

Flows in Vector Fields

Worksheet by © Mike May, S.J. 2006- maymk@slu.edu

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

▼ Overview

Creating flow lines through vector fields is really solving a system of differential equations. We let the vector be the velocity vector at a point, and want to solve for a path so that the velocity vector is the derivative of the parameterized position vector with respect to time.

This can sound intimidating, since it is really solving a system of differential equations, but it is not hard to do graphically. For flows of vector fields in two dimensions, we will use other software, but we want to show Maple for completeness. In particular, we can use Maple when we want to plot flows of vector fields in 3 dimensions, or when we want to do more complicated systems. (We used these tools again when we looked at planetary motion.)

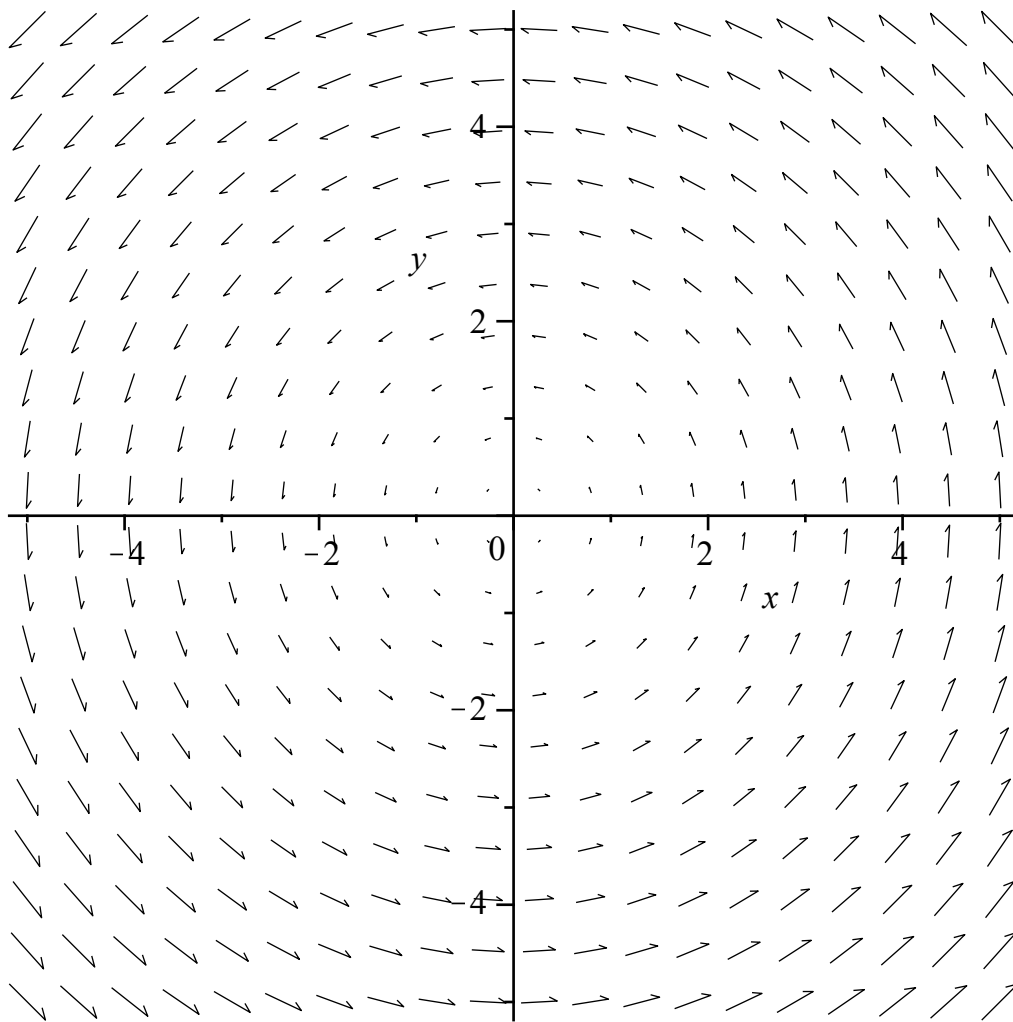
Since we are solving differential equations we loaded the DEtools package when we started the worksheet.

```
[>
```

▼ Plotting flows in 2 dimensions

We start with a simple example. Consider the vector field $[-y, x]$.

```
> fieldplot([-y, x], x=-5..5, y=-5..5);
```



It is clear that a flow path must swirl around the center. If we try to trace a path that is always tangent to the field lines we get circles. We would like Maple to trace out the curve for us. We need to do some set-up.

Given the vector field, $[f(x,y), g(x,y)]$, saying that that field gives the derivative of motion would turn into $x' = f(x,y)$ and $y' = g(x,y)$. Since we will do paths in time, both x and y are functions of time. We now record the two differential equations.

```
> ODE1 := diff(x(t),t) = -y(t);
   ODE2 := diff(y(t),t) = x(t);
      ODE1 :=  $\frac{d}{dt} x(t) = -y(t)$ 
      ODE2 :=  $\frac{d}{dt} y(t) = x(t)$  (2.1)
```

