Differential Equations&*Mathematica*

©1999 Bill Davis and Jerry Uhl
Produced by Bruce Carpenter   Published by Math Everywhere, Inc.
www.matheverywhere.com

DE.04 Modern DiffEq Issues Basics

B.1) Euler's method of faking the plot of the solution of a differential equation and how it highlights the fundamental issue of diffeq

Here's an innocent-looking differential equation:

y'[x] = y[x] (4 Cos[x] − y[x])
with y[0] = starter = 1.

If you want to get a plot of the solution, you ask *Mathematica* for a formula for the solution:

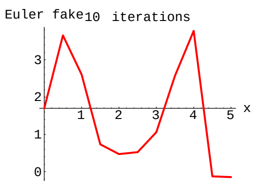
```
Clear[x, y];
starter = 1;
DSolve[{y′[x] == y[x] (4 Cos[x]^2 - y[x]), y[0] == starter}, y[x], x]
```

$$\{\{y[x] \to -\frac{E^{2 x+\text{Sin}[2 x]}}{-1 + \int 1\, d0 - \int E^{2 x+\text{Sin}[2 x]}\, dx}\}\}$$

This output is useless.
Try as it might, *Mathematica* could not come up with a useful formula for the solution y[x]. But there is a way to fake the plot of the solution even when you can't get a formula for it.
It comes from this code which you can play with.
Don't worry about the individual statements in the code right now.

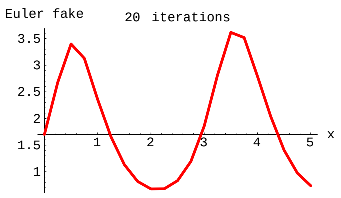
```
a = 0;
b = 5;
starter = 1.7;
y[a] = starter;
beginner = {a, y[a]};
```

```
Clear[x, y, next, jump, iterations, point, k];
jump[iterations_] := (b - a)/iterations;
next[{x_, y_}, iterations_] :=
 {x, y} + jump[iterations] {1, y (4 Cos[x]^2 - y)};
point[0, iterations_] = beginner;
point[k_, iterations_] :=
 point[k, iterations] = N[next[point[k - 1, iterations], iterations]];
Clear[Euler];
Euler[iterations_] := Show[Graphics[{Thickness[0.01], Red,
     Line[Table[point[k, iterations], {k, 0, iterations}]]}],
   PlotRange → All, Axes → True,
   AxesOrigin → beginner, AxesLabel → {"x", "Euler fake"},
   PlotLabel → iterations " iterations", DisplayFunction → Identity];
primitive = Show[Euler[10], DisplayFunction → $DisplayFunction];
```



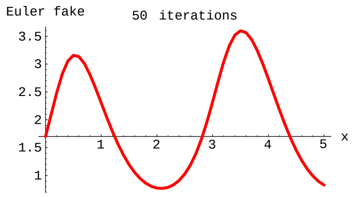
A primitive stick figure resulting from 10 iterations.
Here's what you get with 20 iterations:

```
better = Show[Euler[20], DisplayFunction → $DisplayFunction];
```



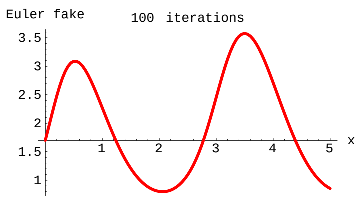
50 iterations:

```
evenbetter = Show[Euler[50], DisplayFunction → $DisplayFunction];
```



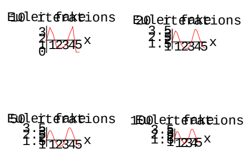
100 iterations:

```
darngood = Show[Euler[100], DisplayFunction → $DisplayFunction];
```



Notice that the last two plots are almost the same.
This is your signal that they are both reasonably good fake plots of the solution of the differential equation.

```
Show[GraphicsArray[{{primitive, better}, {evenbetter, darngood}}]];
```



The more iterations you use, the better the fake plot of the solution of the given differential equation.

□B.1.a)

Here is the exponential differential equation

y'[x] = 0.3 y[x]
with y[0] = 4

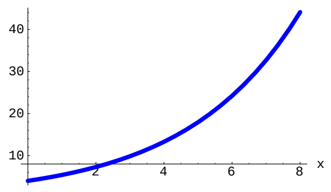
and a formula for its solution:

```
thisr = 0.3;
thisstarter = 4;
Clear[x, r, starter];
solution = DSolve[{y′[x] == r y[x], y[0] == starter}, y[x], x] /.
   {r → thisr, starter → thisstarter};
y[x_] = y[x] /. solution[[1]]
```

$4\, E^{0.3\, x}$

and a plot for $0 \le x \le 8$:

```
realplot =
  Plot[y[x], {x, 0, 8}, PlotStyle → {{Blue, Thickness[0.015]}},
   PlotRange → All, AspectRatio → 1/GoldenRatio, AxesLabel → {"x", ""}];
```



Compare the Euler fake plots of the solution with the true plot.
Discuss what you see.

□ **Answer:**

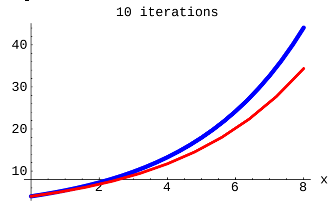
Copy, paste, and edit the code used above to get a Euler faker of a plot of the solution of the differential equation

y'[x] = 0.3 y[x]
with y[0] = 4

and show the fake plots together with the plot of the true solution.

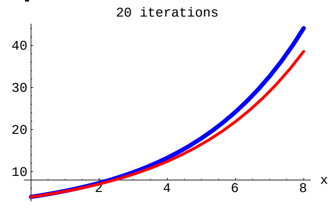
```
a = 0;
b = 8;
```

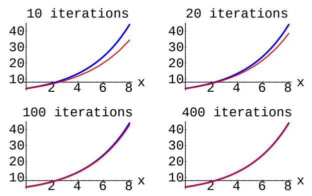
```
starter = 4;
y[a] = starter;
beginner = {a, y[a]};
Clear[x, y, next, jump, iterations, point, k];
                         b - a
jump[iterations_] := ──────────;
                      iterations
next[{x_, y_}, iterations_] := {x, y} + jump[iterations] {1, 0.3 y};

point[0, iterations_] = beginner;
point[k_, iterations_] :=
 point[k, iterations] = N[next[point[k - 1, iterations], iterations]];

Clear[Euler];
Euler[iterations_] := Show[Graphics[{Thickness[0.01], Red,
    Line[Table[point[k, iterations], {k, 0, iterations}]]}],
  PlotRange → All,
  Axes → True, AxesOrigin → beginner, AxesLabel → {"x", ""},
  PlotLabel → iterations " iterations", DisplayFunction → Identity];

rough = Show[realplot, Euler[10], PlotLabel → "10 iterations",
  DisplayFunction → $DisplayFunction];
```



Step up the number of iterations to 20:

```
better = Show[realplot, Euler[20],
     PlotLabel -> "20 iterations",
     DisplayFunction -> $DisplayFunction];
```





This is good stuff.

□B.1.b) The advantage Euler's method gives you

What is the big advantage you get from Euler's method?

□**Answer:**

You get a reasonably good plot of a solution even though you might not be able to come up with a formula for the solution.

This is a big, big plus for you.

□B.1.c) The fundamental issue of diffeq

In Euler's method, you are given a diffeq

y'[x] = f[x, y[x]]

with y[0] = b, a given number

You run Euler's and get a nice approximate plot of the solution starting at {0, b}.

How does this illustrate the fundamental issue of diffeq?

□**Answer:**

In the words of the great American mathematician Peter Lax:

"Every differential equation has a solution and . . . this solution is uniquely determined by initial data [on y[0]]."

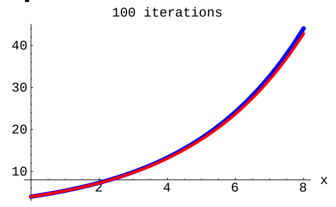
□B.1.d) The idea behind Euler's faker

Try to explain why Euler's method works.

□**Answer:**

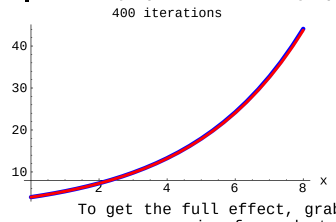
Step up the number of iterations to 100:

```
evenbetter = Show[realplot, Euler[100],
     PlotLabel -> "100 iterations",
     DisplayFunction -> $DisplayFunction];
```



Step up the number of iterations to 400:

```
darngood = Show[realplot, Euler[400],
     PlotLabel -> "400 iterations",
     DisplayFunction -> $DisplayFunction];
```



```
To get the full effect, grab all four plots and animate
         running forward at a fairly slow speed.
```

Right you are, governor.

The more iterations you use, the better the Euler fake plot you get.

One more peek:

```
Show[GraphicsArray[{{rough, better},
        {evenbetter, darngood}}]];
```

Here is a differential equation

y'[x] = Sin[x] y[x]

with y[0] = 2

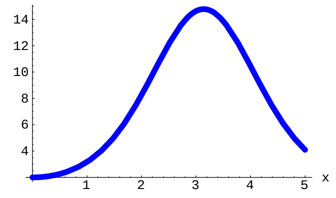
and a formula for its solution:

```
starter = 2;
Clear[x, y];
solution = DSolve[{y'[x] == Sin[x] y[x], y[0] == starter}, y[x], x];
y[x_] = y[x] /. solution[[1]]
```

$2\,E^{1-\text{Cos}[x]}$

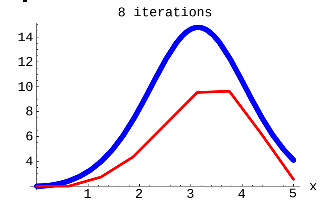
and a plot for 0 ≤ x ≤ 5:

```
realplot = Plot[y[x], {x, 0, 5}, PlotStyle → {{Blue, Thickness[0.02]}},
                                         1
   PlotRange → All, AspectRatio → ─────────────, AxesLabel → {"x", ""}];
                                    GoldenRatio
```



Now look at the Euler faker coming from 8 iterations:

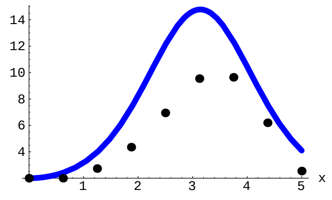
```
a = 0;
b = 5;
y[a] = starter;
beginner = {a, y[a]};
Clear[x, y, next, jump, iterations, point, k];
                         b - a
jump[iterations_] := ──────────;
                      iterations

next[{x_, y_}, iterations_] := {x, y} + jump[iterations] {1, Sin[x] y};

point[0, iterations_] = beginner;
point[k_, iterations_] :=
 point[k, iterations] = N[next[point[k - 1, iterations], iterations]];

Clear[Euler];
Euler[iterations_] := Show[Graphics[{Thickness[0.01], Red,
    Line[Table[point[k, iterations], {k, 0, iterations}]]}],
  PlotRange → All,
  Axes → True, AxesOrigin → beginner, AxesLabel → {"x", ""},
  PlotLabel → iterations " iterations", DisplayFunction → Identity];
veryrough = Show[realplot, Euler[8], PlotLabel → "8 iterations",
  DisplayFunction → $DisplayFunction];
```



Not a very good fake, but this plot exposes what the faker does.

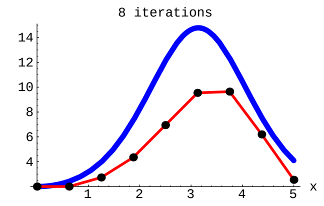
You can see that the faker actually moves by generating a list of points and moving on straight line segments that connect the points. Here are the points:

```
Clear[fakepoints, fakepointplot];
fakepoints[iterations_] :=
 Table[point[k, iterations], {k, 0, iterations}];
fakepointplot[iterations_] :=
 ListPlot[fakepoints[iterations], PlotStyle → {PointSize[0.03]},
  AxesOrigin → {0, 0}, DisplayFunction → Identity];
Show[realplot, fakepointplot[8]];
```



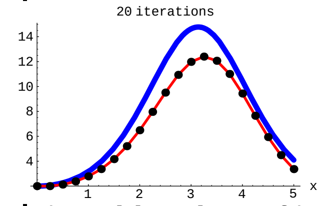
The faker connects these points with line segments:

```
Show[realplot, Euler[8], fakepointplot[8],
 PlotLabel → "8 iterations"];
```

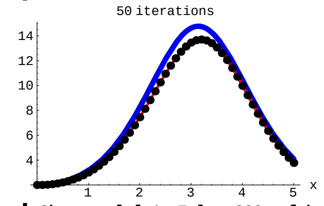


When you step up the iteration number, you shorten the jump between successive points:

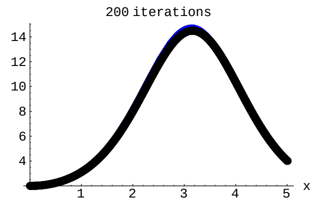
```
Show[realplot, Euler[20], fakepointplot[20],
 PlotLabel → 20 iterations];
```



```
Show[realplot, Euler[50], fakepointplot[50],
 PlotLabel → 50 iterations];
```



```
Show[realplot, Euler[200], fakepointplot[200],
 PlotLabel → 200 iterations];
```



So far, so good.

The real question that remains is:

Why does the fake curve get better and better as you increase the number of iterations?

To answer this, look at the code that generates the fake points:

  next[{x, y}, iterations] =

    {x, y} + jump[iterations] {1, Sin[x] y};

  point[k, iterations] =

    N[next[point[k – 1, iterations], iterations]].

This code tells you that once the Euler faker has settled on a point {x, y}, then it moves off that point with the same growth rate as the solution of the differential equation

    y'[x] = Sin[x] y[x]

that passes through that point. So the big differences between the true plot and the fake plot is:

→ As x advances, the actual solution y[x] updates its growth rate y'[x] instantaneously.

→ But the fake waits for a while to update the growth rate, updating its growth rate only at each fake point.

When you use a small jump, then you can expect the growth rate of the fake to be nearly the same as the actual growth rate of the function.

To get a small jump on a plotting interval [a, b], you use a high iteration number because

$$\text{jump[iterations]} = \frac{\text{b–a}}{\text{iterations}}.$$

This is why high iteration numbers usually give better results than low iteration numbers.

□B.1.e) The person behind the idea

Say a little bit about the great Swiss mathematician Leonhard Euler, 1707-1783, who originally came up with the idea of faking the plots of solutions of differential equations.

□ **Answer:**

Euler (pronounced "Oiler") was certainly one of the greatest mathematical scientists of all time. In addition to his advanced work in geometry, he wrote the first accurate and complete treatise on calculus. Along the way, he introduced the symbols

$$e, \pi, \text{ and } i = \sqrt{-1}$$

and the abbreviations for Sine, Cosine, Tangent, and the other trig functions. Even the notation f[x] owes its life to the fertile mind of Euler.

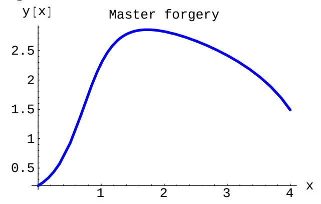
Over and above his mathematics, he had time to take up physics, astronomy, hydrodynamics, optics, electricity, magnetism, ship building, geographical maps, medicine, theology, and oriental languages. Even though he became blind 12 years before his death, nearly half his published works were written during the time he was blind. Not one to neglect the social side of life, Euler spent many years with friends in the famous court of Catherine I in St. Petersburg, Russia.

