

# Eigenmath Manual

February 3, 2008

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

Let us begin by considering the following passage 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, 3529471145760275132301897342055866171392 (I am not sure I have got this right; anyway the root was 212).

We can check Nabokov's arithmetic by typing the following into Eigenmath.

```
212^17
```

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

```
3529471145760275132301897342055866171392
```

So Nabokov did get it right after all. We can enter *float* or click on the float button to scale the number down to size.

```
float
```

```
3.52947 × 1039
```

## Avogadro's constant

There is a proposal to define Avogadro's constant as exactly  $84446886$  to the third power.<sup>1</sup> 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.

$$A=84446886^3$$

$$B=6.0221415 \times 10^{23}$$

$$A-B$$

$$-5.17173 \times 10^{16}$$

$$0.0000010 \times 10^{23}$$

$$1 \times 10^{17}$$

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

$$\text{factor}(A)$$

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

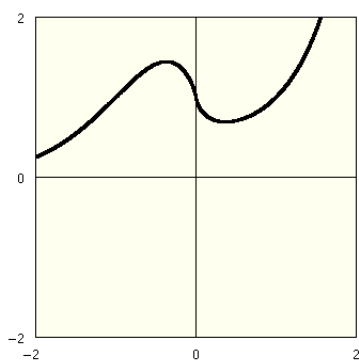
---

<sup>1</sup>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](http://www.physorg.com/news109595312.html)

## Zero to the zero power

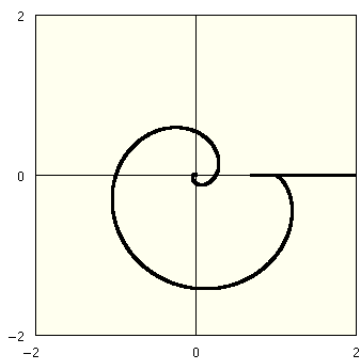
The following example draws a graph of the function  $f(x) = |x^x|$ . The graph shows why the convention  $0^0 = 1$  makes sense.

```
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)
```



## Geometric series

A geometric series converges according to the formula

$$\sum_{k=0}^{\infty} a^k = \frac{1}{1-a}, \quad |a| < 1$$

If we use  $a = -1/2$  and for practical purposes only count up to nine instead of infinity, we should have

$$\sum_{k=0}^9 \left(-\frac{1}{2}\right)^k \approx \frac{2}{3}$$

The above calculation can be done in one line of code using the *sum* function.

```
sum(k,0,9,(-0.5)^k)
0.666016
```

The following example uses an intermediate variable.

```
f=sum(k,0,9,a^k)
f
      f = 1 + a + a^2 + a^3 + a^4 + a^5 + a^6 + a^7 + a^8 + a^9
eval(f,a,-1/2)
      341
      512
float(last)
0.666016
```

As seen on the first line, no result is printed when a symbol is defined. When you do in fact want to see the value of a symbol, just enter it as shown on the second line.

When a result is displayed, it is also stored in the symbol *last*.

The following example shows how to define a function.

```
f(a)=sum(k,0,9,a^k)
f
```

$$f = \text{sum}(k, 0, 9, a^k)$$

```
f(-1/2)
```

$$\frac{341}{512}$$

```
f(-0.5)
```

$$0.666016$$

Eigenmath handles function definitions in a special way. Unlike a normal symbol, a function definition is not evaluated immediately. The following example demonstrates the difference.

```
f=sum(k,0,9,a^k)
f
```

$$f = 1 + a + a^2 + a^3 + a^4 + a^5 + a^6 + a^7 + a^8 + a^9$$

```
f(a)=sum(k,0,9,a^k)
f
```

$$f = \text{sum}(k, 0, 9, a^k)$$

Why are they handled differently? Well, suppose we want to define  $f$  as a function of two variables,  $a$  and  $n$ , like this.

```
f(a,n)=sum(k,0,n,a^k)
```

