Part III. First Order Ordinary Differential Equations

Section 3. Slope Fields: DEplot

A slope field (or direction field) for a first order ordinary differential equation of the form

$$\frac{d}{dt}y(t) = f(t, y(t))$$

is an array of short line segments, at point (t,y) the segment has slope $f(t, y)$. Since these are tangent lines to the solution curves determined by the equation, a slope field can provide a wealth of information about the qualitative behavior of the solutions.

The tool you need: DEplot

The Maple procedures that create direction fields and geometric (i.e. approximate) solution curves are found in the **DEtools** package. All told, there are 115 procedures in the **DEtools** package, but only one will be used now, **DEplot**. This procedure is loaded into the kernel with the entry

```
with(DEtools,DEplot)
```

```
> with(DEtools,DEplot);
```

[*DEplot*]

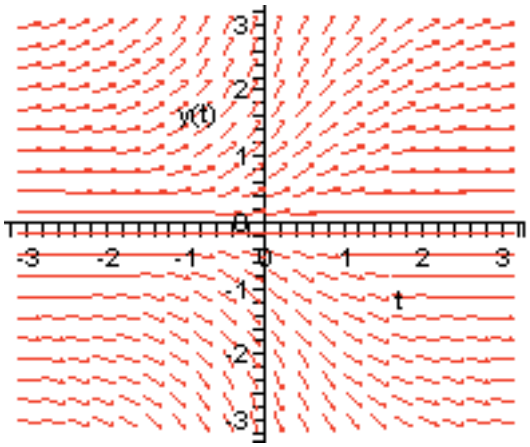
The syntax to generate a direction field for a first order ordinary differential equation like the one displayed above (and named DE) is

```
DEplot( DE, y(t), t=a..b, y=c..d)
```

The default plot for the first equation discussed in Section 1 looks like this (plot window -3..3, -3..3).

```
> DE := diff(y(t),t) = y(t)/(t^2 + 1);
  DEplot( DE, y(t), t=-3..3, y=-3..3);
```

$$DE := \frac{d}{dt}y(t) = \frac{y(t)}{t^2 + 1}$$

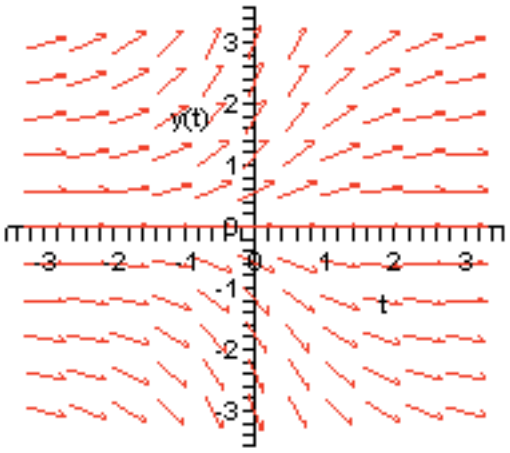


If you have the patience to count them, you will see 400 "harpoon" arrows (20 left to right and 20 top to bottom). The number of arrows in these directions is controlled by an entry of the form

```
dirgrid=[m,n]
```

instructing **DEplot** to draw m arrows in the horizontal direction and n in the vertical direction.

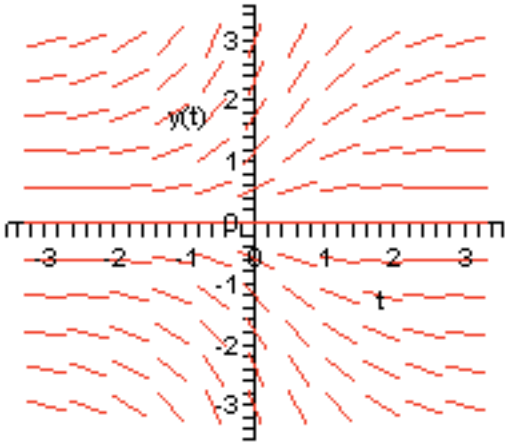
```
> DEplot( DE, y(t), t=-3..3, y=-3..3, dirgrid=[11,11]);
```



