

# Maple Tips

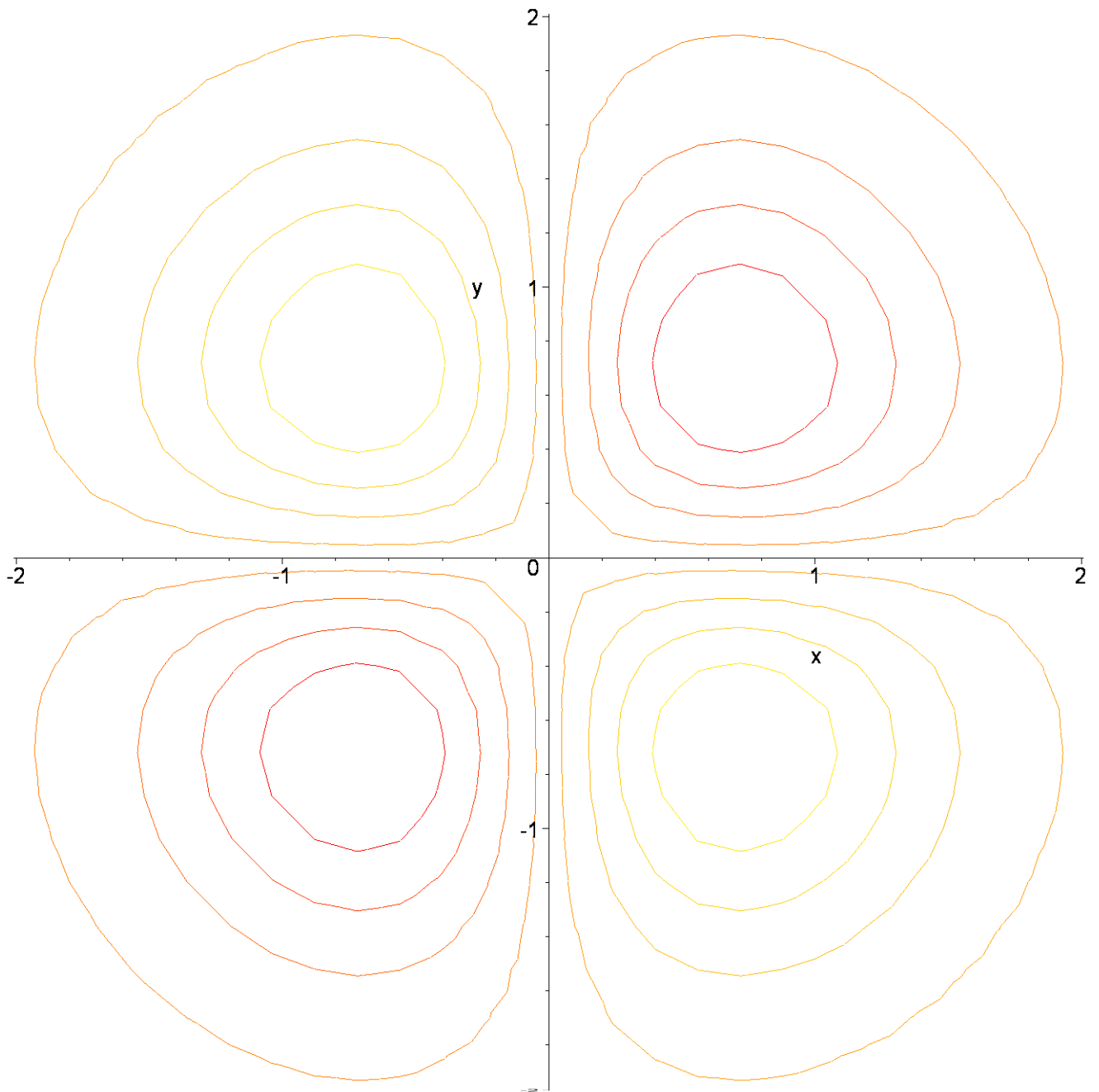
## Chapter 11

### Lesson 11-1

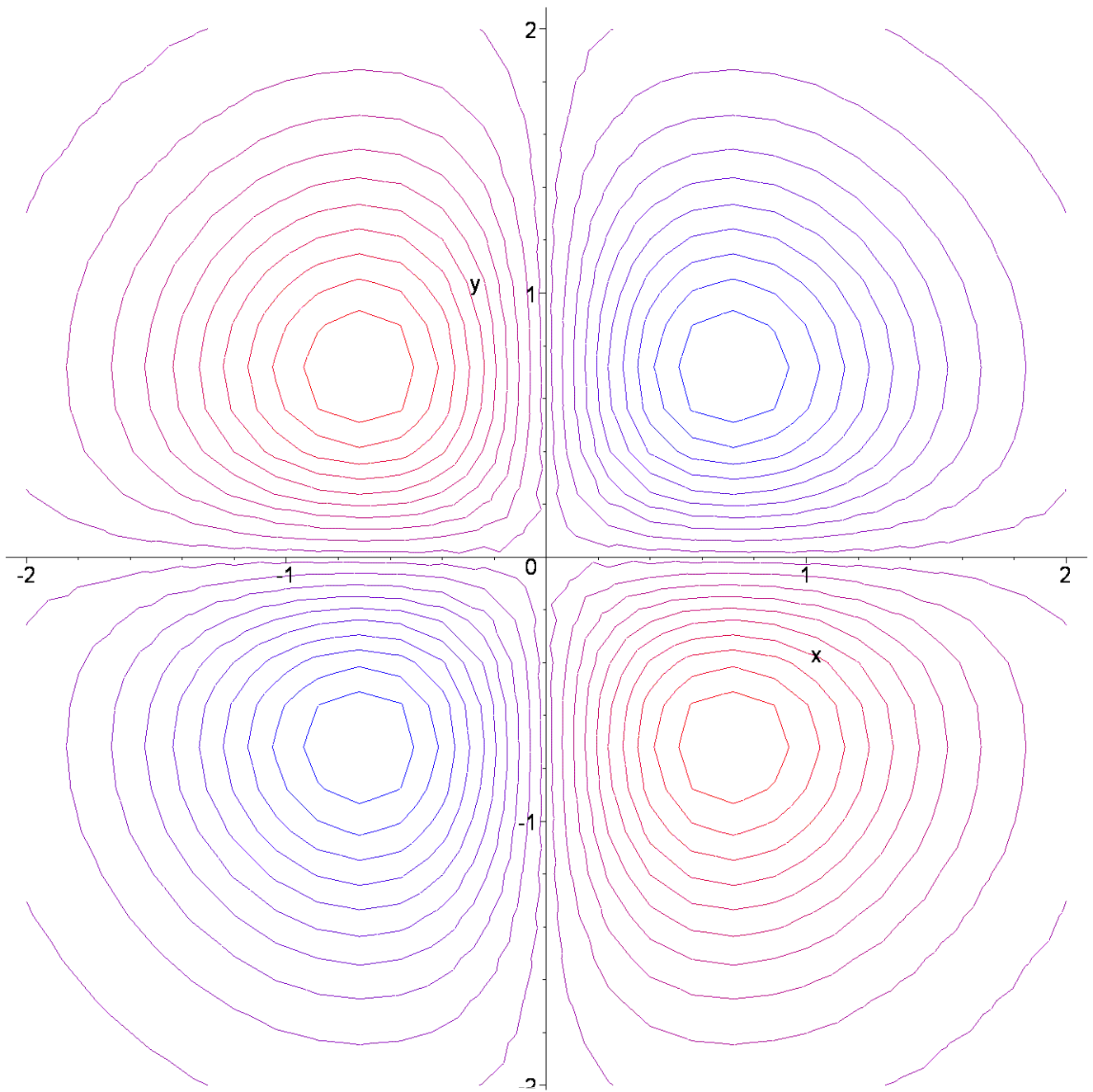
#### Graphing Level Curves of a Function

There are several types and methods for graphing level curves or contour plots of a function. The **contourplot** command in the **plots** package is probably the simplest. The default coloring is red to yellow, but adding the option **coloring=[blue,red]** often looks better. There are several other options available with this command, and some of the more important ones are shown below.