In this case it is not possible to evaluate the sum right away. The value of  $n$  will not be known until  $f$  is actually used somewhere. Consequently, Eigenmath does not normally evaluate a function when it is defined. Now, having said that, there are two work arounds that provide exceptions to the rule. They are *eval* and *quote*.

```
f=quote(sum(k,0,9,a^k))
f
```

$$f = \text{sum}(k, 0, 9, a^k)$$

```
f(a)=eval(sum(k,0,9,a^k))
f
```

$$f = 1 + a + a^2 + a^3 + a^4 + a^5 + a^6 + a^7 + a^8 + a^9$$

## Units of measure

Quoted strings can be used to express units of measurement in a calculation. For example, the space shuttle accelerates from zero to 17,000 miles per hour in 8 minutes. The average acceleration of the space shuttle is

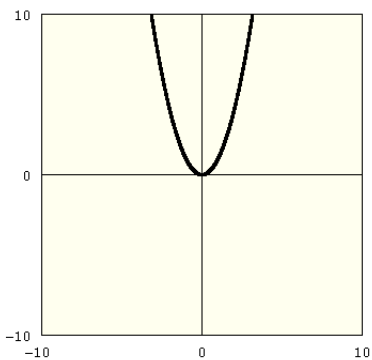
```
v=17000*"mile"/"hr"  
t=8*"min"/(60*"min"/"hr")  
v/t
```



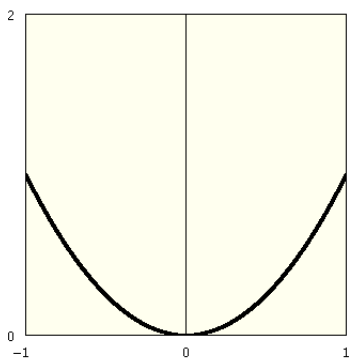
## Draw

`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)
```

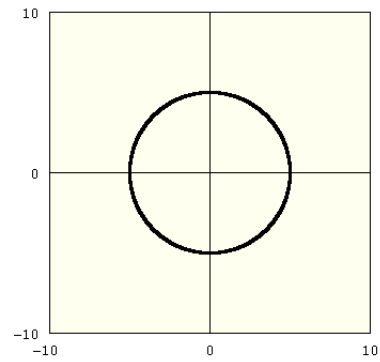


```
clear
```

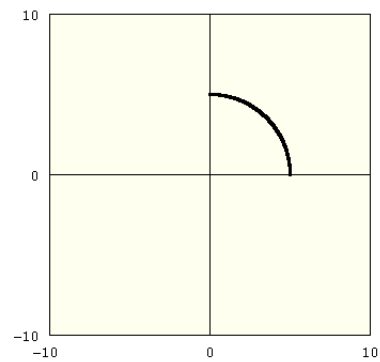
The `clear` command (or a click of the Clear button) resets `xrange` and `yrange`. This needs to be done so that the next graph appears as shown.

Parametric drawing occurs when a function returns a vector. The vector *trange* controls the parameter range. The default range is  $(-\pi, \pi)$ .

```
f=(cos(t),sin(t))  
draw(5*f)
```

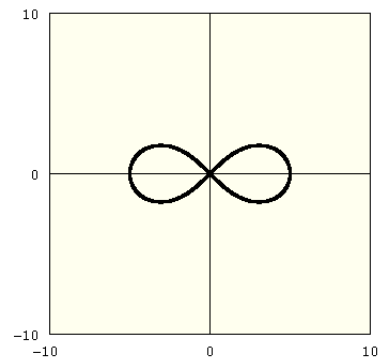


```
trange=(0,pi/2)  
draw(5*f)
```



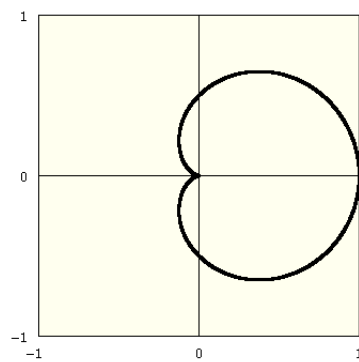
Here are a couple of interesting curves and the code for drawing them. First is a lemniscate.

