Eigenmath Manual

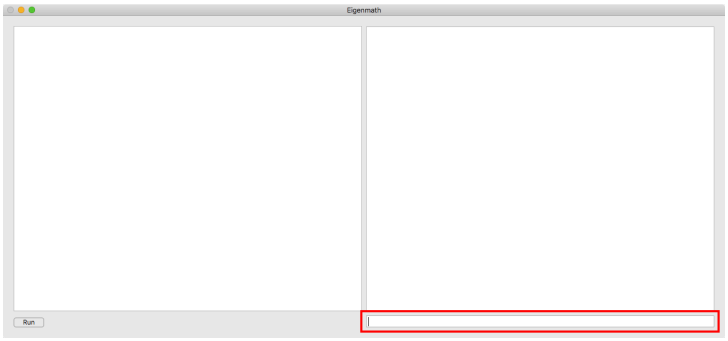
George Weigt

December 13, 2015

Contents

1 Introduction

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



For example, let us 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, 352947114576027513230189734205 5866171392 (I am not sure I have got this right; anyway the root was 212).

We can check Nabokov's arithmetic by entering the following calculation.

```
212^17
```

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

352947114576027513230189734205 5866171392

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
```

$N = 352947114576027513230189734205 5866171392$

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 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
```

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

1.2 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 expresssion. 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)
```

$x^{1/2}$

1.3 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
```

$$A$$

1.4 User-defined functions

The following example shows a user-defined function with a single argument.

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

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

The following example defines a function with two arguments.

```
g(x,y) = abs(x) + abs(y)
g(1,-2)
```

$$3$$

User-defined functions can be evaluated without an argument list. The binding of the function name is returned when there is no argument list.

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

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

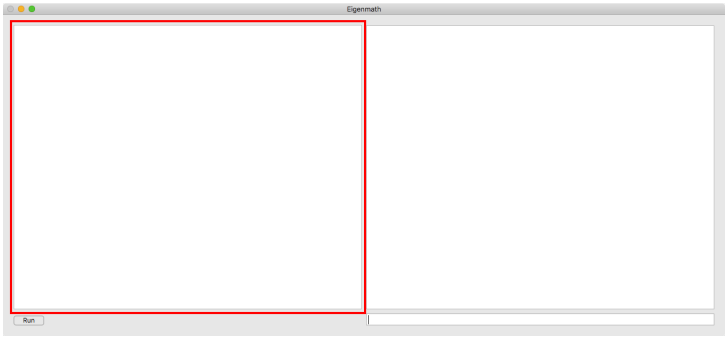
Normally a function body is not evaluated when a function is defined. However, in some cases it is required that the function body be the result of something. The *eval* function is used to accomplish this. For example, the following code causes the function body to be a sixth order Taylor series expansion of $\cos x$.

```
f(x) = eval(taylor(cos(x),x,6))
f
```

$$f = -\frac{1}{720}x^6 + \frac{1}{24}x^4 - \frac{1}{2}x^2 + 1$$

1.5 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

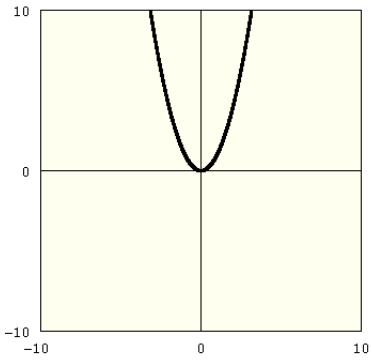
```
Solve for vector X in AX = B
A = ((1,2),(3,4))
B = (5,6)
X = dot(inv(A),B)
X
```

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

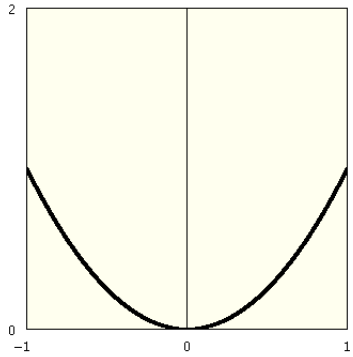
1.6 Draw

$draw(f, x)$ draws a graph of the function f of x. The second argument can be omitted when the dependent variable is literally x or t. The vectors $xrange$ and $yrange$ control the scale of the graph.

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

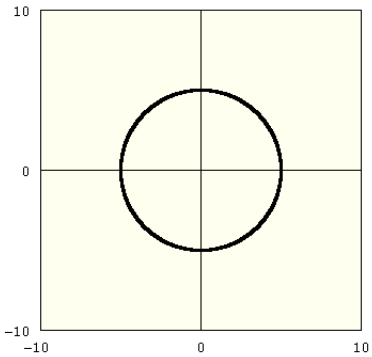



```
xrange = (-1,1)
yrange = (0,2)
draw(x^2)
```



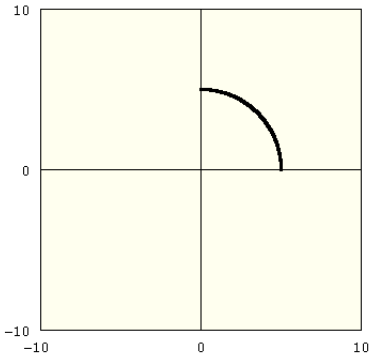
Parametric drawing occurs when a function returns a vector. The vector $trange$ controls the parametric range. The default is $trange = (-\pi, \pi)$. In the following example, $draw$ varies $theta$ over the default range $-\pi$ to $+\pi$.

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



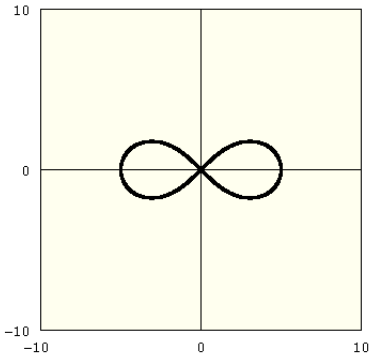
In the following example, *trange* is reduced to draw a quarter circle instead of a full circle.

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



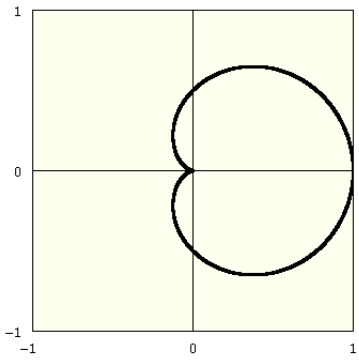
Here are a couple of interesting curves and the code for drawing them. First is a lemniscate.

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



Next is a cardioid.

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



1.7 Complex numbers

When Eigenmath starts up, it defines the symbol i as $i = \sqrt{-1}$. Other than that, there is nothing special about i. It is just a regular symbol that 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(i*pi/3)
```

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

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

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

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

```
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.

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

Of course, this causes i to be removed from the denominator. Here are a few examples.

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

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

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

$$-1 + i$$

The absolute value of a complex number returns its magnitude.

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

5

Since symbols can have complex values, the absolute value of a symbolic expression is not computed.

```
abs(a+b*i)
```

$\text{abs}(a + ib)$

The result is not $\sqrt{a^2 + b^2}$ because that would assume that a and b are real. For example, suppose that $a = 0$ and $b = i$. Then

$$|a + ib| = |-1| = 1$$

and

$$\sqrt{a^2 + b^2} = \sqrt{-1} = i$$

Hence

$$|a + ib| \neq \sqrt{a^2 + b^2} \quad \text{for some } a, b \in \mathbb{C}$$

The *mag* function can be used instead of *abs*. It treats symbols like a and b as real.

```
mag(a+b*i)
```

$(a^2 + b^2)^{1/2}$

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

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

$2j$

1.8 Linear algebra

The function *dot* is used to multiply vectors and matrices. Let

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

The following example computes Ax.

```
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 $A^{-1} = \operatorname{adj} A / \det A$.