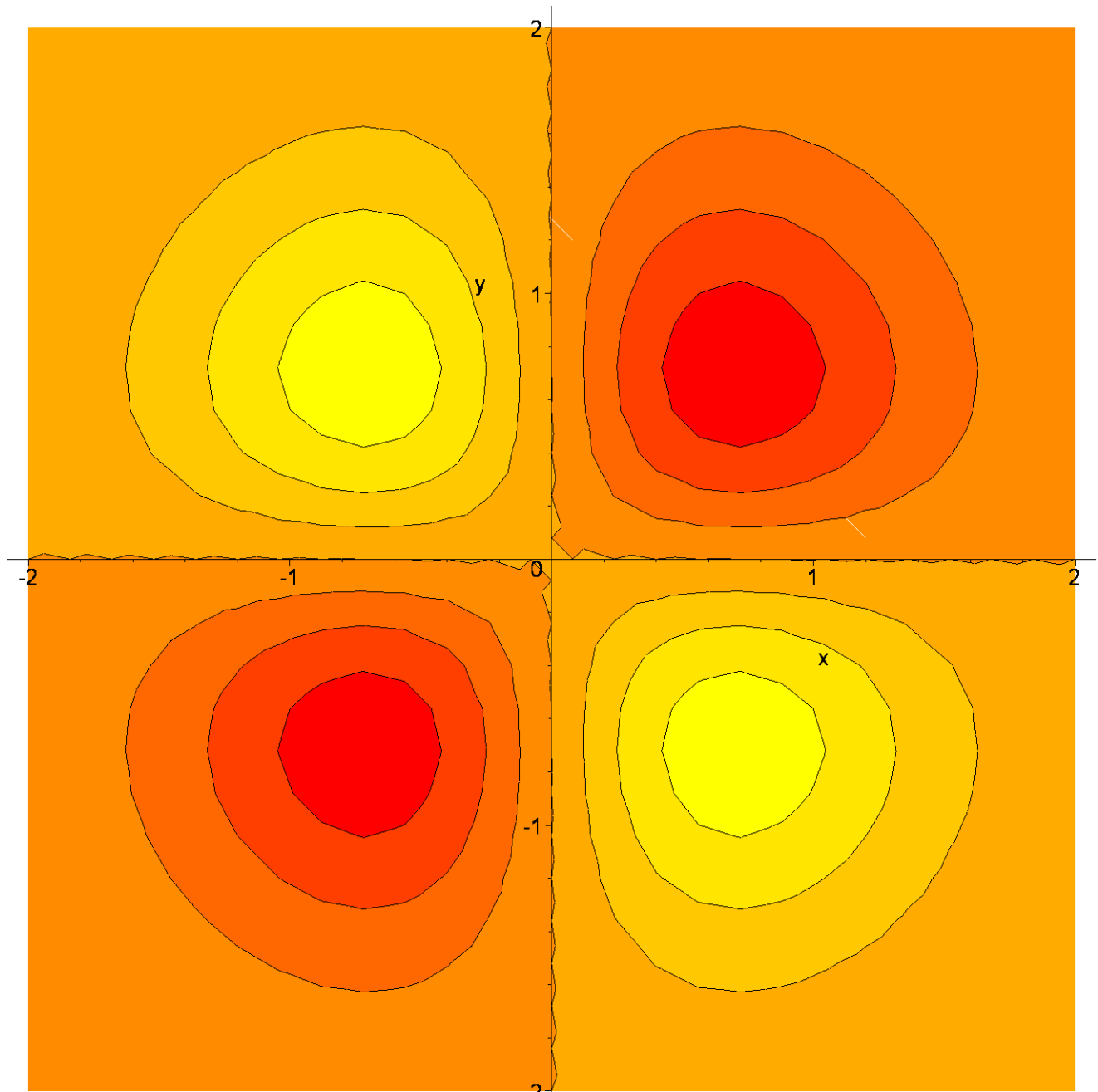
```
> restart: with(plots):  
Warning, the name changecoords has been redefined  
> f:=(x,y)->-x*y*exp(-x^2-y^2): f(x,y);  
-x y e(-x2-y2)  
> contourplot(f(x,y), x=-2..2, y=-2..2);
```



```
> contourplot(f(x,y), x=-2..2, y=-2..2, coloring=[blue,red],  
contours=20);
```



```
> contourplot(f(x,y), x=-2..2, y=-2..2, filled=true,  
contours=[-0.15,-0.1,-0.05,0,0.05,0.1,0.15]);
```