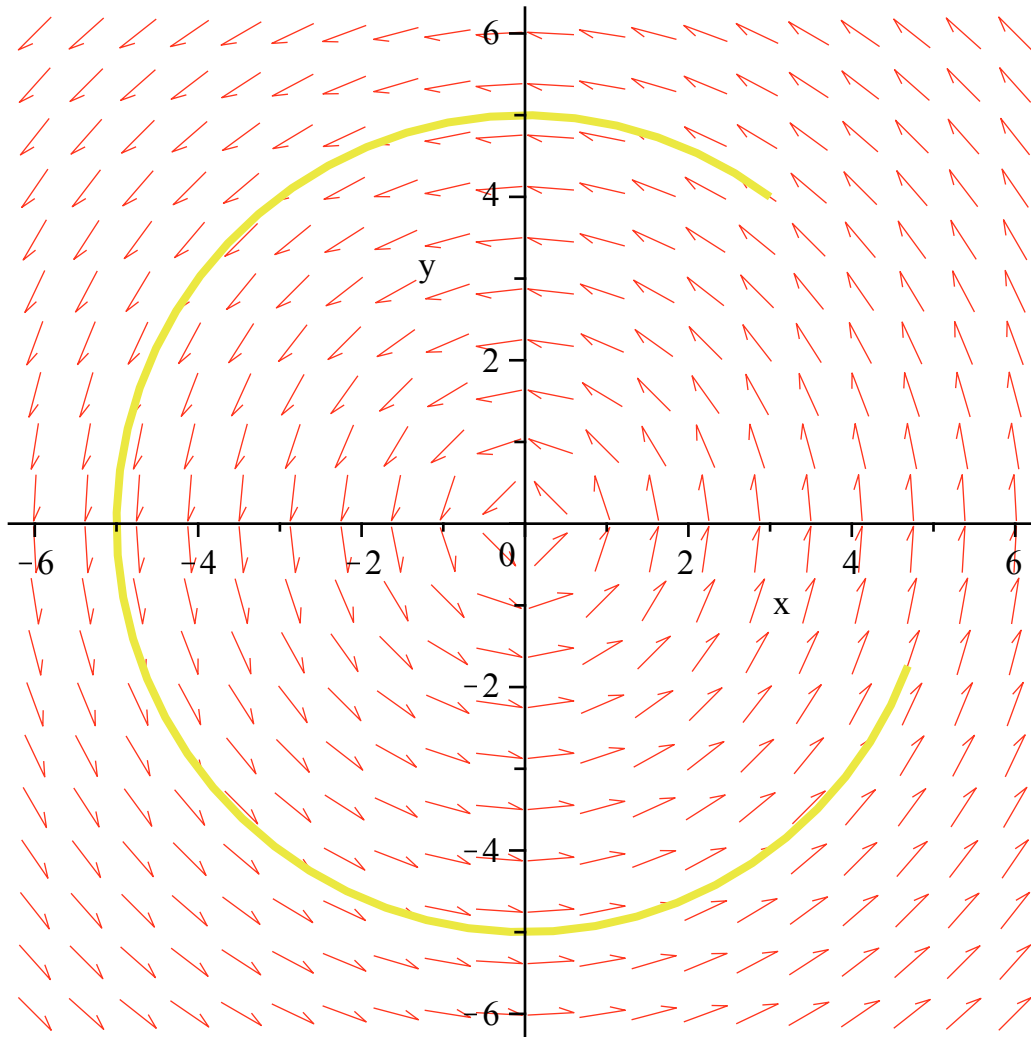
Next we need to specify a starting position. This should be a tell us where our path starts at a given time.

```
> initpos1 := [x(0) = 3, y(0) = 4];
      initpos1 := [x(0) = 3, y(0) = 4] (2.2)
```

Now we try to plot the vector field and curve.

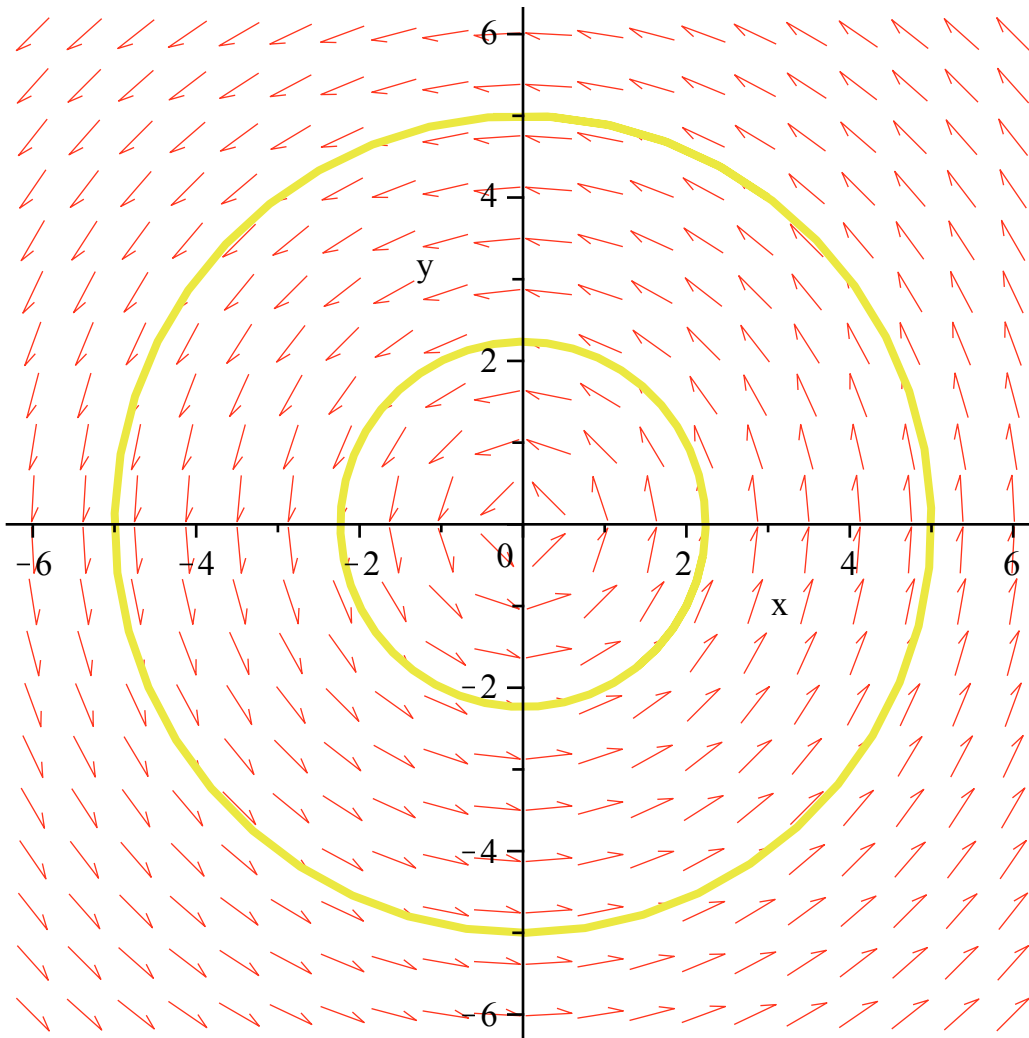
```
> DEplot({ODE1, ODE2}, [x, y], t=0..5, [initpos1], x=-6..6, y=
```

```
-6..6);
```



