TABLE OF CONTENTS

1. A bit of Mathematica and Sky Diving

A review of *Mathematica* 3.0, that introduces solutions to differential equations numerically. The *Mathematica* commands essential to the lab are explained carefully; for additional *Mathematica* review, see the Appendix. A model for the velocity of a sky diver taking into account the difference in air resistance before and after the deployment of his/her parachute is used to look at the solution curve for a relatively simple differential equation. Introduces the direction field and relates that to the behavior of the solution curve, and similarly treats the nullclines of the differential equation.

Comments: Group work by pairs of students is encouraged on this and other labs, provided that both spend some of the time running the keyboard/mouse to learn the *Mathematica* that is being introduced, and both actively think through the problem solutions. This lab and others in the book assume the availability of a "LocalMath" palette and local "diffeq.m" package. Details of these follow in Appendix B. Additional information on using *Mathematica* is included in Appendix A, including a short tutorial on a number of useful *Mathematica*-isms that are used in this and future labs.

2. First-Order Models: Simple Drug Modeling and Solution Structure

Examines a simple first-order (linear) differential equation modeling drug build-up in the blood. A solution is obtained analytically for one input function, and numerical solutions generated for a piecewise continuous one. The numerical solutions are used to examine superposition of solutions, and the division of the problem into solutions to the driven and undriven (free) special cases of the IVP.

Comments: This lab assumes that solution of first-order linear differential equations has been covered. It does not dwell on why superposition works, but should illustrate that it does for linear problems.

3. First-Order Differential Equation Models: World Class Sprinting

Explores a first-order differential equation that models sprinters' velocities for 100 and 200 meter races. Solutions are obtained using *Mathematica*'s DSolve function, and the physical behavior of the model is explored. *Mathematica* is used to find solutions which are then examined.

Comments: All solutions to differential equations are found using *Mathematica*, so that having covered the first-order linear solution methods is not necessary for this lab.

4. The Spruce Budworm, and How Many Solutions Are There Anyway?

Investigates issues of existence and uniqueness of solutions to first order IVPs, by considering a model for the population of the spruce budworm, Torricelli's theorem, and another IVP. These illustrate an initial value problem satisfying the requirements for existence and uniqueness and two that violate different aspects of the theorem. Students hypothesize what the theorem they are discovering is, and check this against the actual theorem as stated at the end of the lab.

Comments: May be done before or after a reading on the theorem for existence and uniqueness of solutions for first-order differential equations. Hypothetically if they have read and remember (or go back and look it up) students will be able to guess the final answer, but this is unlikely to be a significant issue. Assumes that students know how to solve first-order differential equations using separation of variables.

5. Numerical Methods for ODEs, or Solving the Budworm Problem

Investigates the numerical solution of a model for the budworm population in Canadian fir forests, a rather nonlinear first order equation. The Euler approximation for this equation is developed and its relationship to a tangent line approximation noted. The effect of decreasing step size investigated graphically. *Mathematica*'s Do and Table iterators are used to develop solutions over many steps, and its numerical solution algorithm is used as a "good" solution approximation to comment on the accuracy of the solutions obtained.

Comments: This lab is written assuming that the students have read about Euler's method and the idea of advancing a solution in a stepwise manner, though in practice they can probably get along without this. Among the trickier *Mathematica* issues for students are the appearance of the iterators Do and Table, where typographical errors can result in iterating a symbolic rather than numeric calculation for large numbers—which takes huge amounts of time and computer memory.

6. Drugs and Error in Numerical Methods

Continues exploring numerical methods, stopping to examine the error in the approximations. Again, Euler's method is used as an example and *Mathematica*'s numerical solver is used to approximate the exact solution to a model for the concentration of a periodically administered drug in the recipient's system. The size of the local truncation and global errors are examined, with the effect of step size and nonmonotonic solutions on the error noted.

Comments: Students are assumed to remember how to use Euler's method (or to be able to re-read previous labs to figure this out...). This is a short lab that should be able to be finished in well under 2 hours.

7. The Qualitative Return of the Budworm and Such Things

An investigation of how the (qualitative) analysis of the phase (state)-line can show the equilibrium solutions to a problem and the qualitative behavior of time-dependent solutions, and how the equilibria may change as a parameter is varied. (It therefore may serve as an introduction to bifurcation analysis.) It compares this type of information with the quantitative information obtained by the actual solution of the problem (found with *Mathematica*'s numerical solver) that allows finding the behavior of a solution at specific times.

Comments: This lab has significant overlap in content with the "Phase Planes, or the Return of the Budworm Meets the Freeway" lab. Here no development of second-order differential equations is assumed, and the lab focuses on the use of phase and state lines to determine the behavior of the system, without extending the idea to that of a phase plane.

8. Oscillatory Phenomena, or Future Freeways

Here we look at the behavior of a second-order constant coefficient IVP as a model for how cars on a freeway might try to attain a constant velocity and distance between cars. Using this, we consider based on their qualitative behavior the types of solutions that are expected from this type of equation, including oscillatory and exponentially decaying solutions. Introduces a phase-plane and time-state graph.

Comments: As written, this lab uses numerical solutions to the second-order linear differential equations considered and investigates only the behavior and not the functions composing the solutions thereto. It assumes no solution methods for second-order equations have been considered. Numerical solutions are used rather than analytical solutions generated with DSolve, but the latter could easily be used to see also what types of solutions are obtained. If there is no desire to introduce phase-planes and time-space graphs in this lab, the last problem can be omitted without significant loss.

9. Linear Independence, Superposition and Peculiar Pendulums

This is an investigation into the meaning of linear independence of solutions and use of superposition in the linear pendulum problem. Students' intuition is used to develop a solution (through superposition) to an IVP with a linear combination of initial conditions that have already been considered. A comparison with the nonlinear pendulum problem reveals that it is not so readily solved, and that superposition clearly fails for the nonlinear case.

Comments: Again, knowledge of no solution methods for second-order differential equations are assumed, and *Mathematica*'s DSolve and NDSolve commands are used to generate solutions for the linear and nonlinear pendulum equations. *Mathematica*'s replacement operator "`/.`" is used to substitute values into the solutions obtained with both DSolve and NDSolve to increase the accuracy of the results, provide consistency between the the cases considered, and avoid having to define a function into which to plug values.

10. Hitching Another Ride on the Future Freeway

Here we look at the second-order, linear differential equation introduced to model the "future freeway" and examines the effect of a periodic forcing term, looking at resonance and beat phenomena. The periodic forcing is motivated by either a bad computer or bad driver on the freeway, and the lab seeks to finish with some generally stated conclusions of when different types of solutions/phenomena are expected from such problems.

Comments: We assume that students know either how to solve second-order linear, constant coefficient differential equations (with, in this case, some forcing) or can from their previous experience (*e.g.*, the earlier freeway lab) surmise what types of solutions are reasonable to expect in these cases. Even so, *Mathematica*'s DSolve is used to actually solve the equations.

11. More Numerical Methods and the Blasius Boundary Layer

In this lab numerical methods (in particular, Euler's method) for higher order problems are introduced through the solution of the equation for the Blasius boundary layer. It works through the translation of the third order Blasius equation into three first order equations and the implementation of Euler's method. Results are compared with the numerical solver in *Mathematica*, both for the system of first order equations and the original third order equation. The issue of the accuracy of Euler's method is considered in conjunction with the number of steps it will require to obtain a reasonably accurate solution.

Comments: A lab appendix explains in casual detail the origin of the Blasius boundary layer equation. Students will have to remember how to substitute values in to the solution obtained from NDSolve using the replacement operator or some other cunning method to compare the accuracy of the Euler and NDSolve solutions.

12. Phase Planes, or the Return of the Budworm meets the Freeway

This lab investigates the qualitative analysis of higher (second order) differential equations through phase planes. Phase planes are re-motivated starting with the first-order spruce budworm problem (with a "phase line") and then as a true phase plane for a second order equation—the standard second-order constant-coefficient, linear differential equation with damping that is used in earlier labs as a model for the distance between cars on a freeway.

Comments: The introduction to the phase-line that is used at the beginning of this lab is the same as that appearing in the "Qualitative Return of the Budworm and Such Things" lab.

13. Of Birds and War–An Intro to Systems of Differential Equations

This lab begins the investigation of what solutions might be expected for systems of differential equations. A simple linear predator-prey and Lanchester's war model (also linear) are used to experimentally determine the solutions to be for linear constant-coefficient systems of equations. The principle of superposition is implicitly explored through the examination of how the different functions in the solutions appear together.

Comments: No knowledge of solution methods for (linear, constant-coefficient) systems of differential equations is assumed here, as DSolve is again used to solve the differential equations involved.

14. MegaSystems—An Investigation of Nonlinear Species Competition

This lab considers a nonlinear system of equations modeling competition between two species and investigates the qualitative analysis of such a system. Examines the steady states of the system and their stability by examination of the direction field, and then by linearizing about each and solving the linear equations. Looks at behavior of trajectories near each steady state, and at using nullclines to help resolve the appearance of trajectories in the phase plane.

Comments: This is quite a long lab, and works well spread out over two (two-hour) lab periods or used as an extended project.

Lab 1. A bit of Mathematica and Sky-Diving

In this and all of the labs in this volume, text in `typewriter type` indicates text that is input in *Mathematica*, and text in *italics* indicates either things to pay particular attention to or text you will be substituting another value for (*e.g.*, in a *Mathematica* command). Example problems are typeset with a vertical bar to their left, and *slanted* indented type in an example problem explains the use of *Mathematica* in the solution. [Bracketed] text gives a menu option found at the top of the screen. Do not even consider ignoring something in **bold face** type.

All solutions to your labs must by *typed* in complete sentences and *formatted*. The easiest way to do this is using *Mathematica* notebooks, as is shown in the Example problems in this lab. Print out and submit your lab writeup at the end of the lab period, or on completing the lab. In some cases you may need to print out graphs generated in *Mathematica* to sketch on them—in these cases, include the graph as an additional page at the appropriate juncture in the lab report.

Work out the solutions to the Example problems as well as the regular problems. These will show how to use many commands in *Mathematica*, and so should be included in the lab report that you submit.

Our first objective is to remember a little bit about how *Mathematica* works and how to use it to do things related to differential equations. We assume in the following examples that you remember very little about *Mathematica*, calculus, or differential equations. In both calculus and differential equations we are interested in functions, and so begin with functions: in particular, let's consider the function $f(x) = 2\,x\,\sin(3x)$.

Example Problem 1: Find $f'(x)$ by hand and with *Mathematica*.

Solution:

Let's first title our lab: select [Format], *then* [Style], *and* [Title]. *Then begin typing:*

Differential Equations Lab 1

Then, having gotten a title, click with the mouse below the title and add an explanation of what you are doing next. Note that similar to the title, we can make subtitles and section headings. For now, let's type some explanatory text: select [Format] [Style] [Text], *and explain what you're doing—for example,*

First, we consider the function $f(x) = 2x\sin(3x)$. By hand, we find by using the chain and product rules that $f'(x) = 2\sin(3x) + 6x\cos(3x)$. With *Mathematica*, we get

Now we need to take a derivative with Mathematica. The easiest way to do this in version 3.0 is to use Palettes—e.g., the "BasicInput" Palette with the exponential, square root, and calculus buttons[1]*. We want a derivative, so first click below the text you just entered (to allow you to input something else) and click on the* $\boxed{\partial_{\square}\blacksquare}$ *button in the Palette. This pastes*

$\partial_{\blacksquare}\square$

into your notebook. If the focus isn't on the notebook, click on the notebook, and then type "x":

$\partial_x\square$

then press $\boxed{tab}$ *to start entering the function in the other box, and enter the function that we're interested in* **(Note: Capitalization, types of parentheses ( ) and brackets [ ], and spacing matter in** *Mathematica***)***:*

```
∂x (2*x*Sin[3*x])
```

To evaluate this command, press $\boxed{shift}$ $\boxed{return}$.