Asking for 11 arrows in each direction forces **DEplot** to plot arrows centered at points on the *t* axis and the *y* axes. If you would also prefer to see line segments instead of arrows, add the equation

```
arrows=line
```

```
> DEplot( DE, y(t), t=-3..3, y=-3..3, dirgrid=[11,11], arrows=line);
```

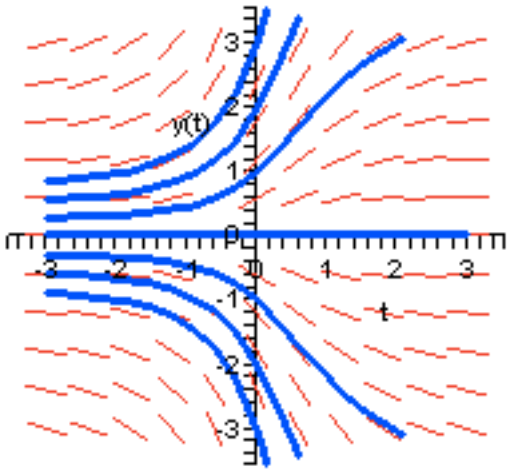


Finally, solution curves can be included in the picture. Just add a *set* containing *lists* of initial conditions. It must be of the form

{ [y(t1) = y1] , [y(t2)=y2] , ... , [y(tn)=yn] }

See below.

```
> DEplot( DE, y(t), t=-3..3, y=-3..3, dirgrid=[11,11], arrows=line,
          {[y(0)=k]$k=-3..3}, linecolor=blue);
```



The default color for the solution curves is yellow. We recommend changing to blue using the equation

`linecolor=blue`

as we did above. The equation **color=blue** would make the arrows blue.

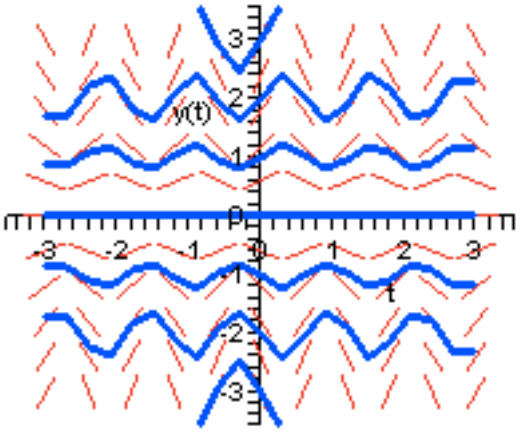
Compare this picture to the first one displayed in Section 1 (page 33).

Stepsize matters

Direction field information is used to plot the curves, slopes are averaged across the plot window. By default, the averaging algorithm is the classical Runge-Kutta fourth order method. See Ledder, Chapter 2, Section 6. When plotting solution curves, **DEplot** takes 20 evenly spaced steps across the plot window. Therefore, when $t = a..b$, each step has length $(b - a)/20$. This may be too large a step size to produce a nice smooth curve. The next example illustrates this.

```
> DE2 := diff(y(t),t) = cos(5*t)*y(t);
  DEplot( DE2, y(t), t=-3..3, y=-3..3, dirgrid=[11,11], arrows=line,
          {[y(0)=k]$k=-3..3}, linecolor=blue);
```

$$DE2 := \frac{d}{dt}y(t) = \cos(5\,t)\,y(t)$$

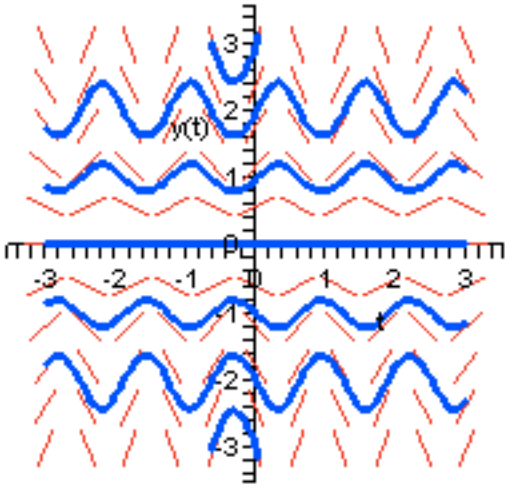


Since b - a = 6, the step size for this plot is 6/20 = 0.3, which is too big to make a smooth curve. To control the step size, insert an equation of the form

```
stepsize=h
```

