

multivariate\_commands

last edited on July 26, 2009 08:55 PM by admin

Save Save & quit  
Discard & quit

Print Worksheet [Edit Text](#)  
[Undo](#) [Share](#) [Publish](#)

File... Action...  
Data... sage  Typeset

> >> Exit

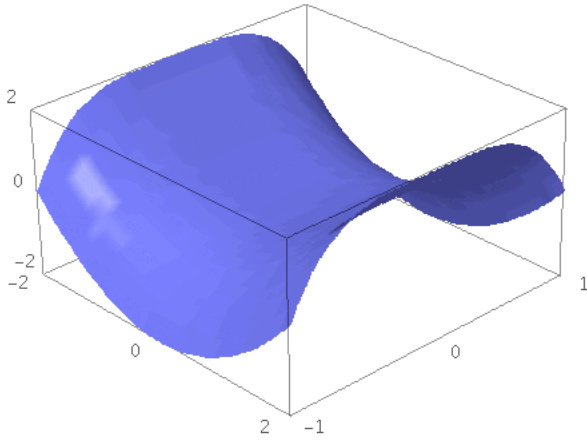
<< <

## A Smattering of Multivariate Calculus Commands

Multivariate calculus is actually one of *Sage's* weaker areas. *Sage's* 3D graphics capability is somewhat awkward. *Sage* lacks a vector field class. The authors of *Sage* are clearly more interested in algebra and in number theory than in analysis. That said, *Sage* can still be used for the basic tools we need in Multivariate Calculus.

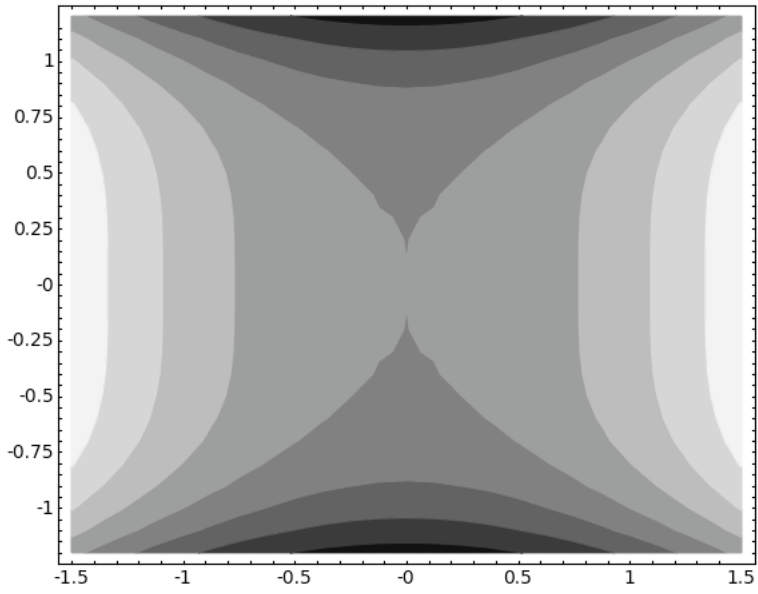
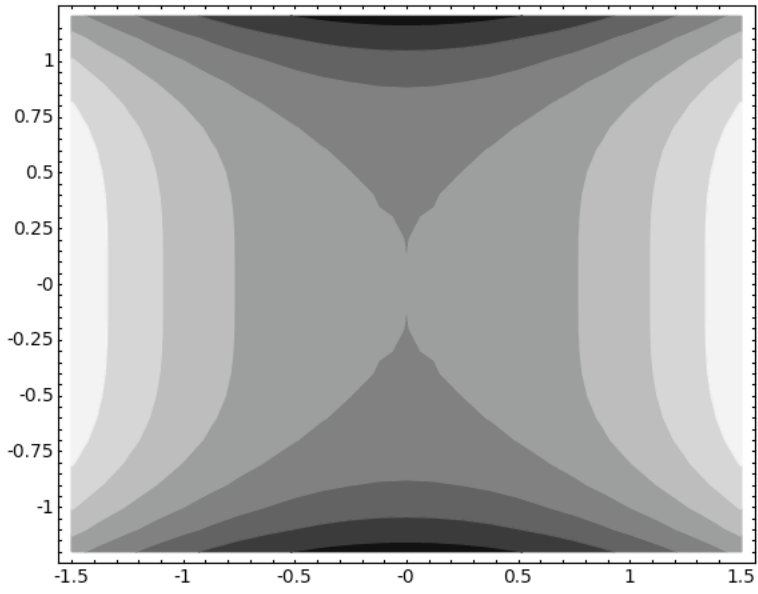
### 3D Plots

```
var('x y')  
plot3d(x^2-y^4, [x,-1.5,1.5], [y,-1.2,1.2]) evaluate
```