B.2) Euler's faker and Mathematica's faker, `NDSolve`

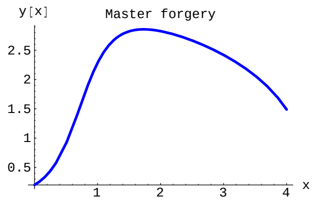
Programmed into *Mathematica* is an instruction called `NDSolve` that will accomplish for you everything that Euler's method did for faking the plot of a solution of a differential equation. And there's an added plus:
You don't have to screw around with setting the number of iterations.
Here's the `NDSolve` instruction working to get an excellent fake plot for $0 \le x \le 6$ of the differential equation

$y'[x] = 3.2\,\mathrm{Sin}[y[x]] - 0.3\,x^2$
with $y[0] = 0.2$:

```
a = 0;
b = 4;
starter = 0.2;
Clear[solution, x, y, fakey];
solution = NDSolve[
  {y'[x] == 3.2 Sin[y[x]] - 0.3 x^2, y[0] == starter}, y[x], {x, a, b}];
fakey[x_] = y[x] /. solution[[1]];

masterfakeplot =
 Plot[fakey[x], {x, a, b}, PlotStyle → {{Blue, Thickness[0.01]}},
  AxesLabel → {"x", "y[x]"}, AxesOrigin → {a, starter}, PlotRange → All,
  AspectRatio → 1/GoldenRatio, PlotLabel → "Master forgery"];
```



The `NDSolve` instruction is just waiting for you to use it.



□B.2.c)

But isn't the actual formula always better than a fake?

□**Answer:**

It all depends on what you want to do.
If what you want is a plot of the solution, then the fake should be as good as the actual formula.
If you want to do theoretical analysis of the true solution, then the actual formula is what you want. The only trouble is that the actual formula is available only in special situations; the fake is nearly always available.

□B.2.d)

Does this mean that you should give up on Euler's method?

□**Answer:**

Hell no!
For serious practical situations, you want something better. But as a theoretical tool to use to unlock more secrets of calculus, Euler's method is just the ticket.

□B.2.a)

Discuss the mathematical principles behind the `NDSolve` instruction.

□**Answer:**

The mathematical principles behind the `NDSolve` instruction are the same as the mathematical principles behind Euler's method. Except in the case of the `NDSolve` instruction, extra features typical of professionally written software are included.
You'll hear more about these extra features later.

□B.2.b)

Look at this:

```
a = 0;
b = 4;
starter = 0.2;
Clear[solution, x, y, fakey];

solution = NDSolve[
  {y'[x] == 3.2 Sin[y[x]] - 0.3 x^2, y[0] == starter}, y[x], {x, a, b}];
fakey[x_] = y[x] /. solution[[1]]
InterpolatingFunction[{{0., 4.}}, <>][x]
```

What does the output mean?

□**Answer:**

This output reflects the fact that `NDSolve` first produces a bunch of points and then strings them together with an interpolating function -- just as Euler's method does. The formula for this interpolating function is not available, but you can plot it:

```
masterfakeplot =
 Plot[fakey[x], {x, a, b}, PlotStyle → {{Blue, Thickness[0.01]}},
  AxesLabel → {"x", "y[x]"}, AxesOrigin → {a, starter}, PlotRange → All,
  AspectRatio → 1/GoldenRatio, PlotLabel → "Master forgery"];
```

B.3) Reading the diffeq through flow plots

□B.3.a.i)

Here's an old friend, the logistic differential equation:

```
Clear[diffeq, y, t, f, starter];
b = 10;
f[t_, y_] = 0.4 y (1 - y/b);
diffeq = {y'[t] == f[t, y[t]], y[0] == starter}
```

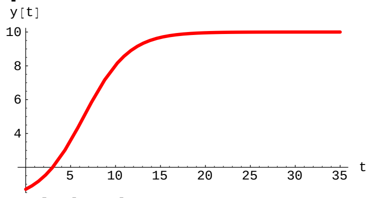
$\{y'[t] == 0.4\left(1 - \frac{y[t]}{10}\right) y[t],\ y[0] == \mathrm{starter}\}$

Here comes a plot of the solution of this differential equation corresponding to given starter data $y[0] = 0.7$:

```
diffeq
```

$\{y'[t] == 0.4\left(1 - \frac{y[t]}{10}\right) y[t],\ y[0] == \mathrm{starter}\}$

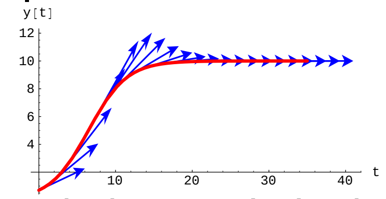
```
Clear[y];
starter = 0.7;
endtime = 35;
y[t_] = y[t] /. NDSolve[diffeq, y[t], {t, 0, endtime}][[1]];

solutionplot = Plot[y[t], {t, 0, endtime}, PlotRange → All,
  PlotStyle → {{Thickness[0.01], Red}}, AxesLabel → {"t", "y[t]"},
  AspectRatio → 1/2];
```



Now look at this:

```
scale = 6;
tangentflow =
 Table[Arrow[{1, f[t, y[t]]}, Tail → {t, y[t]}, VectorColor → Blue,
   ScaleFactor → scale, HeadSize → 1.5], {t, 0, endtime, endtime/20}];
```

```
both = Show[tangentflow, solutionplot, PlotRange → All,
  Axes → True, AxesLabel → {"t", "y[t]"}, AspectRatio → 1/2];
```

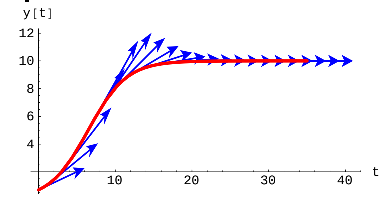


Describe what you see and explain why you see it.

□ **Answer:**

Take another look:

```
Show[both];
```



What you see is the solution plot being guided by its tangent vectors.

This came out this way for the same reason that Euler's method works.

To wit: The vectors plotted are scaled versions of

{1, f [t, y[t]]} with tails at {t, y[t]}.

Because y[t] is a solution of y'[t] = f[t, y[t]],

these vectors are the same as

{1, y'[t]} with tail at {t, y[t]}.

But the slope of

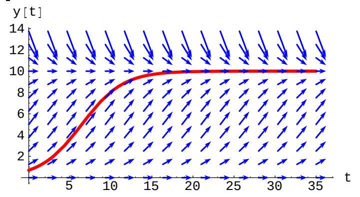
{1, y'[t]} is y'[t],

which is the instantaneous growth rate of y[t] and that's the reason these are tangent vectors.

□B.3.a.ii) Flow plots and corks

Look at this:

```
scalefactor = 1.1;
flowplot = Table[Arrow[scalefactor {1, f[t, y]}, Tail → {t, y},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → 0.8],
  {t, 0, endtime, endtime/15}, {y, 0, 1.4 b, b/8}];

Show[solutionplot, flowplot];
```

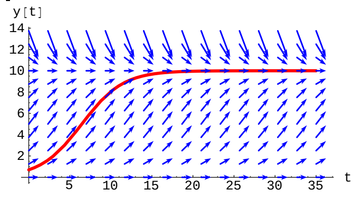


This shows the solution plotted in part i) going with the flow.
What else can you see from this plot?

□ **Answer:**

Take another look:

```
Show[solutionplot, flowplot];
```

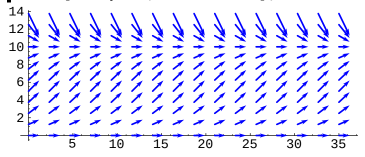


Think of it this way:

Each plotted vector is tangent to the solution plot that goes through its tail. Think of the whole t y-plane as fluid flowing in the directions indicated by the tangent vectors plotted above. The plot of a solution is what you get when you drop a cork into the flow and let it float along.

This way you can use your eyes to see how solutions plot out:

```
Show[flowplot, Axes → True];
```



When you start with

y[0] = starter between 0 and 10,

the corresponding solution increases and eventually snuggles onto a straight line. When you start with

y[0] = starter bigger than 10,

then the corresponding solution decreases and eventually snuggles onto the same straight line.

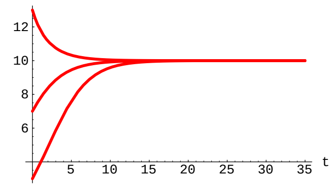
Check it out for a couple of other starting points:

```
Clear[y];
diffeq
```

$$\{y'[t] == 0.4\left(1-\frac{y[t]}{10}\right) y[t],\ y[0] == 0.7\}$$

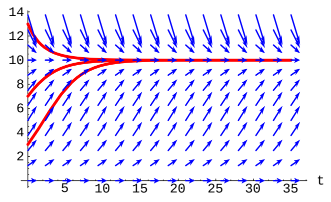
```
Clear[y1, y2, y3, y, t];
{starter1, starter2, starter3} = {3, 7, 13};
y1[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter1, y[t], {t, 0, endtime}][[1]];
y2[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter2, y[t], {t, 0, endtime}][[1]];
y3[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter3, y[t], {t, 0, endtime}][[1]];
```

```
solutionplots = Plot[{y1[t], y2[t], y3[t]}, {t, 0, endtime},
  PlotStyle → {{Thickness[0.01], Red}}, AspectRatio → 1/GoldenRatio,
  PlotRange → All, AxesLabel → {"t", ""}];
```



See these solutions go with the flow:

```
Show[solutionplots, flowplot];
```



Math happens.

Play with other starting values.

□B.3.b.i) More flow plots

Here's a new differential equation:

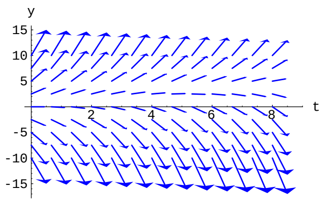
```
Clear[diffeq, y, t, f, starter];
b = 10;
f[t_, y_] = y - 0.5 t;
diffeq = {y'[t] == f[t, y[t]], y[0] == starter}
```

$$\{y'[t] == -0.5\,t + y[t],\ y[0] == starter\}$$

And its flow plot:

```
scalefactor = 0.5;
{ylow, yhigh} = {-10, 10};
{tlow, thigh} = {0, 8};
```

```
flowplot = Table[Arrow[{1, f[t, y]},
    Tail → {t, y}, VectorColor → Blue, ScaleFactor → scalefactor],
  {t, tlow, thigh, (thigh - tlow)/12}, {y, ylow, yhigh, (yhigh - ylow)/8}];

Show[flowplot, Axes → True, AxesLabel → {"t", "y"},
  AspectRatio → 1/GoldenRatio];
```

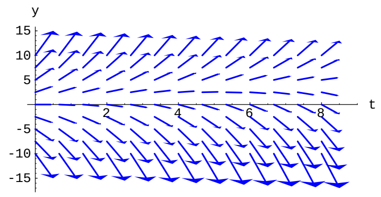


Interpret this flow plot.

□ **Answer:**

Take another look:

```
Show[flowplot, Axes → True, AxesLabel → {"t", "y"}, AspectRatio → 1/2];
```



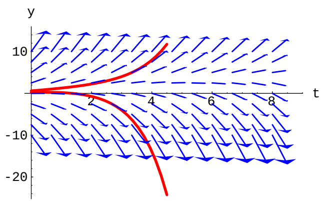
From this flow plot, you can spot at least two types of solutions of the differential equation:

```
diffeq
{y′[t] == -0.5 t + y[t], y[0] == starter}
```

Here is a sample of each shown with the flow plot:

```
Clear[y1, y2, y, t];
{starter1, starter2} = {0.2, 0.6};
endtime = 4.5;
```

```
y1[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter1, y[t], {t, 0, endtime}][[1]];
y2[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter2, y[t], {t, 0, endtime}][[1]];
solutionplots = Plot[{y1[t], y2[t]}, {t, 0, endtime},
  PlotStyle → {{Thickness[0.01], Red}}, DisplayFunction → Identity];

Show[solutionplots, flowplot, Axes → True, AxesLabel → {"t", "y"},
  AspectRatio → 1/GoldenRatio, DisplayFunction → $DisplayFunction];
```



B.4) Systems of interacting differential equations: The predator-prey model

This predator-prey model was orginally proposed and studied by Volterra in 1926 in an effort to explain the oscillatory levels of certain fish harvests in the Adriatic Sea.

Here's the idea:
Two species coexist in a closed environment. One species, the predator, feeds on the other, the prey. There is always plenty of food for the prey, but the predators eat nothing but the prey.
Put

pred[t] = population of predators at time t.

Put

prey[t] = population of prey at time t.

It's reasonable to assume that there are positive constants a and b such that:

$$\text{prey}'[t] = a\,\text{prey}[t] - b\,\text{prey}[t]\,\text{pred}[t]$$

because the abundance of food for the prey allows the birth rate of the prey to be proportional to their current number, and the death rate of prey is proportional to both the current number of prey and the current number of predators.

It also makes some sense to assume that there are positive constants c and d such that:

$$\text{pred}'[t] = -c\,\text{pred}[t] + d\,\text{pred}[t]\,\text{prey}[t]$$

because it's reasonable to assume that the death rate of the predators is likely to be proportional to the current population of predators and that the birth rate of the predators is proportional to both the current number of the predators and the size of the food supply (the prey).
This gives you two simultaneous differential equations

$$\text{prey}'[t] = a\,\text{prey}[t] - b\,\text{prey}[t]\,\text{pred}[t],$$
$$\text{pred}'[t] = -c\,\text{pred}[t] + d\,\text{pred}[t]\,\text{prey}[t].$$

```
Simultaneous differential equations are just the ticket when you want to
            see how one process interacts with another.
```

□B.4.a)

Examine what happens for the sample choice of proportionality constants

$a = 0.7$, $b = 0.3$, $c = 0.44$, and $d = 0.08$

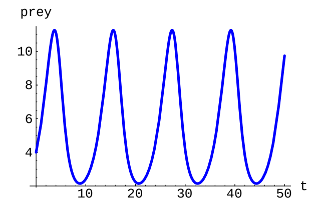
with the prey starting off with a population of 4.0 units and the predators starting out with a population of 1.2 units.

Here the units could be thousands or millions so that a fractional population can make sense.

□ **Answer:**

Here is a plot of *Mathematica*'s faker of the prey population as a function of time t for the first 50 time units:

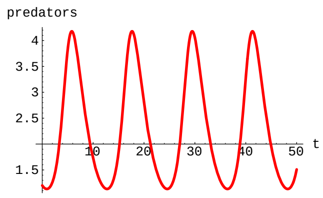
```
endtime = 50;
preystarter = 4.0;
predstarter = 1.2;
a = 0.7;
b = 0.3;
c = 0.44;
d = 0.08;
Clear[pred, prey, t];
approxsolutions = NDSolve[{prey′[t] == a prey[t] - b prey[t] pred[t],
    pred′[t] == -c pred[t] + d pred[t] prey[t],
    prey[0] == preystarter, pred[0] == predstarter},
   {prey[t], pred[t]}, {t, 0, endtime}];
pred[t_] = pred[t] /. approxsolutions[[1]];
prey[t_] = prey[t] /. approxsolutions[[1]];

preyplot =
 Plot[prey[t], {t, 0, endtime}, PlotStyle → {{Blue, Thickness[0.01]}},
  AxesLabel → {"t", "prey"}, AspectRatio → 1/GoldenRatio];
```



Hey! A cyclical pattern.

And the predator population as a function of time t:

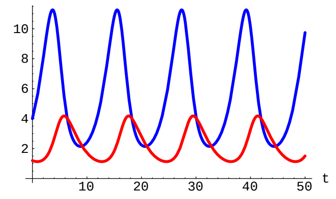
```
predatorplot =
 Plot[pred[t], {t, 0, endtime}, PlotStyle → {{Red, Thickness[0.01]}},
  AxesLabel → {"t", "predators"}, AspectRatio → 1/GoldenRatio];
```



Another cyclical pattern. Here they are together:

The plot of the prey population is thicker than the plot of the predator population.

```
both = Plot[{prey[t], pred[t]}, {t, 0, endtime},
  PlotStyle → {{Blue, Thickness[0.01]}, {Red, Thickness[0.01]}},
  AxesLabel → {"t", ""},
  AspectRatio → 1/GoldenRatio, DisplayFunction → Identity];

Show[both, DisplayFunction → $DisplayFunction];
```



When there are enough prey to sustain strong predator growth, the predators begin to grow so much that they outstrip their food supply. This puts both populations into a decline until the point at which the predators are not a powerful menace; then the prey become numerous enough to support strong predator growth. The cycles go on and on.