```
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.

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

The adjunct of a matrix is related to the cofactors as follows.

```
A = ((a,b),(c,d))
C = ((0,0),(0,0))
C[1,1] = cofactor(A,1,1)
C[1,2] = cofactor(A,1,2)
C[2,1] = cofactor(A,2,1)
C[2,2] = cofactor(A,2,2)
C
```

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

```
adj(A) - transpose(C)
```

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

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 multiderivatives.

$\dfrac{\partial^2 f}{\partial x^2}$	`d(f,x,x)`	`d(f,x,2)`
$\dfrac{\partial^2 f}{\partial x\,\partial y}$	`d(f,x,y)`	
$\dfrac{\partial^{m+n+\cdots} f}{\partial x^m\,\partial y^n \cdots}$	`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+2y,3x+4y)
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,0,2*pi)
```

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

Here is a check of the result.

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

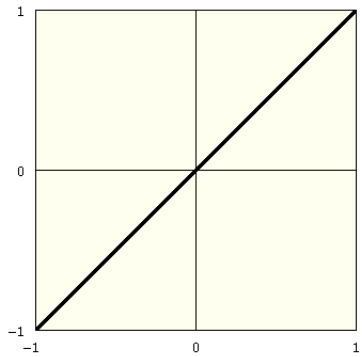
0

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

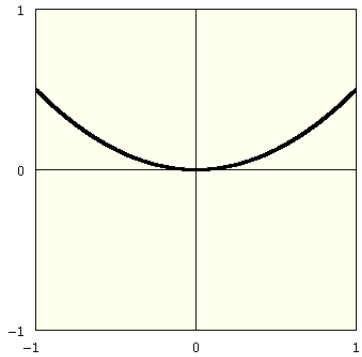
$$\int_a^b f'(x)\,dx = f(b) - f(a)$$

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

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




```
draw(integral(d(f)))
```



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

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

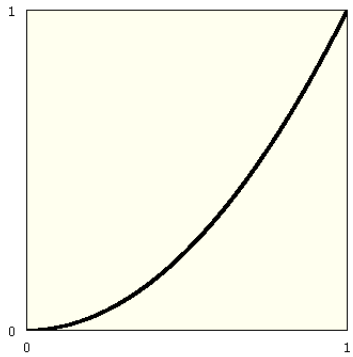
2.5 Arc length

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)
```

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

```
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(t) = abs(g(t)-g(t-0.01))
L = 0
for(k,1,100,L=L+h(k/100.0))
L
```

$L = 1.47894$

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.

```
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.

```
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)
```

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

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

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

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

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

```
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.

```
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.

```
-- 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.

-4π

-4π

3 Examples

A tarball of examples can be downloaded from http://eigenmath.sourceforge.net/eigenmath-examples.tgz

3.1 François Viète

François Viète was the first to discover an exact formula for π. Here is his formula.

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \times \frac{\sqrt{2+\sqrt{2}}}{2} \times \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \times \cdots$$

Let $a_0 = 0$ and $a_n = \sqrt{2 + a_{n-1}}$. Then we can write

$$\frac{2}{\pi} = \frac{a_1}{2} \times \frac{a_2}{2} \times \frac{a_3}{2} \times \cdots$$

Solving for π we have

$$\pi = 2 \times \frac{2}{a_1} \times \frac{2}{a_2} \times \frac{2}{a_3} \times \cdots = 2 \prod_{k=1}^{\infty} \frac{2}{a_k}$$

Let us now use Eigenmath to compute π according to Viète's formula. Of course, we cannot calculate all the way out to infinity, we have to stop somewhere. It turns out that nine factors are just enough to get six digits of accuracy.

```
a(n)=test(n=0,0,sqrt(2+a(n-1)))
float(2*product(k,1,9,2/a(k)))
```

3.14159

The function $a(n)$ calls itself n times so overall there are 54 calls to $a(n)$. By using a different algorithm with temporary variables, we can get the answer in just nine steps.

```
a = 0
b = 2
for(k,1,9,a=sqrt(2+a),b=b*2/a)
float(b)
```

3.14159

3.2 Curl in tensor form

The curl of a vector function can be expressed in tensor form as

$$\operatorname{curl} \mathbf{F} = \epsilon_{ijk} \frac{\partial F_k}{\partial x_j}$$

where ϵ_{ijk} is the Levi-Civita tensor. The following script demonstrates that this formula is equivalent to computing curl the old fashioned way.

```
-- Define epsilon
epsilon = zero(3,3,3)
epsilon[1,2,3] = 1
epsilon[2,3,1] = 1
epsilon[3,1,2] = 1
epsilon[3,2,1] = -1
epsilon[1,3,2] = -1
epsilon[2,1,3] = -1
-- F is a generic vector function
F = (FX(),FY(),FZ())
-- A is the curl of F
A = outer(epsilon,d(F,(x,y,z)))
A = contract(A,3,4) --sum across k
A = contract(A,2,3) --sum across j
-- B is the curl of F computed the old fashioned way
BX = d(F[3],y)-d(F[2],z)
BY = d(F[1],z)-d(F[3],x)
BZ = d(F[2],x)-d(F[1],y)
B = (BX,BY,BZ)
-- Are A and B equal? Subtract to find out.
A-B
```

Here is the result when the script runs.

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

The following is a variation on the previous script. The product $\epsilon_{ijk}\,\partial F_k/\partial x_j$ is computed in just one line of code. In addition, the outer product and the contraction across k are now computed with a dot product.

```
F = (FX(),FY(),FZ())
epsilon = zero(3,3,3)
epsilon[1,2,3] = 1
epsilon[2,3,1] = 1
epsilon[3,1,2] = 1
epsilon[3,2,1] = -1
epsilon[1,3,2] = -1
epsilon[2,1,3] = -1
A = contract(dot(epsilon,d(F,(x,y,z))),2,3)
BX = d(F[3],y)-d(F[2],z)
BY = d(F[1],z)-d(F[3],x)
BZ = d(F[2],x)-d(F[1],y)
B = (BX,BY,BZ)
-- Are A and B equal? Subtract to find out.
A-B
```

This is the result when the script runs.

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

3.3 Quantum harmonic oscillator

For total energy E, kinetic energy K and potential energy V we have

$$E = K + V$$

The corresponding formula for a quantum harmonic oscillator is

$$(2n+1)\psi = -\frac{d^2\psi}{dx^2} + x^2\psi$$

where n is an integer and represents the quantization of energy values. The solution to the above equation is

$$\psi_n(x) = \exp(-x^2/2)H_n(x)$$

where $H_n(x)$ is the nth Hermite polynomial in x. The following Eigenmath code checks $E = K + V$ for $n = 7$.

```
n = 7
psi = exp(-x^2/2)*hermite(x,n)
E = (2*n+1)*psi
K = -d(psi,x,x)
V = x^2*psi
E-K-V
```

0

3.4 Hydrogen wavefunctions

Hydrogen wavefunctions ψ are solutions to the differential equation

$$\frac{\psi}{n^2} = \nabla^2\psi + \frac{2\psi}{r}$$

where n is an integer representing the quantization of total energy and r is the radial distance of the electron. The Laplacian operator in spherical coordinates is

$$\nabla^2 = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{1}{r^2\sin\theta}\frac{\partial}{\partial\theta}\left(\sin\theta\frac{\partial}{\partial\theta}\right) + \frac{1}{r^2\sin^2\theta}\frac{\partial^2}{\partial\phi^2}$$