Example Problem 2: Plot $g(x) = 2\sin(3x) + 6x\cos(3x)$ for $0 \le x \le 2\pi$ using *Mathematica*.

[1] *If this palette isn't in evidence on the desktop, select* [File] [Palettes] [BasicInput].

Solution:

Again, we want to explain what we're doing in our lab write up, and so select [Format] [Style] [Text], *and type an explanation:*

Next we plot the function $g(x) = 2\sin(3x) + 6x\cos(3x)$.

Then click below the text you just entered to get the horizontal line that indicates that Mathematica is ready for you to do something else. To plot, we use the aptly named Plot command in Mathematica. This is in another Palette, so select [File] [Palettes] [BasicCalculations], *and then open the* [▷ Graphics] *menu to paste in the Plot command:*

```
Plot[▣, {□, □, □}]
```

If the focus isn't on the notebook, click on the notebook, and then fill in the blanks, pressing $\boxed{tab}$ *to get between boxes (remember to leave spaces or use "*"s for multiplication):*

```
Plot[2*Sin[3*x] + 6*x*Cos[3*x], {x, 0, 2*π}]
```

(π is in the other palette). Then press $\boxed{shift}$ $\boxed{return}$.

Now, let's stop for a moment to see what we have seen about *Mathematica*:

Rules for building an amicable relationship with Mathematica, part 1:

1. Use the Palettes available.
2. After pasting from a palette button into your notebook, clicking on a box will allow you to enter text in it. Pressing $\boxed{tab}$ will move you to the next box.
3. You can enter something new in *Mathematica* when the notebook has a horizontal line ——— across the page. Click at or below the point you want to enter text to get a line like this. What you type will appear below the line.
4. To set the style of what you type in *Mathematica*, use [Format] [Style] before you start typing.
5. *Mathematica*'s internal functions are Capitalized: `Sin[3*x]`.
6. `[ ]` are used for **function arguments**: `Sin[3*x]`.
7. `( )` are used to **group terms**: `(x+1)(x-Sin[3*x])`.
8. Be sure to use a **space** or * for multiplication.
9. Press $\boxed{shift}$ $\boxed{return}$ to evaluate a command in *Mathematica*.

With this introduction to the niceties of *Mathematica*, let's now do some differential equations. The velocity of a 160 lb sky diver may be given approximately by the differential equation

$$\frac{dv}{dt} = g - \begin{cases} \frac{k_1}{m}v^2, & \text{before the parachute opens} \\ \frac{k_2}{m}v, & \text{after the parachute opens,} \end{cases} \tag{1}$$

where $g = 32.2$ feet/sec^2, $m = 4.97$ slugs, $k_1 = 0.0053$ slugs/foot, and $k_2 = 7.86$ slugs/sec.

Example Problem 3: Before we actually solve (1), let's look at just the right-hand side of the equation to see what it does. To see this, suppose that the velocity of the sky-diver is a constant 150 ft/sec. Then plot the damping function for equation (1) (the last term on the right-hand side, given by the split definition) as a function of time t if the parachute opens at $t = 15$ sec.

Solution:

Before the parachute opens, the damping is given by $0.0053 \cdot 150^2$, and afterwards by $7.86 \cdot 150$. To change between these values when $t = 15$, we use *Mathematica*'s `If` function:

```
Plot[If[t < 15, 0.0053*150^2/4.97, 7.86*150/4.97], {t,0,30}];
```

We can obtain a *solution curve* for the differential equation by using the *Mathematica* function ODESolve, which is an add-on function to numerically solve a differential equation. Before the **first** time we want to do this in *Mathematica*, we must load the Local‘diffeq‘ package with the Needs command. Neither the

specific Needs command we need nor the ODESolve function are on a standard *Mathematica* palette, and so have been added on a “LocalMath” palette: select [File] [Palettes] [LocalMath], and use it to paste the command `Needs["Local‘diffeq‘"]` into your notebook. Arguments for ODESolve are shown in the example:

```
ODESolve[{y'[x]==1 - y[x], y[0]==0}, y[x], {x,0,10}]
```

and are (1) the differential equation ($y' = 1 - y$) and initial condition(s) ($y(0) = 0$), (2) the dependent variable to solve for ($y(x)$), and (3) the range of the independent variable to solve for ($0 \leq x \leq 10$). Execute this command, and then

Problem 1: Plot the solution to the differential equation for the sky-diver's velocity, (1), with initial condition $v(0) = 0$, using ODESolve, assuming that the sky-diver deploys his or her parachute at $t = 15$ seconds. Explain why the solution you obtain makes sense (what are the axes on the plot that ODESolve returns?).

Another way of visualizing the solution of a differential equation is by using a *direction field* (also called a *slope field*) for the differential equation. *Mathematica* will plot these with the `PlotVectorField` command.

Example Problem 4: Plot the direction field for the differential equation $\frac{dy}{dx} = 1 - y$.

Solution:

Now we plot the direction field for the differential equation $\frac{dy}{dx} = 1 - y$.

(note that there is a palette button to format the $\frac{dy}{dx}$ if you want to be really slick). Similar to needing to load Local‘diffeq‘ to run the ODESolve command, we need the Graphics‘PlotField‘ command for PlotVectorField: paste Needs["Graphics‘PlotField‘"] into your notebook and execute it:

```
Needs["Graphics‘PlotField‘"]
```

and then paste in the PlotVectorField command, changing it to:

```
PlotVectorField[{1, 1 - y}, {x, 0, 7},
    {y, 0, 3}, Axes -> True]
```

If you can't see exactly what the direction field looks like, try resizing it: click on the picture, and a box appears around it. By clicking on one of the tabs on the sides of the picture (and holding the mouse button down) you can make it wider or narrower, “the better to see [it] with, my dear” (—Ima Wolf, in Little Red Riding Hood). Then try adding an AspectRatio option to the PlotVectorField command:

```
PlotVectorField[{1, 1 - y}, {x, 0, 7},
    {y, 0, 3}, Axes -> True, AspectRatio -> 1]
```

What does this do?

Problem 2: Plot the direction field for the equation giving the sky-diver's velocity, (1). Then explain what information the direction field gives us about the solution to the differential equation, and how you could draw a picture of solutions using the direction field even if you didn't have equations for the solutions. In particular, be sure to explain how you would know where to begin drawing the solution curve, where it would go from there, and where the slope of the curve comes in.

A *nullcline* is a curve along which the slope of the dependent variable (y) in the differential equation is zero.

Problem 3: Find (by hand) the nullcline(s) of the sky-diver's velocity differential equation. Then plot these with the direction field and solution to the differential equation that you obtained in problems 1 and 2. *To combine multiple plots, it is easiest to name each—when running the Plot or PlotVectorField commands, assign the result to a variable name, e.g.,*

```
solnplot = ODESolve[etc....]
dirfplot = PlotVectorField[{1, etc....]
```

```
nullcplot = Plot[etc....]
```

and then use the Show command to display them together:

```
Show[solnplot, dirfplot, nullcplot]
```

Are the nullclines solutions to the differential equation? Would you expect this to generally be the case? Explain.

Problem 4: Find and plot the direction field and nullcline(s) of the differential equation

$$\frac{dy}{dx} = \frac{3}{2} - 3y + e^{-3x/2}.$$

Use $0 \leq x \leq 1$ and $0 \leq y \leq 1$. Also include in the plot solution curves for several different initial conditions (on $y(0)$) between 0 and 1.

When you are finished with the lab, check that your answers do in fact answer the questions that were asked, and that they are all in **complete sentences**. *Then print out (select* [File] [Print]*) and hand in your lab writeup. If you do not finish the lab today, finish your lab writeup as homework and hand it in on or before our next class meeting. It's a good idea to save your lab before leaving Mathematica—do this with* [File] [Save]*—and then quit Mathematica with* [File] [Quit]*.*

Lab 2. First-Order Models: Simple Drug Modeling and Solution Structure

In this lab we consider a simple model for the amount of a drug in ones bloodstream—but before plunging into the details, a quick review of how we're using *Mathematica*... *Remember that all of your answers should be in* **complete sentences** *written in your Mathematica notebook for the lab, to be handed in at the end of the lab!*

Example Problem 1: How did we get started with *Mathematica*?

> **Solution**:
>
> *With apologies to those who remember clearly how to do it all, we'll start with a bit of Mathematica explanation again: Let's title the lab as we did with Lab 1—select* [Format] [Style] [Title] *and then start typing:*
>
> DiffEq Lab 2: Drug Modeling and Solution Structure
>
> *And then, before plunging into the first problem, let's label it too: Click below the title to get the horizontal line that tells you that Mathematica is ready for additional input, then select* [Format] [Style] [Section] *and type*
>
> ■ Problem 1:
>
> *Now we're ready to answer problem 1. Click below your problem title, and proceed with problem one. Remember that you can get nice text formatting by selecting* [Format] [Style] [Text].

When one is "on drugs," so to speak, there is a rate at which the drug is introduced to ones body—given by some input function—and a rate that it is removed by natural body functions—like the kidneys 'cleaning' the blood to remove the drug. It is therefore reasonable to suppose that the rate at which the drug concentration in the blood changes should increase at a rate proportional the input function (from the gastro-intestinal tract), $\text{in}(t)$, and decrease at a rate proportional to the concentration in the blood.

Problem 1: Write a differential equation for the drug concentration in the blood, $b(t)$ corresponding to the description given above. Explain what each of the terms in your differential equation represent.

a. What is a reasonable initial condition for your differential equation? What would the initial condition $b(0) = 1$ imply?

Problem 2: Let's consider the input function $\text{in}(t) = \sin(2t) + 1$. Solve, by hand, your equation from (1). You may wish to use *Mathematica* to help with the integral on the right-hand side.

a. Your solution has an arbitrary constant in it. Suppose that $k_1 = .693$, $k_2 = .331$ (where k_1 is the constant of proportionality for the rate at which the drug concentration increases because of the drug input, and k_2 that for the rate of decrease), and that the initial condition is $b(0) = 0$. Solve for the constant in this case.
b. Plot your solution for the initial condition $b(0) = 0$ and $0 \le t \le 20$.
c. Check that you get the same result as when you use ODESolve (remember that you need to run Needs["Local'diffeq'"] before using ODESolve, and that running `?ODESolve` returns helpful information about ODESolve).

Problem 3: Now suppose that the input function $\text{in}(t)$ is given by

$$\text{in}(t) = \begin{cases} \sin(2t) + 1, & t < 15 \\ 0, & t \ge 15. \end{cases}$$

a. In words, what do you expect the solution to the differential equation to be like now? Why?
b. Obtain a solution curve for the drug concentration in the bloodstream with this input function and the initial condition $b(0) = 1$ to verify your penetrating insight. (A split-definition function like this can be defined in *Mathematica* with the `If[`*test*`,` *option*`,` *alternate option*`]` command—as seen in Lab 1.)
c. Explain how you could solve the problem with this input function by hand.