The next entry reduces the step size to 0.05. **DEplot** crosses the plot window in 120 steps producing a nice smooth curve.

```
> DEplot( DE2, y(t), t=-3..3, y=-3..3, dirgrid=[11,11], arrows=line,
          {[y(0)=k]$k=-3..3}, linecolor=blue, stepsize=0.05);
```

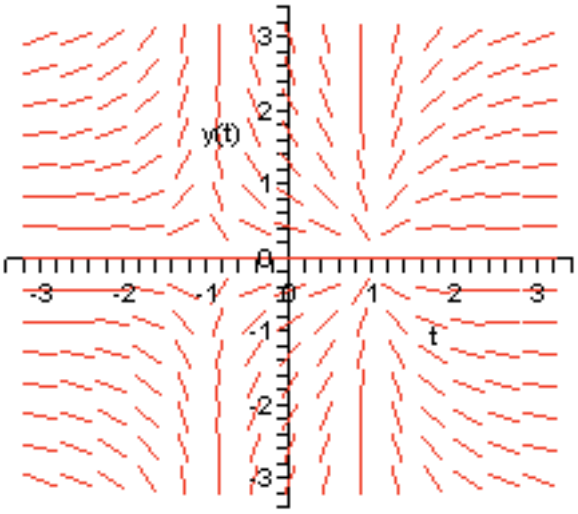


Keep your eye on f (t , y)

In general most points (t, y) in a direction field will have a locally unique solution curve passing through them. "Locally unique" means the only one within a small disc centered at (t, y). However, special attention must be paid to "bad points" where either f or its y partial derivative is discontinuous. The equation named DE2 in Section 1, and DE3 below, has lots of bad points (officially known as "singularities").

```
> DE3 := diff(y(t),t) = y(t)/(t^2 - 1);
  DEplot( DE3, y(t), t=-3..3, y=-3..3, dirgrid=[15,15], arrows=line);
```

$$DE3 := \frac{d}{dt} y(t) = \frac{y(t)}{t^2 - 1}$$



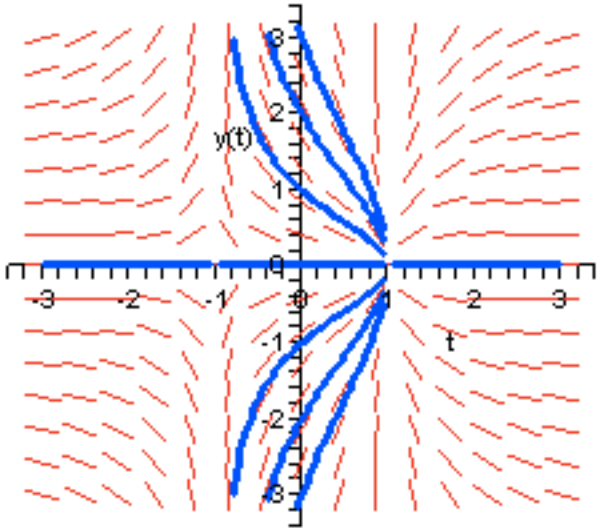
There are potential problems for any solution curve as it approaches the lines $t = 1$ or $t = -1$. This is not surprising, because the function f is not defined at any point on either of these lines; all these points are singular points for DE3:

$$f(t, y) = \frac{y^2}{t^2 - 1}$$

If a step size for solution curves forces DEplot to get too close to a singularity, DEplot may stop the process and issue a warning. The next entry produced 7 warnings, one for each of the initial conditions.