The path is turning out to be a circle, but we need more time to get all the way around. We could also add other initial positions.

```
> DEplot({ODE1, ODE2}, [x, y], t=0..7, [[x(0)=3, y(0)=4], [x(2)  
=1, y(2)=2]], x=-6..6, y=-6..6);
```

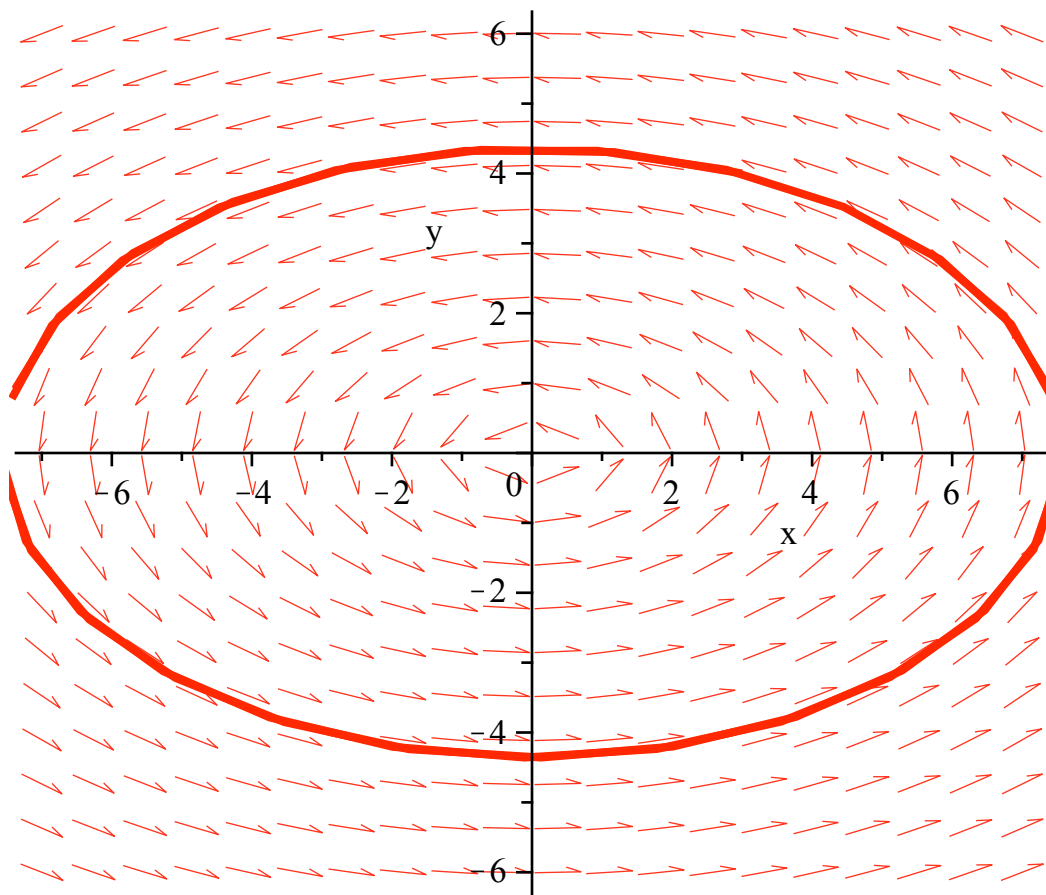


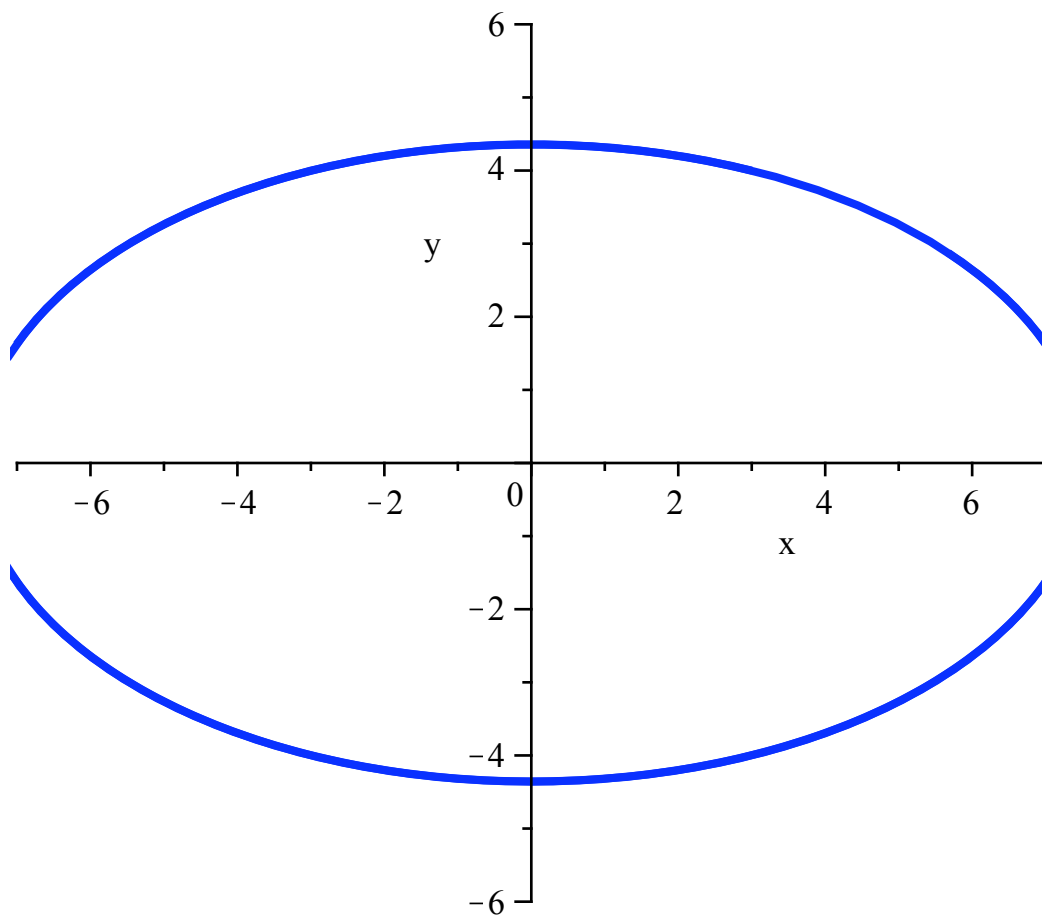
Two tidbits to clean things up if needed. We can suppress the arrows with the `arrows=none` option. We can also specify how smooth the curve is by giving the stepsize of the numerical solution. The default stepsize leads to 20 steps.

```
> ODE1 := diff(x(t),t) = -3*y(t);
   ODE2 := diff(y(t),t) = x(t);
   DEplot({ODE1, ODE2}, [x, y], t=0..7, [[x(0)=3, y(0)=4]], x=
-7..7, y=-6..6, linecolor=red);
   DEplot({ODE1, ODE2}, [x, y], t=0..7, [[x(0)=3, y(0)=4]], x=
-7..7, y=-6..6, arrows=none, linecolor=blue, stepsize=.05);
```