Problem 4: We next want to consider two "sub-cases" of the initial value problem

$$b'(t) = 0.693\text{in}(t) - 0.331b(t) \quad b(0) = b_0$$

that we have been considering.

a. For the first "sub-case," consider $b_0 = 0$ and the input function we used in (3),

$$\text{in}(t) = \begin{cases} \sin(2t) + 1, & t < 15 \\ 0, & t \geq 15. \end{cases}$$

Find a solution for the IVP in this case.

b. For the second "sub-case," suppose that $b_0 = 1$ and $\text{in}(t) = 0$. Find a solution to the IVP in this case.

c. Plot the solutions from the two "sub-cases" you just considered with the solution to the full problem from (3). (Recall that this is most easily done by naming the plots obtained and then using *Mathematica*'s `Show` command—as seen in Lab 1.) How do the three curves appear to be related? (An elementary math operation is involved, and it isn't division.)

We will next verify your observation in (3) by explicitly using *Mathematica* to find an approximate solution function for the differential equation. *Mathematica*'s numerical solver is called "NDSolve," and has as its arguments

`NDSolve[{`*differential eqn*`,` *initial condition(s)*`},` *dependent variable*`, {`*independent variable*`,` *min value*`,` *max value*`}]`

This returns a "rule" for how to solve the differential equation, as shown in the following example problem.

Example Problem 2: Use *Mathematica*'s NDSolve command to plot the solution to $y' = 1 - y$ with initial condition $y(0) = 0.2$.

Solution:

The *Mathematica* command to solve the differential equation given is

Note that this is on the "BasicCalculation" palette, under "Calculus", and then "Differential Equations"— **you will make fewer "==" and "=" errors if you use the palette**

```
NDSolve[{y'[x]==1 - y[x], y[0]==0.2}, y[x], {x,0,5}]
```

Notice that NDSolve returns a line saying "`y[x] -> Interpolating Function`*"—the "*`->`*" indicates a rule: Mathematica is saying "to solve the differential equation, replace all instances of $y(x)$ with the function I've given you." The multiple braces ({ }) indicate that it's really a list of rules (in this case, a list of one rule). It's easiest to use this list of rule(s) if we name it first, so let's change the NDSolve command to*

```
ysolrule = NDSolve[{y'[x]==1 - y[x], y[0]==0.2}, y[x], {x,0,5}]
```

Then, to apply the rule we use the `/.` *operator, which says "apply the rule on the right side of the operator to the expression on the left side." In this case, we apply it to* `y[x]`*, the entity named in the rule, and save it as a new variable:*

```
ysoln = y[x] /.  ysolrule[[1]]
```

In this, the `[[1]]` *indicates that we want to use the first (and in this case, only) item in the list. We can now plot the solution that we obtained:*

This sequence of commands extricates the solution function that *Mathematica* determined for us, and saves it as "ysoln," which we plot to see the solution of the differential equation:

```
Plot[ysoln, {x,0,5}];
```

For further information about this rule application, see Appendix A.

Problem 5: Use NDSolve to obtain solutions to the three IVPs that we considered earlier, namely,

i. $b' = 0.693\text{in}(t) - 0.331b(t), \quad b(0) = 1,$

ii. $b' = -0.331b(t), \quad b(0) = 1,$ and

iii. $b' = 0.693\text{in}(t) - 0.331b(t), \quad b(0) = 0,$ where $\text{in}(t)$ is given as in (3).

Check that you are getting the correct answers by plotting the results that you obtain and verifying that you are getting the same thing as you did before.

a. Use these three solutions to verify your conclusion from (4) about how the three solution curves are related, by applying the elementary operation you divined that problem to some of the 'solutions' that you obtained in (i)–(iii).

Lab 3. First-Order Differential Equation Models: World Class Sprinting

J.B. Keller, an applied mathematician of some renown, published a theory for modeling track sprints of up to 300 meters in which he suggested that the velocity, $v(t)$, of the runner may be described by the differential equation

$$\frac{dv}{dt} = A - \frac{v}{\tau}. \tag{1}$$

In this equation the 1973 values of the constants A (acceleration) and τ (a time constant) were $A = 12.2\ \mathrm{m/s^2}$ and $\tau = 0.892$ sec.[1]

It is interesting to note that this equation has the same form as many other differential equations modeling real-world phenomena—*e.g.*, a population with emigration and a net growth and death rate k is modeled by the equation

$$\frac{dP}{dt} = k\,P - E. \tag{2}$$

> *Remember that all of your answers should be in* **complete sentences** *written in your Mathematica notebook for the lab, to be handed in at the end of the lab!*

Example Problem 1: How did we get started with *Mathematica*?

Solution:

One more time, we'll start with a bit of Mathematica explanation again: Let's title the lab as we did before—select [Format] [Style] [Title] *and then start typing:*

DiffEq Lab: Modeling Sprinting

And then, before plunging into the first problem, let's label it too: Click below the title to get the horizontal line that tells you that Mathematica is ready for additional input, then select [Format] [Style] [Section] *and type*

- Problem 1:

Now we're ready to answer problem 1. Click below your problem title, and proceed with problem one. Remember that you can get nice text formatting by selecting [Format] [Style] [Text].

Problem 1: Consider equation (1) to answer the following questions.

a. What is the physical meaning of $\frac{dv}{dt}$?
b. What effect does each of the terms on the right-hand side of the equation have on the runner's acceleration?
c. Physically, the constant A on the right-hand side of the equation is how fast the runner can initially accelerate. What is the term $\frac{v}{\tau}$?
d. Based on your answers above, do you think that equation (1) is a reasonable model for the velocity of a sprinter?

Problem 2: What initial condition should be imposed for the problem given in equation (1) (*i.e.*, what should the velocity $v(0)$ at $t = 0$ be)? Why?

Problem 3: Now let's see what we can tell about the differential equation (1) with the initial condition you specified in (2) before we try to solve it.

a. What happens to the runner's velocity when $t = 0$? Why?
b. As the runner continues down the track, does his or her velocity continue to increase or decrease? Will it always increase or always decrease?

---

[1] Original problem from S.R. Dunbar, *The ODE of World Class Sprints*, **CODEE Newsletter**, Spring 1994, p.7., which was taken from J.B. Keller, *A Theory of Competitive Running*, **Physics Today**, Sept. 1973, p.43.

c. As the time t gets large, what do you think happens to the velocity? (Does this seem reasonable?)

Problem 4: Plot the direction field for the differential equation (1) when $A = 12.2$ and $\tau = 0.892$.

Recall that to plot the direction field for a differential equation $\frac{dv}{dt} = f(t, v)$ in Mathematica you must first load the 'Plotfield' package with the `Needs["Graphics`PlotField`"]` *command from the "LocalMath" palette in* [File] [Palettes]. *Then (for the rest of the lab!) you can plot direction fields by using the "PlotVectorField" command from the palette, filling in the □s—here we also name the plot 'dfield' so that we can use it again later:*

dfield=PlotVectorField[{1, *f(t, v)*}, {t, *tmin*, *tmax*}, {v, *vmin*, *vmax*}, Axes->True]

After you have found a range of values for t and v that give a good direction field plot, print out the graph (by clicking on the plot, and then selecting [File] [Print Selection]). On the printout, draw the solution $v(t)$ for the velocity as a function of time. Also make notes on the graph indicating (1) the initial condition, (2) the initial slope of the curve for $v(t)$, and (3) what happens for large values of t. Does this seem like a reasonable solution?

Next, we find out how to use *Mathematica* to solve differential equations for us! Let's first do it with an Example Problem.

Example Problem 2: The velocity of a lightweight falling sky diver may be crudely modeled by the differential equation $\frac{dv}{dt} = -32 - 1.58\,v$ and $v(0) = 0$. Solve this equation using *Mathematica* and interpret the result. Is this equation more appropriate before or after the sky diver's parachute opens? *Remember to work out all of the steps to this problem in your lab!*

Solution:

The command to Solve a Differential equation in Mathematica is "DSolve," and is found in the "Basic Calculations" Palette, under "Calculus" and then "Differential Equations." Click on this to paste

```
DSolve[{■==□, □==□}, □, □]
```

into your notebook. Change this to solve the differential equation and initial condition we want to solve:

```
DSolve[{v'[t]==-32 - 1.58 v[t], v[0]==0}, v[t], t]
```

The first argument (`{v'[t]=-32-1.58 v[t], v[0]=0}`*) is a list of equations to solve; the* `v[t]` *tells what to solve for, and the* `t` *tells what the independent variable is. Evaluating this command gives a (nested) list of rules for how to solve the equations—in this case, there's only one rule, so it's a list of a list of one rule:*

```
{{v[t] -> -20.25 + 20.25 E^-1.58 t}}
```

This says "to solve the differential equation, plug in $-20.25+20.25e^{-1.58t}$ for $v(t)$ in the equation." Armed with this, we can answer the question: changing to [Format] [Style] [Text], *we write something like—*

Using *Mathematica*'s DSolve command, we find that the solution to this differential equation with the given initial condition is $v(t) = 20.25(e^{-1.58t} - 1)$. When $t = 0$ this gives $v(0) = 0$, as it should because at $t = 0$ the sky diver has only just stepped out of the plane and isn't falling yet. As $t \to \infty$ the exponential goes to 0 so $v(t) \to 20.25$ft/sec—a downward velocity that is none-too-large, so we expect that this parachutist opened their parachute right away! We can see this by plotting the solution:

...which we can do by using the "Plot" button under "Graphics" in the "BasicCalculations" palette

```
Plot[20.25 (E^(-1.58 t) - 1), {t, 0, 5}]
```

Problem 5: Solve the problem for the velocity of the sprinter using *Mathematica* (Note: if it's messy, you can simplify things with the command `Simplify[%]`—which says to "Simplify" the argument "%"; in

Mathematica, "%" means "the result of the previous calculation") and plot the solution. Think about what range of values for t is appropriate for your plot; explain why you pick the range that you do. Then "Show" the plots of the direction field and solution together. Does your result agree with your estimate based on the direction field that you obtained in Problem 4? Be sure to explain any discrepancies between the two!

Problem 6: Using the solution and plot that you have obtained, estimate where the acceleration of the sprinter is the greatest. Be sure to tell how you know the acceleration from the plot of the velocity.

Problem 7: What is the sprinter's final velocity? How is it different if the sprinter is running a 100m or 200m race? Does this make sense? Why?

In the 1993 World Championships, Linford Christie from England won the men's 100m sprint in 9.87 seconds. (Gail Devers (USA) won the women's sprint in 10.82 seconds.) At the 30 meter mark, Christie was 3.85 seconds into the race.

Problem 8: Let's try to see if the values of A and τ that Keller found for 1973 are appropriate for Linford Christie in 1993. To do this, first use DSolve to find a solution to the differential equation (1) without plugging in any numbers for A and τ:

$$\texttt{DSolve[\{v'[t]==A - }\frac{\texttt{v[t]}}{\texttt{tau}}\texttt{, v[0]==0\}, v[t], t]}$$

Now, to make the work a bit easier, suppose that $\tau = .892$ and find A based on the data for Christie's 30m split. (Hint: there are a couple of ways of doing this; you might want to consider average velocities.)