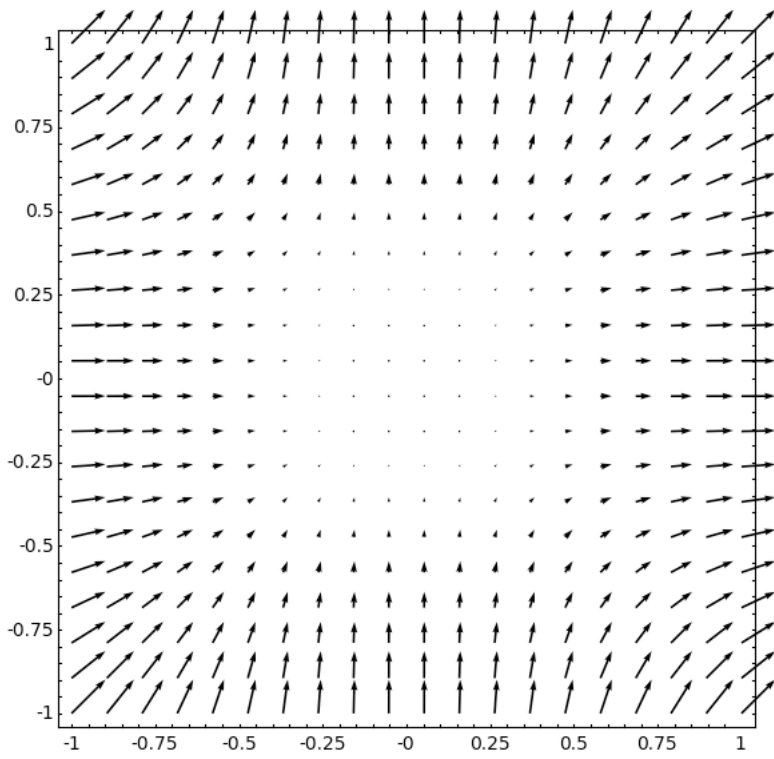
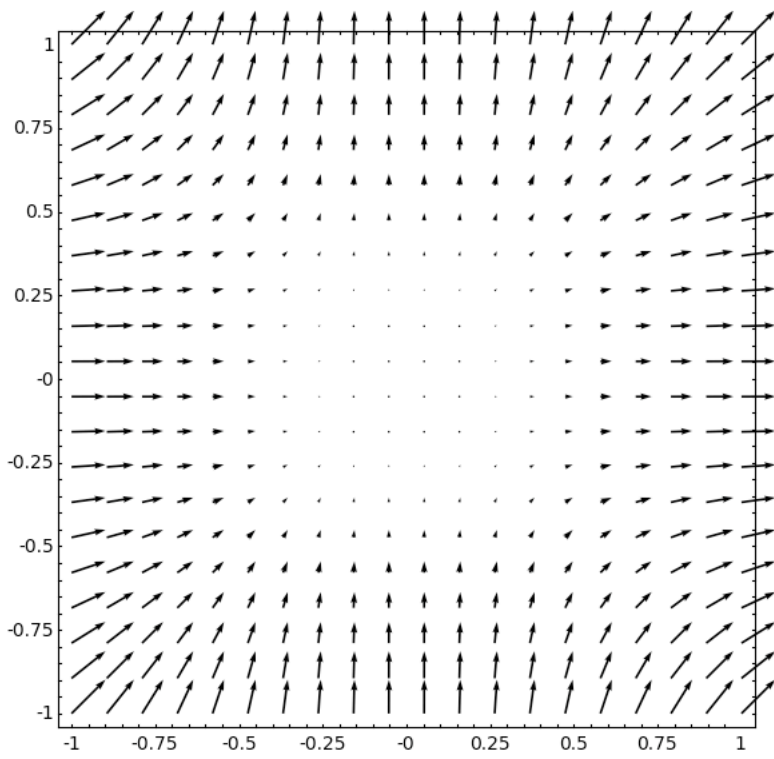