□B.4.c)

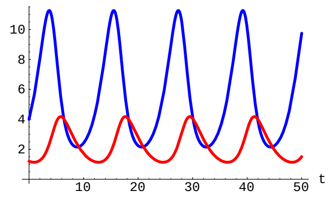
Estimate the length of one cycle in time units.
Plot one cycle, two cycles, three cycles, and four cycles.

□ Answer:

Put in horizontal lines shooting out from the vertical axis reflecting the information that

$$\text{pred}[0] = 0.9 \text{ and } \text{prey}[0] = 4.$$

```
bothwithlines = Plot[{prey[t], pred[t]}, {t, 0, endtime},
  PlotStyle → {{Blue, Thickness[0.01]}, {Red, Thickness[0.01]}},
  AxesLabel → {"t", ""}, AspectRatio → 1/GoldenRatio,
  Epilog → {Line[{{0, predstarter}, {endtime, predstarter}}],
    Line[{{0, preystarter}, {endtime, preystarter}}]}];
```

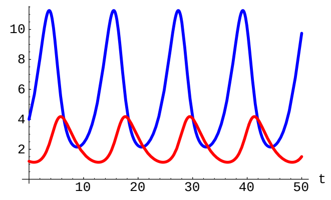


Look at those populations interacting with each other.

□B.4.b)

Here are the plots of both populations shown together again:

```
Show[both, DisplayFunction → $DisplayFunction];
```



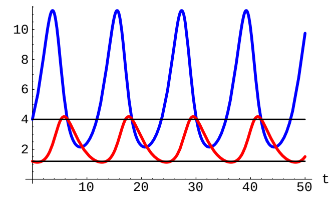
Discuss the relationships between the curves.

□ Answer:

Cyclic oscillations of both populations are evident as all get out.

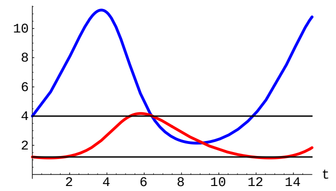
Feast your eyes on the vivid relationship between the crests and dips of the predator population and those of the prey population! The cycles are slightly out of sync. When you think about it, they should be slightly out of sync. Look again:

```
Show[both, DisplayFunction → $DisplayFunction];
```



A closer look:

```
Show[bothwithlines, PlotRange → {{0, 15}, Automatic}];
```



One cycle is about 12 time units by eyeball estimate.

Use *Mathematica* to improve on this:

```
FindRoot[prey[t] == preystarter, {t, 12}]
{t → 11.8355}
```

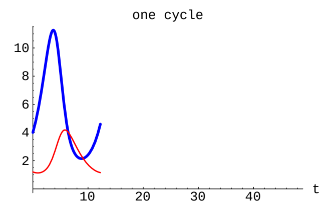
Check:

```
FindRoot[pred[t] == predstarter, {t, 12}]
{t → 11.8354}
```

The eyeball was not so bad.

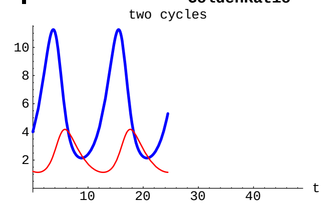
A plot of one cycle:

```
cycle = 12.2355;
one = Plot[{prey[t], pred[t]}, {t, 0, cycle},
  PlotStyle → {{Blue, Thickness[0.01]}, {Red, Thickness[0.005]}},
  AxesLabel → {"t", ""}, PlotRange → {{0, 4 cycle}, Automatic},
  AspectRatio → 1/GoldenRatio, PlotLabel → "one cycle"];
```



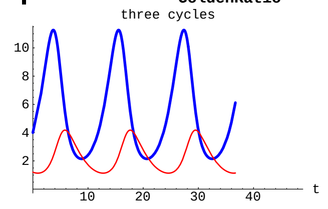
Two cycles:

```
two = Plot[{prey[t], pred[t]}, {t, 0, 2 cycle},
  PlotStyle → {{Blue, Thickness[0.01]}, {Red, Thickness[0.005]}},
  AxesLabel → {"t", ""}, PlotRange → {{0, 4 cycle}, Automatic},
  AspectRatio → 1/GoldenRatio, PlotLabel → "two cycles"];
```



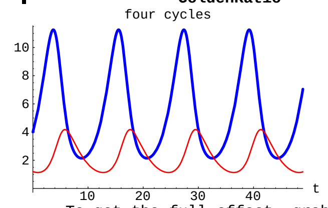
Three cycles:

```
Plot[{prey[t], pred[t]}, {t, 0, 3 cycle},
  PlotStyle → {{Blue, Thickness[0.01]}, {Red, Thickness[0.005]}},
  AxesLabel → {"t", ""}, PlotRange → {{0, 4 cycle}, Automatic},
  AspectRatio → 1/GoldenRatio, PlotLabel → "three cycles"];
```



And four cycles:

```
Plot[{prey[t], pred[t]}, {t, 0, 4 cycle},
  PlotStyle → {{Blue, Thickness[0.01]}, {Red, Thickness[0.005]}},
  AxesLabel → {"t", ""}, PlotRange → {{0, 4 cycle}, Automatic},
  AspectRatio → 1/GoldenRatio, PlotLabel → "four cycles"];
```



To get the full effect, grab all four plots of the cycles, align, animate and run forward slowly.

Nice excursion into mathematical ecology.

As William E. Boyce and Richard DiPrima point out in their book, Elementary Differential Equations (Wiley, 1977), "Cyclic variations of predator and prey as [predicted above] are often observed in nature."

In fact E.P. Odum [Fundamentals of Ecology, Saunders, 1953] was able to use the fur catch records of the Hudson Bay Company from 1850 to 1930 to estimate the Canadian lynx and snowshoe hare populations during these years. Odum's curves are not too different in character from the curves you see above.

All mathematical ecologists agree that the predator-prey model is not an end in itself, but is important as a tool in the quest for asking the right questions.

□B.4.d)

Why all this fooling around with fake plots?
Wouldn't actual formulas for pred[t] and prey[t] be more convenient?

□Answer:

It all depends on what you want to do.

If what you want is a plot, then the fake should be as good as the actual formula.

If you want to do theoretical analysis of the true solution, then the actual formula is what you want. The only trouble is that the actual formula is available only in special situations; the fake is always available.

In the case of the predator-prey model, the formulas so treasured in old-fashioned math classes are not available.

Try it:

```
Clear[a, b, c, d, pred, prey, t];
DSolve[{prey′[t] == a prey[t] - b prey[t] pred[t],
   pred′[t] == -c pred[t] + d pred[t] prey[t], prey[0] == 4, pred[0] == 1},
  {prey[t], pred[t]}, t]
{}
```

Mathematica failed to come up with formulas for pred[t] and prey[t].

You can't chalk this up as a fault of *Mathematica*'s, because no scientist has ever succeeded in finding formulas for pred[t] and prey[t]. For the predator-prey model, the fake is the only game in town.

□B.4.e)

Where do you go for further reading on predator-prey models?

□Answer:

Predator-prey models have been (and still are) under heavy study by biologists and mathematicians. For further reading, here are some good places to begin:

```
1. E. Batschelet,Introduction to Mathematics for Life Scientists,
        Springer-Verlag, 1979. (introductory and clear)
2. Peter Lax,Samuel Burstein and Anneli Lax, Calculus with Applications
        and Computing,Springer-Verlag,1976. (serious mathematics)
```

```
3. J. D. Murray, Mathematical Biology,Springer-Verlag,1990 (serious
        mathematics and serious biology).
4. William E. Boyce and Richard DiPrima, Elementary Differential
   Equations, Wiley, 1977 (serious math although somewhat dated in
        presentation)
```

□B.4.f)

What version of Euler's method underlies *Mathematica*'s fake plots above?

□Answer:

```
None of the instructions below can be activated.
```

To fake a plot of the solution of y'[x] = f[x, y[x]], Euler's method uses

next[{x, y}] = {x, y} + jump {1, f[x, y]}

point[k] = N[next[point[k – 1]]]

To fake a plot of the solution of the simultaneous differential equations

x'[t] = f[t, y[t], x[t]]

y'[t] = g[t, x[t], y[t]],

Euler's method uses:

next[{t, x, y}] = {t, x, y} + jump {1, f[t, x, y], g[t, x, y]}

point[k] = point[k] = N[next[point[k – 1]]].

The fake points for x[t] as a function of t are fished out from the first two slots of point[k], and the fake points for y[t] as a function of t are fished out from the first and the third slots of point[k].

You could program it yourself, but why bother?

DE.04 Modern DiffEq Issues Tutorials

T.1) Sensitive dependence on starter data

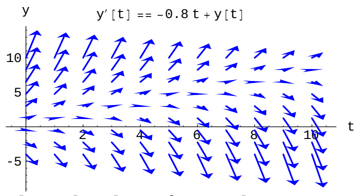
□T.1.a.i) Flow plots

Here's a simple differential equation:

```
Clear[diffeq, y, t, f, starter];
b = 10;
f[t_, y_] = y - 0.8 t;
setup =
 (diffeq = {y'[t] == f[t, y[t]], y[0] == starter}) // ColumnForm
y′[t] == -0.8 t + y[t]
y[0] == starter
```

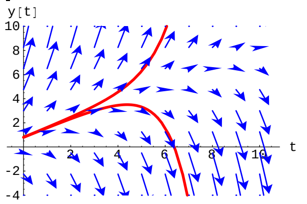
And a flow plot:

```
endtime = 6;
scalefactor = 0.4;
{ylow, yhigh} = {-4, 10};
{tlow, thigh} = {0, 10};

flowplot = Table[Arrow[{1, f[t, y]}, Tail → {t, y},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → 0.8],
  {t, tlow, thigh, (thigh - tlow)/10}, {y, ylow, yhigh, (yhigh - ylow)/8}];

Show[flowplot, Axes → True, AxesLabel → {"t", "y"}, AspectRatio → 1/2,
 PlotLabel → setup⟦1, 1⟧];
```



And see the plots of two solutions:

```
Clear[y1, y2, y3, t];
{starter1, starter2} = {0.79, 0.81};
endtime = 7;

y1[t_] = y[t] /.
  NDSolve[(diffeq /. starter -> starter1), y[t], {t, 0, endtime}][[1]];
y2[t_] = y[t] /.
  NDSolve[(diffeq /. starter -> starter2), y[t], {t, 0, endtime}][[1]];

solutionplots = Plot[{y1[t], y2[t]}, {t, 0, endtime},
   PlotStyle ->
 {{Thickness[0.01], Red}}, AspectRatio -> 1/GoldenRatio,
  PlotRange -> {ylow, yhigh},
  AxesLabel -> {"t", "y[t]"}, DisplayFunction -> Identity];

Show[solutionplots, flowplot, DisplayFunction -> $DisplayFunction];
```



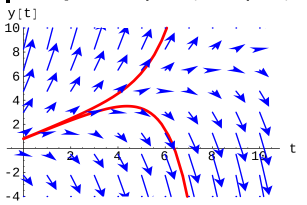
What do folks mean when they talk about "sensitive dependence on starter data?"

□ **Answer:**

You're looking at it.

Take another look:

```
Show[solutionplots, flowplot, DisplayFunction → $DisplayFunction];
```



What you see are two solutions of:

```
diffeq
{y′[t] == -0.8 t + y[t], y[0] == starter}
```

The starting points of the two solutions are almost the same; yet the two solutions are totally different in character.

You gotta agree that small changes in the starting conditions can result in big changes in solutions. This is exactly what folks mean when they talk about senistive dependence on starting data.

□T.1.a.ii)

Do all differential equations exhibit sensitive dependence on starter data?

□ **Answer:**

Not all of them.

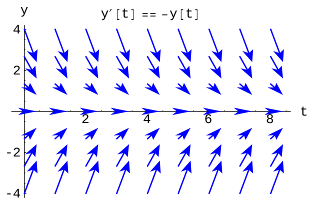
Here's one that doesn't exhibit overly sensitive dependence on starter data:

```
Clear[f, t, y];
b = 10;
f[t_, y_] = -y;
setup =
 (diffeq = {y'[t] == f[t, y[t]], y[0] == starter}) // ColumnForm
y′[t] == -y[t]
y[0] == starter
```

And a flow plot:

```
endtime = 6;
scalefactor = 0.4;
{ylow, yhigh} = {-4, 4};
{tlow, thigh} = {0, 8};

flowplot = Table[Arrow[{1, f[t, y]}, Tail → {t, y},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → 0.6],
  {t, tlow, thigh, (thigh - tlow)/8}, {y, ylow, yhigh, (yhigh - ylow)/6}];
```

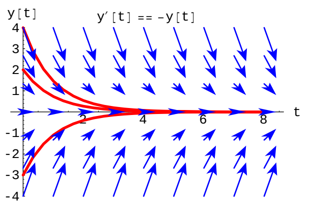
```
Show[flowplot, Axes → True, AxesLabel → {"t", "y"},
 AspectRatio → 1/GoldenRatio, PlotLabel → setup⟦1, 1⟧];
```



The flow plot tells you to expect all solutions to the sucked onto the t-axis.

Check it out:

```
Clear[y1, y2, y3, t];
{starter1, starter2, starter3} = {-3.0, 2.0, 4.0};
endtime = thigh;
y1[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter1, y[t], {t, 0, endtime}]⟦1⟧;
y2[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter2, y[t], {t, 0, endtime}]⟦1⟧;
y3[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter3, y[t], {t, 0, endtime}]⟦1⟧;

solutionplots = Plot[{y1[t], y2[t], y3[t]},
  {t, 0, endtime}, PlotStyle → {{Thickness[0.01], Red}},
  AspectRatio → 1/GoldenRatio, PlotRange → {ylow, yhigh},
  AxesLabel → {"t", "y[t]"}, DisplayFunction → Identity];

Show[solutionplots, flowplot, PlotLabel → setup⟦1, 1⟧,
 DisplayFunction → $DisplayFunction];
```



If two solutions of this differential equation have nearby starting values on y[0], then the corresponding solutions will remain close to each other forever.

This differential equation is nearly insensitive to starter data.

T.2) The drinking versus driving model: Using a diffeq to analyze Bubba's toot

```
                Calculus&Mathematica thanks pharmacokineticist
Professor Al Staubus of the College of Pharmacy at Ohio State University
                                  and
                Medical Doctor Jim Peterson of Urbana, Illinois
                         for help on this problem.
```

□T.2.a)

Here are some facts:
→ A 12 – ounce long neck beer contains 13.6 grams of alcohol.
→ The typical human liver can eliminate 12 grams of alcohol per hour.
→ In all states, if your body contains more than 1 gram of alcohol per liter of body fluids, then you are too drunk to drive legally.
→ In some states, if your body contains more than 0.8 grams of alcohol per liter of body fluids, then you are too drunk to drive legally.
→ Bubba weighs 187 pounds.
→ A college-age male in good shape (as Bubba is) weighing K kilograms has about 0.68 K liters of fluids in his body.
To estimate the number of liters of fluids in Bubba's body, you use:

```
Needs["Miscellaneous`Units`"]
Convert[187 PoundWeight, KilogramWeight]
```

```
84.8217 KilogramWeight
```

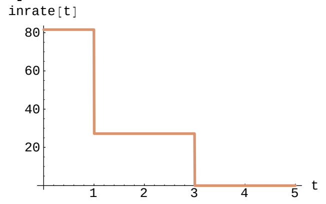
```
Bubbafluids = 0.68 84.8217
```

```
57.6788
```

This is in liters.

