script *file*. Useful for importing function libraries.

```
run("Downloads/EVA.txt")
```

Note: *file* must be in the Downloads folder due to security requirements for apps distributed on the Mac App Store.

simplify(x)

Returns expression x in a simpler form.

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

1

sin(x)

Returns the sine of x.

```
sin(pi/4)
```

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

sinh(x)

Returns the hyperbolic sine of x.

```
circexp(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(i, j, k, f)

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

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

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

Note: 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`.

tan(x)

Returns the tangent of x.

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

0

tanh(x)

Returns the hyperbolic tangent of x.

```
circexp(tanh(x))
```

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

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. 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

trace

Set `trace=1` in a script to print the script as it is evaluated. Useful for debugging.

```
trace = 1
```

Note: The `contract` function is used to obtain the trace of a matrix.

transpose(a, i, j)

Returns the transpose of tensor a with respect to indices i and j. If i and j are omitted then 1 and 2 are used. Hence a matrix can be transposed with a single argument.

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

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

Note: The argument list can be extended for multiple transpose operations. The 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)`

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$(i, j, \ldots)$

Returns a null tensor with dimensions i, j, etc. Useful for creating a tensor and then setting component values.

```
A = zero(3,3)
for(k,1,3,A[k,k]=k)
A
```

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

5 Tricks

1. In the result display, do click-drag-release to copy a selection of the display to the clipboard.

2. In a script, line breaking is allowed provided the line breaks occur immediately after operators. 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. This is useful for debugging.

4. The last result is stored in the 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 π. Set `pi=float(pi)` to evaluate expressions with a numerical value for π. Set `pi=quote(pi)` to make π symbolic again.

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

10. Use `expsin` and `expcos` instead of `sin` and `cos`. Trigonometric simplifications occur automatically when exponentials are used.

11. Use `A==B` or `A-B==0` to test for equality of A and B. The equality operator `==` uses a cross multiply algorithm to eliminate denominators. Hence `==` can typically determine equality even when the unsimplified result of $A-B$ is nonzero. Note: Equality tests involving floating point numbers can be problematic due to roundoff error.

12. If local symbols are needed in a function, they can be appended to *arg-list*. (The caller does not have to supply all the arguments.) The following example uses Rodrigues's formula to compute an associated Legendre function of $\cos\theta$.

$$P_n^m(x) = \frac{1}{2^n\, n!}(1-x^2)^{m/2}\frac{d^{n+m}}{dx^{n+m}}(x^2-1)^n$$

Function P below first computes $P_n^m(x)$ for local variable x and then uses *eval* to replace x with f. In this case, $f = \cos\theta$.

```
x = 123 -- global x in use, need local x in P
P(f,n,m,x) = eval(1/(2^n n!) (1 - x^2)^(m/2) d((x^2 - 1)^n,x,n + m),x,f)
P(cos(theta),2,0) -- arguments f, n, m, but not x
```

$\frac{3}{2}\cos(\theta)^2 - \frac{1}{2}$

Note: The maximum number of arguments is nine.