[Get Image](#)

```
show(contour_plot(x^2-y^4, [x,-1.5,1.5], [y,-1.2,1.2]), aspect_ratio=1) evaluate
```

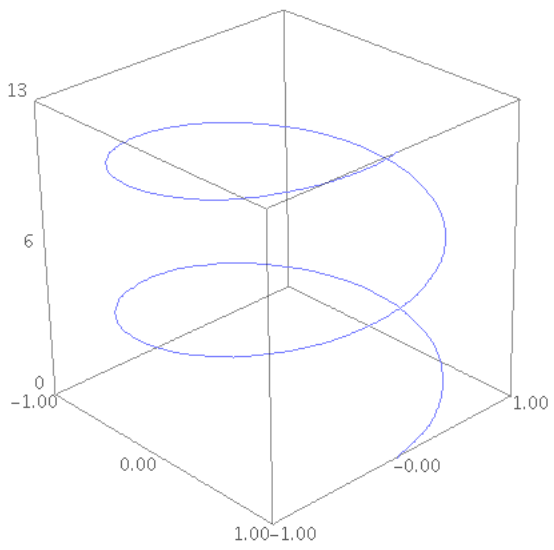


```
show(plot_vector_field([x^2, y^2], [x,-1,1], [y,-1,1]), aspect_ratio=1) evaluate
```



```
var('t')  
parametric_plot3d((cos(t), sin(t), t), [t, 0, 4*pi])
```

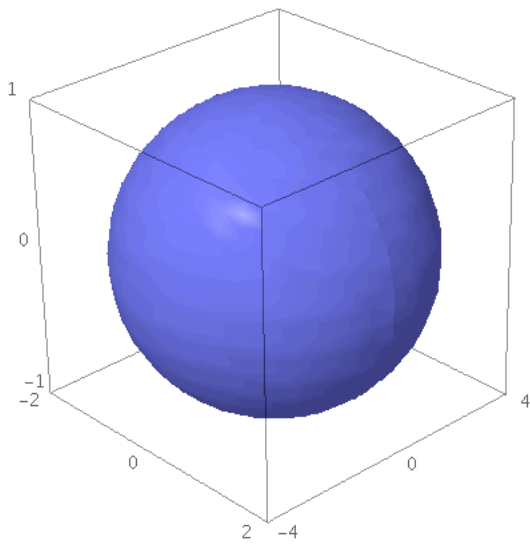
[evaluate](#)



[Get Image](#)

```
var('theta phi')
parametric_plot3d([2*cos(theta)*sin(phi), 4*sin(theta)*sin(phi), cos(phi)],
[theta, 0, 2*pi], [phi, 0, pi])
```

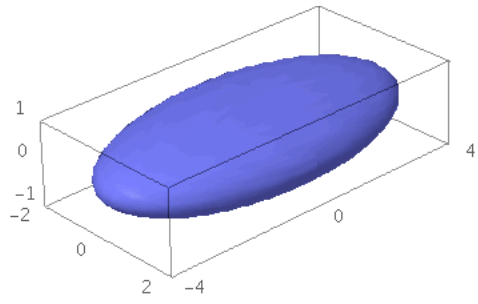
[evaluate](#)



[Get Image](#)

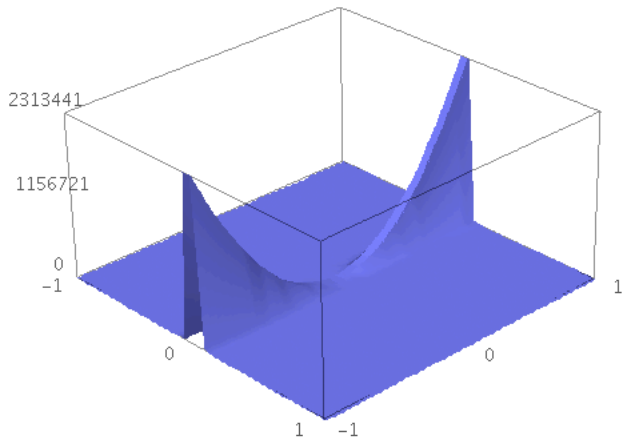
```
show(_, aspect_ratio=1)
```

[evaluate](#)