Here is Bubba's plan for the big party:
He plans to drink plenty of beer at the party. But he is also worried about his driver's license. So he comes up with the scheme:
He'll drink beer gradually at the rate of one 12 – ounce long neck every 10 minutes for the first hour; then he'll nurse one 12 – ounce beer every half hour for the next two hours. After all this beer, he'll stop drinking and just sit around looking cool until he's sure his blood alcohol concentration is down to a safe, legal level for driving.
Here is the function of time (in hours) that measures Bubba's intake rate of alcohol in grams per hour and a plot for the first 5 hours:

```
beer = 13.6;
Clear[inrate, t];
inrate[t_] := 6 beer /; 0 ≤ t < 1;
inrate[t_] := 2 beer /; 1 ≤ t < 3;
inrate[t_] := 0 /; 3 ≤ t;

Plot[inrate[t], {t, 0, 5},
 PlotStyle → {{Thickness[0.01], YellowBrown}}, PlotRange → All,
 AspectRatio → 1/GoldenRatio, AxesLabel → {"t", "inrate[t]"}];
```



Assuming Bubba's liver is normal, you can say the function of time (in hours) that measures his elimination rate of alcohol in grams per hour is:

```
Clear[outrate, t];
outrate[t_] = 12;
```

Use the NDSolve instruction to help give a plot of the function that measures the alcohol (in grams) that will be found in Bubba's body anytime during the first 9 hours.
Then give a plot of the function that measures the concentration of alcohol in grams per liter in Bubba's body fluids.
Estimate how long Bubba will have to stay at the party before he becomes a legal driver.

□Answer:

Call the function that measures the alcohol (in grams) that will be found in Bubba's body anytime during the first 8 hours by the name

alcohol[t].

Assuming Bubba has no residue from the night before, you know that at the start of the party ($t = 0$),

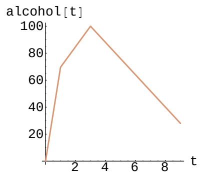
alcohol[0] = 0.

You also know that

alcohol'[t] = inrate[t] − outrate[t].

Here's the plot of alcohol[t] for the first 9 hours:

```
beer = 13.6;
Clear[inrate, outrate, t];
inrate[t_] := 6 beer /; 0 ≤ t < 1;
inrate[t_] := 2 beer /; 1 ≤ t < 3;
inrate[t_] := 0 /; 3 ≤ t;
outrate[t_] = 12;
a = 0;
b = 9;
starter = 0;

Clear[solution, t, y, alcohol];
solution = NDSolve[
  {y′[t] == inrate[t] - outrate[t], y[0] == starter}, y[t], {t, a, b}];
alcohol[t_] = y[t] /. solution[[1]];

Plot[
 alcohol[t], {t, a, b}, PlotStyle → {{YellowBrown, Thickness[0.01]}},
 AxesLabel → {"t", "alcohol[t]"}, AxesOrigin → {a, starter},
 AspectRatio -> 1, PlotRange → All];
```



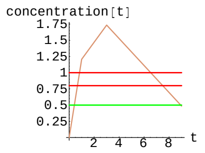
Call the function that measures the concentration of alcohol in grams per liter in Bubba's body fluids by the name concentration[t]. This function is given by:

```
Clear[concentration];
concentration[t_] = alcohol[t] / Bubbafluids
```

```
0.0173374 InterpolatingFunction[{{0., 9.}}, <>][t]
```

Here comes the plot showing concentration[t] together with the legal thresholds 1.0 grams per liter, 0.8 grams per liter, and the conservative threshold 0.5 grams per liter:

```
Plot[{concentration[t], 1.0, 0.8, 0.5}, {t, a, b},
 PlotStyle → {{YellowBrown, Thickness[0.01]}, Red, Red, Green},
 AspectRatio -> 1, AxesLabel → {"t", "concentration[t]"},
 PlotRange → All];
```



In some states, Bubba can legally drive home about 7 hours after the party begins. In other states, he must stay at the party for nearly eight hours.

Just to be on the safe side, it might be a good idea for Bubba to stay at the party a full nine hours.

Go back and run a drinking schedule you or a friend is likely to use. You might pick up some useful information.

T.3) Using parametric plotting to plot the predator population as a function of the prey population

Remember the predator-prey model from the previous lesson. In case you've forgotten about it, here's the scoop again:

```
This mathematical model was orginally studied by Lotka and Volterra.
For a critical analysis of it, get hold of J. D. Murray, Mathematical
Biology, Springer-Verlag, New York,1990 and read.
```

Two species coexist in a closed environment. One species, the predator, feeds on the other, the prey. There is always plenty of food for the prey, but the predators eat nothing but the prey.

Put

pred[t] = population of predators at time t.

Put

prey[t] = population of prey at time t.

It's reasonable to assume that there are positive constants a and b such that:

$$\text{prey}'[t] = a\,\text{prey}[t] - b\,\text{prey}[t]\,\text{pred}[t]$$

because the abundance of food for the prey allows the birth rate of the prey to be proportional to their current number, and the death rate of prey is proportional to both the current number of prey and the current number of predators.

It also makes some sense to assume that there are positive constants c and d such that:

$$\text{pred}'[t] = -c\,\text{pred}[t] + d\,\text{pred}[t]\,\text{prey}[t]$$

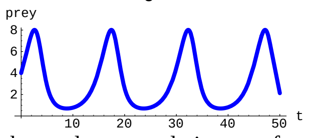
because it's reasonable to assume that the death rate of the predators is likely to be proportional to the current population of predators and that the birth rate of the predators is proportional to both the current number of the predators and the size of the food supply (the prey).

Here is what happens for a sample choice of a, b, c, d with the prey starting off with a population of 4 units and the predators starting out with a population of 1 unit.

```
Here the units could be thousands or millions so that a fractional
population can make sense.
```

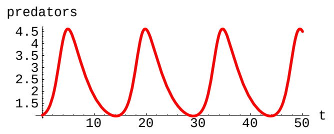
Here is *Mathematica*'s fake plot of the prey population as a function of time[t] for the first 50 time units:

```
endtime = 50;
a = 0.7;
b = 0.3;
c = 0.3;
d = 0.1;

Clear[pred, prey, t];
approxsolutions = NDSolve[{prey′[t] == a prey[t] - b prey[t] pred[t],
    pred′[t] == -c pred[t] + d pred[t] prey[t], prey[0] == 4, pred[0] == 1},
   {prey[t], pred[t]}, {t, 0, endtime}];

Clear[fakepred, fakeprey];
fakepred[t_] = pred[t] /. approxsolutions[[1]];
fakeprey[t_] = prey[t] /. approxsolutions[[1]];

preyplot = Plot[fakeprey[t], {t, 0, endtime},
  PlotStyle → {{Blue, Thickness[0.015]}}, AxesLabel → {"t", "prey"},
  AspectRatio → 1/3];
```



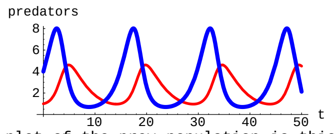
And the predator population as a function of time t:

```
predatorplot = Plot[fakepred[t],
  {t, 0, endtime}, PlotStyle → {{Red, Thickness[0.01]}},
  AxesLabel → {"t", "predators"}, AspectRatio → 1/3];
```



Here they are together:

```
both = Show[predatorplot, preyplot];
```



```
The plot of the prey population is thicker than the plot of the predator
population.
```

Note the relationship between the predator population's crests and dips and prey population's crests and dips. Just after the predators start growing, the prey is just about eaten up.

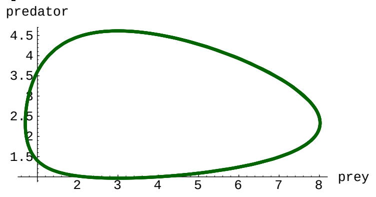
□T.3.a)

Use parametric plotting to display the predator population as a function of the prey population and describe what you see.

□Answer:

Here it is:

```
fullplot = ParametricPlot[{fakeprey[t], fakepred[t]},
  {t, 0, endtime}, PlotStyle → {{Thickness[0.01], DarkGreen}},
  PlotRange → All, AxesLabel → {"prey", "predator"}];
```

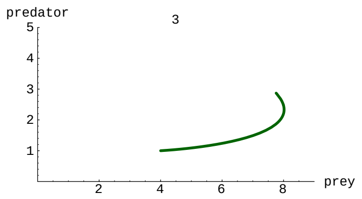


A distorted ellipse that some folks call a closed curve.

This is a little hard to interpret.

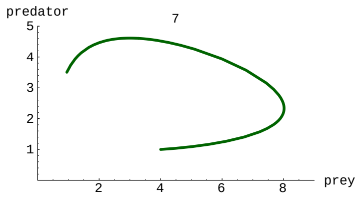
To get a grasp on it, see what happens as t goes from 0 to 3:

```
three = ParametricPlot[{fakeprey[t], fakepred[t]},
  {t, 0, 3}, PlotStyle → {{Thickness[0.01], DarkGreen}},
  AxesLabel → {"prey", "predator"}, PlotRange → {{0, 9}, {0, 5}},
  PlotLabel → 3];
```



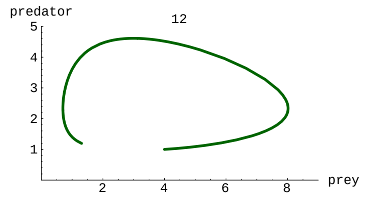
Now see what happens as t goes from 0 to 7:

```
seven = ParametricPlot[{fakeprey[t], fakepred[t]},
  {t, 0, 7}, PlotStyle → {{Thickness[0.01], DarkGreen}},
  AxesLabel → {"prey", "predator"}, PlotRange → {{0, 9}, {0, 5}},
  PlotLabel → 7];
```



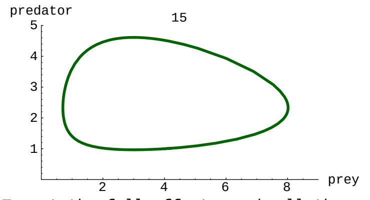
Now see what happens as t goes from 0 to 12:

```
twelve = ParametricPlot[{fakeprey[t], fakepred[t]},
  {t, 0, 12}, PlotStyle → {{Thickness[0.01], DarkGreen}},
  AxesLabel → {"prey", "predator"}, PlotRange → {{0, 9}, {0, 5}},
  PlotLabel → 12];
```



Now see what happens as t goes from 0 to 15:

```
fifteen = ParametricPlot[{fakeprey[t], fakepred[t]},
  {t, 0, 15}, PlotStyle → {{Thickness[0.01], DarkGreen}},
  AxesLabel → {"prey", "predator"}, PlotRange → {{0, 9}, {0, 5}},
  PlotLabel → 15];
```

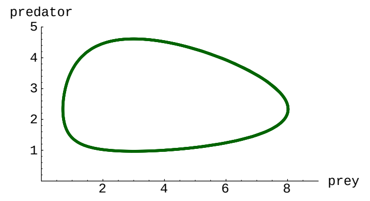


To get the full effect, grab all three plots, animate, and run forward.

Holy smoke!

This is the same as:

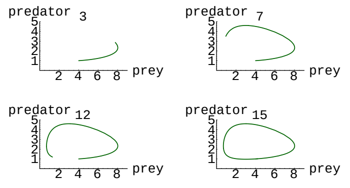
```
Show[fullplot, PlotRange → {{0, 9}, {0, 5}}];
```



The above plot shows what happens as t goes from 0 to 50.

Review the plots:

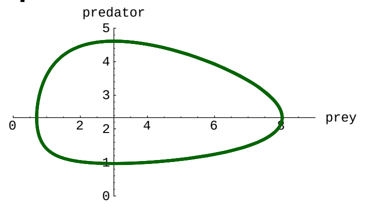
```
Show[GraphicsArray[{{three, seven}, {twelve, fifteen}}]];
```



As time advances, the parametric points {prey[t], pred[t]} advance around the closed curve in a counterclockwise way. As time advances on and on, the parametric points {prey[t], pred[t]} cycle around this curve over and over again.

Take another look but this time setting the axes' origin to $\{\frac{a}{b}, \frac{c}{d}\}$:

```
Show[fullplot, PlotRange → {{0, 9}, {0, 5}}, AxesOrigin → {c/d, a/b}];
```



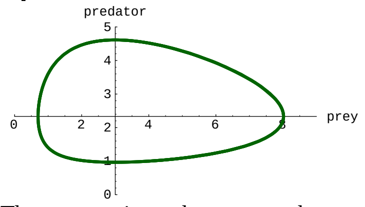
Again the closed curve tells you that predator and prey populations are periodic; they go through repeated cycles. And the time cycle for each is the same. The four sectors above defined by the axes indicate four phases depicting trends that reverse themselves as the curve crosses the axes at prey = $\frac{c}{d}$ and predator = $\frac{a}{b}$.

□T.3.b.i)

Wait a minute!
Take another look at the last plot.

Running this plot successfully requires that the cells in part a) are active.

```
Show[fullplot, PlotRange → {{0, 9}, {0, 5}}, AxesOrigin → {c/d, a/b}];
```



Those axes pierce the curve at the maximum and minimum values of prey and predator populations.
Is this just an accident?

□Answer:

Stupid question.

In mathematics, there are no accidents.

□T.3.b.ii)

Explain why those axes pierce the plot at the maximum and minimum values of prey and predator populations.

□Answer:

Look at the original differential equations

prey'[t] = a prey[t] − b prey[t] pred[t],

pred'[t] = −c pred[t] + d pred[t] prey[t].

Take the first:

prey'[t] = a prey[t] − b prey[t] pred[t].

At the times t at which the prey population is at its maximum or at its minimum, you gotta have

prey'[t] = 0.

Consequently, at the times t at which the prey population is at its maximum or at its minimum,

0 = a prey[t] − b prey[t] pred[t],

= prey[t] (a − b pred[t]).

This tells you that at the times t at which the prey population is at its maximum or at its minimum,

pred[t] = $\frac{a}{b}$.

Similarly, use

pred'[t] = −c pred[t] + d pred[t] prey[t]

to see that for a maximum or minimum predator population, you have

0 = −c pred[t] + d pred[t] prey[t]

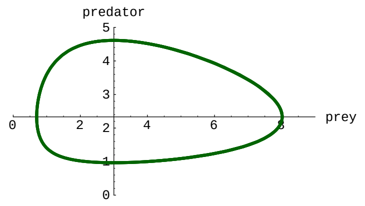
= pred[t] (−c + d prey[t]),

so when the predator population is largest or smallest,

prey[t] = $\frac{c}{d}$.

This explains why when you use AxesOrigin → $\{\frac{c}{d}, \frac{a}{b}\}$; the axes pierce the plot at the maximum and minimum values of prey and predator populations.

```
Show[fullplot, PlotRange → {{0, 9}, {0, 5}}, AxesOrigin → {c/d, a/b}];
```



Math happens again.

T.4) More on reading the predator-prey model

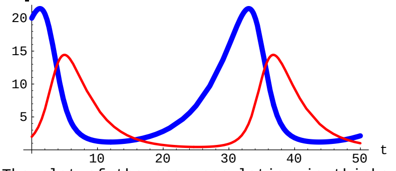
□T.4.a)

Here is what happens in the predator-prey model for the sample choice of proportionality constants

$a = 0.2, \; b = 0.05, \; c = 0.28,$ and $d = 0.04$

with the prey starting off with a population of 20 units and the predators starting out with a population of 2 units.

```
endtime = 50;

a = 0.2;
b = 0.05;
c = 0.28;
d = 0.04;

Clear[pred, prey, t];
approxsolutions =
NDSolve[{prey'[t] == a prey[t] - b prey[t] pred[t],
    pred'[t] == -c pred[t] + d pred[t] prey[t],
    prey[0] == 20,
    pred[0] == 2},
  {prey[t], pred[t]}, {t, 0, endtime}];

Clear[fakepred, fakeprey];
fakepred[t_] = pred[t] /. approxsolutions[[1]];

fakeprey[t_] = prey[t] /. approxsolutions[[1]];

predpreyplot =
Plot[{fakeprey[t], fakepred[t]}, {t, 0, endtime},
    PlotStyle -> {{Blue, Thickness[0.015]},
        {Red, Thickness[0.007]}},
    AxesLabel -> {"t", ""}];
```



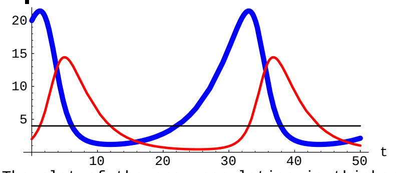
The plot of the prey population is thicker than the plot of the predator population.