The general form of ψ is

$$\psi = r^l e^{-r/n} L_{n-l-1}^{2l+1}(2r/n) P_l^{|m|}(\cos\theta) e^{im\phi}$$

where L is a Laguerre polynomial, P is a Legendre polynomial and l and m are integers such that

$$1 \le l \le n-1, \qquad -l \le m \le l$$

The general form can be expressed as the product of a radial wavefunction R and a spherical harmonic Y.

$$\psi = RY, \qquad R = r^l e^{-r/n} L^{2l+1}_{n-l-1}(2r/n), \qquad Y = P_l^{|m|}(\cos\theta) e^{im\phi}$$

The following script checks $E = K + V$ for $n, l, m = 7, 3, 1$.

```
laplacian(f) = 1/r^2*d(r^2*d(f,r),r)+
  1/(r^2*sin(theta))*d(sin(theta)*d(f,theta),theta)+
  1/(r*sin(theta))^2*d(f,phi,phi)
n = 7
l = 3
m = 1
R = r^l*exp(-r/n)*laguerre(2*r/n,n-l-1,2*l+1)
Y = legendre(cos(theta),l,abs(m))*exp(i*m*phi)
psi = R*Y
E = psi/n^2
K = laplacian(psi)
V = 2*psi/r
simplify(E-K-V)
```

This is the result when the script runs.

0

3.5 Space shuttle and Corvette

The space shuttle accelerates from zero to 17,000 miles per hour in 8 minutes. A Corvette accelerates from zero to 60 miles per hour in 4.5 seconds. The following script compares the two.

```
vs = 17000*"mile"/"hr"
ts = 8*"min"/(60*"min"/"hr")
as = vs/ts
as
vc = 60*"mile"/"hr"
tc = 4.5*"sec"/(3600*"sec"/"hr")
ac = vc/tc
ac
"Time for Corvette to reach orbital velocity:"
vs/ac
vs/ac*60*"min"/"hr"
```

Here is the result when the script runs. It turns out that the space shuttle accelerates more than twice as fast as a Corvette.

$$a_s = \frac{127500 \text{ mile}}{(\text{hr})^2}$$

$$a_c = \frac{48000 \text{ mile}}{(\text{hr})^2}$$

```
Time for Corvette to reach orbital velocity:
```

0.354167 hr

21.25 min

3.6 Avogadro's constant

There is a proposal to define Avogadro's constant as exactly 84446886 to the third power. (Fox, Ronald and Theodore Hill. "An Exact Value for Avogadro's Number." *American Scientist* 95 (2007): 104–107.) The proposed number in the article is actually $(84446888)^3$. In a subsequent addendum the authors reduced it to 84446886^3 to make the number divisible by 12. (See `www.physorg.com/news109595312.html`.) This number corresponds to an ideal cube of atoms with 84,446,886 atoms along each edge. Let us check the difference between the proposed value and the measured value of $(6.0221415 \pm 0.0000010) \times 10^{23}$ atoms.

```
A = 84446886^3
B = 6.0221415*10^23
A-B
```

-5.17173×10^{16}

```
0.0000010*10^23
```

1×10^{17}

We see that the proposed value is within the experimental error. Just for the fun of it, let us factor the proposed value.

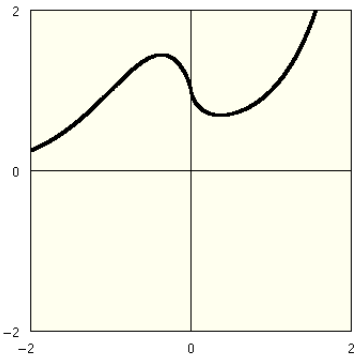
```
factor(A)
```

$2^3 \times 3^3 \times 1667^3 \times 8443^3$

3.7 Zero to the zero power

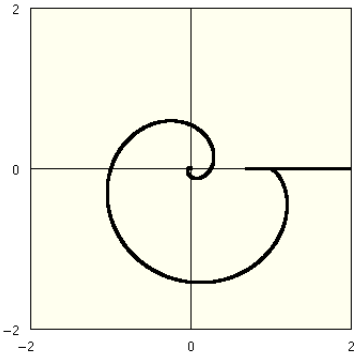
The following example draws a graph of the function $f(x) = |x^x|$. The graph shows why the convention $0^0 = 1$ makes sense.

```
f(x) = abs(x^x)
xrange = (-2,2)
yrange = (-2,2)
draw(f)
```



We can see how $0^0 = 1$ results in a continuous line through $x = 0$. Now let us see how x^x behaves in the complex plane.

```
f(t) = (real(t^t),imag(t^t))
xrange = (-2,2)
yrange = (-2,2)
trange = (-4,2)
draw(f)
```



3.8 Euler's identity

It is easy to "believe" that $e^{i\pi} = -1$ by looking at Taylor series expansions.

First, consider the Taylor series expansion of e^y.

$$e^y = 1 + y + \frac{y^2}{2!} + \frac{y^3}{3!} + \frac{y^4}{4!} + \frac{y^5}{5!} + \frac{y^6}{6!} + \frac{y^7}{7!} + \cdots$$

Next, substitute ix for y.

$$\begin{aligned} e^{ix} &= 1 + ix + \frac{(ix)^2}{2!} + \frac{(ix)^3}{3!} + \frac{(ix)^4}{4!} + \frac{(ix)^5}{5!} + \frac{(ix)^6}{6!} + \frac{(ix)^7}{7!} + \cdots \\ &= 1 + ix - \frac{x^2}{2!} - i\frac{x^3}{3!} + \frac{x^4}{4!} + i\frac{x^5}{5!} - \frac{x^6}{6!} - i\frac{x^7}{7!} + \cdots \end{aligned}$$

Next, collect the real and imaginary terms.

$$\begin{aligned} e^{ix} &= \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots\right) + i\left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots\right) \\ &= \cos x + i \sin x \end{aligned}$$

Finally, substitute π for x.

$$e^{i\pi} = \cos \pi + i \sin \pi = -1$$

The following script checks the identity $e^{ix} = \cos x + i \sin x$ for order n.

```
n = 7
E = taylor(e^y,y,n)
E = eval(E,y,i*x)
C = taylor(cos(x),x,n)
S = taylor(sin(x),x,n)
test(E=C+i*S,"true","false")
```

3.9 Penguin process

From *Quanta Magazine.*

> Most unexpected bumps in data go away as more data accumulate, just as you might get seven heads in your first 10 coin tosses only to end up with a 50-50 ratio after many more tosses. But after tripling their original sample size and analyzing approximately 2,400 of the rare penguin decays, the LHCb scientists say the anomaly hasn't diminished. Instead, it has lingered at an estimated statistical significance of "3.7 sigma" which means it is just as unlikely for such a large fluctuation to happen randomly as it would be to get 69 heads in 100 coin tosses. Physicists require a 5-sigma deviation from their expectations, equivalent to flipping 75 heads in 100 tosses (the odds of which are less than one in a million), to claim the discovery of a real effect.[1]

Recall that the binomial mass function with $p = 1/2$ is the probability of obtaining exactly k heads in n tosses.

$$f(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

In Eigenmath, define the binomial mass function and calculate the probability of getting exactly 75 heads in 100 tosses.

```
f(k) = choose(n,k)*p^k*(1-p)^(n-k)
n = 100
p = 1/2
f(75)
```

$$\frac{1515745435752107006346 9}{7922816251426433759354395033 6}$$

```
float
```

1.91314×10^{-7}

Hence the probability of getting exactly 75 heads in 100 tosses is indeed less than one in a million.