$$ODE1 := \frac{d}{dt} x(t) = -3 y(t)$$

$$ODE2 := \frac{d}{dt} y(t) = x(t)$$





As noted in the last worksheet, when using a new command, it is worthwhile to look at the manual page for that command.

`> ?DEplot`

Exercises:

1) For the vector field $[y^2, 2x^2]$, find a flow line through $(1, 2)$.

`>`

2) Describe typical flow lines for the following vector fields.

(a) $F(x, y) = [y, x]$.

`>`

(b) $F(x, y) = [x, y]$.

`>`

(c) $F(x, y) = -yi + (x + y/10)j$.

`>`

(d) $F(x, y) = (x - y) i + (x - y) j$.

`>`

Flow Lines in 3 dimensions

If we want a flow line in a vector field of three dimensions, most things follow the obvious generalizations. The sticking point is that the `DEplot3d` command won't show the field lines. We

can use `fieldplot3d` to plot the field marks, and `display3d` to put the two together.

```
> ODE1 := diff(x(t),t) = -3*y(t);  
ODE2 := diff(y(t),t) = x(t);  
ODE3 := diff(z(t),t) = 1;  
vfield := [-3*y, x, 1];  
initvals1 := [x(0)=3, y(0)=2, z(0)=-4];
```

$$ODE1 := \frac{d}{dt} x(t) = -3y(t)$$

$$ODE2 := \frac{d}{dt} y(t) = x(t)$$

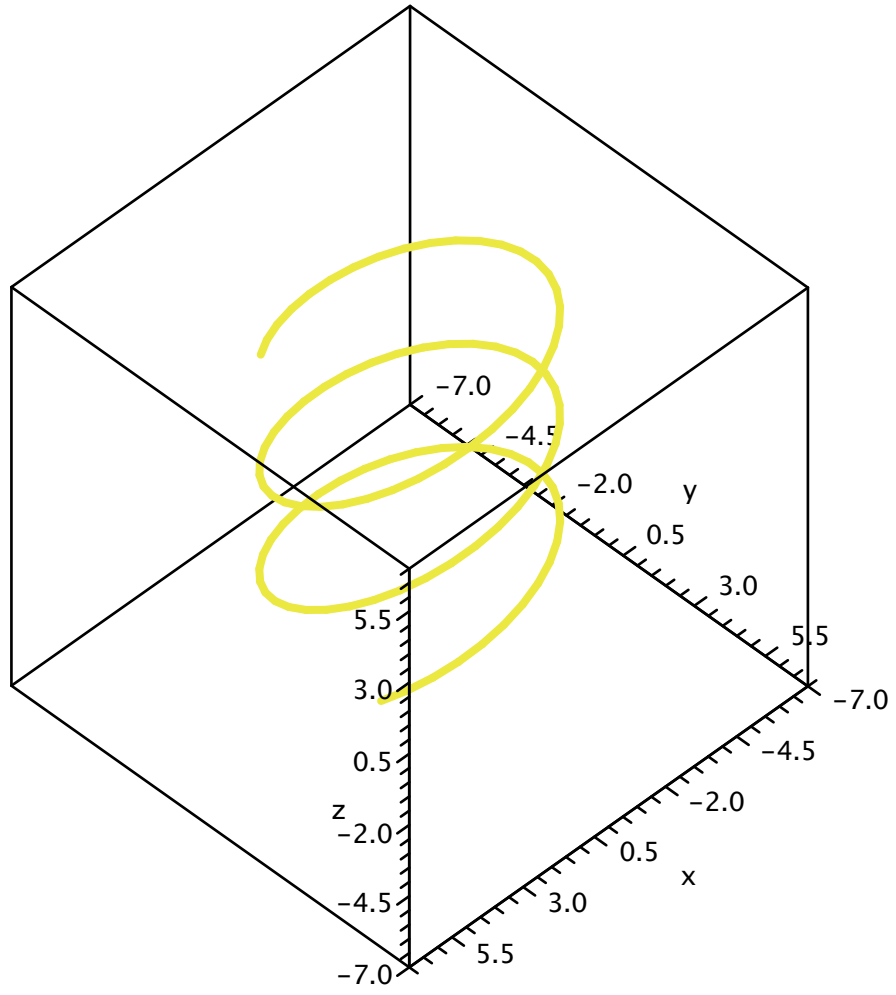
$$ODE3 := \frac{d}{dt} z(t) = 1$$