## Graphing Surfaces to Show Level Curves

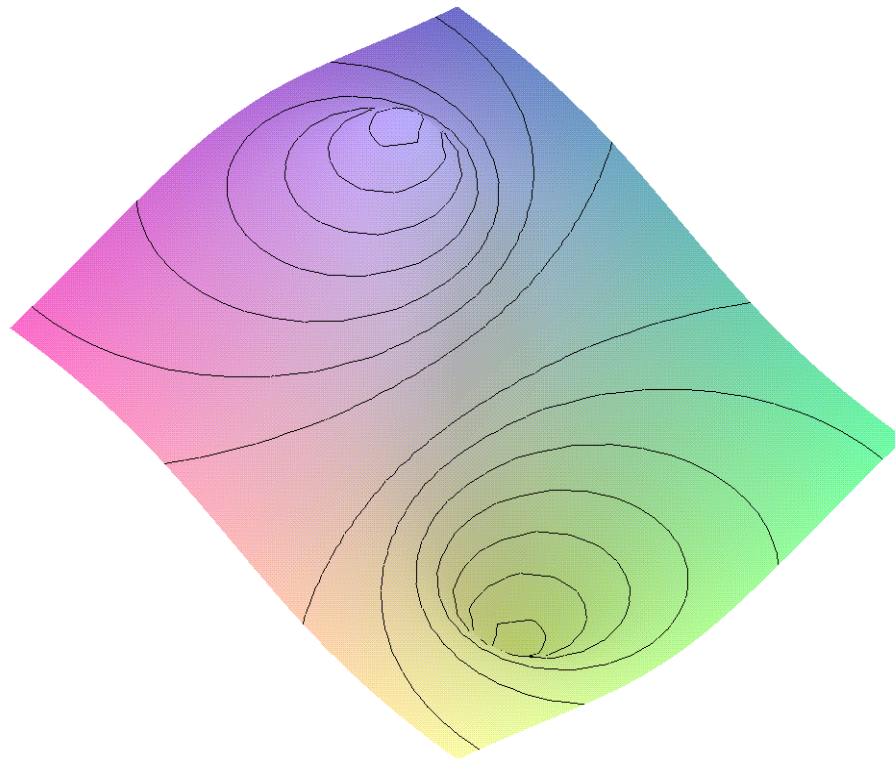
Simply changing the style option in the `plot3d` command with `style=patchcontour` will illustrate the level curves along with the three-dimensional surface. The option `style=contour` graphs only the contours. The `contourplot3d` command in the `plots` package can also be used to graph level curves three-dimensionally, but with more options (such as coloring and adjusting the number of contours drawn).

```
> restart: with(plots):
Warning, the name changecoords has been redefined
```

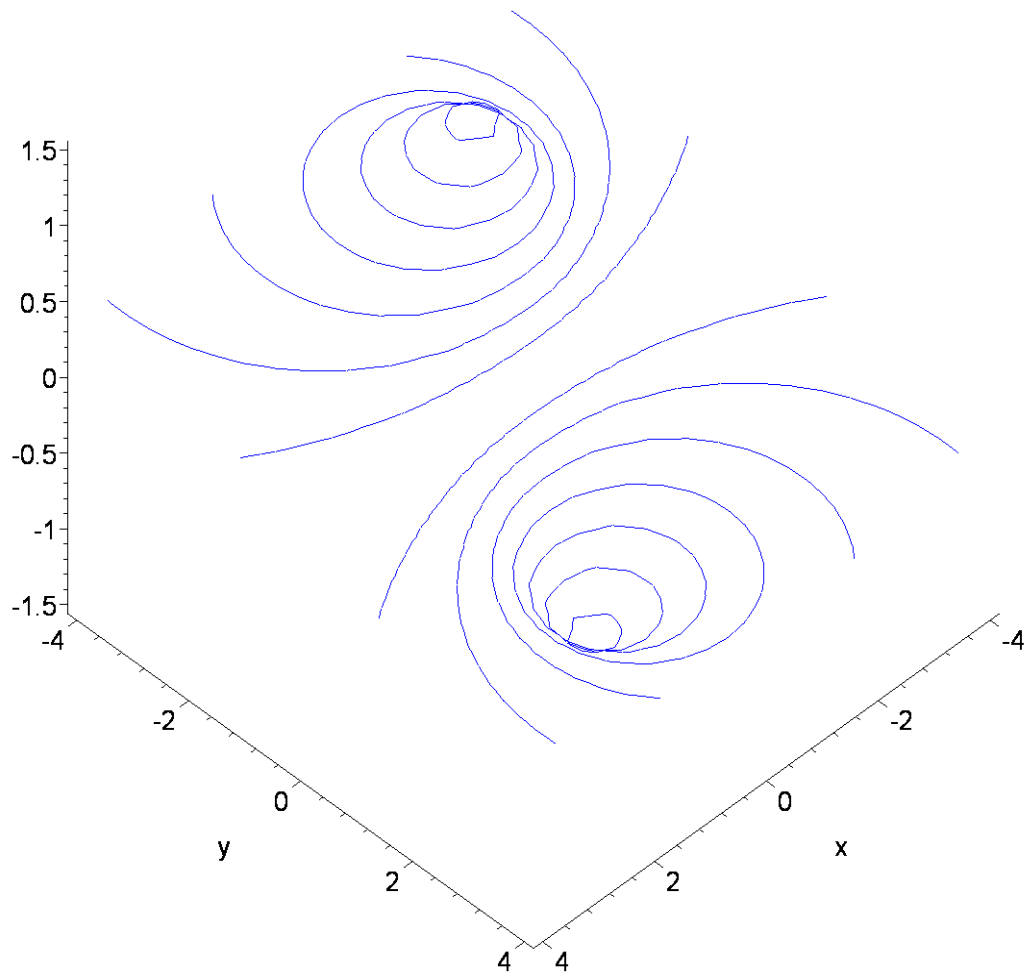
```
> g:=(x,y)->-3*y/(x^2+y^2+1): g(x,y);
```