As the following equation shows, the variance for 100 coin tosses is 25.

$$\sigma^2 = np(1-p) = 100 \times \frac{1}{2} \times \frac{1}{2} = 25$$

Hence $\sigma = 5$. It follows that a 5-sigma deviation from the expected value is $50 + 5\sigma = 75$ and a 3.7-sigma deviation is $50 + 3.7\sigma = 68.5$.

Flipping 75 heads is a rare event, but flipping 25 or 76 is also rare. What we are really interested in is the probability of all the rare events. For this we turn to the cumulative distribution function. The cumulative distribution function sums over the probability mass function as follows.

$$P(X \le x) = F(x) = \sum_{k=0}^{x} f(k)$$

The cumulative distribution function computes the probability of all rare events as follows.

$$P(X \le 25) + P(X \ge 75) = F(25) + 1 - F(74)$$

In Eigenmath, define the cumulative distribution function and compute the probability of fewer than 26 or more than 74 heads in 100 tosses.

[1] Wolchover, Natalie. *'Penguin' Anomaly Hints at Missing Particles.* https://www.quantamagazine.org/20150320-penguin-anomaly-hints-at-missing-particles/

```
F(x) = sum(k,0,x,f(k))
F(25)+1-F(74)
```

$$\frac{8931045379645080593532 5}{1584563250285286751870879006 72}$$

```
float
```

5.63628×10^{-7}

Note that the probability is still less than one in a million.

Exercises

1. What is the probability of tossing exactly 50 heads?
2. Compute $E(1)$ and $E(X)$.
3. Compute $P(25 < X \leq 74)$.

The following script has all of the above calculations and answers to the exercises. It can be copied and pasted into the Eigenmath script window and then run by clicking the run button.

```
f(k) = choose(n,k)*p^k*(1-p)^(n-k)
n = 100
p = 1/2
"Probability of 75 heads"
f(75)
float
"Probability of fewer than 26 or more than 74 heads"
F(x) = sum(k,0,x,f(k))
F(25)+1-F(74)
float
"Answers"
float(f(50))
sum(x,0,100,f(x))
sum(x,0,100,x*f(x))
F(74)-F(25)
```

3.10 Conditional probability

From *The Week*.

> Suppose one out of every million people is a terrorist (if anything, an overestimate), and you've got a machine that can determine whether someone is a terrorist with 99.9 percent accuracy. You've used the machine on Mr. X, and it gives a positive result. What are the odds that Mr. X is a terrorist? Here's the answer: a 0.1 percent chance — which is to say, the 99.9 percent accurate test will give you the wrong answer 99.9 percent of the time.[2]

[2] Cooper, Ryan. *The simple math problem that blows apart the NSA's surveillance justifications.* http://theweek.com/articles/547119/simple-math-problem-that-blows-apart-nsas-surveillance-justifications

Let $X = 1$ be the event that Mr. X is a terrorist with probability one in a million.

$$P(X{=}1) = 0.000001$$
$$P(X{=}0) = 0.999999$$

Let $M = 1$ be the event that the machine gives a positive result. The accuracy of the machine is a conditional probability. Given the condition that Mr. X is a terrorist ($X = 1$), the probability that the machine gives a positive result ($M = 1$) is 99.9 percent.

$$P(M{=}1 \mid X{=}1) = 0.999$$

The trick is to swap M and X. In other words, determine the conditional probability of $X = 1$ given that $M = 1$. A conditional probability can be "turned around" using the following formula.

$$P(X{=}1 \mid M{=}\,1) = \frac{P(M{=}1 \mid X{=}1)P(X{=}1)}{P(M{=}1)}$$

The denominator $P(M{=}1)$ can be obtained from the law of total probability.

$$P(M{=}1) = P(M{=}1 \mid X{=}1)P(X{=}1) + P(M{=}1 \mid X{=}0)P(X{=}0)$$

The term $P(M{=}1 \mid X{=}0)$ is the probability of a false positive. Since the machine is 99.9 percent accurate, the probability the machine is wrong is 0.1 percent.

$$P(M{=}1 \mid X{=}0) = 0.001$$

The following Eigenmath code computes $P(X{=}1 \mid M{=}1)$ which is the probability that the machine is correct when it gives a positive result.

```
PX1 = 1/1000000
PX0 = 999999/1000000
PM1X1 = 999/1000
PM1X0 = 1/1000
PM1 = PM1X1 * PX1 + PM1X0 * PX0
PM1X1 * PX1 / PM1
```

$\frac{1}{1002}$

So the probability is indeed about 0.1 percent.

Exercises

1. Compute $P(X{=}1 \mid M{=}1)$ for an improved machine that is 99.9999 percent accurate.
2. Compute $P(X{=}1 \mid M{=}1)$ for a false positive probability of zero.

3.11 Birthday problem

```
# What is the probability that 23 people have different birthdays?
#
# Probability that birthday #2 is different from #1 is 364/365.
#
# Probability that birthday #3 is different from #1 and #2 is 363/365.
#
```

```
# Probability that birthday #4 ... etc.
#
# For convenience, multiply by 365/365 and write
#
#     365   364   363          343   365! / (365 - 23)!
# p = --- * --- * --- * ... * --- = ------------------
#     365   365   365          365          365^23
#
# for the probability that 23 people have different birthdays.

"Product method"
p = product(k,1,23,(365-k+1)/365)
float(p)

"Factorial method"
p = 365! / (365 - 23)! / 365^23
float(p)

"Probability of at least one shared birthday"
1.0 - p
```