```
clear
X=cos(t)/(1+sin(t)^2)
Y=sin(t)*cos(t)/(1+sin(t)^2)
draw(5*(X,Y))
```



Next is a cardioid.

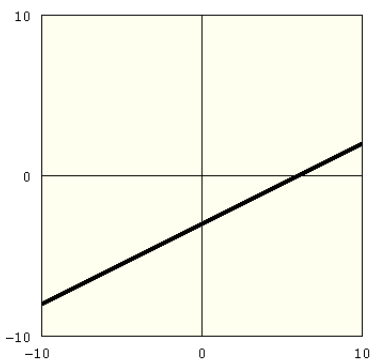
```
r=(1+cos(t))/2
u=(cos(t),sin(t))
xrange=(-1,1)
yrange=(-1,1)
draw(r*u)
```



## Eigenmath scripts

Here is a simple example that draws the graph of  $y = mx + b$ .

```
y=m*x+b  
m=1/2  
b=-3  
draw(y)
```



Now suppose that we want to draw the graph with a different  $m$ . We could type in everything all over again, but it would be easier in the long run to write a script. Then we can go back and quickly change  $m$  and  $b$  as many times as we want.

To prepare a script, click on the Edit Script button. Then enter the script commands, one per line, as shown above. Then click on the Run Script button to see the graph.

Eigenmath runs a script by stepping through it line by line. Each line is evaluated just like a regular command. This continues until the end of the script is reached. After the script runs, you can click Edit Script and go back and change something.

Sometimes it is desirable to have a script print a few comments when it runs. This can be accomplished by placing the desired text in quotes on a single line. For example, the script

```
"Here is the value of pi."  
float(pi)
```

displays the following when run.

```
Here is the value of pi.
```

```
3.14159
```

Eigenmath includes a simple debug facility. Setting the variable *trace* to 1 causes each line of the script to be printed as the script runs. Normally this setting would be the first line in the script.

```
trace=1  
--Now each line of the script is printed as it runs.
```

## 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 rectangular or polar form.

$$\begin{array}{ll} \mathbf{a+i*b} & a + ib \\ \mathbf{exp(i*pi/3)} & \exp(\frac{1}{3}i\pi) \end{array}$$

Converting to rectangular or polar coordinates simplifies mixed forms.

$$\begin{array}{ll} \mathbf{A=1+i} & \\ \mathbf{B=sqrt(2)*exp(i*pi/4)} & \\ \mathbf{A-B} & 1 + i - 2^{1/2} \exp(\frac{1}{4}i\pi) \\ \mathbf{rect} & 0 \end{array}$$

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

$$\begin{array}{ll} \mathbf{(a+i*b)^2} & a^2 - b^2 + 2iab \end{array}$$

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.

$$\begin{array}{ll} \mathbf{1/(2-i)} & \frac{2}{5} + \frac{1}{5}i \\ \mathbf{(-1+3i)/(2-i)} & -1 + i \end{array}$$

The absolute value of a complex number returns its magnitude.

`abs(3+4*i)`

5

In light of this, the following result might be unexpected.

`abs(a+b*i)`

`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

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

The *mag* function is an alternative. It treats symbols like  $a$  and  $b$  as real.

`mag(a+b*i)`

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

## Linear algebra

*dot* is used to multiply vectors and matrices. The following example shows how to use *dot* and *inv* to solve for  $\mathbf{X}$  in  $\mathbf{AX} = \mathbf{B}$ .