$$-\frac{3y}{x^2+y^2+1}$$

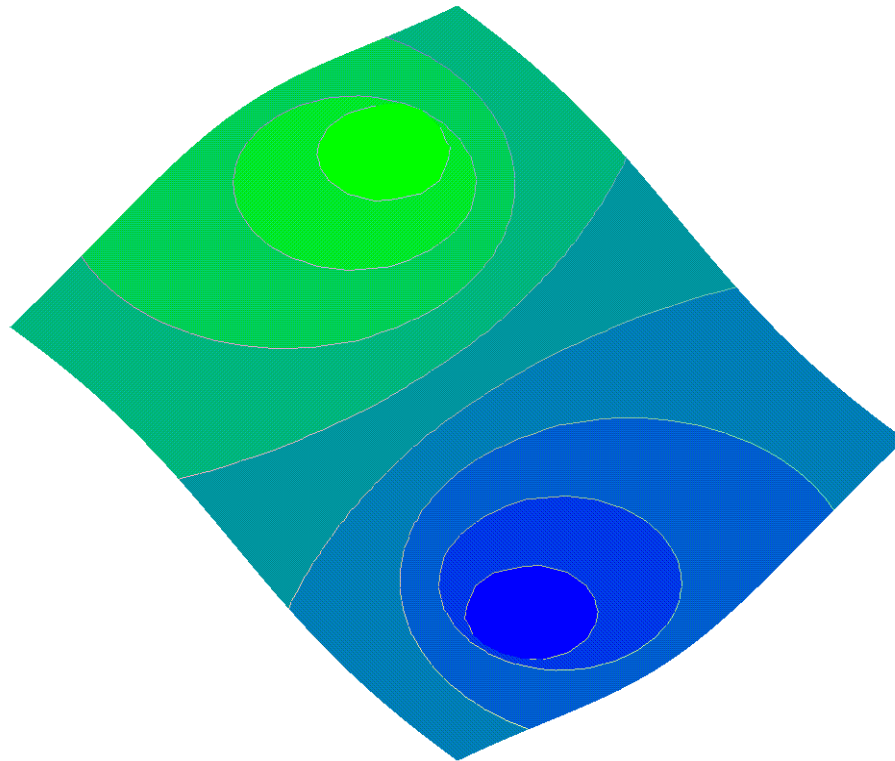
```
> plot3d(g(x,y), x=-4..4, y=-4..4, style=patchcontour);
```



```
> plot3d(g(x,y), x=-4..4, y=-4..4, style=contour, color=blue, axes=frame);
```



```
> contourplot3d(g(x,y), x=-4..4, y=-4..4, filled=true,  
coloring=[blue,green]);
```



## Lesson 11-2

### Finding the Limit of a Multivariable Function

The **limit** command can be used to evaluate some multivariable limits. If the limit evaluation is too "hard" for the CAS, it simply returns the input.

```
> restart;  
> limit(x^2*y^3-x^3*y^2+3*x+2*y, {x=1, y=2});
```

```
> limit((x^2-y^2)/(x^2+y^2), {x=0, y=0});
```

*undefined*

```
> limit(sin(x^2+y^2)/(x^2+y^2), {x=0, y=0});
```

$$\lim_{\{x=0, y=0\}} \left( \frac{\sin(x^2 + y^2)}{x^2 + y^2} \right)$$

## Lesson 11-3

### Calculating the Partial Derivative of a Multivariable Function

The **diff** command can be used to evaluate partial derivatives to functions. The **eval** command is needed for numerical partial derivatives. The **diff** command can be nested for second partial derivatives.

```
> restart;
```

```
> f:=(x,y)->x^3+x^2*y^3-2*y^2; f(x,y);
```

$$f := (x, y) \rightarrow x^3 + x^2 y^3 - 2 y^2$$
$$x^3 + x^2 y^3 - 2 y^2$$

```
> Diff(f(x,y), x)=diff(f(x,y), x);
```

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

```
> Diff(f(x,y), y)=diff(f(x,y), y);
```

$$\frac{\partial}{\partial y} (x^3 + x^2 y^3 - 2 y^2) = 3 x^2 y^2 - 4 y$$

```
> eval(diff(f(x,y), x), {x=1, y=2});
```