3.12 Gamma matrix algebra

```
# This script does a few of the exercises from Feynman's book "Quantum Electrodynamics."

# Define the spacetime metric (for multiplying spacetime vectors).

metric = ((-1, 0, 0, 0),
          ( 0,-1, 0, 0),
          ( 0, 0,-1, 0),
          ( 0, 0, 0, 1))

# Define I, the identity matrix.

I = ((1,0,0,0),(0,1,0,0),(0,0,1,0),(0,0,0,1))

# Define the gamma matrices.

gammax = (( 0, 0, 0, 1),
          ( 0, 0, 1, 0),
          ( 0,-1, 0, 0),
          (-1, 0, 0, 0))

gammay = (( 0, 0, 0,-i),
          ( 0, 0, i, 0),
          ( 0, i, 0, 0),
          (-i, 0, 0, 0))

gammaz = (( 0, 0, 1, 0),
          ( 0, 0, 0,-1),
          (-1, 0, 0, 0),
          ( 0, 1, 0, 0))

gammat = (( 1, 0, 0, 0),
          ( 0, 1, 0, 0),
          ( 0, 0,-1, 0),
          ( 0, 0, 0,-1))

# Define the gamma vector.
#
```

```
# The gamma vector has gamma matrices for its components. We express it here
# as a rank 3 tensor. We set up the tensor so that the vector component index
# is the last (rightmost) index. With this configuration we can left-multiply
# with a Feynman slash matrix using the dot function.
#
# For example, in component notation, this is how we want to multiply with a
# Feynman slash matrix:
#
#     aslash[a,b] gamma[b,c,d]
#
# (summation over the repeated index b)
#
# The summation over b is exactly what the dot function does so we can do the
# above multiply with dot(aslash,gamma).
#
# In the following outer products, placing the basis vector operands on the
# right-hand side results in the desired index ordering.

gamma = outer(gammax,(1,0,0,0)) +
        outer(gammay,(0,1,0,0)) +
        outer(gammaz,(0,0,1,0)) +
        outer(gammat,(0,0,0,1))

# DOT is for multiplying gamma vectors. This is a special multiply because we
# have to dot the individual vector components (the gamma matrices) then we
# have to sum over all the results. In component notation, this is how we want
# to do the multiply:
#
#     T[a,c] = A[a,b,d] * B[b,c,d]
#
# To do this, we start with an outer product which results in the following
# rank 6 tensor:
#
#     T[a,b,d,b,c,d]
#
# Next we sum over b (indices 2 and 4) to get the following:
#
#     T[a,d,c,d]
#
# Then we sum over d (indices 2 and 4 again) to get
#
#     T[a,c]
#
# One final note, dot(B,metric) applies the spacetime metric to the rightmost
# index of B, the vector index.

DOT(A,B) = contract(contract(outer(A,dot(B,metric)),2,4),2,4)

# Define arbitrary spacetime vectors a, b and c.

a = (ax,ay,az,at)
b = (bx,by,bz,bt)
c = (cx,cy,cz,ct)

# Define generic Feynman slash matrices.
# Note: The order of dot operands here is different from the book. This is
# because we defined gamma to have its vector index on the right. Therefore
# we have to right-multiply with the spacetime vector so that dot contracts
# over the correct indices. In component notation we have
#
```

```
#     aslash[u,v] = gamma[u,v,w] * a[w]
#
# where summation is over the repeated index w.

aslash = dot(gamma,metric,a)
bslash = dot(gamma,metric,b)
cslash = dot(gamma,metric,c)

# The Feynman slash matrices are 4x4 matrices. For example, aslash looks like
# this:
#
#     at            0           -az       -ax + i ay
#
#     0             at       -ax - i ay       az
#
#     az       ax - i ay       -at           0
#
# ax + i ay       -az           0           -at

# Now we are ready to try the exercises. We want to show that each of the
# following identities is true.

"Checking the following identities:"

#-------------------------------------------------------------------------------
#
#     aslash = at gammat - ax gammax - ay gammay - az gammaz
#
#-------------------------------------------------------------------------------

quote(aslash = at gammat - ax gammax - ay gammay - az gammaz)

A = aslash

B = at gammat - ax gammax - ay gammay - az gammaz

check(A = B) # if A = B then continue, else stop

#-------------------------------------------------------------------------------
#
#     aslash bslash = -bslash aslash + 2 a b
#
#-------------------------------------------------------------------------------

quote(aslash bslash = -bslash aslash + 2 a b)

A = dot(aslash,bslash)

B = -dot(bslash,aslash) + 2 dot(a,metric,b) I

check(A = B)

#-------------------------------------------------------------------------------
#
#     gamma gamma = 4
#
#-------------------------------------------------------------------------------

quote(gamma gamma = 4)
```

```
A = DOT(gamma,gamma)

B = 4 I

check(A = B)

#-------------------------------------------------------------------------------
#
#     gamma aslash gamma = -2 aslash
#
#-------------------------------------------------------------------------------

quote(gamma aslash gamma = -2 aslash)

A = DOT(gamma,dot(aslash,gamma))

B = -2 aslash

check(A = B)

#-------------------------------------------------------------------------------
#
#     gamma aslash bslash gamma = 4 a b
#
#-------------------------------------------------------------------------------

quote(gamma aslash bslash gamma = 4 a b)

A = DOT(gamma,dot(aslash,bslash,gamma))

B = 4 dot(a,metric,b) I

check(A = B)

#-------------------------------------------------------------------------------
#
#     gamma aslash bslash cslash gamma = -2 cslash bslash aslash
#
#-------------------------------------------------------------------------------

quote(gamma aslash bslash cslash gamma = -2 cslash bslash aslash)

A = DOT(gamma,dot(aslash,bslash,cslash,gamma))

B = -2 dot(cslash,bslash,aslash)

check(A = B)

#-------------------------------------------------------------------------------
#
#     If we get here then everything worked.
#
#-------------------------------------------------------------------------------

"OK"
```

3.13 Free particle Dirac equation

```
# This script demonstrates the free particle Dirac equation and a few of its
# solutions.
```

```
#
# "Free particle" means that there is no force pushing the particle around. In
# other words, the potential energy field V is zero everywhere. What remains is
# just the total energy T and kinetic energy K.
#
#     T = K
#
# The equivalent Dirac equation is
#
#     m Psi = i delslash Psi
#
# where m is the particle's rest mass, Psi is a 4-vector and delslash is a
# differential operator involving gamma matrices.
#
# Verify a few solutions (Psi) to the above free-particle Dirac equation.

# Define the spacetime metric.

metric = ((-1,0,0,0),(0,-1,0,0),(0,0,-1,0),(0,0,0,1))

# Define the gamma matrices.

gammax = (( 0, 0, 0, 1),( 0, 0, 1, 0),( 0,-1, 0, 0),(-1, 0, 0, 0))
gammay = (( 0, 0, 0,-i),( 0, 0, i, 0),( 0, i, 0, 0),(-i, 0, 0, 0))
gammaz = (( 0, 0, 1, 0),( 0, 0, 0,-1),(-1, 0, 0, 0),( 0, 1, 0, 0))
gammat = (( 1, 0, 0, 0),( 0, 1, 0, 0),( 0, 0,-1, 0),( 0, 0, 0,-1))

# Define the delslash operator.

delslash(f) =
  dot(gammax,d(f,x))+
  dot(gammay,d(f,y))+
  dot(gammaz,d(f,z))+
  dot(gammat,d(f,t))

# Define energy E, momentum p and coordinate X.

E = sqrt(px^2 + py^2 + pz^2 + m^2)
p = (px,py,pz,E)
X = (x,y,z,t)

# Verify that p.p = m^2

check(dot(p,metric,p) - m^2 = 0) # continue if true, else stop

# Define the solutions.

PsiA = (E+m,0,pz,px+i py) exp(-i dot(p,metric,X))
PsiB = (0,E+m,px-i py,-pz) exp(-i dot(p,metric,X))
PsiC = (pz,px+i py,E+m,0) exp(i dot(p,metric,X))
PsiD = (px-i py,-pz,0,E+m) exp(i dot(p,metric,X))

# Verify the solutions.

check(m PsiA - i delslash(PsiA) = 0)
check(m PsiB - i delslash(PsiB) = 0)
check(m PsiC - i delslash(PsiC) = 0)
check(m PsiD - i delslash(PsiD) = 0)

# Try a linear combination of all solutions.
```

```
Psi = A PsiA + B PsiB + C PsiC + D PsiD

check(m Psi - i delslash(Psi) = 0)

# For PsiA and PsiB it turns out that
#
#     i delslash Psi = pslash Psi
#
# So another form of the free particle Dirac equation is
#
#     m Psi = pslash Psi
#
# Verify solutions to the above equation.

# Define the gamma tensor. See Gamma Matrix Algebra for details.

gamma =
  outer(gammax,(1,0,0,0)) +
  outer(gammay,(0,1,0,0)) +
  outer(gammaz,(0,0,1,0)) +
  outer(gammat,(0,0,0,1))

# Dot gamma with p to get pslash.

pslash = dot(gamma,metric,p)

# Verify the solutions again.

check(m PsiA - dot(pslash,PsiA) = 0)
check(m PsiB - dot(pslash,PsiB) = 0)
check(m PsiC + dot(pslash,PsiC) = 0)
check(m PsiD + dot(pslash,PsiD) = 0)

# Display pslash on the computer screen.

pslash

# If we get here then everything worked, print OK.

"OK"
```