Problem 9: Plot the solution for the Christie's velocity that you found in Problem 8 along with the solution for the velocity with $A = 12.2$ and $\tau = .892$ to compare the two solutions. Is there a large difference between the two—and is the difference in the direction you expect? Do you think that the solution for A that you found in (8) is correct? Why or why not? Can you explain your result?

Lab 4. The Spruce Budworm, and How Many Solutions Are There Anyway?

We have seen that differential equations often have different, unique solutions for different initial conditions—and that these solutions may tend towards different final (*i.e.*, as $t \to \infty$) values for the dependent variable. A question that may lead from this is that of how many solutions an initial value problem might actually have. In this lab we therefore investigate how many solutions we expect different initial value problems to have (in particular, whether there is no solution, one solution, or more than one solution).

First we examine the issue of *existence and uniqueness*—how many solutions there are to an IVP. The first IVP we will consider is

$$\frac{dp}{dt} = r\,p\left(1 - \frac{p}{k}\right) - \frac{p^2}{1+p^2}, \tag{1}$$

which models the population of the spruce budworm[1], a voracious 'predator' that consumes vast quantities of Canadian fir forests. In equation (1), p is the budworm population (in units having to do with a characteristic predator (bird) population), r is the budworm population growth rate (in units related to a measure of the predation rate), and k is the environmental carrying capacity for budworms (in the predator population unit).

Problem 1: Assuming that the constants r and k in equation (1) are $r = .5$ and $k = 10$, and that $p(0) = 5$, how many solutions are there to this IVP? Plot the direction field (recall that you need to run the appropriate "Needs" command first) for the differential equation and the solution curve together. Looking at the direction field, how many possible values for the final populations $p(t)$ are there?

a. Are the different final population values actually solutions to the differential equation? How do you know? Remind us why you are able to conclude that there is only one solution to the initial value problem (oops—you did conclude that, right?) when there are all of these different possible solutions.

Problem 2: Next, consider the differential equation

$$\frac{dy}{dt} = -k\sqrt{32.2\,y}\,, \tag{2}$$

which is a form of Torricelli's Theorem, which gives the change in the water level, y, in a bucket which is draining through a hole in its bottom (we assume here that when $y = 1$ the bucket is full, and that $k = 1$). What is y when the bucket is empty?

a. Plot the direction field for equation (2) for $0 \le t \le 3$ and $0 \le y \le 1$. Print it out (using `[File][Print Selection]`) and draw (by hand) on the direction field a solution to "backwards IVP" having 'initial condition' $y(3) = 0$ (your solution curve will start at some value of t less than 3). Is there more than one such solution? (Hint: What does the condition $y = 0$ say about how much water is in the bucket at $t = 3$?)

b. Draw two more solutions on the direction field. (Hint: Think about starting at different values of t which are less than 3.) Explain why you choose where they start and end by telling how full the bucket is at either point. In terms of the water level in the bucket, why is there more than one solution with the 'initial condition' $y(3) = 0$?

Problem 3: Now consider the differential equation

$$y\,\frac{dy}{dx} = x. \tag{3}$$

Solve this differential equation by hand. How many solutions are there if we take the initial condition $y(0) = 0$?

a. Plot the direction field for this equation for $-2 \le x \le 2$ and $-2 \le y \le 2$. Plot the solution(s) that you found above and show them together with the direction field.

[1] J.D. Murray, **Mathematical Biology**, Springer-Verlag, 1990, pp. 4–8.

Problem 4: We are now in the position to draw some conclusions from the results of the preceding problems. To compare them, write them all in the form $\frac{dy}{dx} = f(x, y)$. For the first equation (the budworm problem) there is only one solution to the initial value problem; for the other two there is more than one solution. In both of Problems 2 and 3 the initial condition was $y = 0$. What is true about the function $f(x, y)$ for equation (3) when $y = 0$? What is true about the function $\frac{\partial f}{\partial y}$ for equation (2) $y = 0$? Based on these observations, how are these functions $f(x, y)$ different than the $f(x, y)$ for the budworm problem?

a. Formulate the theorem that says when an initial value problem

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0 \tag{4}$$

will have a unique solution. Your theorem should have the form "**If** *some stuff is true about y or f* **then** *something about the number of solutions to equation (4)*".

Problem 5: **After** you have completed Problem 4, consult the appendix to this lab and compare your theorem with the rendition there, which you will also find in your book. Make a list of any differences between the two (even small ones!) and explain how these differences change the meaning of the theorem.

Appendix

Existence and Uniqueness Theorem for First-Order Initial Value Problems. *If a function $f(x, y)$ and its derivative $\frac{\partial f}{\partial y}$ are defined and continuous for a range of x and y values about $x = x_0, y = y_0$, then the initial value problem given in equation (4) has a unique solution in that range of x and y values.*

Lab 5. Numerical Methods for ODEs, or Solving the Budworm Problem

While we spend a significant amount of time in this course considering problems for which we are able to obtain explicit solutions, it is a sad (?) fact that *most* of the problems that are encountered in the course of modeling the real world do *not* have analytical solutions (solutions that we can find by hand). Because of this, other methods have been developed to obtain approximations to the solutions of such problems. Among these are numerical methods, which give us approximate solutions that we may examine graphically to determine the behavior of the system we are studying. In this lab we will consider how some numerical methods work and how to use them.

The equation we are interested in for this lab is

$$\frac{dp}{dt} = r\,p\left(1 - \frac{p}{k}\right) - \frac{p^2}{1+p^2}, \tag{1}$$

which models the population of the spruce budworm[1], a voracious beast that has cost Canada vast expanses of balsam fir forest. Note that this equation is horribly nonlinear (because of the presence of the $p^2/(1+p^2)$ term), and (thus) cannot be solved analytically for an explicit solution. As noted in Lab 4, p is the budworm population (in units having to do with a characteristic predator (bird) population), r is the budworm population growth rate (in units related to a measure of the predation rate), and k is the environmental carrying capacity for budworms (in the predator population unit). The horribly nonlinear $p^2/(1+p^2)$ term reflects predation—primarily by birds. While you do not have to enter this in your lab writeup, stop for a moment to verify that each of these terms has the effect on $p'(t)$ that you would expect. For this lab, we'll consider $r = .5$ and $k = 10$.

Problem 1: We said that equation (1) is "horribly nonlinear" because of the $p^2/(1+p^2)$ term. If this term were omitted, would the equation be linear or nonlinear (and why)? Would it be possible to solve it? How? (Note that you aren't actually being asked to solve it!)

The numerical method we investigate here is Euler's method, which finds an approximate solution to a differential equation by using the difference quotient,

$$\frac{y(x+\Delta x) - y(x)}{\Delta x}, \tag{2}$$

to approximate the derivative in the equation.

Problem 2: Write an approximation to equation (1) by using a difference quotient (as shown in equation (2)) to approximate the derivative. (That is, you should be replacing the derivative in (1) with an approximation like that given by (2).)

a. Rearrange the approximate equation you obtained in (2) to get the formula for Euler's method (*i.e.*, solve for your equivalent to $y(x + \Delta x)$). Explain how you could use this formula to get an approximate solution, *e.g.*, for $p(3)$ or $p(8)$, if you knew that $p(0) = 5$—in particular, what is Δt and what part does it play in the approximation?

b. Explain clearly how the different terms in your formula are related to the terms in the equation for the tangent line to the graph of $p(t)$. *Hint: think about what the quantity multiplying Δt tells you about p—given that it is the right-hand side of equation (1).*

Many numerical methods use a formula such as that you found in (2a)—which allows the estimate for the 'next' value of the population (that at the time Δt later) to be obtained by plugging in the current value. To do this in practice, it is convenient to define a function `pnext` that returns the next value of the population as determined by the Euler formula, as a function of `pprev`, the population at the time `tprev`.

[1] J.D. Murray, **Mathematical Biology**, Springer-Verlag, 1990, pp. 4–8.

More friendly Mathematica stuff:

- Defining functions—the short rule: to define a function $f(x)$ in *Mathematica*, use the command:

  ```
  f[x_] := stuff
  ```

  Note (1) the "_" after the independent variable (x) in the definition *(but only put it on the left-hand side of the definition!)*, and (2) the ":=" used for the "equals" sign. Thus, to define the function $g(x,y) = x^2 \sin(y) + a$, we would use the command

  ```
  g[x_,y_] := x^2 Sin[y] + a
  ```

  For additional details about function definitions in *Mathematica*, see Appendix A.

Example Problem 1: Find a formula for the next Euler step for the equation

$$\frac{dy}{dx} = \sin((y+a)^2). \tag{3}$$

Then obtain an approximate solution to equation (3) if $y(0) = 1$ and $a = 1$ for $0 \le x \le 1$ using Euler's method and a step size of $h = .25$. Plot this solution and comment on its accuracy.

Solution:

By analogy with the work done in problem (2), we will generate successive values for y using the formula

$$y(x + \Delta x) = y(x) + \Delta x \, \sin((y(x) + a)^2)$$

or, with $y(x+\Delta x) = y_{next}$, $y(x) = yprev$ and $\Delta x = h$,

$$y_{next} = yprev + h \, \sin((yprev + a)^2).$$

To define this as a function in Mathematica, we enter.

```
ynext[yprev_] := yprev + h Sin[(yprev + a)^2]
```

Next, we define the x and y points at which we have a solution value:

Remember that you can cut and paste to reduce the amount of typing when entering repetitious entries!

```
h = 0.25; a = 1; x0 = 0; y0 = 1;
x1 = x0 + h; y1 = ynext[y0]
x2 = x1 + h; y2 = ynext[y1]
x3 = x2 + h; y3 = ynext[y2]
x4 = x3 + h; y4 = ynext[y3]
```

Note that the addition of the "`;`" at the end of a command means that Mathematica doesn't print the result of that command, and lets us enter more than one command on a line. Because we don't have an explicit solution, we can't use the `Plot` command to look at our result, and instead use the `ListPlot` command, which plots a list of {x,y} pairs—you can find in the "BasicCalculations" palette, under "▹ Graphics". Using this or by typing it in yourself, execute the command

```
ListPlot[{{x0,y0},{x1,y1},{x2,y2},{x3,y3},{x4,y4}}]
```

If it's hard to see what it's doing with just dots, add lines with the `PlotJoined` option:

```
eplt=ListPlot[{{x0,y0},{x1,y1},{x2,y2},{x3,y3},{x4,y4}}, PlotJoined->True]
```

The solution that we graphed looks pretty ragged! Because the curve is not smooth, we expect that it is inaccurate, and so should solve the equation with greater accuracy (smaller h) to get a better picture of what's going on.

Problem 3: Print out the solution obtained in Example problem 1 and label the points $\{x_0, y_0\}$, $\{x_1, y_1\}$, $\{x_2, y_2\}$, $\{x_3, y_3\}$ and $\{x_4, y_4\}$.

Example Problem 2: Find the solution to the problem in equation 3 using Euler's method with a step size $h = .1$. Compare this with the solution with $h = .25$.

Solution:

First, we step through the interval $0 \le x \le 1$ with a step size $\Delta x = h = .1$, generating the y values at those points using the function y_{next} that we defined in the previous example problem.