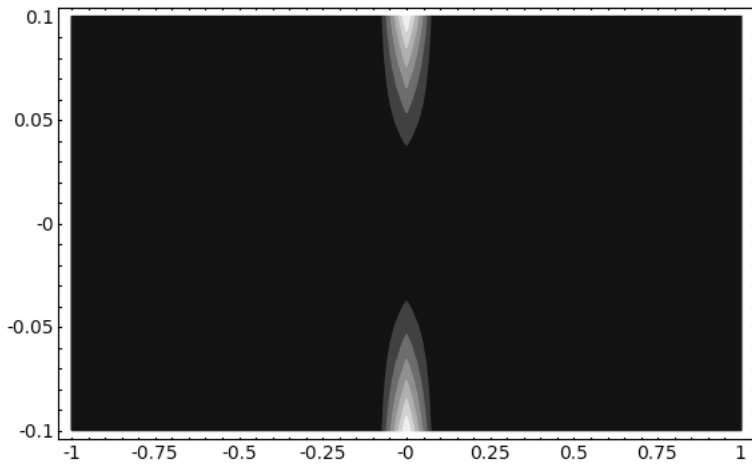
[Get Image](#)

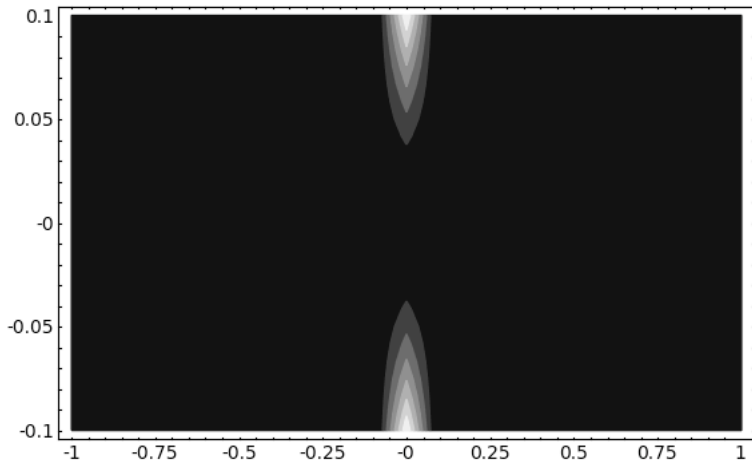
`plot3d(y^2/x^4, [x,-1,1], [y,-1,1])` [evaluate](#)



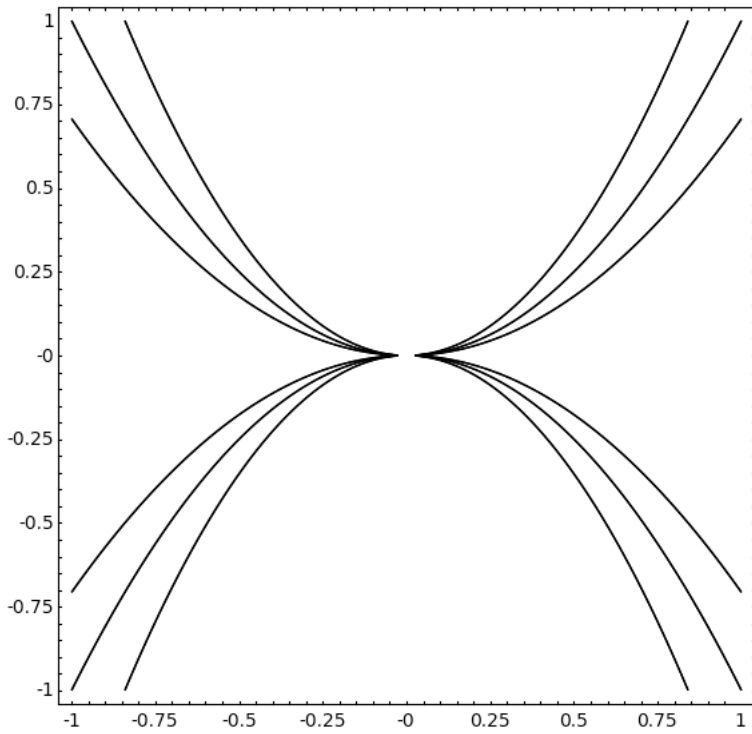
[Get Image](#)