```
A=((3.8,7.2),(1.3,-0.9))  
B=(16.5,-22.1)  
X=dot(inv(A),B)  
X
```

$$\begin{pmatrix} -11.2887 \\ 8.24961 \end{pmatrix}$$

One might wonder why the *dot* function is necessary. Why not simply use  $X = \text{inv}(A) * B$  like scalar multiplication? The reason is that the software normally reorders factors internally to optimize processing. For example,  $\text{inv}(A) * B$  in symbolic form is changed to  $B * \text{inv}(A)$  internally. Since the dot product is not commutative, this reordering would give the wrong result. Using a function to do the multiply avoids the problem because function arguments are not reordered.

It should be noted that *dot* can have more than two arguments. For example,  $\text{dot}(A, B, C)$  can be used for the dot product of three tensors.



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

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$\text{inv}(\mathbf{A})$$

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

$$\text{adj}(\mathbf{A})$$

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

$$\det(\mathbf{A})$$

$$ad - bc$$

$$\text{inv}(\mathbf{A}) - \text{adj}(\mathbf{A}) / \det(\mathbf{A})$$

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

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  $\mathbf{A}$  and  $\mathbf{B}$  you want to check that

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

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

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

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

```

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

adj(A)-transpose(C)

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

## Calculus

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

$$\begin{array}{l} d(x^2) \\ 2x \end{array}$$

The following table summarizes the various ways to obtain multiderivatives.

$$\begin{array}{lll} \frac{\partial^2 f}{\partial x^2} & d(f, x, x) & d(f, x, 2) \\ \frac{\partial^2 f}{\partial x \partial y} & d(f, x, y) & \\ \frac{\partial^{m+n+\dots} f}{\partial x^m \partial y^n \dots} & d(f, x, \dots, y, \dots) & d(f, x, m, y, n, \dots) \end{array}$$

The gradient of  $f$  is obtained by using a vector for  $x$  in  $d(f, x)$ .

$$\begin{array}{l} r = \text{sqrt}(x^2 + y^2) \\ d(r, (x, y)) \\ \left( \begin{array}{c} \frac{x}{(x^2 + y^2)^{1/2}} \\ \frac{y}{(x^2 + y^2)^{1/2}} \end{array} \right) \end{array}$$

The  $f$  in  $d(f, x)$  can be a tensor function. Gradient raises the rank by one.

$$\begin{array}{l} F = (x + 2y, 3x + 4y) \\ X = (x, y) \\ d(F, X) \\ \left( \begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array} \right) \end{array}$$

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  $y$ .

$$\begin{array}{ll} d(f(x), x) & \partial(f(x), x) \\ d(f(x), y) & 0 \\ d(f(x, y), y) & \partial(f(x, y), y) \\ d(f(), t) & \partial(f(), t) \end{array}$$

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

$$\text{div}(\text{curl } \mathbf{F}) = 0$$

for an arbitrary vector function  $\mathbf{F}$ .

$$\begin{array}{l} F=(F1(x,y,z),F2(x,y,z),F3(x,y,z)) \\ \text{curl}(U)=(d(U[3],y)-d(U[2],z),d(U[1],z)-d(U[3],x),d(U[2],x)-d(U[1],y)) \\ \text{div}(U)=d(U[1],x)+d(U[2],y)+d(U[3],z) \\ \text{div}(\text{curl}(F)) \\ 0 \end{array}$$

*integral*(*f*, *x*) returns the integral of *f* with respect to *x*. The *x* can be omitted for expressions in *x*. A multi-integral can be obtained by extending the argument list.

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

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

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

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

*defint*(*f*, *x*, *a*, *b*, ...) computes the definite integral of *f* with respect to *x* evaluated from *a* to *b*. The argument list can be extended for multiple integrals.

The following example computes the integral of  $f = x^2$  over the domain of a semicircle. For each *x* along the abscissa, *y* ranges from 0 to  $\sqrt{1 - x^2}$ .

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

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

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.

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

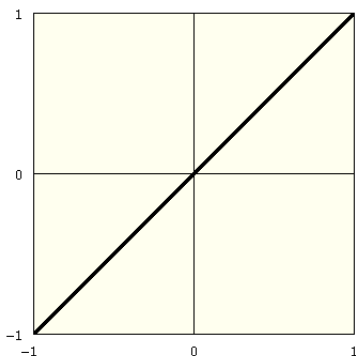
0

The fundamental theorem of calculus was established by James Gregory, a contemporary of Newton. The theorem is a formal expression of the inverse relation between integrals and derivatives.