To actually do the Euler stepping, we let Mathematica do all of the work: the `Do[■,{□,□,□}]` *command is also in the "LocalMath" palette, and says "Do ■ a bunch of times in succession, first with □ (the second one)=□ (the third one), and so on up to □ (the second one)=□ (the last one). We can use this to generate the different x and y points at which we find our Euler solution:*

```
h=.1; a=1; x[0]=0; y[0]=1;
Do[x[i] = x[i-1] + h, {i, 1, 10}];
Do[y[i] = ynext[y[i-1]], {i, 1, 10}];
```

(Iterators such as the Do command are introduced in Appendix A.) Now we put the list of x and y values that we found into a list by using the `Table[■,{□,□,□}]` *command, which is in the "Basic Calculations" palette under "▷ Lists and Matrices," "▷ Creating Lists and Matrices." This is very similar to the Do command, except that it puts all of the ■s into a list (i.e., between a { and }). So that we can plot things in color, we also run the* `Needs["Graphics'Colors'"]` *command—which is on the "LocalMath" palette and is similar to the Needs["Graphics'PlotField'"] command.*

```
list0points = Table[{x[i], y[i]}, {i, 0, 10}];
Needs["Graphics'Colors'"]
eplt2 = ListPlot[list0points, PlotJoined->True, PlotStyle->Blue]
Show[eplt,eplt2]
```

The result that we obtain with a greater number of points is much smoother and, we expect, a better approximation to the actual solution. Looking closely, however, it is still possible to discern the points and straight line segments of the solution, which leads us to believe that it is still not so accurate as all that.

Problem 4: Solve the spruce budworm problem for $0 \le t \le 20$ using Euler's method with a step size of .1 and initial condition $p(0) = 5$, by (i) defining a function for "pnext" in terms of "pprev"; (ii) defining any parameters you might have that need sorting out, and (iii) running an appropriate "Do" command. What happens to the population as t gets large? What does this mean for the survival of the Canadian fir tree forests?

Simple as it is, Euler's method is not a spectacularly accurate numerical method. In fact, it is in practice never used because there are far better methods lurking in the wild plains of engineering and applied science (*e.g.*, Runge-Kutta). In some respects the most significant difference between Euler's method and these more sophisticated numerical solvers is in the definition of the function y_{next}. *Mathematica*'s sophisticated numerical solver is called NDSolve, which we now use. The `NDSolve[{■==□,□==□},□,{□,□,□}]` palette button is found in the "Basic Calculations" palette, under "Calculus" and "Differential Equations." It obtains a numerical solution to the equations {■==□,□==□}, solving for □ (the one after the equations). The second set of braces is like that for the Plot command, defining the independent variable and range of values for that variable. We use this in the next problems. (Note that NDSolve is used in the LocalMath palette function ODESolve. Use NDSolve in the following example problem; you may want to use ODESolve for problem 5.)

Example Problem 3: Numerically solve the differential equation

$$\frac{dy}{dx} = \sin((y + a)^2).$$

for $0 < x < 2$ (and $a = 1$) with an initial condition $y(0) = 1$ using NDSolve.

Solution:

To find a numerical solution to this problem, we use *Mathematica*'s NDSolve command to obtain a solution rule, which we use to extract a 'function' that solves the problem, and then plot it with the Euler's solutions that we had before.

First, we clear some of the variables that we've used before, to avoid their being confused between the two uses.

```
Clear[x,y,eqn,nsolnrule,yfunc,solnplot]
```

Then we obtain a solution rule from NDSolve:

```
nsolnrule = NDSolve[{y'[x]==Sin[(y[x]+1)^2], y[0] == 1}, y[x], {x, 0, 2}]
```

This returns a solution rule, which we use to substitute for "y[x]" to define a solution function, and then plot it along with the Euler method plots from before:

```
yfunc = y[x] /. nsolnrule[[1]]
solnplot = Plot[ yfunc, {x, 0, 2}, PlotStyle->Red]
Show[solnplot,eplt,eplt2]
```

We see that the solution with NDSolve is in fact noticably different than both of the Euler's method solutions, and suspect that it is in fact that which is the most accurate.

Problem 5: Use NDSolve or ODESolve to solve the spruce budworm problem for $0 \leq t \leq 20$ and initial condition $p(0) = 5$. Compare with your Euler solution and comment on the accuracy of the latter.

Lab 6. Drugs and Error in Numerical Methods

Related to the issues of how to use numerical methods to solve differential equations is that of the accuracy (or, equivalently, error) in the methods used. To illustrate this, in this lab we consider a differential equation modeling the buildup of a regularly administered drug in a patient's system. If the drug is administered, say by an I.V. drip, at intervals of 2 hours and in a peak concentration of .5 drug units, then $d(t)$ (the amount of the drug in the patient's system) may be sort of modeled[1] by the differential equation

$$d'(t) = -.5d(t) + .5e^{-15\sin^2(\frac{1}{2}\pi t)}. \tag{1}$$

The first of the terms on the right-hand side models the decay of the drug concentration and the second the regular addition of the drug. Stop for a moment and check that these terms have the effects that you expect on the amount of drug $d(t)$ in the patient's system.

In the following problems, we will investigate the types of error encountered when numerical methods are used to solve equations such as (1).

Problem 1: What initial condition should be added to equation (1) to model a patient who is beginning treatment? How necessary are numerical methods in the solution to the differential equation (1) with the initial condition that you posed (could you solve the problem analytically)? Explain.

Problem 2: Use the *Mathematica* command `NDSolve` (in the "Basic Calculations" "▷ Calculus" "▷ Differential Equations" palette) to obtain a solution for the drug concentration problem for $0 \le t \le 20$. Plot the solution and explain how or whether this is what you would expect based on the differential equation. (In this case we need an actual function giving the solution to the differential equation for comparison with the approximation we generate below, and so use NDSolve rather than ODESolve. Recall that NDSolve was introduced in Lab 2—see that lab if you don't remember how to use the solution rule that you obtain from it.)

Problem 3: Now use Euler's method to approximate the solution to this differential equation with a step size $h = .2$, for $0 \le t \le 5$. You may want to define a function giving the next value of d as a function of the previous value and the value of t to do this, and then define a set of values for d and the corresponding values of t—*e.g.*, after defining the function "dnext,"

```
h = 1.; teul[0] = 0; deul[0] = 0;
teul[1] = teul[0] + h; deul[1] = dnext[deul[0], teul[0]]
```

etc.

a. Compare your results with the Euler method with those of the closer-to-exact solution that NDSolve returned by making a table of the values for t, the solution given by NDSolve, and the values that you obtained in (3). If you named your solution (obtained with NDSolve) "dsoln," you can plug values of t into it by using a rule: `dsoln /. t ->` *value*. This can be done for a number of values at once by using the Table command:

```
Table[{i, dsoln /. t->i, deul[i]}, {i,0,5}] // MatrixForm
```

Comment on the accuracy of this solution—in particular, what is the *local truncation error* for the first step (how far off is the first step away from the exact initial condition)? What is the *global error* for $t = 5$ (how far off is the Euler estimate at $t = 5$)?

b. The theory associated with Euler's method indicates that the local truncation error is proportional to Δt^2 and the global error to $(\Delta t)(\textit{local error}) = \Delta t$. Predict what the local error and global error at $t = 2, 3, 4,$ and 5 will be if you decrease h from 1 to $h = .2$.

Problem 4: Now let's check your predictions from (3) for $h = .2$. Using the `Do` command, find the Euler approximation to the solution to the problem for $h = .2$ and $0 \le t \le 5$:

[1] by which we mean that the qualitative behavior of the solution is approximately correct...

It is possible to do this in a particularly cunning manner if you like, so that it is easy to compare this estimate with the solution obtained by NDSolve—e.g.,

```
h=.2; deul2[0] = 0; teul2[0] = 0;
Do[ teul2[i/5] = teul2[(i-1)/5] + h;
    deul2[i/5] = dnext[deul2[(i-1)/5], teul2[(i-1)/5]],
    {i, 1, 100}];
Table[{i, dsoln /. t->i, deul2[i]}, {i,0,5}] // MatrixForm
```

a. Find the local truncation error and the global error for each of $t = 2, 3, 4$, and 5 for the case of $h = .2$. Are they what you expect? Explain.

Problem 5: Finally, plot the two estimates that you obtained in the preceding two problems along with the solution from NDSolve, which we will gamely assume is close to the exact solution. To do this, you may want to first use the Table command to generate lists of (t, d) values to plot with ListPlot for each of the Euler solutions. Explain based on the graphs how the errors that you predicted and observed are consistent with the behavior of the solution. In particular, if your estimates from (3) were higher or lower than the values you found in (4), use your graph to explain why this might be.

Lab 7. The Qualitative Return of the Budworm and Such Things

We have in the past considered a tremendously interesting equation modeling the population of the spruce budworm[1],

$$\frac{dp}{dt} = rp\left(1 - \frac{p}{k}\right) - \frac{p^2}{1+p^2}, \qquad p(0) = p_0. \tag{1}$$

In this lab we consider what we may determine about the qualitative behavior of solutions to the equation and how this relates to the information provided by a specific solution to the differential equation (howsoever we might obtain it!). Among the tools we have at our disposal for the qualitative analysis of an equation are direction field plots and the analysis of what the differential equation tells us about the slope, concavity and possible equilibrium states of the solution. In this lab we concentrate on the information provided by the equation for the slope provided by a first-order equation and see what it might tell us about the equilibrium states of a differential equation.

Problem 1: To begin with, let's consider equation (1) with $r = .5$ and $k = 10$. Plot the right hand side of the equation with these values of r and k. Use the `AxesLabel` option to label the axes of your plot to show what variables lie along each (so your plot command will look something like `Plot[■,{□,□,□}, AxesLabel->{`*xlabel*`,` *ylabel*`}]`). Explain in your write up what the equilibrium solutions to equation (1) appear to be and why.

a. Find the equilibrium solutions of the equation more precisely by using the `NSolve[■==□,□]` command (in the "Basic Calculations" palette, under "▷ Algebra" "▷ Solving Equations"—the last □ is the variable to solve for). Explain which equilibrium will be reached for the two initial conditions $p(0) = 2.1$ and $p(0) = 1.9$ by referring to your plot of the right hand side of equation (1). Is the equilibrium solution $p = 2$ stable or unstable?

b. Classify each of the equilibrium solutions that you found in (1a) as stable or unstable.

c. Suppose that the growth rate of the budworms, r, increases from $r = .5$ to $r = .51$. Plot the right hand side of the equation in this case. Is there any significant change in the equilibrium solutions? Then determine what happens to the solutions with initial conditions $p(0) = 1.9$ and $p(0) = 2.1$. Is there a change from the behavior you found before?[2]

Problem 2: Having noticed that there may be a fairly significant change in the behavior of the solutions to the problem with only a small change of the growth rate r, we might ask ourselves what will happen if r is increased further. Let's investigate this a bit more here.

a. Plot the graph of the right hand side of the equation for the completely randomly chosen value of $r = .559525$ and slightly less logical value of $r = .6$. Find the equilibrium states and comment on how they change as the value of r increases.

b. Print out each of the four graphs of the right hand sides of the equation that you obtained in this and the previous problem, one to a graph (it would be nice if you could do this two to a page—or better! Appendix A explains how to use the GraphicsArray command in *Mathematica* to graph things side-by-side, which would easily allow one to print four to a page). Along the p axis of each graph, draw arrows showing for each region between equilibrium states the direction that an initial population will move (*i.e.*, whether it will increase or decrease). Add a comment in your lab writeup explaining how you draw the arrows that you do. Label each equilibrium solution as stable or unstable (assume that for the case $r = .559525$ the two equilibria have exactly coalesced into one).