Add the horizontal line that hits the vertical axis at $\frac{a}{b}$ to this plot.
Inspect the plot; then describe what you see and explain why you see it.

□Answer:

Here you go:

```
Show[predpreyplot, Graphics[Line[{{0, a/b}, {endtime, a/b}}]]];
```



The plot of the prey population is thicker than the plot of the predator population.

Interesting.

When pred[t] $> \frac{a}{b}$, then prey[t] goes down.

When pred[t] $< \frac{a}{b}$, then prey[t] goes up.

And prey[t] takes its largest and smallest values when

pred[t] $= \frac{a}{b}$.

To explain this phenomenom, look at one of the differential equations that went into this plot:

prey'[t] = a prey[t] − b prey[t] pred[t]

= prey[t] (a − b pred[t]).

When pred[t] $> \frac{a}{b}$, then (a − b pred[t]) < 0.

So when pred[t] $> \frac{a}{b}$, then

prey'[t] = prey[t] (a − b pred[t]) < 0.

That's why the prey curve goes down when pred[t] $> \frac{a}{b}$.

Similarly when pred[t] $< \frac{a}{b}$, then prey'[t] > 0.

That's why the prey curve goes up when pred[t] $< \frac{a}{b}$.

When pred[t] $= \frac{a}{b}$, then

prey'[t] = prey[t] (a − b pred[t]) $= 0$.

That's why prey[t] takes its largest and smallest values when

pred[t] $= \frac{a}{b}$.

Amazing what a little calculus can do.

DE.04 Modern DiffEq Issues
Give It a Try!

G.1) Visual flows and trajectories: Quickies

□G.1.a.i)

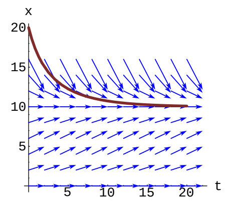
Here's a differential equation of the logistic type:

```
Clear[f, x, t];
f[t_, x_] = 0.2 x (1 - x/10);
diffeq = (x'[t] == f[t, x[t]])
```

```
x'[t] == 0.2 (1 - x[t]/10) x[t]
```

Here is a flow plot for this diffeq shown with the plot of a certain curve:

```
Clear[f, x, t];
f[t_, x_] = 0.2 x (1 - x/10);
scalefactor = 2;
flowplot = Table[Arrow[{1, f[t, x]}, Tail → {t, x},
     VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → 0.9],
   {t, 0, 20, 2}, {x, 0, 16, 2}];
curveplot = Plot[10 Exp[0.2 t]/(Exp[0.2 t] - 0.5), {t, 0, 20}, PlotStyle →
    {{Thickness[0.015], Brown}}, DisplayFunction → Identity];

Show[flowplot, curveplot, Axes → True, AxesLabel → {"t", "x"},
 DisplayFunction → $DisplayFunction];
```

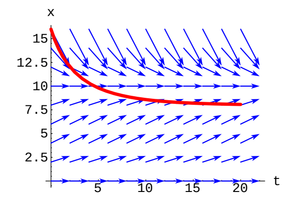


Using only your eyes, what's your guess?
Is that curve a reasonable approximation of a solution of this diffeq?
Explain what visual evidence you used to arrive at your opinion.

□G.1.a.ii)

Stay with the same diffeq and flow plot as in part i) above and look at the flowplot together with the plot of a certain new curve:

```
newcurveplot = Plot[8 Exp[0.2 t]/(Exp[0.2 t] - 0.5), {t, 0, 20},
  PlotStyle → {{Thickness[0.015], Red}}, DisplayFunction → Identity];
Show[flowplot, newcurveplot, Axes → True, AxesLabel → {"t", "x"},
 DisplayFunction → $DisplayFunction];
```



Using only your eyes, what's your guess?
Is that curve a reasonable approximation of a solution of this diffeq?
Explain what visual evidence you used to arrive at your opinion.

□G.1.b.i)

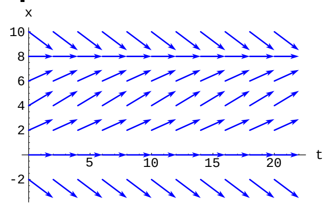
Here's a new differential equation of the logistic type:

```
Clear[f, x, t];
f[t_, x_] = 0.3 x (1 - x/8);
diffeq = (x′[t] == f[t, x[t]])
```

```
x′[t] == 0.3 (1 - x[t]/8) x[t]
```

Here is a flow plot for this diffeq:

```
Clear[f, x, t];
f[t_, x_] = 0.3 x (1 - x/8);
scalefactor = 2;
flowplot = Table[Arrow[{1, f[t, x]}, Tail → {t, x},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → 0.7],
   {t, 0, 20, 2}, {x, -2, 10, 2}];

Show[flowplot, Axes → True, AxesLabel → {"t", "x"},
 DisplayFunction → $DisplayFunction];
```

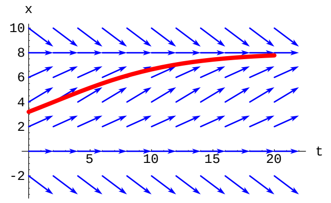


Got any idea why those arrows corresponding to x = 0 and x = 8 are not tilted at all? If so, then say why.
Got any idea why the arrows corresponding to 0 < x < 8 are tilted up, but the otheres are tilted down? If so, then say why.

□G.1.b.ii)

Stay with the same diffeq as in part i) and look at the same flow plot together with the plot of a certain curve:

```
curveplot = Plot[8 Exp[0.2 t]/(Exp[0.2 t] + 1.5), {t, 0, 20},
  PlotStyle → {{Thickness[0.015], Red}}, DisplayFunction → Identity];

Show[flowplot, curveplot, Axes → True, AxesLabel → {"t", "x"},
 DisplayFunction → $DisplayFunction];
```



Using only your eyes, what's your guess?
Is that curve a reasonable approximation of a solution of this diffeq?
Explain what visual evidence you used to arrive at your opinion.

□G.1.b.iii)

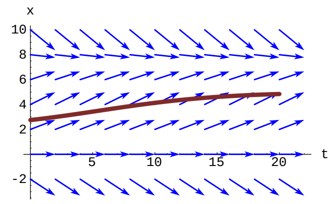
Here's a new differential equation of the logistic type:

```
Clear[f, x, t];
f[t_, x_] = 0.26 x (1 - x/8.6);
diffeq = (x′[t] == f[t, x[t]])
```

```
x′[t] == 0.26 (1 - 0.116279 x[t]) x[t]
```

Here is a flow plot for this diffeq shown with a plot of a certain curve:

```
Clear[f, x, t];
f[t_, x_] = 0.26 x (1 - x/7.6);
scalefactor = 2;
flowplot = Table[Arrow[{1, f[t, x]}, Tail → {t, x},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → 0.8],
   {t, 0, 20, 2}, {x, -2, 10, 2}];
curveplot = Plot[3 Exp[0.2 t]/(Exp[0.2 t] + 3) + 2, {t, 0, 20}, PlotStyle →
    {{Thickness[0.015], Brown}}, DisplayFunction → Identity];

Show[flowplot, curveplot, Axes → True, AxesLabel → {"t", "x"},
 DisplayFunction → $DisplayFunction];
```



Using only your eyes, what's your guess?
Is that curve a reasonable approximation of a solution of this diffeq?
Explain what visual evidence you used to arrive at your opinion.

□G.1.c.i) Systems

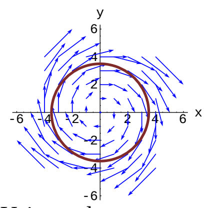
Here's a system of differential equations:

```
Clear[x, y, t, desystem];
desystem = {x′[t] == y[t], y′[t] == -x[t]};
ColumnForm[desystem]
```

```
x′[t] == y[t]
y′[t] == -x[t]
```

Here is a flow plot for this diffeq system shown with a plot of a certain curve:

```
Clear[flow];
flow[x_, y_] = {y, -x};
scalefactor = 0.5;
flowplot = Table[Arrow[flow[x, y], Tail → {x, y}, VectorColor → Blue,
    ScaleFactor → scalefactor, HeadSize → 0.4], {x, -4, 4}, {y, -4, 4}];
curveplot =
 ParametricPlot[{3.5 Sin[t], 3.5 Cos[t]}, {t, 0, 2 π}, PlotStyle →
   {{Thickness[0.015], Brown}}, DisplayFunction → Identity];

Show[flowplot, curveplot, Axes → True, AxesLabel → {"x", "y"},
 DisplayFunction → $DisplayFunction];
```

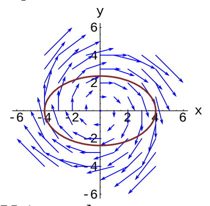


Using only your eyes, what's your guess?
Is that curve a reasonable approximation of a parametric plot of a solution {x[t], y[t]} of this diffeq system?
Explain what visual evidence you used to arrive at your opinion.

□G.1.c.ii)

Stay with the same system and flow plot, but with the plot of a new curve:

```
newcurveplot =
 ParametricPlot[{4 Sin[t], 2.5 Cos[t]}, {t, 0, 2 π}, PlotStyle →
   {{Thickness[0.009], Brown}}, DisplayFunction → Identity];
Show[flowplot, newcurveplot, Axes → True, AxesLabel → {"x", "y"},
 DisplayFunction → $DisplayFunction];
```



Using only your eyes, what's your guess?
Is that curve a reasonable approximation of a parametric plot of a solution {x[t], y[t]} of this diffeq system?
Explain what visual evidence you used to arrive at your opinion.

□G.1.d.i)

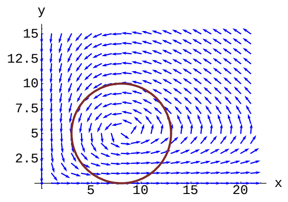
Here's a system of differential equations closely related to the predator-prey system

```
Clear[x, y, t, desystem];
desystem = {x'[t] == 1.2 x[t] (5.2 - y[t]),
 y'[t] == 0.9 y[t] (x[t] - 8.3)};
desystem // ColumnForm
```

```
x′[t] == 1.2 x[t] (5.2 - y[t])
y′[t] == 0.9 (-8.3 + x[t]) y[t]
```

Here is a flow plot for this diffeq system shown with a plot of a certain curve:

```
Clear[flow];
flow[x_, y_] = {1.2 x (5.2 - y), 0.9 (-8.3 + x) y};
flowplot = Table[Arrow[flow[x, y], Tail → {x, y}, VectorColor → Blue,
   ScaleFactor → Normalize, HeadSize → 0.5], {x, 0, 21}, {y, 0, 15}];
curveplot =
 ParametricPlot[{8, 5} + 5 {Cos[t], Sin[t]}, {t, 0, 2 π}, PlotStyle →
   {{Thickness[0.009], Brown}}, DisplayFunction → Identity];

Show[flowplot, curveplot, Axes → True, AxesLabel → {"x", "y"},
 DisplayFunction → $DisplayFunction];
```



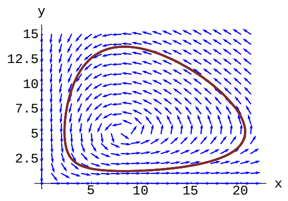
Using only your eyes, what's your guess?
Is that curve a reasonable approximation of a parametric plot of a solution {x[t], y[t]} of this diffeq system?
Explain what visual evidence you used to arrive at your opinion.

□G.1.d.ii)

Stay with the same system and flow plot, but with the plot of a new curve:

```
Clear[x, y, t, desystem];
desystem = {x′[t] == 1.2 x[t] (5.2 - y[t]), y′[t] == 0.9 y[t] (x[t] - 8.3)};
Clear[f, t]
f[t_] = {x[t], y[t]} /. NDSolve[Join[desystem,
     {x[0] == 3.4, y[0] == 2.1}], {x[t], y[t]}, {t, 0, 3}][[1]];
newcurveplot = ParametricPlot[f[t], {t, 0, 3}, PlotStyle →
   {{Thickness[0.009], Brown}}, DisplayFunction → Identity];
```

```
Show[flowplot, newcurveplot, Axes → True, AxesLabel → {"x", "y"},
 DisplayFunction → $DisplayFunction];
```



Using only your eyes, what's your guess?
Is that curve a reasonable approximation of a parametric plot of a solution {x[t], y[t]} of this diffeq system?
Explain what visual evidence you used to arrive at your opinion.

□G.1.e)

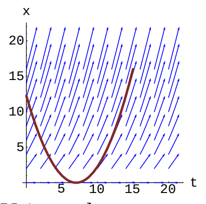
Here's a new diffeq

```
Clear[f, x, t];
f[t_, x_] = Sqrt[x];
diffeq = (x'[t] == f[t, x[t]])
```

$x'[t] == \sqrt{x[t]}$

Here is a flow plot for this diffeq shown with the plot of a certain curve:

```
Clear[f, x, t, function];
f[t_, x_] = √x;
scalefactor = 1.5;

flowplot =
 Table[Arrow[{1, f[t, x]}, Tail -> {t, x}, VectorColor -> Blue,
   ScaleFactor -> scalefactor, HeadSize -> 0.5],
  {t, 0, 20, 2}, {x, 0, 16, 2}];


function[t_] = x[t] /. DSolve[{diffeq, x[15] == 16}, x[t], t][[2]];

curveplot = Plot[function[t], {t, 0, 15},
  PlotStyle -> {{Thickness[0.015], Brown}},
  DisplayFunction -> Identity];

Show[flowplot, curveplot, Axes -> True, AxesLabel -> {"t", "x"},
 DisplayFunction -> $DisplayFunction];
```



Using only your eyes, what's your guess?
Is that curve a reasonable approximation of a solution of this diffeq?
Explain what visual evidence you used to arrive at your opinion.

G.2) Reading a diffeq through flow plots

□G.2.a.i)

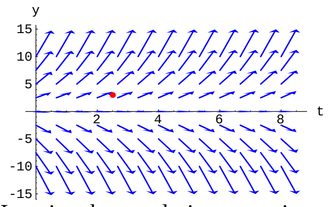
Here's a differential equation:

```
Clear[diffeq, y, t, f, starter];
f[t_, y_] = y - 0.5 E^(-0.3 t);
(diffeq = {y'[t] == f[t, y[t]], y[0] == starter}) // ColumnForm
```

$y'[t] == -0.5\, E^{-0.3\, t} + y[t]$
$y[0] == starter$

And its flow plot shown with one specially marked point:

```
scalefactor = 0.5;
{ylow, yhigh} = {-10, 10};
{tlow, thigh} = {0, 8};
specialpoint = {2.5, 3.0};
flowplot = Table[Arrow[{1, f[t, y]}, Tail → {t, y},
   VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → 0.6],
                   thigh - tlow                      yhigh - ylow
  {t, tlow, thigh, ────────────}, {y, ylow, yhigh, ────────────}]];
                        12                                8

Show[flowplot, Axes → True,
                                                  1
 AxesLabel → {"t", "y"}, AspectRatio → ───────────,
                                             GoldenRatio
 Epilog → {Red, PointSize[0.02], Point[specialpoint]}];
```

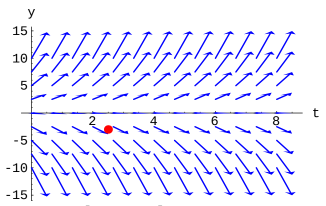


Imagine that a solution curve is coming from the left and passes through this specially marked point. What happens to the solution as it leaves this point?
Does it go up or does it go down?

□G.2.a.ii)

Stay with the same diffeq and flow plot, but go with this new specially marked point:

```
specialpoint = {2.5, -3.0};
Show[flowplot, Axes → True,
                                                  1
 AxesLabel → {"t", "y"}, AspectRatio → ───────────,
                                             GoldenRatio
 Epilog → {Red, PointSize[0.03], Point[specialpoint]}];
```

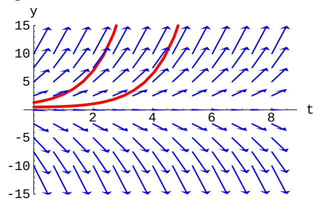


Imagine that a solution curve is coming from the left and passes through this specially marked point. What happens to the solution as it leaves this point?
Does it go up or does it go down?