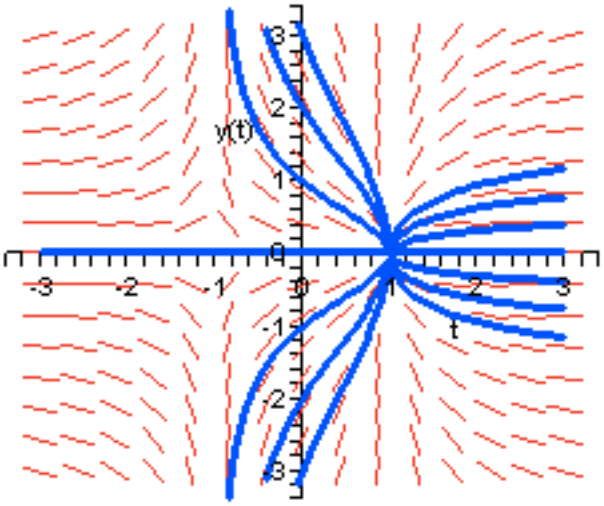
• For printing purposes, warnings like these are deleted in the rest of th e section.

```
> DEplot( DE3, y(t), t=-3..3, y=-3..3, dirgrid=[15,15], arrows=line,
          {[y(0)=k]$k=-3..3}, linecolor=blue, stepsize=0.05);
Warning, plot may be incomplete, the following errors(s) were issued:
   division by zero
Warning, plot may be incomplete, the following errors(s) were issued:
   division by zero
Warning, plot may be incomplete, the following errors(s) were issued:
   division by zero
Warning, plot may be incomplete, the following errors(s) were issued:
   division by zero
Warning, plot may be incomplete, the following errors(s) were issued:
   division by zero
Warning, plot may be incomplete, the following errors(s) were issued:
   division by zero
Warning, plot may be incomplete, the following errors(s) were issued:
   division by zero
```



We actually got lucky with the stepsize in this **DEplot** (compare it to the plot at the top of page 37 in Section 1). To see why, examine the output when the stepsize is increased only slightly to 0.06.

```
> DEplot( DE3, y(t), t=-3..3, y=-3..3, dirgrid=[15,15], arrows=line,
          {[y(0)=k]$k=-3..3}, linecolor=blue, stepsize=0.06);
```



This time no singularities are directly encountered, and the ploting algorithm simply stepped over the line $t = 1$ and continued on. The input requested solutions satisfying initial conditions starting at $t = 0$. **DEplot** delivered them, but also plotted extraneous solution curves that are not actually connected to the ones that were requested.

Using a more sophisticated plotting algorithm might help. The next entry is just like the last one except for two things.

1. The stepsize is set back to 0.05.

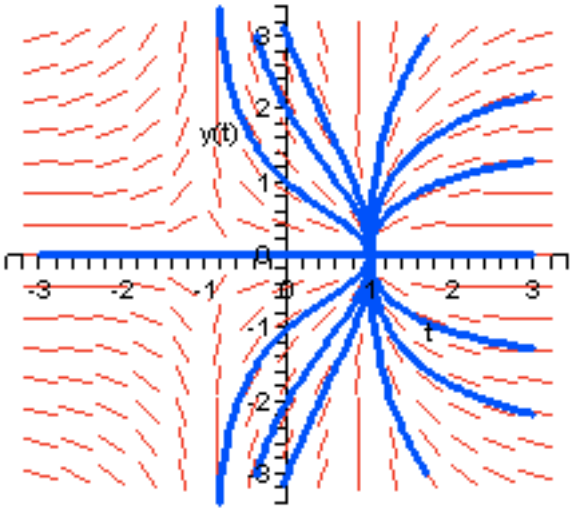
2. The equation

```
method=rkf45
```

has been added, telling **DEplot** to plot points using the more advanced Runge-Kutta-Fehlberg 4-5 algorithm

(Ledder, page 114). This algorithm varies the step size according to the difficulties it encounters in the field.

```
> DEplot( DE3, y(t), t=-3..3, y=-3..3, dirgrid=[15,15], arrows=line,
          {[y(0)=k]$k=-3..3}, linecolor=blue, stepsize=0.05,
          method=rkf45);
```