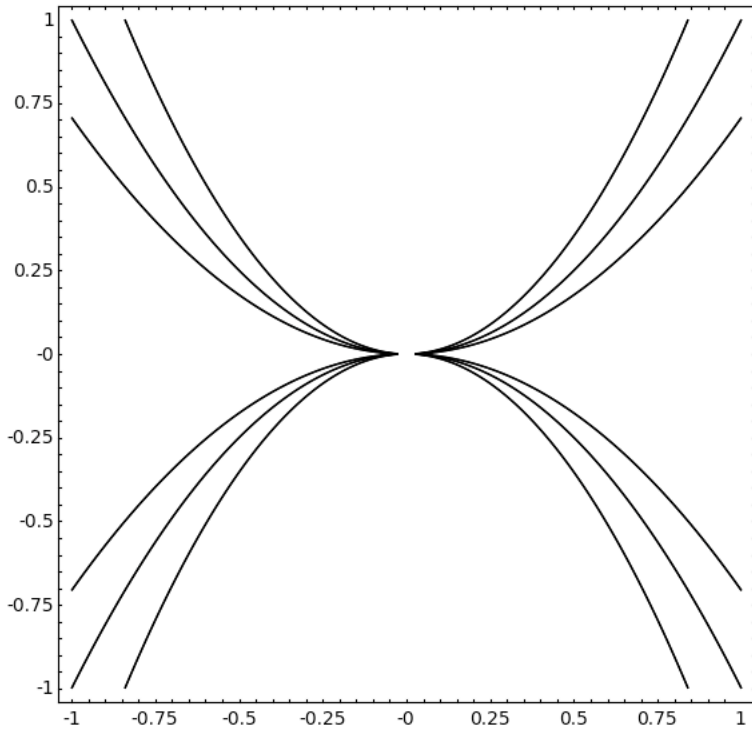
`contour_plot(y^2/x^4, [x,-1,1], [y,-0.1,0.1])` [evaluate](#)





```
p1 = implicit_plot(y^2/x^4==1, (x,-1,1), (y,-1,1), plot_points=1000)
p2 = implicit_plot(y^2/x^4==2, (x,-1,1), (y,-1,1), plot_points=1000)
phalf = implicit_plot(y^2/x^4==1/2, (x,-1,1), (y,-1,1), plot_points=1000)
show(p1+p2+phalf, aspect_ratio=1) evaluate
```





## Vectors and Matrices

Matrices and vectors work pretty much the way you would expect, except that in keeping with a number of programming languages, entries and rows and columns are numbered starting at 0, not 1.

`v = vector([3,1,4])`  
`v` [evaluate](#)

(3, 1, 4)

(3, 1, 4)

`v[0]` [evaluate](#)

3

3

`v[1]` [evaluate](#)

1

1

`v[2]` [evaluate](#)

4

4

`u = vector([1,-3,2])` [evaluate](#)

`v.dot_product(u)` [evaluate](#)

8

8

`v*u` [evaluate](#)

8

8

Notice that in this last product,  $v$  was treated as a row vector, and  $u$  as a column vector.

`v.cross_product(u)` [evaluate](#)

$(14, -2, -10)$

$(14, -2, -10)$

`M = matrix([[1,2,3],[3,1,4],[-1,1,1]])`  
`M` [evaluate](#)

$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 4 \\ -1 & 1 & 1 \end{pmatrix}$

$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 4 \\ -1 & 1 & 1 \end{pmatrix}$

`M[1,2]` [evaluate](#)

4

4

`M.determinant()` [evaluate](#)

-5

-5

`M.inverse()` [evaluate](#)

$\begin{pmatrix} \frac{3}{5} & -\frac{1}{5} & -1 \\ \frac{7}{5} & -\frac{4}{5} & -1 \\ -\frac{4}{5} & \frac{3}{5} & 1 \end{pmatrix}$

$\begin{pmatrix} \frac{3}{5} & -\frac{1}{5} & -1 \\ \frac{7}{5} & -\frac{4}{5} & -1 \\ -\frac{4}{5} & \frac{3}{5} & 1 \end{pmatrix}$

`M^2` [evaluate](#)