19

```
> eval(diff(f(x,y), y), {x=1, y=2});
```

4

```
> Diff(Diff(f(x,y), x), x)=diff(diff(f(x,y), x), x);
```

$$\frac{\partial^2}{\partial x^2} (x^3 + x^2 y^3 - 2 y^2) = 6 x + 2 y^3$$

```
> Diff(Diff(f(x,y), y), y)=diff(diff(f(x,y), y), y);
```

$$\frac{\partial^2}{\partial y^2} (x^3 + x^2 y^3 - 2 y^2) = 6 x^2 y - 4$$

```
> Diff(Diff(f(x,y), x), y)=diff(diff(f(x,y), x), y);
```

$$\frac{\partial^2}{\partial y \partial x} (x^3 + x^2 y^3 - 2 y^2) = 6 x y^2$$

```
> Diff(Diff(f(x,y), y), x)=diff(diff(f(x,y), y), x);
```

$$\frac{\partial^2}{\partial x \partial y} (x^3 + x^2 y^3 - 2 y^2) = 6 x y^2$$

## Implicit Differentiation Using Partial Derivatives

The **implicitdiff** command can be used to evaluate partial derivatives to when given an implicitly defined relation.

```
> restart;
```

```
> eq:=x^3+y^3+z^3+6*x*y*z=1;
```

$$eq := x^3 + y^3 + z^3 + 6 x y z = 1$$

```
> Diff(z,x)=implicitdiff(eq, z, x);
```

$$\frac{\partial}{\partial x} z = -\frac{x^2 + 2 y z}{z^2 + 2 x y}$$

```
> Diff(z,y)=implicitdiff(eq, z, y);
```

$$\frac{\partial}{\partial y} z = -\frac{y^2 + 2 x z}{z^2 + 2 x y}$$

## Lesson 11-4

### Finding and Graphing the Tangent Plane to a Surface

The following example shows how to use previously defined commands to find and graph the tangent plane to a function's surface at a point.

```
> restart;
```

```
> f:=(x,y)->2*x^2+y^2: f(x,y);
```

$$2 x^2 + y^2$$

```
> a:=1; b:=1;
```

$$a := 1$$

$$b := 1$$

```
> fx:=eval(diff(f(x,y), x), {x=a, y=b});
```

$$fx := 4$$

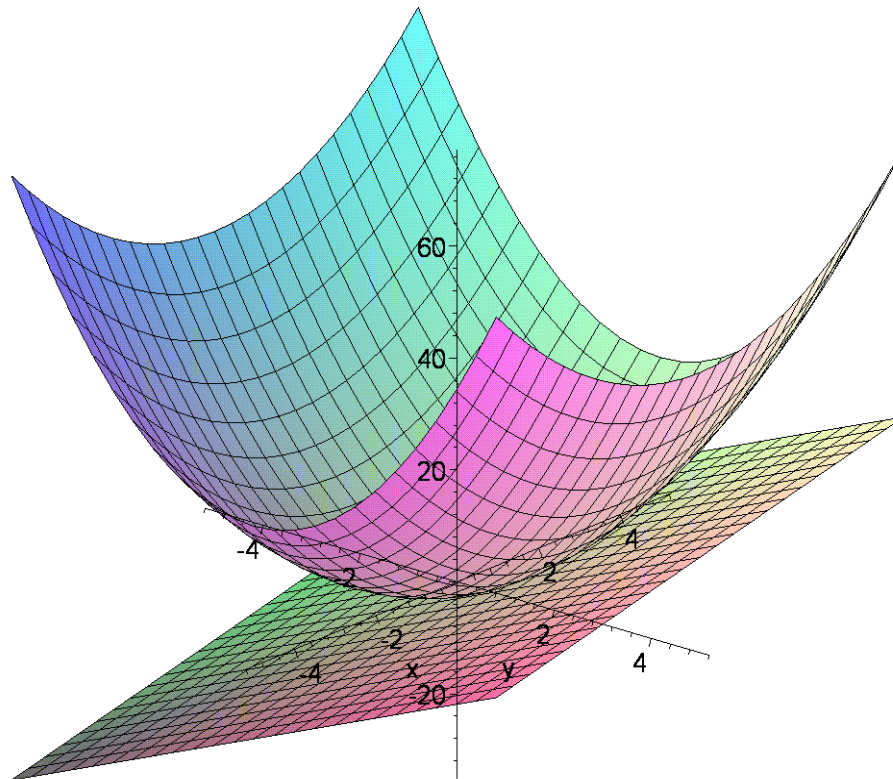
```
> fy:=eval(diff(f(x,y), y), {x=a, y=b});
```

$$fy := 2$$

```
> l:=(x,y)->fx*(x-a)+fy*(y-b)+f(a,b): l(x,y);
```

$$4 x - 3 + 2 y$$

```
> plot3d([f(x,y), l(x,y)], x=-5..5, y=-5..5, axes=normal,
orientation=[-50,70]);
```