□G.2.a.iii)

Stay with the same diffeq and flow plot and look at plots of two solutions:

```
Clear[y1, y2, t];
{starter1, starter2} = {0.5, 1.3};
endtime = thigh;
y1[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter1, y[t], {t, 0, endtime}][[1]];
y2[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter2, y[t], {t, 0, endtime}][[1]];
solutionplots = Plot[{y1[t], y2[t]}, {t, 0, endtime},
  PlotStyle → {{Thickness[0.01], Red}}, DisplayFunction → Identity];

Show[solutionplots, flowplot, Axes → True, AxesLabel → {"t", "y"},
 AspectRatio → 1/GoldenRatio, PlotRange → {-15, 15},
 DisplayFunction → $DisplayFunction];
```



```
Disregard any error messages.
```

Visible in this flow plot are two distinct families of solutions, but both plotted solutions are of the same type. Copy, paste and edit in effort to show sample plots of each family.

□G.2.a.iv)

Folks say that a diffeq is extremely sensitive to errors in starting data on y[0] if small changes in y[0] can result in radically different behavior of the corresponding solutions.
For what values of y[0] do you see extreme sensitivity to errors on y[0]?

□G.2.b.i)

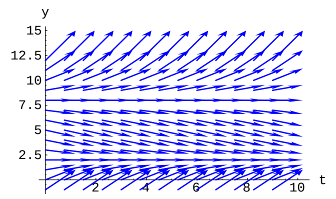
Here's a new differential equation:

```
Clear[diffeq, y, t, f, starter];
f[t_, y_] = f[t_, y_] = (1 - y/2) (1 - y/8);
(diffeq = {y'[t] == f[t, y[t]], y[0] == starter}) // ColumnForm
```

```
y′[t] == (1 - y[t]/2) (1 - y[t]/8)
y[0] == starter
```

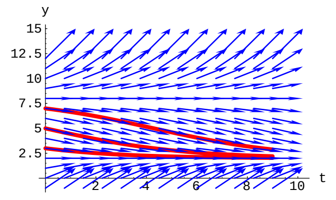
And its flow plot:

```
scalefactor = 1.2;
{ylow, yhigh} = {-1, 12};
{tlow, thigh} = {0, 9};
flowplot = Table[Arrow[{1, f[t, y]}, Tail → {t, y},
    VectorColor → Blue, ScaleFactor → scalefactor, HeadSize → 0.6],
   {t, tlow, thigh, (thigh - tlow)/12}, {y, ylow, yhigh, (yhigh - ylow)/13}];

Show[flowplot, Axes → True, AxesLabel → {"t", "y"},
 AspectRatio → 1/GoldenRatio];
```



Here are the plots of three solutions:

```
Clear[y1, y2, y3, t];
{starter1, starter2, starter3} = {3.0, 5.0, 7.0};
endtime = thigh;
y1[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter1, y[t], {t, 0, endtime}][[1]];
y2[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter2, y[t], {t, 0, endtime}][[1]];
y3[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter3, y[t], {t, 0, endtime}][[1]];
solutionplots = Plot[{y1[t], y2[t], y3[t]}, {t, 0, endtime},
  PlotStyle → {{Thickness[0.015], Red}}, DisplayFunction → Identity];

Show[solutionplots, flowplot, Axes → True, AxesLabel → {"t", "y"},
 AspectRatio → 1/GoldenRatio, DisplayFunction → $DisplayFunction];
```



```
Disregard any error messages.
```

Visible in this flow plot are three distinct families of solutions, but the three plotted solutions are of the same type.
Copy, paste and edit in effort to show sample plots of each type.

□G.2.b.ii)

Stay with the same diffeq as in part ii) immediately above and discuss where you see extreme sensitivity to errors in the starter value on y[0]. Illustrate with a decisive plot.

G.3) Reading the diffeq through flow plots and phase lines

□G.3.a.i) Flow plots

Here's an old friend, the logistic differential equation:

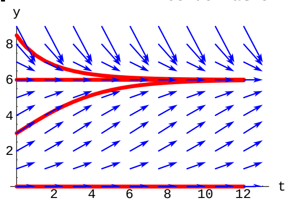
```
Clear[diffeq, y, t, f, starter];
b = 6;
f[t_, y_] = 0.45 y (1 - y/b);
 diffeq
 {y′[t] == f[t, y[t]], y[0] == starter}
```

```
y′[t] == 0.45 (1 - y[t]/6) y[t]
y[0] == starter
```

Here is a flow plot shown with plots of four solutions of this diffeq.

```
{ylow, yhigh} = {0, 9};
{tlow, thigh} = {0, 12};
scalefactor = 1.4;
flowplot = Table[Arrow[{1, f[t, y]}, Tail → {t, y},
    VectorColor → Blue, Scalefactor → scalefactor, HeadSize → 0.5],
   {t, tlow, thigh, (thigh - tlow)/8}, {y, ylow, yhigh, (yhigh - ylow)/9}];

Clear[y1, y2, y3, y, t];
{starter1, starter2, starter3, starter4} = {0, 3.0, 6.0, 8.5};
endtime = thigh;
y1[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter1, y[t], {t, 0, endtime}][[1]];
y2[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter2, y[t], {t, 0, endtime}][[1]];
y3[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter3, y[t], {t, 0, endtime}][[1]];
y4[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter4, y[t], {t, 0, endtime}][[1]];
solutionplots = Plot[{y1[t], y2[t], y3[t], y4[t]}, {t, 0, endtime},
  PlotStyle → {{Thickness[0.015], Red}}, DisplayFunction → Identity];
```

```
setup = Show[solutionplots, flowplot,
  Axes → True, AxesLabel → {"t", "y"}, PlotRange → {ylow, yhigh},
  AspectRatio → 1/GoldenRatio, DisplayFunction → $DisplayFunction];
```

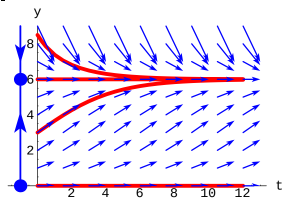


Describe how the solution plots are in harmony with the flow plot. How does the flow plot confirms that the solutions starting at y = 0 and y = 6 stay flat all the way?

□G.3.a.ii) The phase line

Look at this embellishment of the graphic immediately above.

```
phaseline = PhaseLine[f[t, y], {y, 0, yhigh}, Blue, tlow - 1];
Show[setup, phaseline];
```



Folks like to call that gadget on the far left by the name "phase line" for the diffeq.

What do you think that the upward arrowhead on the phaseline signifies?
What do you think that the downward arrowhead on the phase line signifies?
What do you think that the centers of the two dots on the phase line signify?

□G.3.b.i) Flow plots

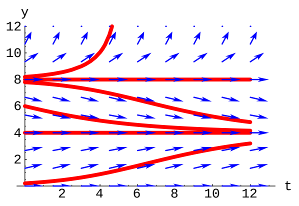
Here's a a diffeq which is related to the logistic differential equation:

```
Clear[diffeq, y, t, f, starter];
b = 6;
f[t_, y_] = 0.45 y (1 - y/4) (1 - y/8);
 diffeq
 {y'[t] == f[t, y[t]], y[0] == starter}
```

```
y'[t] == 0.45 (1 - y[t]/4) (1 - y[t]/8) y[t]
y[0] == starter
```

Here is a flow plot shown with plots of four solutions of this diffeq.

```
{ylow, yhigh} = {0, 12};
{tlow, thigh} = {0, 12};
flowplot = Table[Arrow[{1, f[t, y]}, Tail → {t, y},
    VectorColor → Blue, ScaleFactor → Normalize, HeadSize → 0.6],
  {t, tlow, thigh, (thigh - tlow)/8}, {y, ylow, yhigh, (yhigh - ylow)/9}];

Clear[y1, y2, y3, y4, y5, y6, y, t];
{starter1, starter2, starter3, starter4, starter5, starter6} =
  {0.2, 4.0, 6.0, 7.8, 8.0, 8.2};
endtime = thigh;
y1[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter1, y[t], {t, 0, endtime}][[1]];
y2[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter2, y[t], {t, 0, endtime}][[1]];
y3[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter3, y[t], {t, 0, endtime}][[1]];
y4[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter4, y[t], {t, 0, endtime}][[1]];
y5[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter5, y[t], {t, 0, endtime}][[1]];
y6[t_] = y[t] /.
  NDSolve[diffeq /. starter → starter6, y[t], {t, 0, endtime}][[1]];
solutionplots =
 Plot[{y1[t], y2[t], y3[t], y4[t], y5[t], y6[t]}, {t, 0, endtime},
  PlotStyle → {{Thickness[0.015], Red}}, DisplayFunction → Identity];

setup = Show[solutionplots, flowplot,
  Axes → True, AxesLabel → {"t", "y"}, PlotRange → {ylow, yhigh},
  AspectRatio → 1/GoldenRatio, DisplayFunction → $DisplayFunction];
```



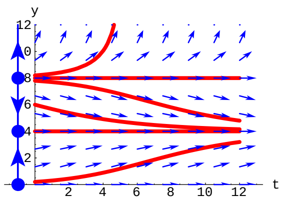
Don't worry about the error messages.

Folks say that a diffeq is extremely sensitive to errors in starting data on y[0] if small changes in y[0] can result in radically different behavior of the corresponding solutions. For what values of y[0] do you see extreme sensitive to errors on y[0]?

□G.3.b.ii) The phase line

Look at this embellishment of the graphic immediately above.

```
phaseline = PhaseLine[f[t, y], {y, 0, 12}, Blue, -1];
Show[setup, phaseline];
```



Folks like to call that gadget on the far left by the name "phase line" for the diffeq.

What do you think that the upward arrowheads on the phaseline signify?
What do you think that the downward arrowhead on the phase line signifies?
What do you think that the centers of the two dots on the phase line signify?

G.4) Shooters: Firing for effect

A chemostat is a well-stirred vessel that contains microorganisms into which fresh medium is pumped at a constant rate F. The contents are pumped out at the same rate so that the volume V of the fluid in the chemostat remains constant.

Chemostat experiments performed by Hansen and Hubbell (Science, 20 (1980), pp 1491-1493)) in which the amino acid tryptophan was pumped at a constant rate into a chemostat containing a certain strain of E. coli bacteria indicated a growth model:

```
Clear[diffeq, y, t, f, s, d, starter];
f[t_, y_] = (0.81 (s - y)/(3 + (s - y)) - d) y;
 diffeq
 {y'[t] == f[t, y[t]], y[0] == starter}
```

```
y'[t] == (-d + 0.81 (s-y[t])/(3+s-y[t])) y[t]
y[0] == starter
```

Here:
→ y[t] measures the population of the E. coli at time t after the pumping of the amino acid started.
→ s is the concentration (in millionths of grams per liter) in of the amino acid in the fluid pumped into the chemostat.
→ d $= \frac{F}{V}$, so larger values of d correspond to a fast running pump, and low values of d correspond to a slow running pump.

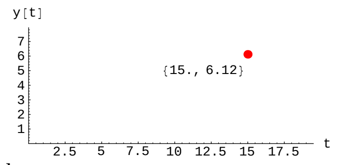
In the experiment under study here, s = 9.8 and d = 0.3 ; so the model becomes:

```
s = 9.8;
d = 0.3;
Clear[diffeq, y, t, f, starter];
f[t_, y_] = (0.81 (s - y)/(3 + (s - y)) - d) y;
 diffeq
 {y'[t] == f[t, y[t]], y[0] == starter}
```

```
y′[t] == (-0.3 + 0.81 (9.8-y[t])/(12.8-y[t])) y[t]
y[0] == starter
```

You are given the job of coming up with a starter y[0] so that the E. coli population at time t = 15.0 is 6.12 to two accurate decimals.

```
endtime = 15.0;
goal = 6.12;
target = {endtime, goal};
bullseye = Graphics[{PointSize[0.03], Red, Point[target]}];
targetlabel = Graphics[Text[target, target, {1, 2}]];

Show[bullseye, targetlabel, Axes → True, AxesLabel → {"t", "y[t]"},
 PlotRange → {{0, 1.3 target⟦1⟧}, {0, 1.3 target⟦2⟧}}];
```

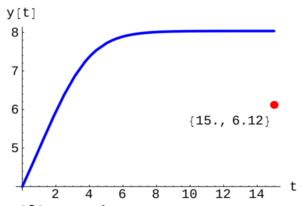


Make a guess

y[0] = starter = 4

and activate the following solution plotter code which plots and remembers your first try:

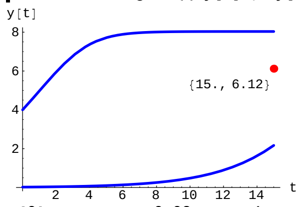
```
starter = 4.0;
Clear[thistry, sollist];
thistry[t_] = y[t] /. NDSolve[diffeq, y[t], {t, 0, endtime}]⟦1⟧;
sollist = {thistry[t]};
numsols = Length[sollist]; Null;
startingpoints = N[sollist /. t → 0, 3];
names = Table[startingpoints⟦k⟧, {k, numsols}];
plotsols = Plot[Evaluate[sollist], {t, 0, endtime},
  AspectRatio → 1/GoldenRatio, PlotStyle → {{Thickness[0.01], Blue}},
  AxesLabel → {"t", "y[t]"}, DisplayFunction → Identity];

Show[plotsols, bullseye, targetlabel,
 PlotRange → All, DisplayFunction → $DisplayFunction];

TableForm[{startingpoints, N[sollist /. t → endtime, 3]},
 TableHeadings → {{"y[0]", "y[1]"}, None}]
```



```
y[0]    4.
y[1]    8.04
```

The solution corresponding to y[0] = 4 overshoots the target. You respond by decreasing your starting value on y[0] and trying again:

```
starter = 0.02;
Clear[thistry];
thistry[t_] = y[t] /. NDSolve[diffeq, y[t], {t, 0, endtime}]⟦1⟧;
PrependTo[sollist, thistry[t]];
numsols = Length[sollist];
If[numsols > 3, numsols -= 1; sollist = Drop[sollist, -1]];
startingpoints = N[sollist /. t → 0, 3];
names = Table[startingpoints⟦k⟧, {k, numsols}];
plotsols = Plot[Evaluate[sollist], {t, 0, endtime},
  AspectRatio → 1/GoldenRatio, PlotStyle → {{Thickness[0.01], Blue}},
  AxesLabel → {"t", "y[t]"}, DisplayFunction → Identity];

Show[plotsols, bullseye, targetlabel,
 PlotRange → All, DisplayFunction → $DisplayFunction];

TableForm[{startingpoints, N[sollist /. t → endtime, 4]},
 TableHeadings → {{"y[0]", "y[endtime]"}, None}]
```



```
y[0]          0.02      4.
y[endtime]    2.169     8.035
```

Interact with this last code by changing the starter values and re-running. The code is written to keep track of your last three attempts.
Remember, your job is to come up with a starting value that makes y[endtime] agree with 6.12 to two accurate decimals (after rounding).

G.5) Logistic harvesting*

Take the logistic equation

$y'[t] = a\,y[t]\left(1 - \frac{y[t]}{b}\right)$ with $0 < a$ and $0 < b$

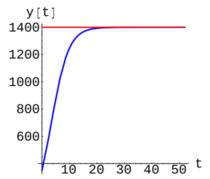
and start with $0 < y[0] < b$.

As t advances from 0, then you are guaranteed that y[t] grows with some pep until y[t] gets near b.
Once y[t] gets near b, then y[t] settles into global scale with

$y[t] \to b$ as $t \to \infty$.

