

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

$$\frac{d}{dx} y(x) = f(x, y(x))$$

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

The tool you need: DEplot

The procedures that create direction fields and geometric (i.e. approximate) solution curves are in the **DEtools** package. All told, there are 125 procedures in this package, but only one will be used now, **DEplot**. This is loaded into the kernel using the entry `with(DEtools, DEplot)`

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

`[DEplot]` (1)

To generate a direction field for a first order ordinary differential equation named *DE* enter the following.

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

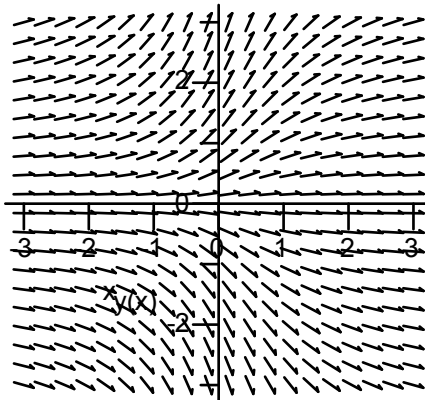
We first set some options for **DEplot**. This will save us the trouble of typing them into every **DEplot** procedure. These set the color of the line segments, and the color and thickness of the solution curves.

```
> DFoptions := color=black, linecolor=black, thickness=1;
```

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

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

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



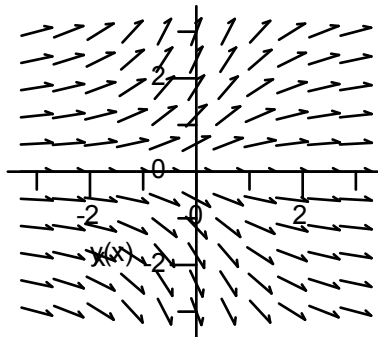
If you have the patience to count them you will find that there are 400 "harpoon" arrows (20 from left to

right and 20 from top to bottom). The number of arrows in each of these directions is controlled by an entry of the form

$$\text{dirgrid}=[m,n]$$

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

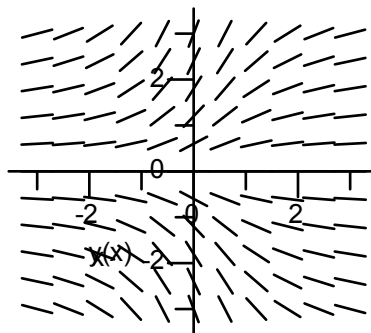
```
> DEplot( DE, y(x), x=-3..3, y=-3..3, DFOptions, dirgrid=[11,11] );
```



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

$$\text{arrows}=\text{line}$$

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



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

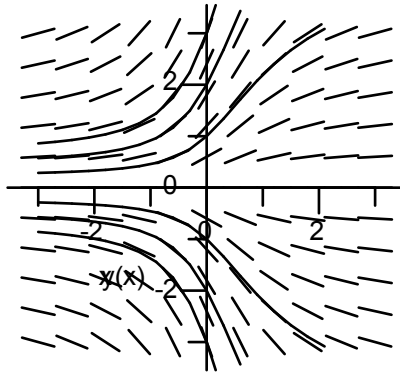
$$\{ [y(x_1)=y_1], [y(x_2)=y_2], \dots, [y(x_n)=y_n] \}$$

or the form

$$\{ [x_1, y_1], [x_2, y_2], \dots, [x_n, y_n] \}$$

See below.

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



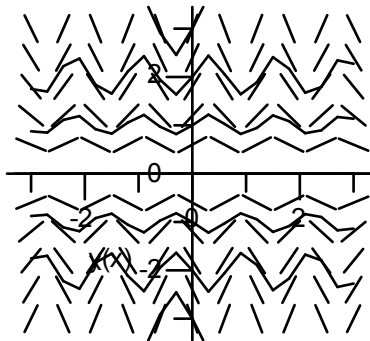
Compare this picture to the first one displayed in Section 1 (page 33).

Stepsize matters

Direction field information is used to plot the curves generated by **DEplot**. Slopes are averaged across the plot window. By default, the averaging algorithm is the classical Runge-Kutta fourth order method. See Simmons/Kranz, Chapter 9, Section 5. 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 smooth curves. The next example illustrates this.

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

$$DE2 := \frac{d}{dx} y(x) = \cos(5x) y(x)$$

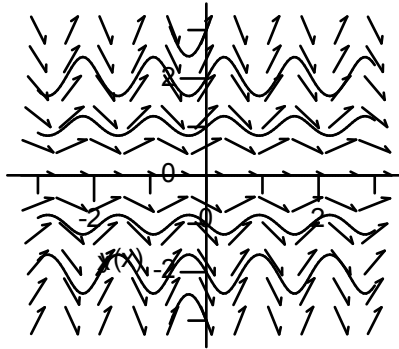


Since $b - a = 6$, the step size for this plot is $6/20 = 0.3$, which is too large to make a smooth curve. To control the step size, insert an equation of the form

stepsize=h

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

```
> DEplot( DE2, y(x), x=-3..3, y=-3..3, dirgrid=[11,11],
  {[y(0)=k]$k=-3..3}, DFoptions, stepsize=0.05);
```

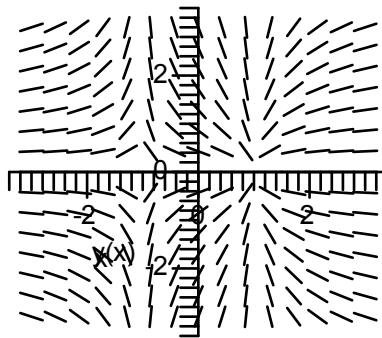


Keep your eye on $f(x, y)$

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

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

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



There are potential problems for any solution curve as it approaches the line $x = 1$ or the line $x = -1$. 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 $DE3$:

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

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

```
> DEplot( DE3, y(x), x=-3..3, y=-3..3, DFOptions, dirgrid=[15,15],
  arrows=line, {[y(0)=k]$k=-3..3}, stepsize=0.05);
```

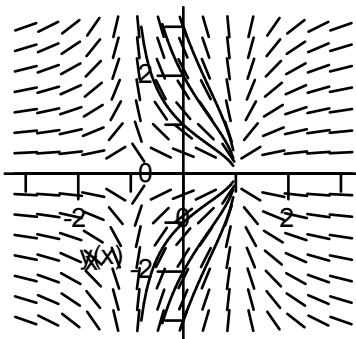
Warning, plot may be incomplete, the following error(s) were issued:
division by zero

Warning, plot may be incomplete, the following error(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

```

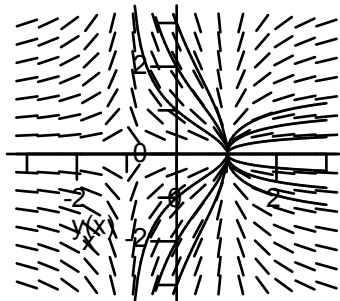


We actually got lucky with the stepsize in this **DEplot** (compare it to the plot at the bottom of page 38 in Section 1). To see why, examine the output when the stepsize is increased only slightly to 0.06.

```

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

```



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

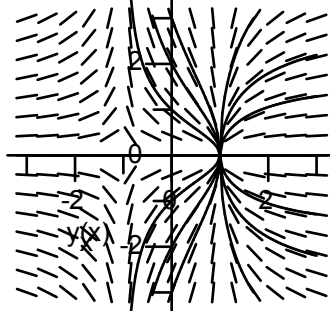
Using a more sophisticated plotting algorithm will help. The next entry is just like the last one except for two things.

1. The stepsize is set back to 0.05.
2. The equation `method=rkfk45` has been added, telling **DEplot** to plot points using a more

advanced algorithm called the *Runge-Kutta-Fehlberg 4-5 algorithm*.

The *rkf45* algorithm varies the step size according to the difficulties it encounters in the field.

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



The output shows that the *rkf45* algorithm also steps over the line $x = 1$ and *no warning messages were issued*.

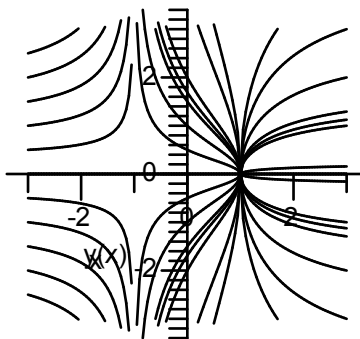
Bottom line. **Maintain a skeptical attitude towards approximate solution curves as they pass near singularities in the direction field.**

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*. Two warnings are issued by **DEplot**.

```
> init1 := [y(-3)=k/2]$k=-5..5: #Initial points on the line x = -3
  init2 := [y(0)=k/2]$k=-4..4: #Initial points on the line x = 0
  init3 := [y(3)=k]$k=-3..3: #Initial points on the line x = 3
  DEplot( DE3, y(x), x=-3..3, y=-3..3, DFoptions, dirgrid=[15,15],
          arrows=none, {init1,init2,init3}, stepsize=0.05,
          method=rkf45, xtickmarks=5);
```

```
Warning, plot may be incomplete, the following error(s) were issued:
  cannot evaluate the solution further right of -1.0000001,
  probably a singularity
Warning, plot may be incomplete, the following error(s) were issued:
  cannot evaluate the solution further right of -1.0000001,
  probably a singularity
```



Note that the equation `arrows = none` is used to suppress the arrows in the plot.

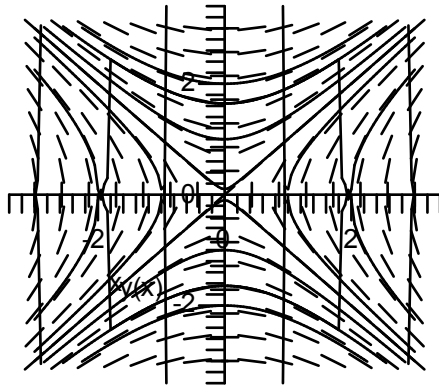
We conclude with the last equation in Section 1.

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

$$DE4 := \frac{d}{dx} y(x) = \frac{x}{y(x)} \quad (2)$$

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

```
> 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(x), x=-3..3, y=-3..3, dirgrid=[15,15], DFOptions,
        arrows=line, inits, stepsize=0.05);
```



Note that extraneous vertical lines appear. This is because the default algorithm has jumped over the singularities.

Question: Where are the singularities in this field? Answer: Every point on the x -axis is a singular point.

The second plot suppresses the arrows and the axes and uses `method = rkf45` thereby requesting the more sophisticated Runge-Kutta-Fehlberg algorithm. The picture shows that the algorithm has stopped short of the singularities. Moreover, if you run the code in your own worksheet you will see that **DEplot** issues several warnings about the singularities.

```
> DEplot( DE4, y(x), x=-3..3, y=-3..3, DFOptions, dirgrid=[15,15],
        arrows=none, axes=none, inits, stepsize=0.05,
        method=rkf45);
```