## Tangent Planes to Parametric Surfaces

This example also uses previously defined commands to find and graph the tangent plane to a parametrically defined surface at a point.

```
> restart: with(VectorCalculus): with(plots):
```

```
Warning, the assigned names <, > and <|> now have a global binding
```

```
Warning, these protected names have been redefined and unprotected: *, +, -, ., D, Vector, diff, int, limit, series
```

```
Warning, the name changecoords has been redefined
```

```

[ > r:=(u,v)-><u^2, v^2, u+2*v>: r(u,v);
      
$$u^2 \mathbf{e}_x + v^2 \mathbf{e}_y + (u+2v) \mathbf{e}_z$$

[ > a:=1; b:=1;
      
$$a := 1$$

      
$$b := 1$$

[ > r(a,b);
      
$$\mathbf{e}_x + \mathbf{e}_y + 3 \mathbf{e}_z$$

[ > ru:=diff(r(u,v), u);
      
$$ru := 2u \mathbf{e}_x + \mathbf{e}_z$$

[ > rv:=diff(r(u,v), v);
      
$$rv := 2v \mathbf{e}_y + 2 \mathbf{e}_z$$

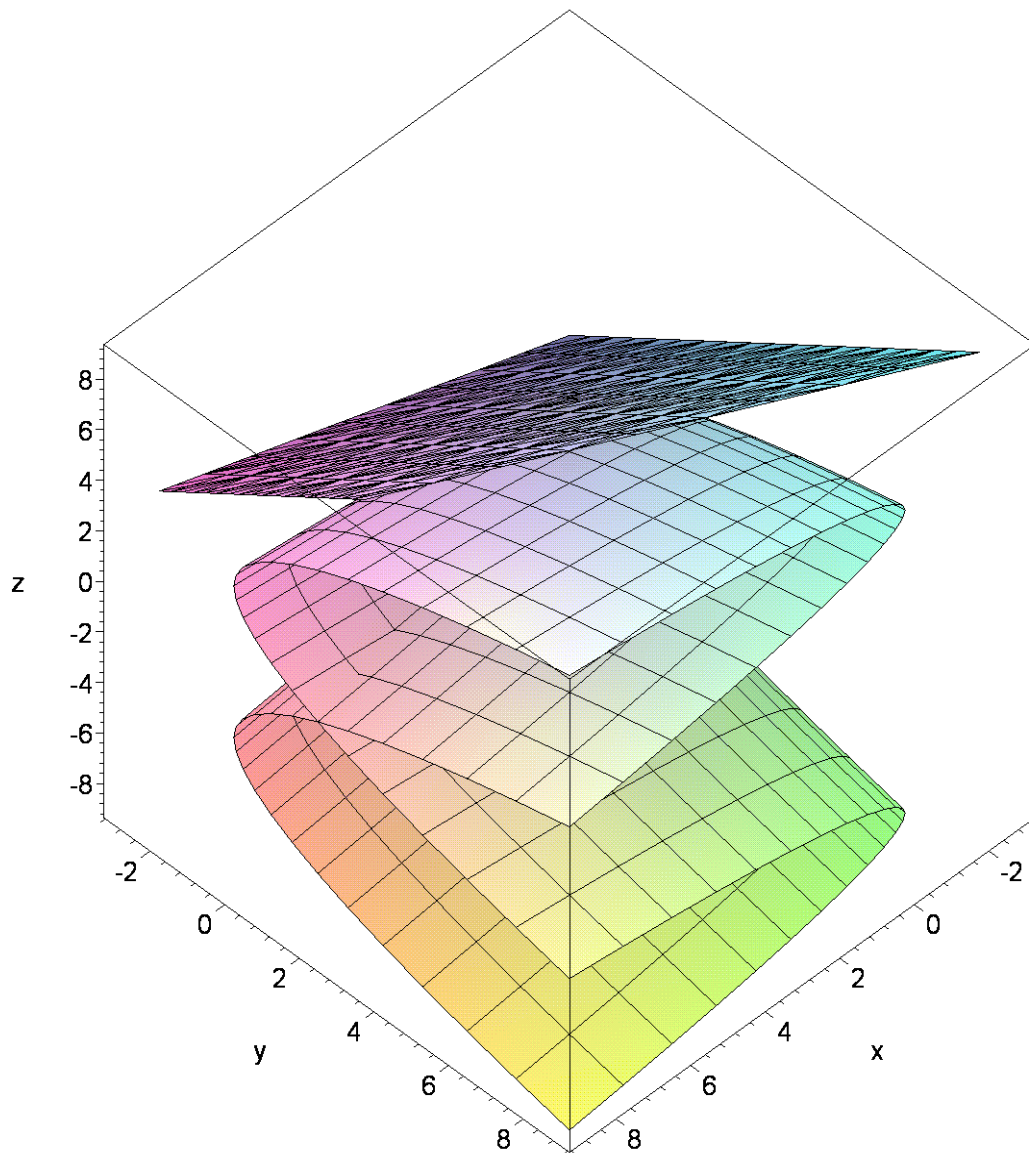
[ > cross:=ru &x rv;
      
$$cross := -2v \mathbf{e}_x - 4u \mathbf{e}_y + 4uv \mathbf{e}_z$$

[ > n:=eval(cross, {u=a, v=b});
      
$$n := (-2) \mathbf{e}_x - 4 \mathbf{e}_y + 4 \mathbf{e}_z$$

[ > t:=n . (<x,y,z> - r(a,b)) = 0;
      
$$t := -2x - 6 - 4y + 4z = 0$$

[ > pl1:=plot3d(r(u,v), u=-3..3, v=-3..3):
[ > pl2:=implicitplot3d(t, x=-3..8, y=-3..8, z=-3..8):
[ > display(pl1, pl2, axes=boxed);

```



## Lesson 11-5

### Calculating a Derivatives Using the Chain Rule (Case 1)

The following example shows the evaluation of a numerical derivative of a function defined by two one-variable functions.

```
> restart;
```

```
> z:=(x,y)->x^2*y+3*x*y^4: x:=sin(2*t): y:=cos(t): z(x,y);
```

$$\sin(2t)^2 \cos(t) + 3 \sin(2t) \cos(t)^4$$

```
> diff(z(x,y), t);
```

$$4 \sin(2t) \cos(t) \cos(2t) - \sin(2t)^2 \sin(t) + 6 \cos(2t) \cos(t)^4 - 12 \sin(2t) \cos(t)^3 \sin(t)$$

```
> eval(diff(z(x,y), t), t=0);
```