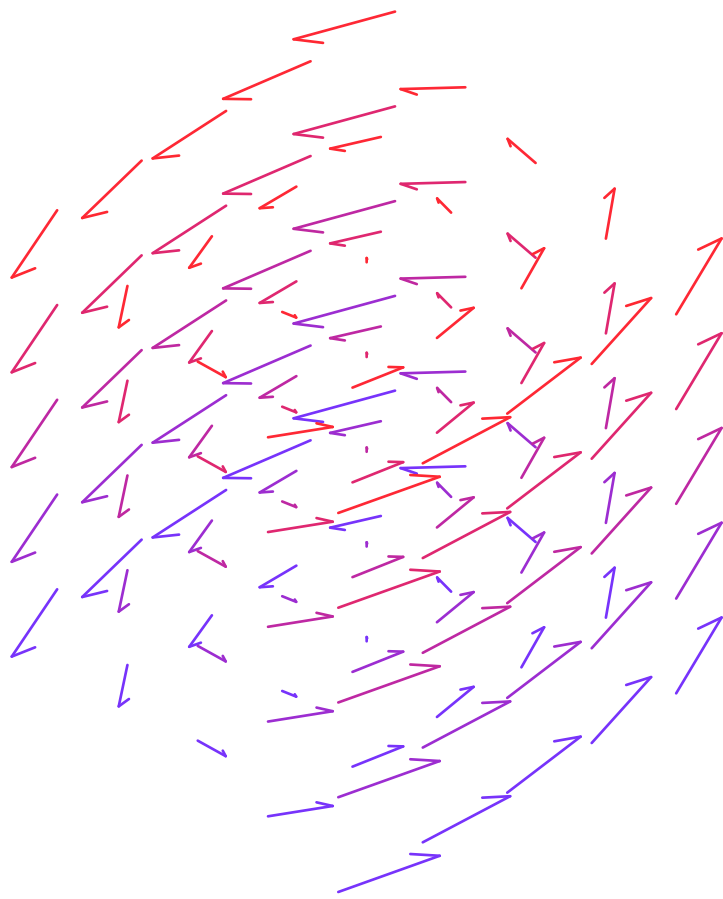
$$vfield := [-3y, x, 1]$$

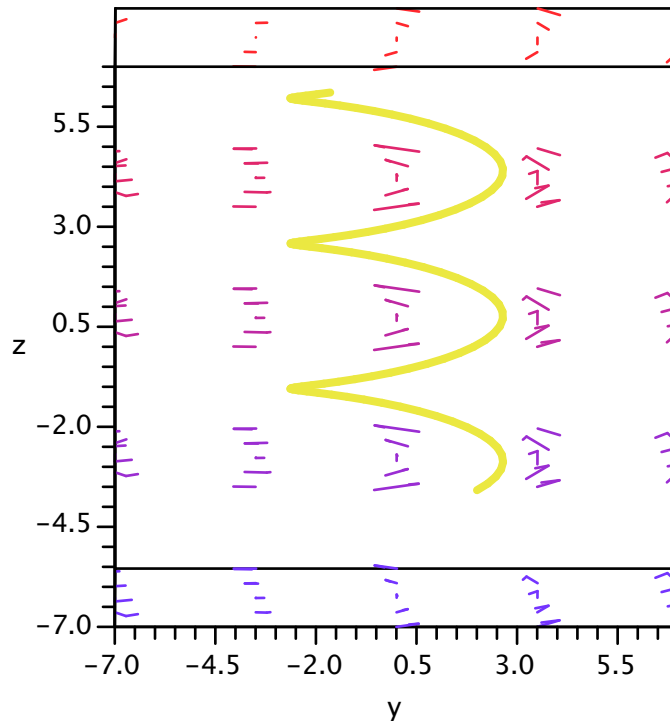
$$initvals1 := [x(0) = 3, y(0) = 2, z(0) = -4]$$

(4.1)

```
> DEplot3d({ODE1, ODE2, ODE3}, [x,y,z], t=0..10, [initvals1],  
           x=-7..7, y=-7..7, z=-7..7, shading=z, stepsize=.1 );  
fieldplot3d(vfield, x=-7..7, y=-7..7, z=-7..7, shading=z,  
            grid=[5,5,5], shading=z);  
patha := DEplot3d({ODE1, ODE2, ODE3}, [x,y,z], t=0..10,  
                 [initvals1],  
                 x=-7..7, y=-7..7, z=-7..7, shading=z, stepsize=.1 ):  
vfielda := fieldplot3d(vfield, x=-7..7, y=-7..7, z=-7..7,  
                      shading=z,  
                      grid=[5,5,5], shading=z):  
display3d({patha, vfielda});
```







We see that the flow line tries to form a helix

Exercise:

3) Describe a flow line generated by the field $[y-z, z-x, x-y]$ through the point $[1, 2, 3]$.

[>

Numerical approaches

Since the book looks at numerical computation of flow lines, we want to look at them a little bit. Maple uses advanced techniques that are beyond the scope of this course. We can however do Euler's method to find specified points.