Check it out:

```
endtime = 52;
a = 0.32;
b = 1400;
starter = 350;
Clear[t, y];
solution = NDSolve[
  {y′[t] == a y[t] (1 - y[t]/b), y[0] == starter}, y[t], {t, 0, endtime}];

Clear[fakey];
fakey[t_] = y[t] /. solution⟦1⟧;
fakeplot = Plot[{fakey[t]}, {t, 0, endtime},
  PlotStyle → {{Blue, Thickness[0.01]}}, DisplayFunction → Identity];
globalscale = Graphics[{Red, Line[{{0, b}, {endtime, b}}]}];

Show[fakeplot, globalscale, AxesLabel → {"t", "y[t]"},
 PlotRange → All, AspectRatio → 1, DisplayFunction → $DisplayFunction];
```



You get a reasonable interpretation of the logistic differential equation

$y'[t] = a\,y[t]\left(1 - \frac{y[t]}{b}\right)$ with $0 < y[0] < b$

by imagining that y[t] is the number of catfish in a given lake on a catfish farm t weeks after the lake was stocked with y[0] catfish. As time goes on, the catfish population increases until it reaches its steady-state population of b catfish.
But the catfish farmer doesn't grow catfish as pets; the farmer is in business to harvest catfish and to sell them so that hungry people can fry them up and then wash them down with a couple of cold beers or iced teas.

□G.5.a.i)

Measure time in weeks and assume the farmer wants to harvest r fish per week and explain why

$y'[t] = a\,y[t]\left(1 - \frac{y[t]}{b}\right) - r$

lays the base for a reasonable model.

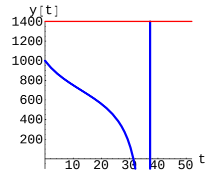
□G.5.a.ii)

The farmer opens the lake and lets the fish population increase to a level of 1000 fish. Then the farmer begins to harvest at a rate of r = 130 fish per week. Here's what happens over the next year:

This goes with specific data $a = 0.32$ and $b = 1400$.
The reason that $b = 1400$ is that 1400 fish is the greatest number of fish the lake can support.

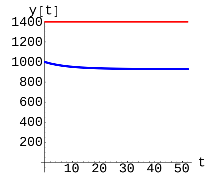
```
endtime = 52;
r = 130;
a = 0.32;
b = 1400;
starter = 1000;
```

```
Clear[t, y];
solution = NDSolve[{y′[t] == a y[t] (1 - y[t]/b) - r,
    y[0] == starter}, y[t], {t, 0, endtime}];

Clear[fakey];
fakey[t_] = y[t] /. solution[[1]];
fakeplot = Plot[{fakey[t]}, {t, 0, endtime},
   PlotStyle → {{Blue, Thickness[0.015]}}, DisplayFunction → Identity];
withoutharvest = Graphics[{Red, Line[{{0, b}, {endtime, b}}]}];

Show[
 fakeplot, withoutharvest, AxesLabel → {"t", "y[t]"}, AspectRatio → 1,
 PlotRange → {{0, 52}, {-100, b}}, DisplayFunction → $DisplayFunction];
```



Disaster.
In a little more than 30 weeks, the fish supply is gone.
Here's what happens over a one-year period if the farmer lowers the weekly harvest from 130 to 100 fish:

```
endtime = 52;
r = 100;
starter = 1000;
Clear[t, y];
solution = NDSolve[{y′[t] == a y[t] (1 - y[t]/b) - r,
    y[0] == starter}, y[t], {t, 0, endtime}];

Clear[fakey];
fakey[t_] = y[t] /. solution[[1]];
fakeplot = Plot[{fakey[t]}, {t, 0, endtime},
   PlotStyle → {{Blue, Thickness[0.015]}}, DisplayFunction → Identity];
withoutharvest = Graphics[{Red, Line[{{0, b}, {endtime, b}}]}];

Show[
 fakeplot, withoutharvest, AxesLabel → {"t", "y[t]"}, AspectRatio → 1,
 PlotRange → {-100, b}, DisplayFunction → $DisplayFunction];
```



That's nice.
The lake produces 100 fish per week for harvest and maintains a fish population of about 1000 fish at all times.
Can the farmer harvest significantly more than 100 fish per week and avoid disaster?
If so, then estimate how big the weekly harvest r can be before disaster sets in.

□Tip:

If you can't come up with a grand theory, prospect a little bit.

Is r = 110 OK?

How about r = 120?

Remember, you are after an estimate, not an exact number.

□G.5.a.iii)

Take your biggest possible weekly harvest estimate r from part ii) above and estimate how small y[0] = starter can be before disaster sets in.

□Tip:

Evidently y[0] = 1000 is OK.

Can the lake still be harvested at the same weekly rate r if you start with y[0] = 800? How about y[0] = 700? How low can you go?

□G.5.b.i)

The diffeq model is

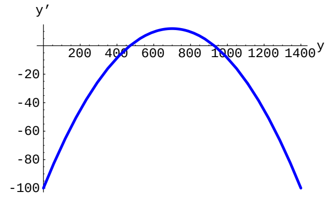
$$y'[t] = a\,y[t]\left(1 - \frac{y[t]}{b}\right) - r.$$

When you look at it, you can see that you can realize y' as a function of population size y through the formula

$$y' = f[y] = a\,y\,(1 - \tfrac{y}{b}) - r$$

Look at this plot which is related to the case with r = 100 above.

```
r = 100;
Clear[f, y];
f[y_] = a y (1 - y/b) - r;

Plot[f[y], {y, 0, b}, PlotStyle → {{Thickness[0.01], Blue}},
 AxesOrigin → {0, 0}, AxesLabel → {"y", "y'"},
 AspectRatio → 1/GoldenRatio];
```



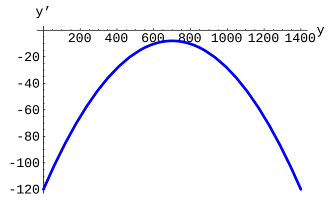
Some folks call this a phase plot, not to be confused with a phase line.

Remembering that y′ = f[y], report on how this plot tells you that if the fish population is ever more than 500, then you can continue a weekly harvest of 100 fish and the predicted long term fish population will settle in at a little more than 900.
Also remembering that y′ = f[y], report on how this plot tells you that if the fish popoulation ever falls below 400 then a weekly harvest of 100 fish will drive the fish population to 0.

□G.5.b.ii)

Look at this plot which is related to the case with harvest rate r = 120.

```
r = 120;
Clear[f, y];
f[y_] = a y (1 - y/b) - r;

Plot[f[y], {y, 0, b}, PlotStyle → {{Thickness[0.01], Blue}},
 AxesOrigin → {0, 0}, AxesLabel → {"y", "y'"},
 AspectRatio → 1/GoldenRatio];
```



Remembering that y′ = f[y], report on how this plot tells you that continuing to harvest 120 fish per week will eventually drive the fish population to 0.

G.6) Running Euler's faker

□G.6.a.i)

Here's an innocent-looking differential equation:

y'[x] = y[x] (3 + 3 Sin[x] − y[x])
with y[0] = starter = 0.5.

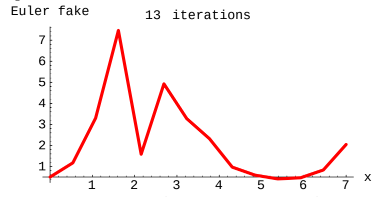
If you want to get a plot of the solution, you ask *Mathematica* for a formula for the solution:

```
Clear[x, y, starter];
thisstarter = 0.5;
DSolve[{y′[x] == y[x] (3 + 3 Sin[x] - y[x]), y[0] == starter}, y[x], x] /.
 starter → thisstarter
```

$$\{\{y[x] \to -(0.5\,E^{3+3x-3\,\mathrm{Cos}[x]}) \Big/ \Big(-1 + 10.0428 \int \frac{1}{E^3}\, d0 - 10.0428 \int E^{3x-3\left(\frac{\mathrm{Cos}[x]}{2}-\frac{1}{2} I\,\mathrm{Sin}[x]\right)-3\left(\frac{\mathrm{Cos}[x]}{2}+\frac{1}{2} I\,\mathrm{Sin}[x]\right)}\, dx\Big)\}\}$$

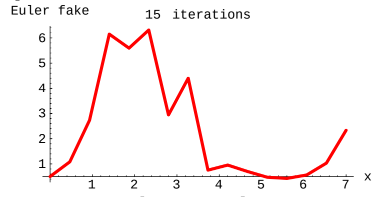
This made *Mathematica* barf. The output is useless.
Try as it might, *Mathematica* could not come up with a useful formula for the true solution y[x]. But students like you are not easily dissuaded. Use Euler's method to come up with what you believe to be a reasonably accurate plot of the true solution for 0 ≤ x ≤ 7.

```
a = 0;
b = 7;
starter = 0.5;
y[a] = starter;
beginner = {a, y[a]};

Clear[x, y, next, jump, iterations, point, k];
jump[iterations_] := (b - a)/iterations;
next[{x_, y_}, iterations_] :=
 {x, y} + jump[iterations] {1, y (3 + 3 Sin[x] - y)};
point[0, iterations_] = beginner;
point[k_, iterations_] :=
 point[k, iterations] = N[next[point[k - 1, iterations], iterations]];

Clear[Euler];
Euler[iterations_] := Show[Graphics[{Thickness[0.01], Red,
    Line[Table[point[k, iterations], {k, 0, iterations}]]}],
  PlotRange → All, Axes → True,
  AxesOrigin → beginner, AxesLabel → {"x", "Euler fake"},
  PlotLabel → iterations " iterations",
  AspectRatio → 1/2, DisplayFunction → Identity];
primitive = Show[Euler[13], DisplayFunction → $DisplayFunction];
```



A primitive stick figure resulting from 13 iterations.
Here's what you get with 15 iterations:

```
better = Show[Euler[15], DisplayFunction → $DisplayFunction];
```



Experiment with more and more iterations until you are satisfied with the Euler fake plot of the solution of this diffeq.

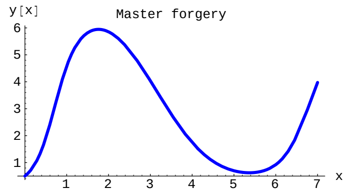
□G.6.a.ii)

Here is *Mathematica*'s `NDSolve` instruction at work to come up with a good approximate plot of the solution of

y'[x] = y[x] (3 + 3 Sin[x] − y[x])
with y[0] = starter = 0.5,

the same diffeq as you worked with in part i):

```
Clear[solution, x, y, fakey];
solution = NDSolve[
  {y′[x] == y[x] (3 + 3 Sin[x] - y[x]), y[0] == starter}, y[x], {x, a, b}];
fakey[x_] = y[x] /. solution[[1]];

masterfakeplot =
 Plot[fakey[x], {x, a, b}, PlotStyle → {{Blue, Thickness[0.01]}},
  AxesLabel → {"x", "y[x]"}, AxesOrigin → {a, starter},
  PlotRange → All, AspectRatio → 1/2, PlotLabel → "Master forgery"];
```



Discuss how this plot compares with the plot you came up with in part i).

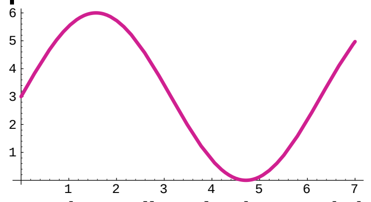
□G.6.a.iii)

Then take another look at the differential equation

y'[x] = y[x] (3 + 3 Sin[x] − y[x])

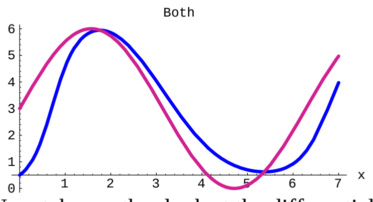
and then take a look at this plot of 3 + 3 Sin[x]:

```
Clear[x];
auxillaryplot = Plot[3 + 3 Sin[x], {x, a, b}, AspectRatio → 1/2,
  PlotStyle → {{Thickness[0.01], VioletRed}}];
```



Here is this auxillary plot shown with the master fake plot:

```
Show[masterfakeplot, auxillaryplot, AxesLabel → {"x", ""},
 PlotLabel → "Both"];
```



Now take another look at the differential equation

y'[x] = y[x] (3 + 3 Sin[x] − y[x])

and explain why it is automatic that:
When the solution curve is below the plotted 3 + 3 Sin[x] curve, the solution curve is going up.
When the solution curve is above the 3 + 3 Sin[x] curve, the solution curve is going down.

G.7) Lanchester's war model

The first scientist who tried to apply calculus to warfare in the way you will see below was F. W. Lanchester, who worked in Britain during World War I. In his honor, fancy folks call all these models by the name Lanchesterian models.

□G.7.a)

The good guys and the bad guys are engaged in combat. Agree that good[t] is the number of good guys still fighting t days after combat begins, and let bad[t] be the number of bad guys still fighting t days after combat begins.
If neither side receives reinforcing troops, then why is it natural to propose a model

good'[t] = − a bad[t]
bad'[t] = − b good[t]

where a and b are positive numbers?

□G.7.b.i)

If the good guys bring in new troops at a rate of f[t] new troops per day and the bad guys bring in new troops at a rate of g[t] new troops per day, then why is it natural to propose a model

$$good'[t] = -a\,bad[t] + f[t]$$
$$bad'[t] = -b\,good[t] + g[t]$$

where a and b are positive numbers?

□G.7.b.ii)

Agree that the good guys win if bad[t] becomes 0 before good[t] becomes 0 and that the bad guys win if good[t] becomes 0 before bad[t] becomes 0.

The model

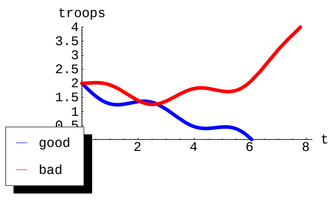
$$good'[t] = -0.5\,bad[t] + Sin[t]^2$$
$$bad'[t] = -0.5\,good[t] + Cos[t]^2$$
$$\text{with } good[0] = bad[0] = 2$$

suggests the situation in which the troops on both sides are equally effective and at the start (t = 0) they are of equal strength. The main difference is that the reinforcement rates are out of phase, with the bad guys holding the initial advantage because

$$Cos[0]^2 = 1 > 0 = Sin[0]^2$$

This suggests that the good guys lose.
Check it out:

```
endtime = 8;
a = -0.5;
b = -0.5;

Clear[good, bad, t];
approxsolutions =
 NDSolve[{good'[t] == a bad[t] + Sin[t]^2, bad'[t] == b good[t] + Cos[t]^2,
   good[0] == 2, bad[0] == 2}, {good[t], bad[t]}, {t, 0, endtime}];

good[t_] = good[t] /. approxsolutions⟦1⟧;
bad[t_] = bad[t] /. approxsolutions⟦1⟧;

warplot = Plot[{good[t], bad[t]}, {t, 0, endtime},
  PlotStyle → {{Blue, Thickness[0.015]}, {Red, Thickness[0.015]}},
  PlotRange → {0, 4}, AxesLabel → {"t", "troops"},
  PlotLegend → {"good", "bad"}, LegendSize → 0.5];
```



Sure enough!
The bad guys beat the daylights out of the good guys. The good guys are off the battlefield in about 6 days.
But the good guys are not so stupid. They ran the model and saw that their reinfocement rate was not enough. They react by doubling their reinforcement rate. This gives the model:

$$good'[t] = -0.5\,bad[t] + 2\,Sin[t]^2$$
$$bad'[t] = -0.5\,good[t] + Cos[t]^2$$
$$\text{with } good[0] = bad[0] = 2.$$

Who wins and how long does the battle last?

□G.7.b.iii)

This time the good guys have antiquated ammunition and heavy bureaucratic problems. As a result, their troops are only half as effective as the troops of the bad guys.
This suggests the model:

$$good'[t] = -0.5\,bad[t] + 2\,Sin[t]^2$$
$$bad'[t] = -0.25\,good[t] + Cos[t]^2$$
$$\text{with } good[0] = bad[0] = 2.$$

Who wins and how long does the battle last?