3.14 Maxwell in tensor form

```
--Maxwell equations in tensor form.
--See the book Gravitation p. 81.
--
--   F     + F     + F     = 0
--    ab,c    bc,a    ca,b
--
--    ab           a
--   F     = 4 pi J
--      ,b
--
--For this demo, use circular polarized light.
--
EX = sin(t+z)
EY = cos(t+z)
EZ = 0
BX = cos(t+z)
```

```
BY = -sin(t+z)
BZ = 0
FDD = ((  0, -EX, -EY, -EZ),
       ( EX,   0,  BZ, -BY),
       ( EY, -BZ,   0,  BX),
       ( EZ,  BY, -BX,   0))   --See p. 74. Here, DD means "down down" indices.
X = (t,x,y,z)   --Coordinate system
FDDD = d(FDD,X)   --Gradient of F
T1 = transpose(transpose(FDDD,2,3),1,2)   --Transpose bca to abc
T2 = transpose(transpose(FDDD,1,2),2,3)   --Transpose cab to abc
check(FDDD + T1 + T2 = 0)
guu = ((-1,0,0,0),(0,1,0,0),(0,0,1,0),(0,0,0,1))
FDDU = contract(outer(FDDD,guu),3,4)   --Easier to make FDDU than FUUD.
check(contract(FDDU,2,3) = 0)   --For light J is zero.
"OK"
```

3.15 Static spherical metric

```
# This script calculates the Einstein tensor for a static spherically symmetric
# metric.
#
# Cf. "A first course in general relativity," Bernard F. Schutz, p. 255.
#
# This is the line element for the metric (Equation 10.7)
#
#   2     2 Phi    2    2 Lambda   2    2        2
# ds  = -e      dt  + e         dr  + r  d Omega
#
# where
#
#  2        2    2         2      2            2
# r  d Omega  = r  (d theta  + sin  theta d phi )
#
# Note: Phi and Lambda are both functions of r.

gdd = ((-exp(2 Phi(r)),                0,   0,                0),
       (             0, exp(2 Lambda(r)),   0,                0),
       (             0,                0, r^2,                0),
       (             0,                0,   0, r^2 sin(theta)^2))

# Note: "dd" stands for two "down" indices, "uu" stands for two "up" indices.

# X is our coordinate system. We need this for computing gradients.

X = (t,r,theta,phi)

# Step 1: Calculate guu.

guu = inv(gdd)

# Step 2: Calculate the connection coefficients. Cf. Gravitation, p. 210.
#
# Gamma    = 1/2 (g     + g     - g    )
#      abc          ab,c    ac,b    bc,a
#
# Note: The comma means gradient which increases the rank of gdd by 1.

gddd = d(gdd,X)

# Note: We transpose indices so they match up with Gamma, i.e., we put them in
```

```
# alphabetical order.

GAMDDD = 1/2 (gddd +                # indices are already in correct order
transpose(gddd,2,3) -               # transpose c and b
transpose(transpose(gddd,2,3),1,2)) # transpose c and a, then b and a

# Raise first index.
#
#      a      au
# Gamma    = g   Gamma
#       bc            ubc
#
# Note: Sum over index u means contraction.

GAMUDD = contract(outer(guu,GAMDDD),2,3)

# Step 3. Calculate the Riemann tensor. Cf. Gravitation, p. 219.
#
# a is alpha
# b is beta
# c is gamma
# d is delta
# u is mu
#
#  a           a            a            a        u          a        u
# R     = Gamma      - Gamma      + Gamma    Gamma    - Gamma    Gamma
#   bcd          bd,c         bc,d         uc       bd         ud       bc
#
# Do the gradient once and save in a temporary variable.

tmp1 = d(GAMUDD,X)

# The Gamma Gamma product is a rank 6 tensor with dim 4 per rank.
# That works out to 4 to the 6th or 4,096 elements.
# Of course, we'll do the outer product and contract over u just once and save
# the result in a second temporary variable.

tmp2 = contract(outer(GAMUDD,GAMUDD),2,4)

# Now put it all together. Do the transpositions so the indices get matched up
# with R on the left, i.e., put them in alphabetical order.

RUDDD = transpose(tmp1,3,4) -             # transpose d and c
  tmp1 +                                  # already in correct order
  transpose(tmp2,2,3) -                   # transpose c and b
  transpose(transpose(tmp2,2,3),3,4)      # transpose d and b, then d and c

# Step 4: Calculate the Ricci tensor. Cf. Gravitation, p. 343.
#
#        a
# R   = R
#  uv     uav
#
# Contract over "a" (1st and 3rd indices).

RDD = contract(RUDDD,1,3)

# Step 5: Calculate the Ricci scalar. Cf. Gravitation, p. 343.
#
#      uv
```

```
# R = g   R
#          vu  ...the book has uv, does it give the same result?
#              Yes because the metric tensor is symmetric so it's ok to
#              transpose.

R = contract(contract(outer(guu,RDD),2,3),1,2)

# Step 6: Finally, calculate the Einstein tensor. Cf. Gravitation, p. 343.
#
# G   = R   - 1/2 g   R
#  uv    uv        uv

GDD = RDD - 1/2 gdd R

# Next we compare this result with Schutz' book. Schutz p. 255 gives the
# following Einstein tensor components (all other components are zero):
#
#        1                d
# G   = ----  exp(2 Phi) ---- [r (1 - exp(-2 Lambda))]
#  tt     2               dr
#        r
#
#          1                                          2
# G   = - ---- exp(2 Lambda) (1 - exp(-2 Lambda)) + --- Phi'
#  rr       2                                        r
#          r
#
#                  2                                  2
# G             = r  exp(-2 Lambda) [Phi'' + (Phi')  + Phi'/r
#  theta theta
#
#                                                       - Phi' Lambda' - Lamda'/r]
#
#               2
# G        = sin  theta G
#  phi phi               theta theta

Gtt = 1/r^2 exp(2 Phi(r)) d(r (1 - exp(-2 Lambda(r))),r)

Grr = -1/r^2 exp(2 Lambda(r)) (1 - exp(-2 Lambda(r))) + 2/r d(Phi(r),r)

Gthetatheta = r^2 exp(-2 Lambda(r)) (
  d(d(Phi(r),r),r) +
  d(Phi(r),r)^2 +
  d(Phi(r),r) / r -
  d(Phi(r),r) d(Lambda(r),r) -
  d(Lambda(r),r) / r)

Gphiphi = sin(theta)^2 Gthetatheta

# Put together the expected tensor:

expect = ((Gtt,   0,           0,       0),
          (  0, Grr,           0,       0),
          (  0,   0, Gthetatheta,       0),
          (  0,   0,           0, Gphiphi))

# Check that GDD is correct.

check(GDD = expect)
```

```
# Display the non-zero components of GDD.

Gtt
Grr
Gthetatheta
Gphiphi

"OK"
```

3.16 Circular polarization

```
--Simple example of clockwise (right hand) circular polarization.
--Both E and B rotate clockwise while remaining at right angles to each other.
--To see the rotation, suppose you are standing at location z=0 at time t=0.
--You observe E=(0,1,0), that is, E pointing to 12 o'clock.
--Now increase t a little bit by just waiting.
--What happens is the y component decreases and the x component increases.
--So now, instead of pointing straight up, E points to the right a little.
--Keep waiting and eventually observe E=(1,0,0), E pointing to 3 o'clock.
--E has rotated a quarter turn.
--Let t run freely and the rotation continues.
E = (sin(t+z), cos(t+z), 0)
B = (cos(t+z), -sin(t+z), 0)
--Check Maxwell's equations.
check(div(E) = 0)
check(curl(E) + d(B,t) = 0)
check(div(B) = 0)
check(curl(B) - d(E,t) = 0)
check(dot(E,B) = 0)
--Already shown to be true by Maxwell above but do this anyway...
check(dot(E,B) = 0)               --Check right angle
check(dot(E,E) - dot(B,B) = 0)    --Check equal length
--If we get here then everything checks out, print OK.
"OK"
```