Euler's Method

Consider our favorite vector field, $[-y, x]$. The flow lines should be circles around the origin. The vector field corresponds to the system $x' = -y$, $y' = x$, where both x and y are understood to be functions of t . Euler's method uses linear approximation to say that

$$x[t+\text{delt}] = x[t] + (\text{delt}) * (-y[t]);$$

$$y[t+\text{delt}] = y[t] + (\text{delt}) * (x[t]);$$

Then we put the instructions in a loop to compute to our desired time.

```
> pt[0.0] := [3,4]; delt := .1;
  for i from 0 to 63 do
```

```
pt[i*delt+delt] :=  
  [pt[i*delt][1]-delt*pt[i*delt][2],  
   pt[i*delt][2]+delt*pt[i*delt][1]];  
end do;
```

$$pt_0 := [3, 4]$$

$$delt := 0.1$$

$$pt_{0.1} := [2.6, 4.3]$$

$$pt_{0.2} := [2.17, 4.56]$$

$$pt_{0.3} := [1.714, 4.777]$$

$$pt_{0.4} := [1.2363, 4.9484]$$

$$pt_{0.5} := [0.74146, 5.07203]$$

$$pt_{0.6} := [0.234257, 5.146176]$$

$$pt_{0.7} := [-.2803606, 5.1696017]$$

$$pt_{0.8} := [-.79732077, 5.14156564]$$

$$pt_{0.9} := [-1.311477334, 5.061833563]$$

$$pt_{1.0} := [-1.817660690, 4.930685830]$$

$$pt_{1.1} := [-2.310729273, 4.748919761]$$

$$pt_{1.2} := [-2.785621249, 4.517846834]$$

$$pt_{1.3} := [-3.237405932, 4.239284709]$$

$$pt_{1.4} := [-3.661334403, 3.915544116]$$

$$pt_{1.5} := [-4.052888815, 3.549410676]$$

$$pt_{1.6} := [-4.407829883, 3.144121794]$$

$$pt_{1.7} := [-4.722242062, 2.703338806]$$

$$pt_{1.8} := [-4.992575943, 2.231114600]$$

$$pt_{1.9} := [-5.215687403, 1.731857006]$$

$$pt_{2.0} := [-5.388873104, 1.210288266]$$

$$pt_{2.1} := [-5.509901931, 0.6714009556]$$

$$pt_{2.2} := [-5.577042027, 0.1204107625]$$

$$pt_{2.3} := [-5.589083103, -.4372934402]$$

$$pt_{2.4} := [-5.545353759, -.9962017505]$$

$$pt_{2.5} := [-5.445733584, -1.550737126]$$

$$pt_{2.6} := [-5.290659871, -2.095310484]$$

$$pt_{2.7} := [-5.081128823, -2.624376471]$$

$pt_{2,8} := [-4.818691176, -3.132489353]$
 $pt_{2,9} := [-4.505442241, -3.614358471]$
 $pt_{3,0} := [-4.144006394, -4.064902695]$
 $pt_{3,1} := [-3.737516124, -4.479303334]$
 $pt_{3,2} := [-3.289585791, -4.853054946]$
 $pt_{3,3} := [-2.804280296, -5.182013525]$
 $pt_{3,4} := [-2.286078944, -5.462441555]$
 $pt_{3,5} := [-1.739834788, -5.691049449]$
 $pt_{3,6} := [-1.170729843, -5.865032928]$
 $pt_{3,7} := [-.5842265502, -5.982105912]$
 $pt_{3,8} := [0.0139840410, -6.040528567]$
 $pt_{3,9} := [0.6180368977, -6.039130163]$
 $pt_{4,0} := [1.221949914, -5.977326473]$
 $pt_{4,1} := [1.819682561, -5.855131482]$
 $pt_{4,2} := [2.405195709, -5.673163226]$
 $pt_{4,3} := [2.972512032, -5.432643655]$
 $pt_{4,4} := [3.515776398, -5.135392452]$
 $pt_{4,5} := [4.029315643, -4.783814812]$
 $pt_{4,6} := [4.507697124, -4.380883248]$
 $pt_{4,7} := [4.945785449, -3.930113536]$
 $pt_{4,8} := [5.338796803, -3.435534991]$
 $pt_{4,9} := [5.682350302, -2.901655311]$
 $pt_{5,0} := [5.972515833, -2.333420281]$
 $pt_{5,1} := [6.205857861, -1.736168698]$
 $pt_{5,2} := [6.379474731, -1.115582912]$
 $pt_{5,3} := [6.491033022, -.4776354389]$
 $pt_{5,4} := [6.538796566, 0.1714678633]$
 $pt_{5,5} := [6.521649780, 0.8253475199]$
 $pt_{5,6} := [6.439115028, 1.477512498]$
 $pt_{5,7} := [6.291363778, 2.121424001]$
 $pt_{5,8} := [6.079221378, 2.750560379]$

```

pt5,9 := [5.804165340, 3.358482517]
pt6,0 := [5.468317088, 3.938899051]
pt6,1 := [5.074427183, 4.485730760]
pt6,2 := [4.625854107, 4.993173478]
pt6,3 := [4.126536759, 5.455758889]
pt6,4 := [3.580960870, 5.868412565]