$\begin{pmatrix} 4 & 7 & 14 \\ 2 & 11 & 17 \\ 1 & 0 & 2 \end{pmatrix}$

$\begin{pmatrix} 4 & 7 & 14 \\ 2 & 11 & 17 \\ 1 & 0 & 2 \end{pmatrix}$

`M.charpoly()` [evaluate](#)

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

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

`M.eigenvalues()` [evaluate](#)

$[-1.571201422548122?, 0.8567226781603571?, 3.714478744387765?]$

$[-1.571201422548122?, 0.8567226781603571?, 3.714478744387765?]$

`print(M.eigenvectors_left())` [evaluate](#)

$[(-1.571201422548122?, [(1, -0.8205055024661761?, 0.10968491514959334?)], 1), (0.8567226781603571?, [(1, -0.6818607149130776?, -1.902304822899590?)], 1), (3.714478744387765?, [(1, 2.502366217379254?, 4.792619907749997?)], 1)]$

$[(-1.571201422548122?, [(1, -0.8205055024661761?, 0.10968491514959334?)], 1), (0.8567226781603571?, [(1, -0.6818607149130776?, -1.902304822899590?)], 1)]$

`M*v` [evaluate](#)

(17, 26, 2)

(17, 26, 2)

Here  $\mathbf{v}$  was viewed as a column vector.

`v*M` [evaluate](#)

(2, 11, 17)

(2, 11, 17)

And here  $\mathbf{v}$  was a row vector.

Although matrices can be written as simply as we did above, *Sage* would really rather we specified what ring the entries of any matrix lie in: the integers  $\mathbb{Z}$ , which *Sage* writes as  $\mathbf{ZZ}$ , the rationals  $\mathbb{Q}$ , which *Sage* calls  $\mathbf{QQ}$ , the reals  $\mathbb{R}$ , which is roughly *Sage*'s  $\mathbf{RR}$ , the integers mod 7,  $\mathbb{Z}_7$ , which is *Sage*'s  $\mathbf{Zmod}(7)$ , etc. Depending on what ring the entries lie in, some matrix operations will behave differently.

`M = matrix(QQ, [[1,2],[3,4]])`  
`M` [evaluate](#)

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

`M^2` [evaluate](#)

$$\begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$$
$$\begin{pmatrix} 7 & 10 \\ 15 & 22 \end{pmatrix}$$

`M^(-1)` [evaluate](#)

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

`M7 = matrix(Zmod(7), [[1,2],[3,4]])`  
`M7` [evaluate](#)

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

`M7^2` [evaluate](#)

$$\begin{pmatrix} 0 & 3 \\ 1 & 1 \end{pmatrix}$$
$$\begin{pmatrix} 0 & 3 \\ 1 & 1 \end{pmatrix}$$

`M7^(-1)` [evaluate](#)

$$\begin{pmatrix} 5 & 1 \\ 5 & 3 \end{pmatrix}$$
$$\begin{pmatrix} 5 & 1 \\ 5 & 3 \end{pmatrix}$$

`MQ = matrix(QQ, [[6,2,1],[10,3,6],[4,1,5]])`  
`MQ` [evaluate](#)

$$\begin{pmatrix} 6 & 2 & 1 \\ 10 & 3 & 6 \\ 4 & 1 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 6 & 2 & 1 \\ 10 & 3 & 6 \\ 4 & 1 & 5 \end{pmatrix}$$

[evaluate](#)

[evaluate](#)

$$\begin{pmatrix} 1 & 0 & \frac{9}{2} \\ 0 & 1 & -13 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & \frac{9}{2} \\ 0 & 1 & -13 \\ 0 & 0 & 0 \end{pmatrix}$$

[evaluate](#)

$$\begin{pmatrix} 6 & 2 & 1 \\ 10 & 3 & 6 \\ 4 & 1 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 6 & 2 & 1 \\ 10 & 3 & 6 \\ 4 & 1 & 5 \end{pmatrix}$$

[evaluate](#)

$$\begin{pmatrix} 2 & 0 & 9 \\ 0 & 1 & -13 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 2 & 0 & 9 \\ 0 & 1 & -13 \\ 0 & 0 & 0 \end{pmatrix}$$

[evaluate](#)

[0, 1]

[0, 1]

As a tool in teaching and learning, it's cute that *Sage* lets one do Gaussian elimination a step at a time, like this:

(Notice that the row operations don't return a new matrix, but that they modify the original matrix in place.)