[1] J.D. Murray, **Mathematical Biology**, Springer-Verlag, 1990, pp. 4–8.

[2] The analysis of how equilibrium solutions of a problem change as a parameter is varied is called *bifurcation analysis*.

Problem 3: Finally, let's consider how the qualitative information that we obtained by looking at a graph of the *phase lines*[3] that we've graphed in the preceding problems is different than the behavior exemplified by a solution to the differential equation. In particular, let's consider the problem with $r = .6$ and initial condition $p(0) = .1$. Based on your work above, can you determine how long it take for p to grow to within .1 of the steady state $p = 7.9$? Explain how or why not.

a. Now that you've decided that we don't actually have the resources at our disposal to answer that question yet, let's get a real answer. To do this, we need a solution to the differential equation! Use *Mathematica*'s NDSolve (not ODESolve) command (in the "Basic Calculations" palette, under "▹ Calculus" "▹ Differential Equations") to find a solution rule for equation (1) with initial condition $p(0) = .1$. Then...

b. Use the solution rule that you obtained with the replacement operator `/.` to cunningly define a new variable `soln` that is the solution to the differential equation (see Lab 2 for an example of this if need be). Then use this to find how long it will take for the population to reach 7.8. You might want to use `FindRoot[■==□, {□,□}]` (where the □s in the { }s are, respectively, the variable to solve for and an initial guess for that—FindRoot is in the "Basic Calculations" "▹ Calculus" "▹ Numerical Root Finding" palette menu) to find a more-or-less exact value. Is the value you obtain what you expect?

Problem 4: Summarize the type of information you are able to obtain from a phase plane plot. In bifurcation analysis (see footnote 2), is phase plane analysis or the numerical solution of the problem (as we did in problem 3) most useful? Explain.

[3] A "phase plane" is a graph of the derivative of the solution to a differential equation, *e.g.*, $\frac{dp}{dt}$, against the solution itself, *e.g.*, p—as we have done here. For first-order problems, we call this a phase line because the motion is really only in one dimension. In second order equations we see a true phase plane!

Lab 8. Oscillatory Phenomena, or Future Freeways

Let us suppose for the moment that it is the year 2032... You are commuting to work from your spacious suburban home on the outskirts of Lincoln, Nebraska, to one of the industrial suburbs of Chicago, some 200 miles away. You ease your car onto the entrance ramp to I-80 and the computer on the dashboard locks into the highway supercomputer and takes over the control of the car. You swing your chair around to the coffee pot in the rear of the car as you accelerate to the highway cruising speed of 100 mph and slide in to follow the car in front of you at the standard following distance of 20 feet.

What do you think? "Smart" roadways of this type have already been tested in California to remove the hassles and delays of commuting. But we probably want to be sure before you get into your computer controlled car that everything is going to work as planned—a small 100 mph miscalculation could be messy! In this lab we consider a model for how the computer in your car might try to make sure that your car is going the same speed as the one in front of you (probably a good thing) and get it reliably to the "standard following distance" so that the highway will operate efficiently for a high volume of traffic.

How will the computer adjust the speed of your car? Let's suppose that there is a laser on your car that the computer can use to determine the distance between your car and the car in front of you (which we'll assume is moving at a constant 100 mph) by timing how long it takes the laser light to travel there and back. Then the computer in your car can cause it to accelerate or decelerate according to whether that distance is greater or less than 20 feet. If $x(t)$ is how much the distance (in feet) between your car and the next one differs from 20 feet, the acceleration of your car is $x''(t)$, and this leads to the differential equation

$$\frac{d^2x}{dt^2} + k\,x = 0, \tag{1}$$

where k is a measure of how fast your car accelerates or decelerates. Note that if $x = -10$, your car is 10 feet from the one in front of you and if $x = 30$, you are 50 feet behind it.

Problem 1: Rewrite equation (1) in the form $\frac{d^2x}{dt^2} = \mathit{stuff}$ and explain how it is the correct equation for the description in the preceding paragraph of how the car's computer accelerates or decelerates the car. *(Hint—recall that $x(t)$ is a distance between the cars, not an absolute position. If $x'(t) > 0$, what is happening to $x(t)$? What does this imply about the distance between the cars? Suppose that $x''(t) > 0$: what does this say about $x'(t)$? How is this related to the distance between the cars? Then take these ideas and think about what should be happening to x' and therefore x'' when $x < 0$ or $x > 0$.)*

Problem 2: Let's assume that initially your car is 100 feet behind the one in front, and is traveling 20 mph (1760 feet/min) slower. What initial conditions (values for $x(0)$ and $x'(0)$) do these correspond to? Why? (Work in units of feet and minutes.) Think carefully about the initial velocity $x'(0)$! Again, recall that $x(t)$ is a distance between cars, not an absolute position.

Problem 3: Use *Mathematica*'s NDSolve command or the ODESolve command from the "LocalMath" palette to solve equation (1) with the initial values that you found in Problem (2) and $k = 1$. (Note that for either command, because we have a second-order differential equation with two initial conditions, you will have to add a third equation in the first set of braces, to have there {`x''[t] + x[t]==0, x[0]==80, x'[0]==1760`}. Use the AxesLabel option to add appropriate labels to the axes of your plot, by including `AxesLabel->`{*xlabel, ylabel*} in the command to plot the solution (ODESolve or Plot).

a. Do you get what you expect? What does the plot tell you about the distance between your car and the one in front? Does this in any way disturb you? (Why?)

b. Putting aside (for the moment, at least) any reservations you might have about this observation, does the distance between your car and the next ever get to 20 feet? Why or why not?

Problem 4: To try and alleviate the difficulties discovered in Problem (3), let's try increasing the value of k to increase the amount the computer tells the car to accelerate or decelerate when the distance between the cars is not equal to 20. Take $k = 10$ and solve the problem again. Does the increased value of k take care of the problem? What does it do? Explain why (or why not) you expect (not) to be able to get a usable solution by changing k. Try another value of k if it is useful to substantiate your answer.

Problem 5: You have now considered the differential equation

$$\begin{aligned} \frac{d^2x}{dt^2} + k\,x = 0, \\ x(0) = x_0, \quad x'(0) = v_0 \end{aligned} \tag{2}$$

for a couple of different cases. Make a statement about the types of solutions that you expect to see for this initial value problem (*i.e.*, do they settle down to a constant value, grow without bound...?). Based on this, what types of functions to you expect are part of the formula for $x(t)$?

To cause your car to settle down to a constant following distance and speed we must modify the model we formulated before (equation (1)). One way to do this is to have your car's computer monitor not only the distance between your car and the car in front but also your velocity—it can then try to make the velocity converge to 100 mph at the same time as it tries to make the distance equal to 20 feet. If we add this to the model, the differential equation becomes

$$\frac{d^2x}{dt^2} + c\,\frac{dx}{dt} + k\,x = 0. \tag{3}$$

Problem 6: Solve and plot equation (3) with the initial conditions that you found originally, and with $k = 1$ and $c = 1$, using NDSolve or ODESolve. Has this modification to our model has resulted in a workable solution?

Problem 7: If the solution still isn't quite right, we might try changing c to improve things. Should we increase or decrease it from 1? Why? Try solving equation (3) with $c = 2$, $c = 4$, and $c = 8$. What happens to the solutions that you find? How are they different? (Plot them together with the solution for $c = 1$ to help answer this.) Would any of them be appropriate for the freeway?

Problem 8: Explain how the solution to the initial value problem

$$\begin{aligned} \frac{d^2x}{dt^2} + c\,\frac{dx}{dt} + k\,x = 0, \\ x(0) = x_0, \quad x'(0) = v_0 \end{aligned} \tag{4}$$

differs from the solution to the problem given by equation (2). Based on the appearance of the solutions, what types of functions would you expect to be in them?

Problem 9: Finally, let's see how to obtain orbits in the phase-plane (this graph is a phase-portrait) and time-state curves. Generate solutions for the *state variables*—$x(t)$ and $x'(t)$—in (3) by solving the IVP with $k = 1$, $c = .1$, $x(0) = 100$ and $x'(0) = 0$ with NDSolve:

```
solnrule = NDSolve[{x''[t] + 0.1 x'[t] + x[t] == 0, x[0] == 80, x'[0] == 0},
        x[t], {t,0,100}];
xsoln = x[t] /.  solnrule[[1]]; xprimesoln = D[xsoln, t];
```

Then use *Mathematica*'s ParametricPlot command (under "▹Graphics", in the "Basic Calculations" palette) to generate an orbit in the phase plane—that is, a plot of $x'(t)$ *vs* $x(t)$. In general a phase portrait is just a plot with the state variables on the axes.

a. Finally use *Mathematica*'s ParametricPlot3D command to plot the Time-Space curve: essentially just a phase plane, but with the t dependence along an additional axis.

```
ParametricPlot3D[{xsoln,xprimesoln,t}, {t,0,100}];
```

You might want to add options such as `PlotPoints->500` and `AspectRatio->1` to get a nice plot.

Lab 9. Linear Independence, Superposition and Peculiar Pendulums

We can discuss at great length the implications of linearity of equations, linear dependence and independence, superposition and so on—in fact, everything you ever wanted to know about linearity and then some. In this lab we will investigate what that means with the assistance of a pendulum or two. Recall that the motion of a pendulum is governed by the equation

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\sin(\theta) = 0 \tag{1}$$

where θ is the angle that the pendulum makes with the vertical, g is the acceleration due to gravity (9.81m/s^2) and L is the length (in meters) of the pendulum.

This is, however, a nonlinear equation, and so we often use its linearization,

$$\frac{d^2\theta}{dt^2} + \frac{g}{L}\theta = 0, \tag{2}$$

which is valid for small angles θ.

And, without further *adieu*, let's get started!

Problem 1: Suppose we start by pulling the pendulum back an angle $\frac{\pi}{4}$ and then release it. What initial conditions would we add to equation (1) or (2) in this case? Use *Mathematica*'s DSolve command (found in the "Basic Calculations" palette, under "▹ Calculus", "▹ Differential Equations") to solve the linear differential equation (2) with these initial conditions and $\frac{g}{L} = 9.81$.[1] (Remember that you'll have to add another initial condition in the list of things that you're solving, so that you execute a command like `DSolve[{θ''[t] + 9.81 θ[t] == 0, θ[0] == π/4, θ'[0] == 0},θ[t],t]`.)

a. Use the replacement operator `/.` to extract the solution from the rule that DSolve gives you. Is it what you expect? (Hint: what's $\sqrt{9.81}$?) Plot the solution to verify that it behaves as you expect.

Problem 2: In that that was so much fun, let's try something slightly different! Suppose that instead of pulling the pendulum bob back and letting go, we tap it so that it starts moving with some velocity, say 5 m/s. Write out the initial value problem (differential equation (2) plus some new initial conditions). What solution do you expect to get for this (answer this without solving the differential equation). Then use DSolve to solve the problem as you did in problem (1). Does your solution make sense—in particular, why did you get the coefficient that you did? Plot this solution as well to verify that it appears reasonable.