```

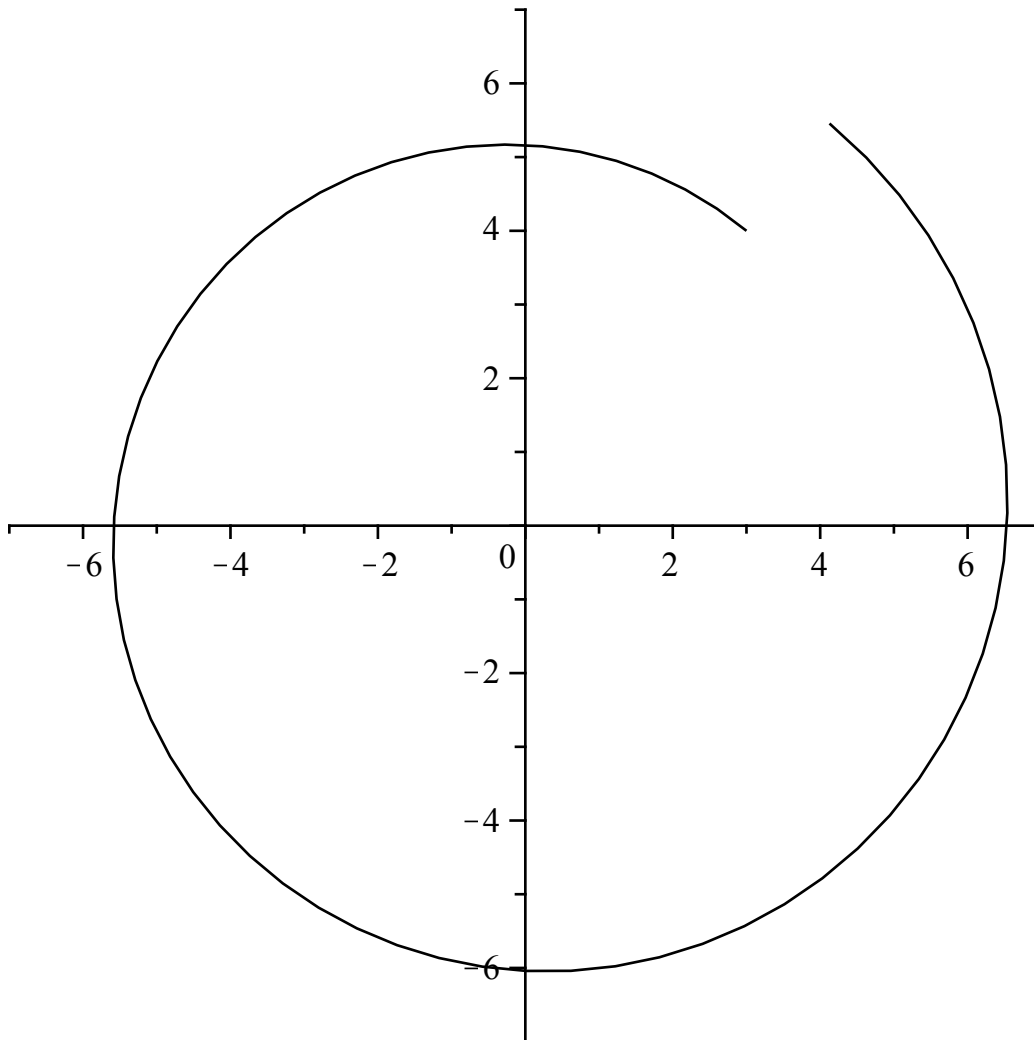
(6.1.1)

We would like to plot the points we computed.

```

> pointplot([seq(pt[i*delta], i=0..63)], connect=true, view=
[-7..7, -7..7]);

```



It becomes clear that our plot wanders off the true path. In fact Euler's method is related to integration by the left hand rule and is not very accurate unless we make δ quite small.

```

> pta[0.0] := [3,4]; delta := .1:
clump := 50:
truedel := delta/clump;
for i from 0 to 63 do
ptb[0] := pta[i*delta]:
for j from 1 to clump do
ptb[j] := [ptb[j-1][1]-(delta/clump)*ptb[j-1][2],

```

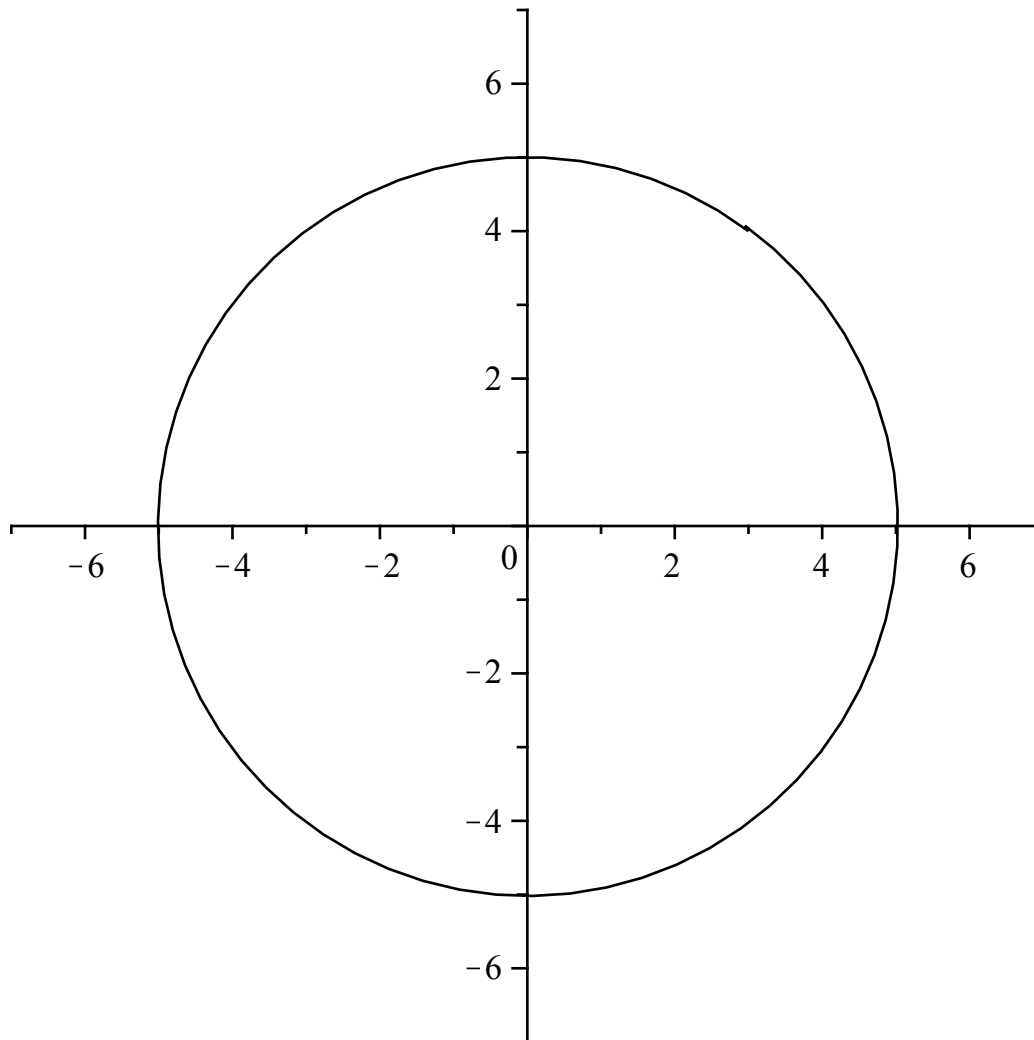
```

    ptb[j-1][2]+(delt/clump)*ptb[j-1][1]]:
od:
pta[(i+1)*delt] := ptb[clump]:
od:
pointplot([seq(pta[i*delt],i=0..63)], connect=true, view=
[-7..7, -7..7]);