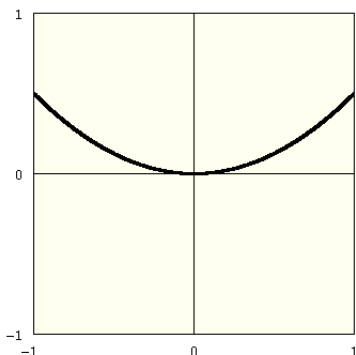
$$\int_a^b f'(x) dx = f(b) - f(a)$$

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

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

## Line integrals

Line integrals are easily computed by converting the coordinates  $x$ ,  $y$  and  $z$  into functions of  $t$ . This has the effect of changing the measure as well. For instance,  $dx$  becomes  $(dx/dt) dt$ . 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
defint(y^2*d(x,t),t,0,1)
```

$$\frac{8}{3}$$

Evaluate  $\int y dx$  along the straight line from  $(2,1)$  to  $(1,2)$ .

```
x=2-t
y=t+1
defint(y*d(x),t,0,1)
```

$$-\frac{3}{2}$$

Evaluate  $\int x dy$  along the straight line from  $(1,1)$  to  $(2,1)$ .

```
x=t+1
y=1
defint(x*d(y),t,0,1)
```

$$0$$

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

```
x=2t+1
y=t^2
z=1+t^3
f=z*d(x)+x*d(y)+y*d(z)
defint(f,t,0,1)
```

$$\frac{163}{30}$$



## 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)
```

$$\pi$$

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

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

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

As a practical matter,  $f(x, y)$  must be very simple in order for Eigenmath to solve the double integral.

Find the area of the spiral ramp defined by<sup>2</sup>

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

In this example, the coordinates  $x$ ,  $y$  and  $z$  are all functions of an independent parameter space.

```
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

---

<sup>2</sup>Williamson and Trotter, *Multivariable Mathematics*, p. 598.

## 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  $dA$ .

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

Hence

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

For example, 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.<sup>3</sup>

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)
```

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

---

<sup>3</sup>Kaplan, *Advanced Calculus*, p. 313.

## Green's theorem

Green's theorem tells us that

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

Evaluate  $\oint (2x^3 - y^3) dx + (x^3 + y^3) dy$  around the circle  $x^2 + y^2 = 1$  using Green's theorem.<sup>4</sup>

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)
f=eval(f)
defint(f*r,r,0,1,theta,0,2pi)
```

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

A few words of explanation are in order. The line  $f = eval(f)$  is necessary to update  $f$  with the polar substitutions for  $x$  and  $y$ . 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)
```

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

---

<sup>4</sup>Wilfred Kaplan, *Advanced Calculus, 5th Edition*, 287.

## Stokes' theorem

Stokes' theorem proves the following equivalence of line and surface integrals.

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

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

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

In many cases, converting an integral according to Stokes' theorem can turn a difficult problem into an easy one.

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  $x^2 + y^2 = 2$ . Calculate both the line and surface integrals. It turns out that we need to use polar coordinates 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)
f=eval(f)
defint(f*r,r,0,2,theta,0,2pi)
```

−4π

```
--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)
```

−4π

## François Viète

François Viète was the first to discover an exact formula for  $\pi$ . Here is his formula.

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

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

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

Solving for  $\pi$  we have

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

Let us now use Eigenmath to compute  $\pi$  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

## Curl in tensor form

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

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

where  $\epsilon_{ijk}$  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
```

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

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
```

$$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$



## 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  $n$ th 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

## Hydrogen wavefunctions

Hydrogen wavefunctions  $\psi$  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  $\psi$  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

$$1 \leq l \leq n-1, \quad -l \leq m \leq l$$

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

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

The following code 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)
```

## 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

## **abs**

$\text{abs}(x)$  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**

$\text{adj}(m)$  returns the adjunct of matrix  $m$ .