6

## Calculating a Derivatives Using the Chain Rule (Case 2)

This example finds the partial derivatives of a function defined by two two-variable functions.

```
> restart;
```

```
> z:=(x,y)->exp(x)*sin(y): x:=s*t^2: y:=s^2*t: z(x,y);
```

$$e^{(s^2 t)} \sin(s^2 t)$$

```
> diff(z(x,y), s);
```

$$t^2 e^{(s^2 t)} \sin(s^2 t) + 2 e^{(s^2 t)} \cos(s^2 t) s t$$

```
> diff(z(x,y), t);
```

$$2 s t e^{(s^2 t)} \sin(s^2 t) + e^{(s^2 t)} \cos(s^2 t) s^2$$

## Lesson 11-6

### Calculating a Directional Derivative

The **DirectionalDiff** command in the **VectorCalculus** package can be used to calculate the directional derivative of a function. The specified direction is given as a vector, and the coordinates **[x, y]** must be stated.

```
> restart: with(VectorCalculus):
```

Warning, the assigned names <, > and <|> now have a global binding

Warning, these protected names have been redefined and unprotected: \*, +, -, ., D, Vector, diff, int, limit, series

```
> f:=(x,y)->x^3-3*x*y+4*y^2: f(x,y);
```

$$x^3 - 3xy + 4y^2$$

```
> DirectionalDiff(f(x,y), <cos(Pi/6), sin(Pi/6)>, [x,y]);
```

$$\frac{(3x^2 - 3y)\sqrt{3}}{2} - \frac{3x}{2} + 4y$$

```
> eval(%, {x=1, y=2});
```

$$-\frac{3\sqrt{3}}{2} + \frac{13}{2}$$

### Finding the Gradient of a Function

The **Gradient** command in the **VectorCalculus** package can be used to calculate the gradient of a

function. The coordinates  $[x, y]$  must be stated.

```
> restart: with(VectorCalculus):
```

```
Warning, the assigned names <, > and <|> now have a global binding
```

```
Warning, these protected names have been redefined and unprotected: *, +, -, ., D, Vector, diff, int, limit, series
```

```
> g:=(x,y)->sin(x)+exp(x*y): g(x,y);
```

$$\sin(x) + e^{(xy)}$$

```
> Gradient(g(x,y), [x,y]);
```

$$(\cos(x) + y e^{(xy)}) \mathbf{e}_x + x e^{(xy)} \mathbf{e}_y$$

```
> eval(%, {x=0, y=1});
```

$$2 \mathbf{e}_x$$

## Creating a Gradient Vector Field

The `gradplot` command in the `plots` package can be used to create a gradient vector field for a given function. It is often informative to graph the vector field with a contour map.

```
> restart: with(plots):
```

```
Warning, the name changecoords has been redefined
```