3.17 Elliptical polarization

```
--Simple example of clockwise (right hand) elliptical polarization.
--As in "circular polarization" except E and B trace out ellipses.
--Becomes linear polarization for EX=0 or EY=0.
--Becomes circular polarization for EX=EY.
--Note that at any given time E and B have the same length.
E = (EX sin(t+z), EY cos(t+z), 0)
B = (EY cos(t+z), -EX sin(t+z), 0)
--Check Maxwell's equations.
check(div(E) = 0)
check(curl(E) + d(B,t) = 0)
check(div(B) = 0)
check(curl(B) - d(E,t) = 0)
check(dot(E,B) = 0)
--Already shown to be true by Maxwell above but do this anyway...
check(dot(E,B) = 0)               --Check right angle
check(dot(E,E) - dot(B,B) = 0)    --Check equal length
--If we get here then everything checks out, print OK.
"OK"
```

3.18 Vector calculus

```
# This script tests 10 vector calculus identities.

# Define the cross product, div, grad, curl and laplacian for
# rectangular coordinates.

cross(u,v) = (
  u[2] v[3] - u[3] v[2],
  u[3] v[1] - u[1] v[3],
  u[1] v[2] - u[2] v[1]
)

div(v) = contract(d(v,(x,y,z)),1,2)

grad(v) = d(v,(x,y,z))

curl(f) = (
  d(f[3],y) - d(f[2],z),
  d(f[1],z) - d(f[3],x),
  d(f[2],x) - d(f[1],y)
)

laplacian(f) = d(d(f,x),x) + d(d(f,y),y) + d(d(f,z),z)

# Note: Functions can be left undefined, such as FX(), FY(), etc.
# These "generic" functions, when evaluated by the derivative function d(),
# are considered to be dependent on all variables.
# Basically what this means is that d() does no evaluation at all.
# For example, d(FX(),x) returns the expression d(FX(),x).

# Define generic vector functions F and G.

F = (FX(),FY(),FZ())
G = (GX(),GY(),GZ())

# Now check the 10 identities.

"Checking the following identities:"

"1. div(curl F) = 0"

A = div(curl(F))

check(A = 0)

"2. curl(grad f) = 0"

A = curl(grad(f())) # Note the use of generic scalar function f() here.

check(A = 0)

"3. div(grad f) = laplacian f"

A = div(grad(f()))

B = laplacian(f())

check(A = B)

"4. curl(curl F) = grad(div F) - laplacian F"
```

```
A = curl(curl(F))

B = grad(div(F)) - laplacian(F)

check(A = B)

"5. grad(fg) = f grad g + g grad(f)"

A = grad(f() g())

B = f() grad(g()) + g() grad(f())

check(A = B)

"6. grad(F . G) = (G . grad)F + (F . grad)G + G x curl F + F x curl G"

A = grad(dot(F,G))

B = dot(grad(F),G)+dot(grad(G),F)+cross(G,curl(F))+cross(F,curl(G))

check(A = B)

# Note: It turns out that (G . grad)F actually means (grad F) . G

"7. div(fF) = f div F + grad f . F"

A = div(f() F)

B = f() div(F) + dot(grad(f()),F)

check(A = B)

"8. div(F x G) = G . curl F - F . curl G"

A = div(cross(F,G))

B = dot(G,curl(F)) - dot(F,curl(G))

check(A = B)

"9. curl(fF) = f curl F + grad f x F"

A = curl(f() F)

B = f() curl(F) + cross(grad(f()),F)

check(A = B)

"10. curl(F x G) = F div G - G div F + (G . grad)F - (F . grad)G"

A = curl(cross(F,G))

B = F div(G) - G div(F) + dot(grad(F),G) - dot(grad(G),F)

check(A = B)

# If we get here then everything worked.

"OK"
```

3.19 Rotation matrix

```
-- This script demonstrates some properties of a rotation matrix.

-- These are the matrix components.

R11 = expcos(phi2) expcos(phi1) - expcos(theta) expsin(phi1) expsin(phi2)
R12 = -expcos(phi2) expsin(phi1) - expcos(theta) expcos(phi1) expsin(phi2)
R13 = expsin(phi2) expsin(theta)

R21 = expsin(phi2) expcos(phi1) + expcos(theta) expsin(phi1) expcos(phi2)
R22 = -expsin(phi2) expsin(phi1) + expcos(theta) expcos(phi1) expcos(phi2)
R23 = -expcos(phi2) expsin(theta)

R31 = expsin(theta) expsin(phi1)
R32 = expsin(theta) expcos(phi1)
R33 = expcos(theta)

-- R is a rotation matrix.

R = ((R11,R12,R13),(R21,R22,R23),(R31,R32,R33))

-- Inverse and transpose are equivalent.

check(inv(R) = transpose(R))

-- Determinant is 1.

check(det(R) = 1)

-- Does not change the length of a vector.

U = (U1,U2,U3)

check(U^2 = dot(R,U)^2)

-- If we get here then everything worked, print OK.

"OK"
```

3.20 Quantile function

```
# This demo implements pnorm and qnorm from the R language.
#
# For a normal distribution...
#
#       pnorm is the cumulative distribution function
#
#       qnorm is the quantile function
#
# Recall that a quantile function is the inverse of a cdf:
#
#       p = pnorm(x,mu,sigma)
#
#       x = qnorm(p,mu,sigma)
#
# p is the probability of observing an X less than or equal to x, that is,
#
#       p = P(X <= x)
#
```

```
# for X normally distributed with mean mu and variance sigma squared.

pnorm(x,mu,sigma) = float((1+erf((x-mu)/sigma/sqrt(2)))/2)

# Note: qnorm clobbers a, b, and m

qnorm(p,mu,sigma) = do(
  a = float(mu-3*sigma),
  b = float(mu+3*sigma),
  for(k,1,20,m=(a+b)/2,test(pnorm(m,mu,sigma)<p,a=m,b=m)),
  (a+b)/2 # return value
)

qnorm(0.975,0,1)
qnorm(0.995,0,1)

# Compare above results with R...
#
# > qnorm(0.975,0,1)
# [1] 1.959964
# > qnorm(0.995,0,1)
# [1] 2.575829
```

4 Built-in functions

abs

abs(x) returns the absolute value or vector length of x. The mag function should be used for complex x.

```
P = (x,y)
abs(P)
```

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

adj

adj(m) returns the adjunct of matrix m.

and

and($a, b, \ldots$) returns the logical "and" of predicate expressions.

arccos

arccos(x) returns the inverse cosine of x.

arccosh

arccosh(x) returns the inverse hyperbolic cosine of x.

arcsin

arcsin(x) returns the inverse sine of x.

arcsinh

arcsinh(x) returns the inverse hyperbolic sine of x.

arctan

arcttan(x) returns the inverse tangent of x.

arctanh

arctanh(x) returns the inverse hyperbolic tangent of x.

arg

arg(z) returns the angle of complex z.

ceiling

ceiling(x) returns the smallest integer not less than x.

check

check(x) In a script, if the predicate x is true then continue, else stop.

choose

choose(n, k) returns $\binom{n}{k}$

circexp

circexp(x) returns expression x with circular functions converted to exponential forms. Sometimes this will simplify an expression.

coeff

coeff(p, x, n) returns the coefficient of x^n in polynomial p.

cofactor

cofactor(m, i, j) returns of the cofactor of matrix m with respect to row i and column j.

conj

conj(z) returns the complex conjugate of z.

contract

contract(a, i, j) returns tensor a summed over indices i and j. If i and j are omitted then indices 1 and 2 are used. contract(m) is equivalent to the trace of matrix m.

cos

cos(x) returns the cosine of x.

cosh

cosh(x) returns the hyperbolic cosine of x.

cross

cross(u, v) returns the cross product of vectors u and v.