[evaluate](#)

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

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

[evaluate](#)

$$\begin{pmatrix} 1 & 2 \\ 0 & -2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 0 & -2 \end{pmatrix}$$

```
M.rescale_row(1, -1/2)
M
```

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

```
M.add_multiple_of_row(0,1,-2)
M
```

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

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

```
var('x y z w')
vandermonde = matrix(list(list(k^n for n in range(4)) for k in [x,y,z,w]))
vandermonde
```

$$\begin{pmatrix} 1 & x & x^2 & x^3 \\ 1 & y & y^2 & y^3 \\ 1 & z & z^2 & z^3 \\ 1 & w & w^2 & w^3 \end{pmatrix}$$

$$\begin{pmatrix} 1 & x & x^2 & x^3 \\ 1 & y & y^2 & y^3 \\ 1 & z & z^2 & z^3 \\ 1 & w & w^2 & w^3 \end{pmatrix}$$

```
vdet = vandermonde.det()
print(vdet)
```

$$\begin{aligned} & ((x - y)*w^2 - (w - y)*x^2 + (w - x)*y^2)*z^3 - ((x - z)*w^2 - (w - z)*x^2 + (w - x)*z^2)*y^3 - ((y - z)*x^2 - (x - z)*y^2 + (x - y)*z^2)*w^3 + ((y - z)*w^2 - (w - z)*y^2 + (w - y)*z^2)*x^3 \\ & ((x - y)*w^2 - (w - y)*x^2 + (w - x)*y^2)*z^3 - ((x - z)*w^2 - (w - z)*x^2 + (w - x)*z^2)*y^3 - ((y - z)*x^2 - (x - z)*y^2 + (x - y)*z^2)*w^3 + \end{aligned}$$

```
factor(vdet)
```

$$\begin{aligned} & -(y - z)(x - z)(x - y)(w - z)(w - y)(w - x) \\ & -(y - z)(x - z)(x - y)(w - z)(w - y)(w - x) \end{aligned}$$

### Div, Grad, Curl, and All That

Weirdly, Sage has gradients built in, but not divergence or curl or even the vector fields for which these operators would make sense. Of course, we can implement these things ourselves, but it's somewhat shocking that we need to. Here are some very quick and dirty definitions of these functions assuming that the variables involved are  $x, y,$  and  $z$ . The Sage wiki has a somewhat more professional version, but obviously Sage needs someone to build a proper vector field class.

```
var('t x y z')
f(x,y,z) = sin(x) - cos(y) + z
f(x,y,z)
```

$$\begin{aligned} & z + \sin(x) - \cos(y) \\ & z + \sin(x) - \cos(y) \end{aligned}$$

```
gr = f(x,y,z).gradient()
gr
```

$$\begin{aligned} & (\cos(x), \sin(y), 1) \\ & (\cos(x), \sin(y), 1) \end{aligned}$$

```
gr.derivative(x)
```

$$\begin{aligned} & (-\sin(x), 0, 0) \\ & (-\sin(x), 0, 0) \end{aligned}$$

```
def divergence(F):
    assert(len(F) == 3)
    return diff(F[0],x) + diff(F[1],y) + diff(F[2],z)
```

[evaluate](#)

```
gr
```

[evaluate](#)

$(\cos(x), \sin(y), 1)$

$(\cos(x), \sin(y), 1)$

```
divergence(gr)
```

[evaluate](#)

$-\sin(x) + \cos(y)$

$-\sin(x) + \cos(y)$

```
divergence([x,y,z])
```

[evaluate](#)

3

3

```
def curl(F):
    assert(len(F) == 3)
    return vector([diff(F[2],y)-diff(F[1],z), diff(F[0],z)-diff(F[2],x), diff(F[1],x)-diff(F[0],y)])
```

[evaluate](#)

```
curl(gr)
```

[evaluate](#)

$(0, 0, 0)$

$(0, 0, 0)$

```
curl([-y, x, 0])
```

[evaluate](#)

$(0, 0, 2)$

$(0, 0, 2)$

```
gr[1]
```

[evaluate](#)

$\sin(y)$

$\sin(y)$

[evaluate](#)