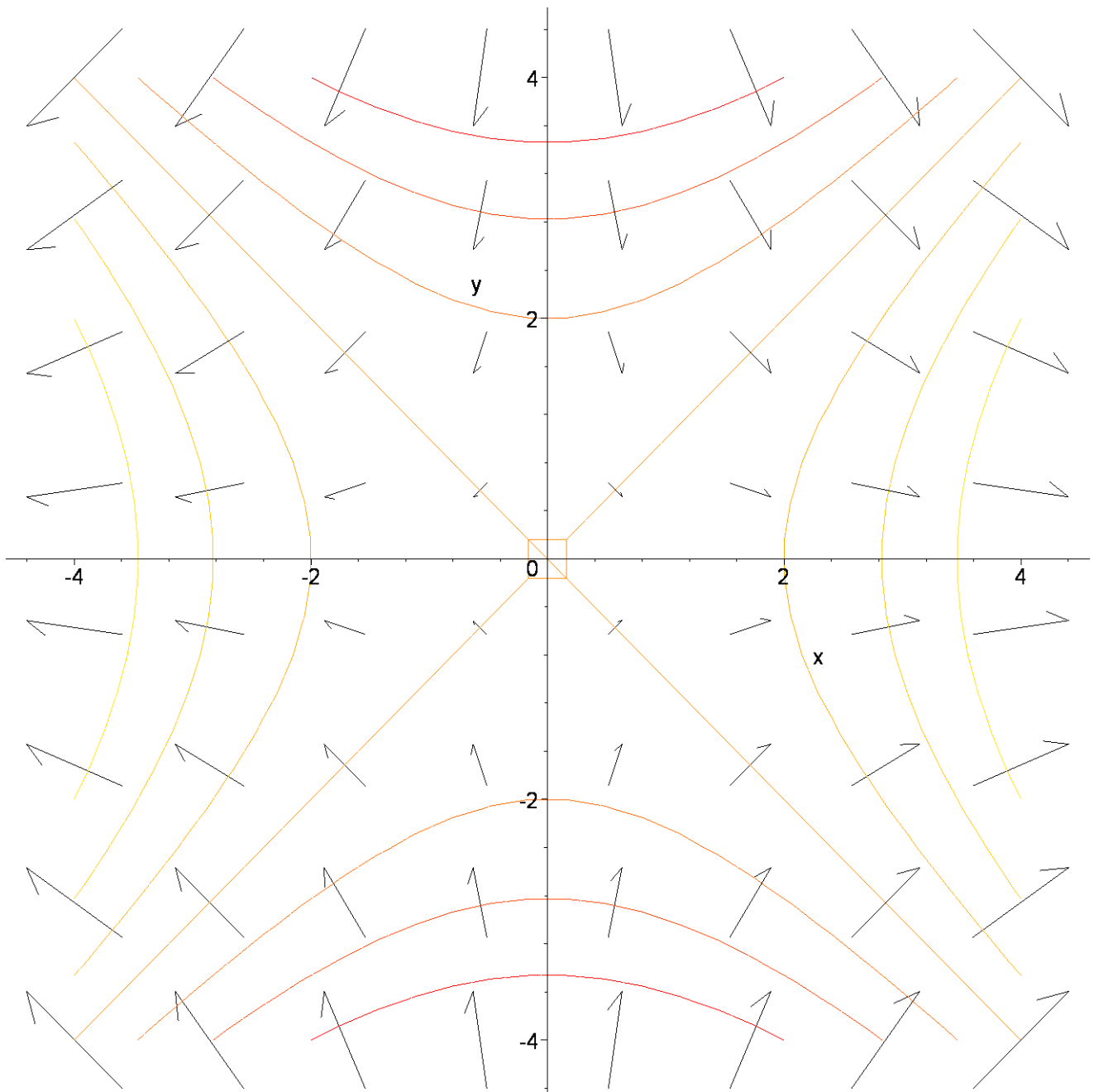
```
> f:=(x,y)->x^2-y^2: f(x,y);
```

$$x^2 - y^2$$

```
> a:=contourplot(f(x,y), x=-4..4, y=-4..4, contours=[-12,-8,-4,0,4,8,12]):
```

```
> b:=gradplot(f(x,y), x=-4..4, y=-4..4, grid=[8,8]):
```

```
> display(a,b);
```



## Lesson 11-7

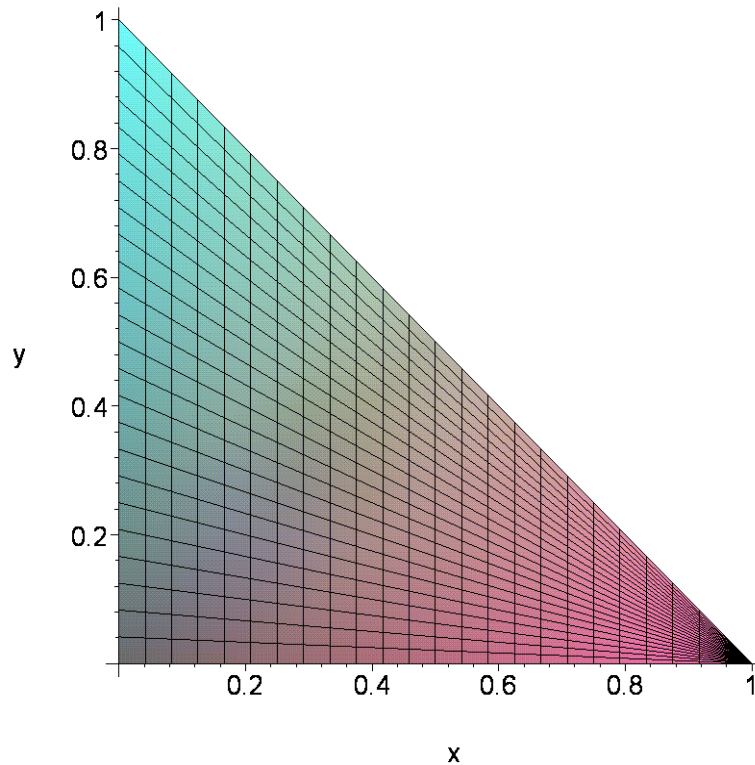
### Graphing an Optimization Problem's Domain

The  $x$  and  $y$  parameters can be adjusted to give an exact representation of most optimization applications (let one be a function of the other). The option `orientation=[-90, 0]` can be used to view just the domain.

```
> restart;
> f:=(x,y)->x^2-2*x*y+2*y: f(x,y);
```

$$x^2 - 2xy + 2y$$

```
> plot3d(f(x,y), x=0..1, y=1-x..0, axes=normal,  
orientation=[-90,0]);
```



## Lesson 11-8

### Optimizing a Function Using Lagrange Multipliers

The `LagrangeMultipliers` command in the `Student[MultivariateCalculus]` package can be used to optimize an objective function with given constraints using the Lagrange multipliers method. Note that the list of constraint functions are all assumed to be equal to zero, and the domain variables must

be explicitly stated. The option **output=detailed** can be included to show all variables and the objective function's output.

```
> restart: with(Student[MultivariateCalculus]):
> obj:=(x,y,z)->x*y*z: obj(x,y,z);
                                     x y z
> constr:=(x,y,z)->2*x*z+2*y*z+x*y-12: constr(x,y,z);
                                     2 x z + 2 y z + x y - 12
> LagrangeMultipliers(obj(x,y,z), [constr(x,y,z)], [x,y,z],
  output=detailed);
  [ x = 2, y = 2, z = 1, λ1 = 1/2, x y z = 4 ], [ x = -2, y = -2, z = -1, λ1 = -1/2, x y z = -4 ]
```

### Optimizing an Objective Function Given Two Constraints.

The **LagrangeMultipliers** command can also be used if there are two constraint functions.

```
> restart: with(Student[MultivariateCalculus]):
> obj:=(x,y,z)->x+2*y+3*z: obj(x,y,z);
                                     x + 2 y + 3 z
> constr1:=(x,y,z)->x-y+z-1: constr1(x,y,z);
                                     x - y + z - 1
> constr2:=(x,y,z)->x^2+y^2-1: constr2(x,y,z);
                                     x2 + y2 - 1
> LagrangeMultipliers(obj(x,y,z), [constr1(x,y,z),constr2(x,y,z)],
  [x,y,z], output=detailed);
  [ x = -2 RootOf(29 _Z2 - 1, label = _L1), y = 5 RootOf(29 _Z2 - 1, label = _L1),
    z = 7 RootOf(29 _Z2 - 1, label = _L1) + 1, λ1 = 3, λ2 = 29/2 RootOf(29 _Z2 - 1, label = _L1),
    x + 2 y + 3 z = 29 RootOf(29 _Z2 - 1, label = _L1) + 3 ]
> evalf(%);
[x = -0.3713906764, y = 0.9284766910, z = 2.299867367, λ1 = 3., λ2 = 2.692582404,
  x + 2. y + 3. z = 8.385164808 ]
>
```