Problem 3: Compare the two solution formulas and graphs that you got for the solutions to problems (1) and (2). How are they similar and different (answer this question from the point of view of the formulas that you obtained and the graphs of the solutions)? Could the formula from problem (2) be used as part of a solution to the inital value problem (which consists of the differential equation and initial condition(s)) that you solved in problem (1) or vice versa? Explain your answer—in particular, which parts of the initial value problem would it solve, and which parts would it not solve? With which part of the title to this lab does this have to do?

Problem 4: Now, because we thrive on running the ragged edge of disaster, let's concoct a new problem that includes parts of the two that we've already considered. Suppose that we pull the bob of the pendulum back a distance of $\frac{\pi}{4}$ and, instead of just releasing it, push it outwards with a speed of 5 m/s. Write out the initial value problem for this situation and then, *without* using *Mathematica*, write out the solution to the problem—based only on your solutions to the previous problems. Explain how you know that this is the solution.

Problem 5: One way of checking that your solution to Problem (4) is correct is by directly substituting it into the initial value problem. While we could, of course, do this by hand with ease and enthusiasm,

[1] *Mathematica* will solve the problem for arbitrary g and L also! If you try this, do you get the same thing as when you specify g and L to start with?

we will instead use *Mathematica*. To check the initial conditions, you can use the `/.` operator—name your solution guess from (4) something cunning, say `solnguess` (be sure to keep at least four decimal places in your solution), and then to check the first initial condition substitute 0 for t:

```
ic1check = solnguess /. t -> 0
```

Similarly, to check the second initial condition, first take the derivative of `solnguess` with the `D` command, and then substitute for t—

```
ic2check = D[solnguess,t] /. t -> 0
```

Do you get what you expect for these checks? Next substitute `solnguess` into differential equation (2). To do this you will need to use the `D` command twice to get the second derivative. You can do this by typing `D[ D[solnguess,t], t]` (so that you are taking the derivative with respect to t of the derivative with respect to t), or by typing `D[solnguess, {t,2}]` (which tells *Mathematica* to take the second derivative):

```
eqncheck = D[solnguess, {t,2}] + 9.81 solnguess
```

Is the result obviously zero? Plot it to see, and then use the Simplify command to see if it really is zero.

Problem 6: Now let's try the ideas that we explored in the preceding problem with the "real" pendulum equation, equation (1). Start with the initial conditions for the pendulum that is pulled back and released, namely $\theta(0) = \frac{\pi}{4}$ and $\theta'(0) = 0$. Try to solve the initial value problem with these initial conditions and equation (1) using DSolve. What happens? Why?

Use *Mathematica*'s NDSolve (not ODESolve—we need the function solving the differential equation in this case) to get a solution. Call the solution that you extract (using the replacement operator `/.`) from the rule that NDSolve gives you something descriptive (we'll call it `nlsoln1` below). Then plot this solution with the solution to the corresponding linear problem. Do you think that the solution that you got with the linear equation is a good approximation to the actual solution? You will of course remember that the requirement for being able to use the linear equation as an approximation to the nonlinear equation is that the angle θ is small—is this the case here?

Problem 7: Solve the nonlinear problem with the initial conditions for when the pendulum was initially thrown "outwards" with a speed 5 m/s, $\theta(0) = 0$ and $\theta'(0) = 5$, using NDSolve. Call the solution something (we'll call it `nlsoln2` below) and plot it and its linear equivalent together and comment on how reasonable the linear approximation is in this case—explain on the basis of the amplitude of the solution why the linear approximation is better or worse in this case than the result you found in problem (6).

Problem 8: Suppose you now wanted to solve the initial value problem

$$\frac{d^2\theta}{dt^2}+9.81\,\sin(\theta) = 0$$

with initial conditions

$$\theta(0) = \frac{\pi}{4} \quad \text{and} \quad \theta'(0) = 5.$$

What solution would you guess based on your results from problem (4) (you will have to express this in terms of `nlsoln1` and `nlsoln2`)? Define your guess in *Mathematica* in terms of `nlsoln1` and `nlsoln2`.

a. Check if the initial conditions are satisfied by using *Mathematica*. Then plug your guess into the differential equation (the nonlinear equation!) using *Mathematica* and then plot the result to see if the equation is satisfied. Do you get what you expect? Why or why not? Explain whether you think it is reasonable to say that it is really zero.

b. Say in a couple of sentences why things are different for this case than the first that we considered in this lab.

Lab 10. Hitching Another Ride on the Future Freeway

Let's return, so to speak, to that year (2032) when you were commuting from your spacious suburban home on the outskirts of Lincoln, Nebraska, to one of the industrial suburbs of Chicago, some 200 miles away. Commuting is a breeze because the car automatically adjusts its speed to come to a "safe"cruising speed of 100 mph and following distance of, um, well, let's say 100 feet instead of 20.

In this lab, we examine the effect of a capricious other driver who changes his or her car's position at variance to the "will of the highway"—or perhaps the computer on that car is malfunctioning. In any event, if this change is periodic with amplitude A and period $2\pi/\beta$, the resulting model for $x(t)$, *how much the distance between your car and the equilibrium position of the one in front differs from 100 feet* might be given by

$$\frac{d^2x}{dt^2} + c\,\frac{dx}{dt} + k\,(x - 100) = A\sin(\beta t). \tag{1}$$

We consider this equation in the following. We will for the purposes of our examination (though not, perhaps, for the purposes of the freeway!) consider $k = 2$, with a variety of zero and non-zero values for c, *et al.*

Problem 1: First suppose that $A = 0$ (the driver in front of us forgets to be capricious, or the computer is working A-OK) and $c = 0$. Write out the equation in this case and explain what type of solutions you expect to see, with what equilibria.

Problem 2: Suppose that the initial conditions for the problem are $x(0) = 0$ (initially your car is 100 feet behind that in front of it), and $x'(0) = 1760$ (and it is moving 20 mph slower). Use *Mathematica*'s DSolve command to check your intuition (as demonstrated in problem (1)) in this case. Plot the solution you obtain for $x(t)$. Remind yourself (and your lab report) why or why not you would be happy to be a rider in this car.

Problem 3: What are the functions that appear in your solution from (2)? Now suppose that we take $A = 100$ and $\beta = \sqrt{2}$. Without solving the problem, what type of solutions would you expect to see now? (Think about both the driven and undriven part of the problem.) Solve the problem with DSolve to again check your intuition, and then plot the solution for $0 \leq t \leq 150$. What phenomenon do you see? Does this look like a reasonable model for what should go on with the computer-controlled highway?

Problem 4: Next, let $c = 1$. Again solve the differential equations and indicate how the solution you obtain is different than that you saw before. In particular, how do the functions in it differ, and what is the long-term behavior of the solution? (You may also want to plot $100\sin(\sqrt{2}t)$, the variation in the leading car's position, to check what you are seeing.) Explain why physically this long-term behavior would make sense given that $x(t)$ is the distance between your car and the equilibrium position of the car in front of it.

Problem 5: As a final "freeway" example, let $c = 0$ again and take $\beta = 1.5$. Find a solution with DSolve and explain as you did in (4) how the solution obtained is different (be sure to plot it for a large enough t range). Does it converge to a given value? What phenomenon are we witnessing in rapt amazement now? Then take $c = 1$ and check that you get what you would expect.

Problem 6: Explain what you have seen in the previous problems: What solutions (and phenomena) do you see, and what was the long-term behavior, when

i. you considered the equation $x'' + kx = 0$?
ii. you considered the equation $x'' + kx = \sin(\sqrt{k}t)$?
iii. you considered the equation $x'' + kx = \sin(bt)$ ($b \neq \sqrt{k}$)?
iv. you considered the equation $x'' + cx' + kx = \sin(bt)$?

Lab 11. More Numerical Methods and the Blasius Boundary Layer

A problem of particular interest in many different fields is that of finding the velocity of a stream of a "fluid" over a flat surface, at different distances above the surface. For airplane or ship design the fluid is air or water, which is moving over the plane or ship. Air moving over the "flat" inside surface of the lung airways is interesting in biomedical applications, and wind passing over the ocean to generate waves is of general scientific interest. In each case, the "Blasius boundary layer" equation

$$\frac{d^3 f}{d\eta^3} + \frac{1}{2} f \frac{d^2 f}{d\eta^2} = 0 \tag{1}$$

appears in some form. In this equation, η is like distance above the flat surface (its interpretation changes in different cases), and the *derivative* of f, $f'(\eta)$, is the velocity of the fluid (air or water). The initial conditions for this problem may be found to be

$$f(0) = 0, \quad f'(0) = 0, \quad \text{and} \quad f''(0) = .332058. \tag{2}$$

Problem 1: Does the boundary condition $f'(0) = 0$ make physical sense? In particular, explain what it says about the air or water at the flat surface and why you might expect this to be the case.

Of course, even as simple as it looks, equation (1) is both nonlinear and third order and does not have an analytical solution (one that may be found by hand). Therefore we must resort to numerical methods to solve it, which we do in this lab. Recall that for first-order equations such as

$$\frac{dy}{dx} = f(x, y), \quad y(0) = 1 \tag{3}$$

a numerical method generates a series of points (x_0, y_0), (x_1, y_1), and so on, approximating the actual solution to the problem by starting with a known point and estimating what the next point should be based on the differential equation. In particular, Euler's method, which is a simple (if inaccurate) numerical method, uses the equation

$$y(x + \Delta x) \approx y(x) + \Delta x\, f(x, y), \tag{4}$$

derived from a difference quotient, to approximate later values of y.

Problem 2: Let's rewrite equation (1) so that we can use a numerical method that applies to first-order equations. Let $g(\eta) = f'(\eta)$ and $h(\eta) = f''(\eta)$, and use these to write equation (1) as a system of three first-order equations for $f'(\eta)$, $g'(\eta)$, and $h'(\eta)$. (Hint: The first equation will be $f'(\eta) = g(\eta)$, and the right-hand-side of your three equations should not involve derivatives of f, g or h.) Be sure that it is clear how your equations are derived.

a. What are the initial conditions for your equations?

b. Write down what the Euler approximations corresponding to equation (4) are for the three equations that you obtain. (So you will have $h(\eta + \Delta\eta) =$ *something*, $g(\eta + \Delta\eta) =$ *something else*, *etc.*)

Problem 3: Now comes the fun stuff! Recall that when we were using Euler's method before we defined a function based on equation (4) to find the next y value $y(x + \Delta x)$ given $y(x)$—`ynext[x,y]`. Define functions in *Mathematica* that generate the Euler approximations for the next value of f, g and h, based on your results in (2)—they should depend on the previous values of f, g and h, and the step size, which we will call `del` in the following.

a. Verify that your functions work: in *Mathematica*, define `del = .1` and evaluate your functions for `fnext`, `gnext` and `hnext` by plugging in the initial conditions for f, g, and h. You should get 0, .0332058 and .332058.