```

$pta_0 := [3, 4]$

$truedel := 0.002000000000$



This method lets us evaluate points. The point $pta[3.1]$ should be halfway around the circle.

```
> pta[3.1];
```

$[-3.264969497, -3.992330540]$

Exercise:

4) How far is the point $pta[6.2]$ from its expected value, which is back at the starting point? How does this compare to $pt[6.2]$?

```
>
```

The dsolve command

(Optional material - this is for completeness for the curious. It will not be on the test.)

You should be suspicious that Maple has a cleaner way of doing this. Maple uses the command

dsolve.

We start by defining our DEs and initial conditions.

```
> DE1 := diff(x(t),t)=-y(t);  
DE2 := diff(y(t),t)=x(t);  
IC1 := x(0)=3;  
IC2 := y(0)=4;
```

$$DE1 := \frac{d}{dt} x(t) = -y(t)$$

$$DE2 := \frac{d}{dt} y(t) = x(t)$$

$$IC1 := x(0) = 3$$

$$IC2 := y(0) = 4$$

(6.3.1)

We now use the dsolve command to create a function, named solcurve), that is our solution curve.

```
> solcurve := dsolve({DE1, DE2, IC1, IC2}, [x(t), y(t)],  
numeric);
```

```
solcurve := proc(x_rkf45) ... end proc
```

(6.3.2)

To test the function we can evaluate at 2Pi, which should be back to our starting point.

```
> solcurve(2*Pi);
```

```
[t=6.28318530717958, x(t) = 3.00000235512243130, y(t)
```

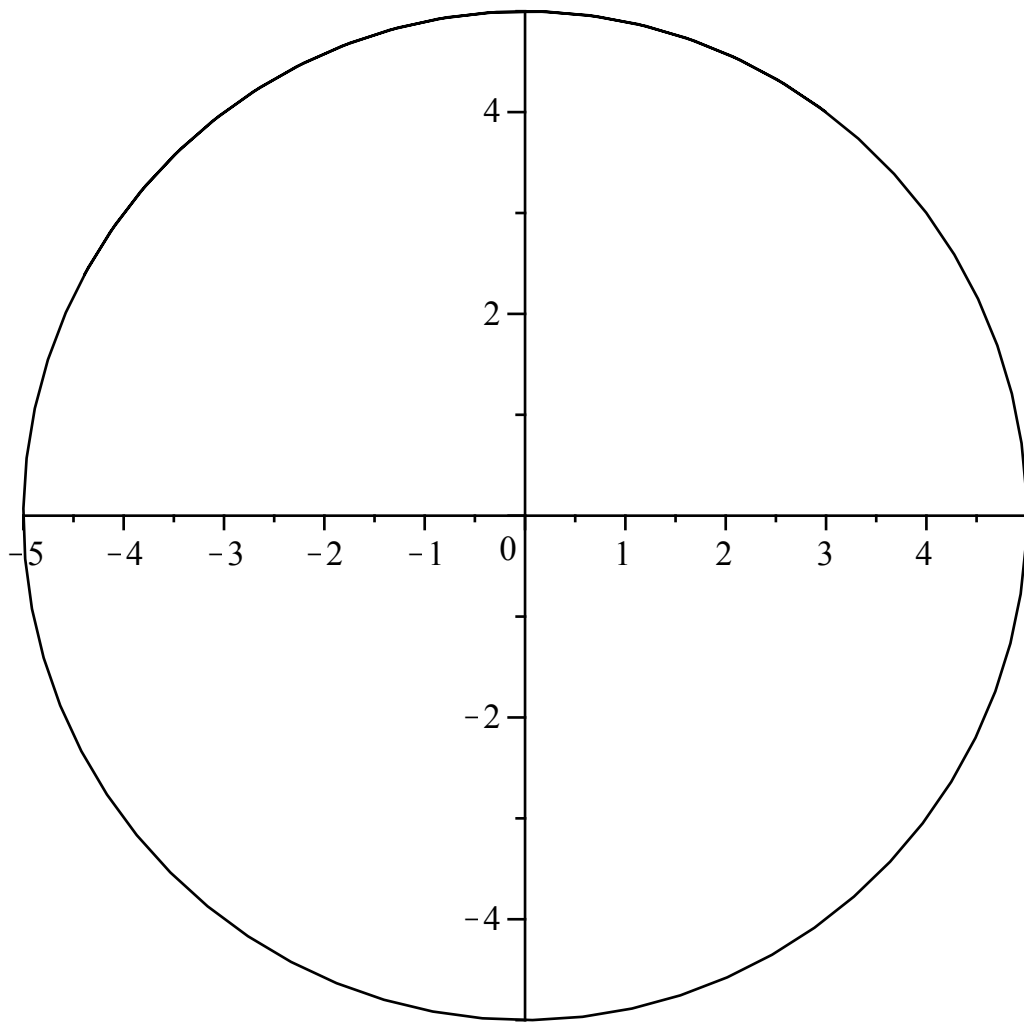
```
= 4.00000245977151980]
```

(6.3.3)

The answer is accurate to 8 decimal places which will be good enough for us.

You should notice that solcurve is a vector valued function giving 3 values, each of which is an equation in t, x, or y. We can use the second and third values to create a plot of the solution curve.

```
> pointplot([seq(  
[op(2,solcurve(t*.1)[2]),op(2,solcurve(t*.1)[3])],  
t=0..80)], connect=true);
```



This is the technique that was used in the worksheet on planetary motion to find plots of orbits starting only with the force of gravity, initial position, and initial velocity.

[>

[>
[>