curl

curl(u) returns the curl of vector u.

d

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

defint

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. For example, $d(f, x, a, b, y, c, d)$.

deg

deg(p, x) returns the degree of polynomial p in x.

denominator

denominator(x) returns the denominator of expression x.

det

det(m) returns the determinant of matrix m.

do

do($a, b, \ldots$) evaluates the argument list from left to right. Returns the result of the last argument.

dot

dot($a, b, \ldots$) returns the dot product of tensors.

draw

draw(f, x) draws the function f with respect to x.

erf

erf(x) returns the error function of x.

erfc

erf(x) returns the complementary error function of x.

eval

eval(f, x, n) returns f evaluated at $x = n$.

exp

exp(x) returns e^x.

expand

expand(r, x) returns the partial fraction expansion of the ratio of polynomials r in x.

```
expand(1/(x^3+x^2),x)
```

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

expcos

expcos(x) returns the cosine of x in exponential form.

```
expcos(x)
```

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

expsin

expsin(x) returns the sine of x in exponential form.

```
expsin(x)
```

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

factor

factor(n) factors the integer n.

```
factor(12345)
```

$3\times 5\times 823$

$factor(p,x)$ factors polynomial p in x. The last argument can be omitted for polynomials in x. The argument list can be extended for multivariate polynomials. For example, factor(p,x,y) factors p over x and then over y.

```
factor(125*x^3-1)
```

$(5x-1)(25x^2+5x+1)$

factorial

Example:

```
10!
```

3628800

filter

filter$(f,a,b,\ldots)$ returns f with terms involving a, b, etc. removed.

```
1/a+1/b+1/c
```

$$\frac{1}{a}+\frac{1}{b}+\frac{1}{c}$$

```
filter(last,a)
```

$$\frac{1}{b}+\frac{1}{c}$$

float

float(x) converts x to a floating point value.

```
sum(n,0,20,(-1/2)^n)
```

$$\frac{699051}{1048576}$$

```
float(last)
```

0.666667

floor

floor(x) returns the largest integer not greater than x.

for

for($i, j, k, a, b, \ldots$) For i equals j through k evaluate a, b, etc.

```
x = 0
y = 2
for(k,1,9,x=sqrt(2+x),y=2*y/x)
float(y)
```

3.14159

gcd

gcd($a, b, \ldots$) returns the greatest common divisor.

hermite

hermite(x, n) returns the nth Hermite polynomial in x.

hilbert

hilbert(n) returns a Hilbert matrix of order n.

imag

imag(z) returns the imaginary part of complex z.

inner

inner($a, b, \ldots$) returns the inner product of tensors. Same as the dot product.

integral

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

inv

inv(m) returns the inverse of matrix m.

isprime

isprime(n) returns 1 if n is prime, zero otherwise.

```
isprime(2^53-111)
```

1

laguerre

laguerre(x, n, a) returns the nth Laguerre polynomial in x. If a is omitted then $a = 0$ is used.

lcm

lcm($a, b, \ldots$) returns the least common multiple.

leading

leading(p, x) returns the leading coefficient of polynomial p in x.

```
leading(5x^2+x+1,x)
```

5

legendre

legendre(x, n, m) returns the nth Legendre polynomial in x. If m is omitted then $m = 0$ is used.

log

log(x) returns the natural logarithm of x.

mag

mag(z) returns the magnitude of complex z.

mod

mod(a, b) returns the remainder of a divided by b.

not

not(x) negates the result of predicate expression x.

nroots

nroots(p, x) returns all of the roots, both real and complex, of polynomial p in x. The roots are computed numerically. The coefficients of p can be real or complex.

numerator

numerator(x) returns the numerator of expression x.

or

or$(a, b, \ldots)$ returns the logical "or" of predicate expressions.

outer

outer$(a, b, \ldots)$ returns the outer product of tensors.

polar

polar(z) converts complex z to polar form.

prime

prime(n) returns the nth prime number, $1 \leq n \leq 10{,}000$.

print

print($a, b, \ldots$) evaluates expressions and prints the results.. Useful for printing from inside a "for" loop.

product

product(i, j, k, f) returns $\displaystyle\prod_{i=j}^{k} f$

quote

quote(x) returns expression x unevaluated.

quotient

quotient(p, q, x) returns the quotient of polynomials in x.

rank

rank(a) returns the number of indices that tensor a has. A scalar has no indices so its rank is zero.

rationalize

rationalize(x) puts everything over a common denominator.

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

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

real

real(z) returns the real part of complex z.

rect

rect(z) returns complex z in rectangular form.

roots

roots(p, x) returns the values of x such that the polynomial $p(x) = 0$. The polynomial should be factorable over integers.

simplify

simplify(x) returns x in a simpler form.

sin

sin(x) returns the sine of x.

sinh

sinh(x) returns the hyperbolic sine of x.

sqrt

sqrt(x) returns the square root of x.

stop

In a script, it does what it says.

subst

subst(a, b, c) substitutes a for b in c and returns the result.

sum

sum(i, j, k, f) returns $\sum_{i=j}^{k} f$

tan

tan(x) returns the tangent of x.

tanh

tanh(x) returns the hyperbolic tangent of x.

taylor

taylor(f, x, n, a) returns the Taylor expansion of f of x at a. The argument n is the degree of the expansion. If a is omitted then $a = 0$ is used.

```
taylor(1/cos(x),x,4)
```

$$\frac{5}{24}x^4 + \frac{1}{2}x^2 + 1$$

test

test$(a, b, c, d, \ldots)$ If a is true then b is returned else if c is true then d is returned, etc. If the number of arguments is odd then the last argument is returned when all else fails.

transpose

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)
```

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

unit

unit(n) returns an $n \times n$ identity matrix.

```
unit(2)
```

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

zero

zero($i, j, \ldots$) returns a null tensor with dimensions i, j, etc. Useful for creating a tensor and then setting the component values.

5 Syntax

Math	*Eigenmath*	*Alternate form and/or comment*
$-a$	`-a`	
$a+b$	`a+b`	
$a-b$	`a-b`	
ab	`a*b`	`a b` *with a space in between*
$\frac{a}{b}$	`a/b`	
$\frac{a}{bc}$	`a/b/c`	
a^2	`a^2`	
$\sqrt{a}$	`a^(1/2)`	`sqrt(a)`
$\frac{1}{\sqrt{a}}$	`a^(-1/2)`	`1/sqrt(a)`
$a(b+c)$	`a*(b+c)`	`a (b+c)` *with a space in between*
$f(a)$	`f(a)`	
$\begin{pmatrix} a \\ b \\ c \end{pmatrix}$	`(a,b,c)`	
$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$	`((a,b),(c,d))`	
T^{12}	`T[1,2]`	*tensor component access*
$2\,\mathrm{km}$	`2*"km"`	*units of measure are quoted*

6 Tricks

1. The Eigenmath result field can be copied to the pasteboard by click, drag, then release.

2. The last result is stored in the symbol `last`.

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

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

5. Calculations in a script can span multiple lines. The trick is to arrange things so the parser will keep going. For example, if a calculation ends with a plus sign, the parser will go to the next line to get another term. Also, the parser will keep going when it expects a close parenthesis.

6. Normally a function body is not evaluated when a function is defined. However, in some cases it is required that the function body be the result of something. The trick is to use eval. For example, the following code causes the function body to be a sixth order Taylor series expansion of $cos(x)$.

   ```
   f(x) = eval(taylor(cos(x),x,6))
   ```

7. Use `binding` to see the unevaluated binding of a symbol.

   ```
   binding(f)
   ```

8. This is how to clear a symbol.

   ```
   f = quote(f)
   ```