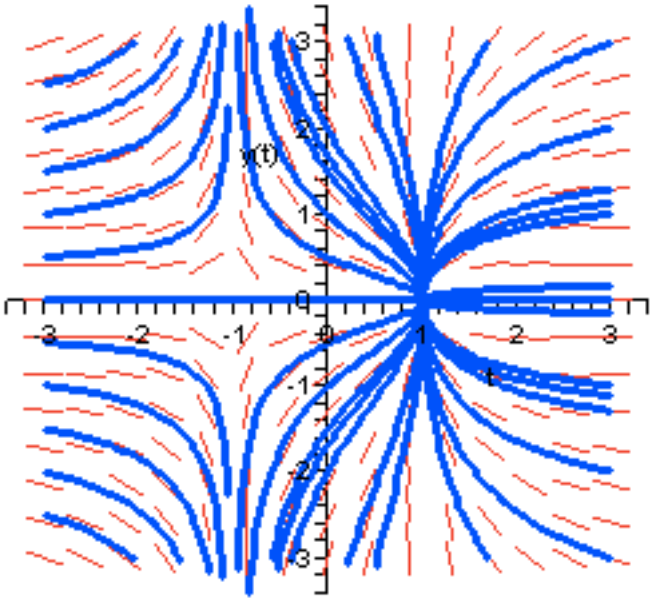
The output shows that the rkf45 algorithm also steps over the line $t = 1$,

Bottom line: **Maintain a skeptical attitude towards approximate solution curves as they pass near singularities in the direction field** .

Go with the flow

Having issued a fair warning, the last two examples in this section illustrate how to get a decent global view of solution behaviour, starting with our friend DE3.

```
> init1 := [y(-3)=k/2]$k=-5..5:  #Initial points on the line t = -3
  init2 := [y(0)=k/2]$k=-4..4:   #Initial points on the line t = 0
  init3 := [y(3)=k]$k=-3..3:     #Initial points on the line t = 3
  DEplot( DE3, y(t), t=-3..3, y=-3..3, dirgrid=[15,15], arrows=line,
         {init1,init2,init3}, linecolor=blue, stepsize=0.05,
          method=rkf45);
```



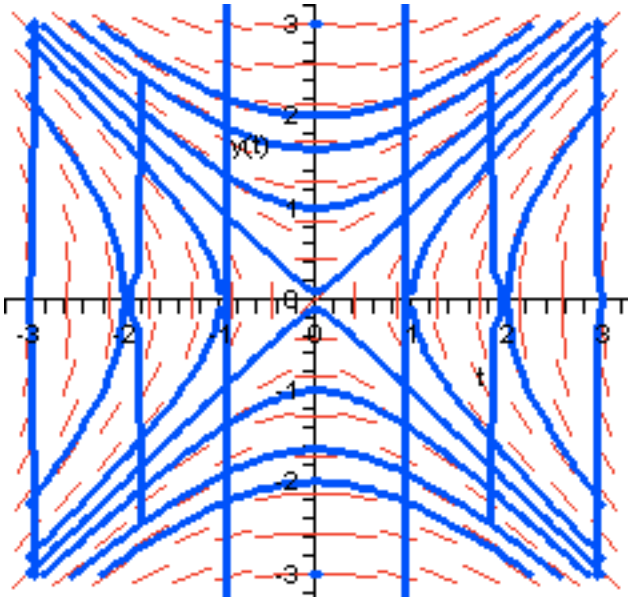
And concluding with the last equation in Section 1.

```
> DE4 := diff(y(t),t) = t/y(t);
```

$$DE4 := \frac{d}{dt} y(t) = \frac{t}{y(t)}$$

Solutions are first plotted using the default fixed step size Runge-Kutta algorighm.

```
> inits := { [y(0)=k]$k=1..3, [y(0)=-k]$k=1..3,
             [y(k)=0.1]$k=-3..3, [y(k)=-0.1]$k=-3..3 }:
  DEplot( DE4, y(t), t=-3..3, y=-3..3, dirgrid=[15,15], arrows=line,
          inits, linecolor=blue, stepsize=0.05);
```



The extraneous vertical lines appear because the default algorithm has jumped over the singularities.

Question: Where are the singularities in this field?

The second plot includes **method** = **rkf45** requesting the more sophisticated Runge-Kutta-Felhberg algorithm. The following picture shows that the algorithm has stopped short of the singularities. If you run the worksheet you will also see that **DEplot** issues several warnings about the singularities.

```
> DEplot( DE4, y(t), t=-3..3, y=-3..3, dirgrid=[15,15], arrows=line,
          inits, linecolor=blue, stepsize=0.05, method=rkf45);
```