## **and**

$\text{and}(a, b, \dots)$  returns the logical “and” of predicate expressions.

## **arccos**

$\text{arccos}(x)$  returns the inverse cosine of  $x$ .

## **arccosh**

$\text{arccosh}(x)$  returns the inverse hyperbolic cosine of  $x$ .

## **arcsin**

$\text{arcsin}(x)$  returns the inverse sine of  $x$ .

## **arcsinh**

$\text{arcsinh}(x)$  returns the inverse hyperbolic sine of  $x$ .

## **arctan**

$\text{arctan}(x)$  returns the inverse tangent of  $x$ .

## **arctanh**

$\text{arctanh}(x)$  returns the inverse hyperbolic tangent of  $x$ .

## **arg**

$\text{arg}(z)$  returns the angle of complex  $z$ .

## **ceiling**

$\text{ceiling}(x)$  returns the smallest integer not less than  $x$ .

## **check**

$\text{check}(x)$  In a script, if the predicate  $x$  is true then continue, else stop.

## **choose**

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

## **circexp**

$\text{circexp}(x)$  returns expression  $x$  with circular functions converted to exponential forms. Sometimes this will simplify an expression.

## **coeff**

$\text{coeff}(p, x, n)$  returns the coefficient of  $x^n$  in polynomial  $p$ .

## **cofactor**

$\text{cofactor}(m, i, j)$  returns of the cofactor of matrix  $m$  with respect to row  $i$  and column  $j$ .

## **conj**

$\text{conj}(z)$  returns the complex conjugate of  $z$ .

## **contract**

$\text{contract}(a, i, j)$  returns tensor  $a$  summed over indices  $i$  and  $j$ . If  $i$  and  $j$  are omitted then indices 1 and 2 are used.  $\text{contract}(m)$  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**

$\text{cross}(u, v)$  returns the cross product of vectors  $u$  and  $v$ .

## **curl**

$\text{curl}(u)$  returns the curl of vector  $u$ .

## **d**

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

## **defint**

$\text{defint}(f, x, a, b, \dots)$  returns the definite integral of  $f$  with respect to  $x$  evaluated from  $a$  to  $b$ . The argument list can be extended for multiple integrals. For example,  $d(f, x, a, b, y, c, d)$ .

## **deg**

$\text{deg}(p, x)$  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( $a, b, \dots$ )` evaluates the argument list from left to right. Returns the result of the last argument.

## **dot**

`dot( $a, b, \dots$ )` 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  $e^x$ .

## **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, ...)` 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, ...)` 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( $a, b, \dots$ )` returns the greatest common divisor.

## **hermite**

`hermite( $x, n$ )` returns the  $n$ th 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( $a, b, \dots$ )` 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( $2^{53}-111$ )`

## **laguerre**

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

## **lcm**

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

## **legendre**

`legendre( $x, n, m$ )` returns the  $n$ th 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( $a, b, \dots$ )` returns the logical “or” of predicate expressions.

## **outer**

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

## **polar**

`polar( $z$ )` converts complex  $z$  to polar form.

## **prime**

`prime( $n$ )` returns the  $n$ th prime number,  $1 \leq n \leq 10,000$ .

## **print**

`print( $a, b, \dots$ )` evaluates expressions and prints the results.. Useful for printing from inside a “for” loop.

## **product**

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

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

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

## unit

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

```
unit(2)
```

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

## zero

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

## Index

built-in functions, 36  
complex numbers, 14  
derivative, 19  
fundamental theorem of calculus, 23  
gradient, 19  
Green's theorem, 28  
hydrogen wavefunctions, 34  
integral, 21  
line integral, 24  
linear algebra, 16  
scripts, 12  
Stokes' theorem, 29  
surface integral, 27  
trace, 38  
units of measure, 8