□G.7.b.iv)

Again the good guys have antiquated ammunition and heavy bureaucratic problems. As a result, their troops are only half as effective as the troops of the bad guys, so they decide to find a positive number r as small as practical such that if

$$good'[t] = -0.5\,bad[t] + r\,Sin[t]^2$$
$$bad'[t] = -0.25\,good[t] + Cos[t]^2$$
$$\text{with } good[0] = bad[0] = 2,$$

then the good guys do not lose during the first 15 days.
Estimate how small r can be.

□Tip:

Experiment with different r's.

□G.7.c.i) Iwo Jima

The authors are pleased to acknowledge
the heavy influence of the book
Differential Equations and Their Applications
by Martin Braun, Springer-Verlag, 1978.
If you like this stuff, you'll want to look at this book,
paying special attention to section 4.3.

Iwo Jima is the largest of the volcano islands in the western Pacific Ocean. During World War II, it was a big Japanese air base. During February and March, 1945, American marines took it at great cost in a bloody assault. If you have been to Washington, D.C., you may have seen the Iwo Jima memorial statue which depicts U.S. Marines planting the Stars and Stripes on the beach during a very bloody stage of the battle.
The Americans kept careful records of their losses, and the records confirm that the Iwo Jima battle is correctly modeled by the ideas in the Lanchesterian model above.

Here comes the nitty-gritty:
Agree that usa[t] measures the number of active American troops and japan[t] measures the number of active Japanese troops engaged in the battle t days after the battle began.
For this battle, military scientists have arrived at the model

$$usa'[t] = -0.0544\,japan[t] + f[t]$$
$$japan'[t] = -0.0106\,usa[t] + g[t]$$

where f[t] and g[t] are the reinforcement rates.

The individual Japanese troop was 5 times more effective
than the individual American troop because the
Japanese held a defensive position.

At the beginning of the battle,

$$usa[0] = 5.4 \text{ and } japan[0] = 2.15$$

with units measured in tens of thousands.
The reinforcement rate for Japan was

$$g[t] = 0.$$

There were no Japanese reinforcements.

```
Clear[g, t];
g[t_] = 0
```

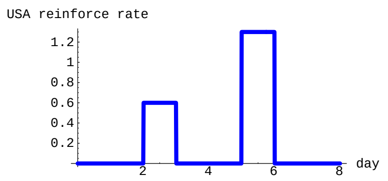
```
0
```

The Americans piled in 5.4 (tens of thousands of) troops at the beginning of the battle (t = 0), 0 extra troops during the first and second day, 0.6 troops spread out over the third day, 0 troops during the fourth and fifth days, 1.3 troops spread out over the sixth day, and no other reinforcements.

Remember that troops are measured in tens of thousands.

Here is a plot of f[t]:

```
Clear[f, t];
f[t_] := 0.0 /; t < 2;
f[t_] := 0.6 /; 2 ≤ t ≤ 3;
f[t_] := 0.0 /; 3 < t < 5;
f[t_] := 1.3 /; 5 ≤ t ≤ 6;
f[t_] := 0.0 /; 6 < t;

Plot[f[t], {t, 0, 8}, PlotStyle → {{Thickness[0.015], Blue}},
 AxesLabel → {"day", "USA reinforce rate"}, PlotRange → All,
 AspectRatio → 1/2];
```



The Americans considered the island secure after 28 days of battle. Replay the battle of Iwo Jima by giving plots of usa[t] and japan[t] for the first 28 days.

□G.7.c.ii)

In actuality, there were 5.28 (tens of thousands) American troops in the field on day 28. Compare this with the value predicted by the model.

□Tip:

Don't expect the model to be totally accurate. There is always a gap between models and reality. If you got a big discrepancy, then you did something wrong.

□G.7.c.iii)

Estimate:
How many American troops were knocked out of action?
How many Japanese troops bit the dust?

G.8) More on the predator-prey model

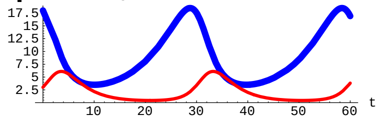
□G.8.a.i)

Here is what happens in the predator-prey model for the sample choice of proportionality constants

a = 0.15, b = 0.07, c = 0.36, and d = 0.04

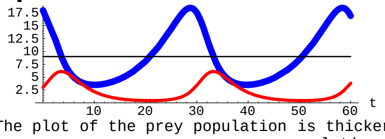
with the prey starting off with a population of 18 units and the predators starting out with a population of 3 units.

```
endtime = 60;
a = 0.15;
b = 0.07;
c = 0.36;
d = 0.04;
Clear[pred, prey, t];
approxsolutions =
 NDSolve[{prey′[t] == a prey[t] - b prey[t] pred[t], pred′[t] ==
    -c pred[t] + d pred[t] prey[t], prey[0] == 18, pred[0] == 3},
  {prey[t], pred[t]}, {t, 0, endtime}];
pred[t_] = pred[t] /. approxsolutions[[1]];
prey[t_] = prey[t] /. approxsolutions[[1]];

predpreyplot = Plot[{prey[t], pred[t]}, {t, 0, endtime},
  PlotStyle → {{Blue, Thickness[0.02]}, {Red, Thickness[0.01]}},
  PlotRange → All, AxesLabel → {"t", ""}];
```



Here is what you get when you add to the plot the horizontal line that hits the vertical axis at $\frac{c}{d}$:

```
Show[predpreyplot, Graphics[Line[{{0, c/d}, {endtime, c/d}}]]];
```



The plot of the prey population is thicker than the plot of the predator population.

Inspect the plot; then describe what you see and explain why you see it.

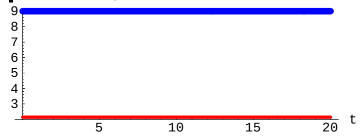
G.8.a.ii)

Here is what happens in the predator-prey model for the same proportionality constants

a = 0.15, b = 0.07, c = 0.36, and d = 0.04

as in part i) but with the prey starting off with a population of $\frac{c}{d}$ units and the predators starting out with a population of $\frac{a}{b}$ units.

```
endtime = 20;
a = 0.15;
b = 0.07;
c = 0.36;
d = 0.04;
Clear[pred, prey, t];
approxsolutions =
 NDSolve[{prey′[t] == a prey[t] - b prey[t] pred[t], pred′[t] ==
    -c pred[t] + d pred[t] prey[t], prey[0] == c/d, pred[0] == a/b},
  {prey[t], pred[t]}, {t, 0, endtime}];
pred[t_] = pred[t] /. approxsolutions[[1]];
prey[t_] = prey[t] /. approxsolutions[[1]];

predpreyplot = Plot[{prey[t], pred[t]}, {t, 0, endtime},
  PlotStyle → {{Blue, Thickness[0.02]}, {Red, Thickness[0.01]}},
  PlotRange → All, AxesLabel → {"t", ""}];
```



Fancy folks call this "equilibrium."
Play with other choices of a, b, c, and d.
Try to explain why this happened.

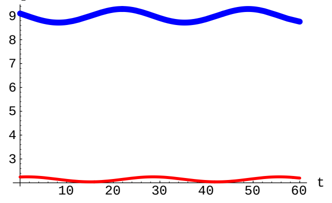
G.8.a.iii)

Here is what happens in the predator-prey model for the same proportionality constants

a = 0.15, b = 0.07, c = 0.36, and d = 0.04

as in part i) but with the prey starting off with a population of $\frac{c}{d} + 0.1$ units and the predators starting out with a population of $\frac{a}{b} + 0.1$ units.

```
endtime = 60;
a = 0.15;
b = 0.07;
c = 0.36;
d = 0.04;
Clear[pred, prey, t];
approxsolutions = NDSolve[{prey′[t] == a prey[t] - b prey[t] pred[t],
   pred′[t] == -c pred[t] + d pred[t] prey[t],
   prey[0] == c/d + 0.1, pred[0] == a/b + 0.1},
  {prey[t], pred[t]}, {t, 0, endtime}];
pred[t_] = pred[t] /. approxsolutions[[1]];
prey[t_] = prey[t] /. approxsolutions[[1]];

predpreyplot = Plot[{prey[t], pred[t]}, {t, 0, endtime},
  PlotStyle → {{Blue, Thickness[0.02]}, {Red, Thickness[0.01]}},
  PlotRange → All, AxesLabel → {"t", ""}, AspectRatio → 1/GoldenRatio];
```



Got any idea why this happened?

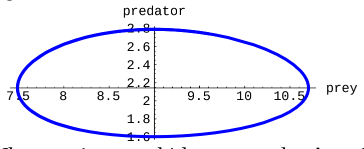
G.8.b)

Stay with the same predator-prey model as above. But start with

$\text{prey}[0] \; = \; \frac{c}{d} + h,$

$\text{pred}[0] \; = \; \frac{a}{b} + h$

```
h = 0.6;
endtime = 40;


Clear[pred, prey, t];
approxsolutions =
NDSolve[{prey'[t] == a prey[t] - b prey[t] pred[t],
    pred'[t] == -c pred[t] + d pred[t] prey[t],
    prey[0] == c/d + h,
    pred[0] == a/b + h},
  {prey[t], pred[t]}, {t, 0, endtime}];

Clear[fakepred, fakeprey];
pred[t_] = pred[t] /. approxsolutions[[1]];

prey[t_] = prey[t] /. approxsolutions[[1]];

 ParametricPlot[{prey[t], pred[t]},
    {t, 0, endtime}, PlotStyle -> {{Blue, Thickness[0.01]}},
    PlotRange -> All,
    AxesLabel -> {"prey", "predator"},
    AxesOrigin -> {c/d, a/b}];
```



Why was it a good idea to put the AxesOrigin at $\{\frac{a}{b}, \frac{c}{d}\}$?
Reducing the size of h has what effect on the fluctuations of the two populations? What happens when you make h = 0.01?

How about h = 0?
Try to explain, in your own terms, why this turned out the way it did.

□ Tip:

As you plot, pay special attention to the numbers along the axes.

□ G.8.c.i)

The authors are pleased to acknowledge that the idea for this problem came from the book Elementary Differential Equations by William Boyce and Richard DiPrima (Wiley, New York, 1977). These gentlemen wrote a really good book. You should be able to find it in your school's library.

Most folks call the populations

$\text{prey} \; = \; \frac{c}{d}$ and $\text{predators} \; = \; \frac{a}{b}$

for the predator-prey model

$\text{prey}'[t] \; = \; a\,\text{prey}[t] \; - \; b\,\text{prey}[t]\,\text{pred}[t]$

$\text{pred}'[t] \; = \; -c\,\text{pred}[t] \; + \; d\,\text{pred}[t]\,\text{prey}[t]$

that you worked with above by the name "equilibrium populations."

Imagine that the prey are insects and that the predators are birds or fish. The government goes into action, spreading an insecticide like DDT all over the place. The insecticide kills both predator and prey in proportion to their current population, so the new model becomes:

$\text{prey}'[t] \; = \; a\,\text{prey}[t] \; - \; b\,\text{prey}[t]\,\text{pred}[t] \; - \; u\,\text{prey}[t]$

$\text{pred}'[t] \; = \; -c\,\text{pred}[t] \; + \; d\,\text{pred}[t]\,\text{prey}[t] \; - \; v\,\text{pred}[t]$

where a, b, c, d, u,and v are all positive constants.

This is the same as

$\text{prey}'[t] \; = \; (a \; - \; u)\,\text{prey}[t] \; - \; b\,\text{prey}[t]\,\text{pred}[t]$

$\text{pred}'[t] \; = \; -(c \; + \; v)\,\text{pred}[t] \; + \; d\,\text{pred}[t]\,\text{prey}[t]$

What are the new equilibrium populations?
How do these new equilibrium populations tell you that the net effect of the government action was to raise the equilbrium population of the prey (insects) but to lower the equilibrium population of the predators? Is that what the government had in mind?

□ Tip:

You don't need the machine for this one.

□ G.8.c.ii)

In the Tutorials, you saw a rationale for looking at the differential equations

$\text{prey}'[t] \; = \; a\,\text{prey}[t] \; - \; b\,\text{prey}[t]\,\text{pred}[t]$

$\text{pred}'[t] \; = \; -c\,\text{pred}[t] \; + \; d\,\text{pred}[t]\,\text{prey}[t]$

for modeling the predator-prey relationship. These differential equations are not carved in marble. In fact, current literature pays attention to trying to modify these equations to make them more realistic.

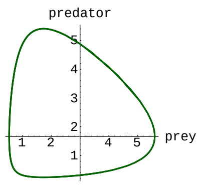
See J. D. Murray, Mathematical Biology, Springer-Verlag, New York, 1990, page 71.

For example, you can decide to replace the second equation by

$\text{pred}'[t] \; = \; -c\,\text{pred}[t]\,/\,\text{prey}[t] \; + \; d\,\text{pred}[t]\,\text{prey}[t].$

Rationale:
It's reasonable to assume that the birth rate of the predators is proportional both to the current number of the predators and the size of the food supply, and the death rate of the predators is likely to be directly proportional to the current population of predators and inversely proportional to the number of prey.
Here's a try:

```
endtime = 30;
a = 0.5;
b = 0.3;
c = 0.6;
d = 0.2;
Clear[pred, prey, t];
approxsolutions =
 NDSolve[{prey′[t] == a prey[t] - b prey[t] pred[t], pred′[t] ==
     - (c pred[t])/(prey[t]) + d pred[t] prey[t], prey[0] == 5, pred[0] == 3},
   {prey[t], pred[t]}, {t, 0, endtime}];
Clear[fakepred, fakeprey]
pred[t_] = pred[t] /. approxsolutions⟦1⟧;
prey[t_] = prey[t] /. approxsolutions⟦1⟧;

ParametricPlot[{prey[t], pred[t]}, {t, 0, endtime},
 PlotStyle → {{Thickness[0.01], DarkGreen}}, PlotRange → All,
 AxesLabel → {"prey", "predator"}, AxesOrigin → {c/d, a/b}];
```



Come up with the equillibrium populations in this model and reset the axes origin accordingly.

□ Tip:

Look at

$$0 \; = \; a\,\text{prey}[t] \; - \; b\,\text{prey}[t]\,\text{pred}[t]$$
$$= \; \text{prey}[t]\,(a \; - \; b\,\text{pred}[t]).$$

And look at

$$0 = -\frac{c}{\text{prey}[t]} + d\,\text{prey}[t]$$
$$=\frac{(-c + d\,\text{prey}[t]\,)^2}{\text{prey}[t]}.$$

Don't be afraid to take a square root.

G.9) Drinking versus driving

◻G.9.a)

Melanie, a female student weighing 125 pounds, drives to the party with the idea of nursing three 12 – ounce beers for the first hour and then nursing two 12 – ounce beers each hour for the next three hours.
If she starts with no alcohol in her system, how many grams of alcohol are there in each liter of her body fluids at the end of the four hours?
After the four hours, about how long must she go without drinking alcohol in order to reduce her body fluid alcohol concentration to a legal driving level of under $\frac{0.8\text{ grams}}{\text{liter}}$?
Each 12 – ounce beer Melanie is drinking contains 13.0 grams of alcohol.

◻Tip:

If you don't use a plot of a solution of a differential equation to do this, you are doing too much work.

Activate the following cell before you try to convert Melanie's weight into kilograms:

```
Needs["Miscellaneous`Units`"]
```

◻G.9.b)

The authors thank Junior L. Duitsman, owner of Duitsman Construction Company, St. Joseph, Illinois for asking this question.

On your way home, you stop at a local pub and gradually drink 12 – ounce regular beers for one hour with the intention of driving home immediately at the end of the hour.
Estimate the greatest number of beers you can drink during this hour and remain a legal driver in your state.

◻Tip:

Use your own weight and sex.

Activate the following cell before you try to convert Melanie's weight into kilograms:

```
Needs["Miscellaneous`Units`"]
```

To estimate your body fluids, multiply your kilogram weight by 0.65 if you are female and you are in good shape and multiply your kilogram weight by 0.68 weight into kilograms if you are male and in good shape. If your are fat, use lower multipliers.