Problem 4: Next we want to solve the the set of three differential equations that you found in problem 2 for $0 \le \eta \le 10$.

a. Use the Do command to generate a set of points `η[0]`, `η[1]`,... between $\eta = 0$ and $\eta = 10$ corresponding to a step size of `del=.1`—first define `η[0]=0` and then `Do[η[i] =` *something*`,{i,1,`*some number*`}]`.

b. Use the Do command to generate the corresponding values of `f[0]`, `f[1]`,..., `g[0]`, `g[1]`,..., etc. Your command should look something like the following. *Note that the form of your* `fnext`, `gnext`, *and* `hnext` *commands will depend on how you defined your functions in (3)!*

```
f[0] = 0; g[0] = 0; h[0] = .332058; del = .1;
Do[ f[i] = fnext[g[i-1]];
    g[i] = gnext[f[i-1]];
    h[i] = hnext[f[i-1],h[i-1]], {i,1,100}]
```

c. Use the Table command to generate a big `listOfpoints` with ordered $(\eta, velocity)$ pairs (remember which variable is the velocity) and then use the ListPlot command to plot the velocity as a function of η. Use the `AxesLabel` option to label your axes appropriately. Explain why (or why not) the solution you obtain is what you expect. What is the maximum velocity that you obtain? (Find this by looking that the last five velocity points that you found!)

Problem 5: Next solve the problem again using `del = .05`. Is there a large difference between the two solutions? Plot both solutions together to see how the decreased step size changes the solution that you obtain. Compare the maximum velocity with the result that you found in (4).

Problem 6: Now use *Mathematica*'s NDSolve command to solve the same system of equations. *Note: if you want use the variables f, g, and h when solving with NDSolve,* `Clear[f,g,h]` *before proceeding!* To solve the (three) first-order equations that you found for the Blasius boundary layer, we must give NDSolve a list of *six* equations to solve—each of the differential equations and the initial conditions—and tell it to solve for *three* dependent variables, `f[η]`, `g[η]` and `h[η]`:

`NDSolve[{`*list of equations and initial conditions*`}, {f[η],g[η],h[η]}, {η,0,10}]`

a. Plot the velocity that you obtain with the solutions that you got with the Euler method. (Recall that you can extract the function from a rule that *Mathematica* gives you by using something like `func = f[x] /. rulename[[1]]`.) What does this say about the accuracy of the Euler solution?

b. The solution from NDSolve should be about as good a solution as you can get. Compare the final velocity you found with that obtained with Euler—how big is the error in the Euler approximation at $\eta = 10$? If we were solving a "real-life" problem, we would likely need the solution to be accurate to at least six decimal places. Recalling that the accuracy of Euler's method goes as $1/n$ (where n is the number of steps), how many steps would be required to have an error of 0.000001?

In theory, we have just found how to solve **any** differential equation or set of differential equations numerically! This is a powerful skill! This allows us to solve (almost) any problem at all—either (if we have *Mathematica* handy) using *Mathematica*'s NDSolve, or (if we don't) by writing a program to use some method like Euler's method *(though in practice we would never use Euler's method to solve something for which we needed an accurate solution)* to solve the problem. Here we wrote the program in *Mathematica* using the Do command and the definitions of the functions that we wrote, but it could be equally well done in other ways, and we could easily use more accurate numerical methods such as Runge-Kutta. The idea would be the same—essentially the only thing that would change would be the functions `fnext`, `gnext`, and `hnext`.

Problem 7: Try solving the original problem (equation (1) with the initial conditions given by equation (2)) using NDSolve, without first breaking the problem down into first-order equations. What happens? Can you plot the velocity in this case? (Remember that *Mathematica* can take derivatives of functions.)

Plot the velocity and see if it is the same thing that you got before. Could you use ODESolve to plot the velocity? Why or why not?

a. If *Mathematica* will solve the original problem for us without rewriting it as first-order equations, we might ask why we should bother rewriting things. Give two reasons why we might want to do so anyway. (Hints: Could you solve the original problem using Euler's method? How did you find the velocity from *Mathematica*'s solution of the full problem? What do you think this will do to the accuracy of the answer? Why?)

Lab 11 Appendix

This section is just a bit of an explanation of what the Blasius equation actually says, for the benefit of anyone who wants to know a bit more about it all.

Let's first think for a moment about the solution that we found in this lab. We saw that the velocity of the fluid was zero at the flat plate, and then increased as we got higher and higher above the plate, until it reached some constant value. What would the velocity be like if the plate weren't there? Well, there would be no reason for the fluid to slow down, so it would just be a constant speed no matter where we were—a constant, or, when graphed, a straight (horizontal) line. The reason we don't have that when the plate is there is that the plate slows down the fluid that is right next to it.

So far, so good? Ok, now suppose that we moved further along the plate and looked at the fluid velocity there. Would it be the same as the picture that we saw before? Because the plate slows down the fluid that is right next to it, the velocity would have to be slower near the plate than it was before. To think about what this means, take the picture of the fluid velocity that you got in the lab and imagine grabbing the flat part (far away from the plate) and pulling to the right. This will not change the velocity far away from the plate, but will stretch out the region where the velocity increases from zero (at the plate) to one (far from the plate). The result is what the velocity looks like further along the plate.

But how does this come from the equation we solved? This is the really cunning part—the variable η that we had in the equation is actually a ratio of the distance above the plate and the distance along it! In fact, $\eta \sim y/\sqrt{x}$. So if we pick a value for x, we know what the velocity is as a function of y. Conversely, if we pick a y, we know what the velocity is as a function of x. Suppose that we picked a value of y: then as x increases, η goes *down*, and we move to the left on the graph of the velocity as a function of η. So the velocity goes down, too, which is what we said would happen when we were talking about it in the previous paragraph! This result means that the solution that we have for the fluid velocity is a very convenient one indeed: it captures both the x *and* the y behavior in one variable, η. (This type of solution is called a *similarity solution*.)

Now, where is the distinction between the air over an airplane fuselage and water over a ship's hull? You will have noted that we didn't write $\eta = y/\sqrt{x}$ in the preceding paragraph. Instead we wrote $\eta \sim y/\sqrt{x}$—η is *similar* to $y/\sqrt{x}$. Really $\eta = y/\sqrt{Rx}$, where R is a constant (called the Reynolds number) related to the fluid and flat plate that we are considering. For air flowing past a jet airplane moving at 300 mph (about half the speed of sound), R is about 100 million. For a book (probably not a differential equations book) being dragged through honey, R is around 0.5.

Lab 12. Phase Planes, or the Return of the Budworm meets the Freeway

In this lab we consider some ways that we might graphically analyze higher-order equations. One way to do this graphical analysis is with the phase plane, a graph with the state variables along its axes—if x and x' are the state variables for the problem (which is the case for second-order problems), the phase plane is then a graph of $x'(t)$ *vs.* $x(t)$. To illustrate the idea behind the phase plane, we go back for a moment to a first order differential equation—for which there is only one state variable, say $p(t)$, so that our phase plane becomes a phase line. We consider, again, the model for the population of spruce budworms in the Canadian fir forests[1],

$$\frac{dp}{dt} = r\,p\left(1-\frac{p}{k}\right) - \frac{p^2}{1+p^2}. \tag{1}$$

(Pause for a moment to remember what p is!)

Problem 1: Using *Mathematica*, plot the right hand side of equation (1) (use $r = .5$ and $k = 10$). This lets us see the "phase portrait" (graph of the phase line) for this problem. Label the vertical and horizontal axes of your graph with the `AxesLabel->{"`*xlabel*`","`*ylabel*`"}` option to indicate what variables are represented by each.

Print out the phase portrait you found and label the equilibrium (steady-state) values for the population. Explain in your lab what an equilibrium point is. Then draw arrows on the horizontal axis to the left and right of each such point showing the behavior of the population there and explain how this tells the stability of the point in question. (The resulting figure given by the horizontal axis and the arrows you've drawn is the phase line.)

Problem 2: In this case, we've sort of summed up the behavior of the system with a "state line," or "phase line": the horizontal axis of the graph that you obtained in problem (1) (with the arrows you've drawn).

Find the value of the largest steady state by using *Mathematica*'s NSolve command on the right hand side of equation (1) (NSolve is in the "Basic Calculations" palette under "▷ Algebra" "▷ Solving Equations", and takes as its arguments the equation to solve and variable to solve for). Then solve equation (1) with initial conditions above and below the largest steady state using *Mathematica*'s NDSolve (or ODESolve) command(s). Show both of these along with the steady state solution. Label the axes on your graph using the AxesLabel option on one of the plots.

Which of the axes on this graph correspond to an axis from your graph in (1)? Where would the arrows that you drew there lie on this graph? What other information besides that of the arrows does this graph have (*i.e.*, what is the other axis on your graph)?

Now that we've reconsidered the case of a one-dimensional "phase space" (Note!: a state-line is one-dimensional, because it is a line; a "phase plane" is two-dimensional; and for three and higher dimensional systems we end up with a "phase space"), let's consider a real phase plane. Recall that our model for the computer speed-and-position maintenance algorithm for the cars on the freeways of the future used the differential equation

$$\frac{d^2x}{dt^2} + c\,\frac{dx}{dt} + k\,x = 20\,k\,, \tag{2}$$

where x is the distance between one car and the car in front of it, which is traveling at a constant speed of 100mph. We defined the "standard following distance" to be 20 feet, which is why the "20" shows up on the right hand side of the equation.

Problem 3: Find the general solution to equation (2) by hand. Assume that $c^2 - 4k < 0$. After finding the general solution, substitute $c = 1$ and $k = 1$, and define it in *Mathematica*—e.g.,

[1] J.D. Murray, **Mathematical Biology**, Springer-Verlag, 1990, pp. 4–8.

```
xgen = c1*stuff + c2*other stuff + more stuff
```

Problem 4: Next we want to graph the phase plane (a graph of x' as a function of x) for this problem. In this case we first have to solve for x and x'! Explain why this wasn't the case for problem (1) but is the case here.

Use *Mathematica* to solve for c_1 and c_2 in your general solution so that the initial conditions $x(0) = 100$ and $x'(0) = 80$ are satisfied, as follows: First substitute $t = 0$ into your expression for `xgen`:

```
x0 = xgen /. t->0
```

and similarly define `xp0` to be $x'_{gen}(0)$. Then use `Solve` to find what `c1` and `c2` are, and apply the rules that it gives you to define `x` and `xprime`:

```
csoln = Solve[{x0 == 100, xp0 == 80}, {c1, c2}]
x = xgen /.  csoln[[1]]
xprime = ∂x ( xgen ) /.  csoln[[1]]
```

What was the definition of the phase plane? Just a graph of "dp/dt" vs. "p", or, here, $x'(t)$ vs. x. This is a *parametric* curve (does everyone remember their calculus?), which is parameterized by time t and is therefore graphed with *Mathematica*'s `ParametricPlot` command (in the "Basic Calculations" palette, under "▹ Graphics"):

```
phaseplane = ParametricPlot[{x, xprime}, {t,0,10}]
```

Print out your phase portrait. Label the axes, and put a dot on the graph that shows (approximately) where the initial condition is (if the exact value of the initial condition doesn't appear in the plot, put a dot as close to it as you can). Draw an arrow on the trajectory that shows what happens as t increases from zero. Explain what the position and velocity of the car are for a point at the beginning of the curve you obtained and for points in the fourth quadrant of your graph (in particular, are they increasing or decreasing?).

Problem 5: Next, suppose we start with the initial conditions $x(0) = -100$, $x'(0) = -80$ (ignoring the non-physical nature of these initial conditions!). Proceeding as you did in problem (4), add this phase plane to the graph you obtained there. (You may want to consider adding the option `PlotRange-> {{-130,130},{-80,80}}` to get all of your graphs to show up when shown together.)

What does the phase plane tell us about the behavior of the system—in particular, are there any attractors? If so, what?

Problem 6: Finally, more generally, what type of information do we get from a phase plane? (What can we say about different initial conditions? About the behavior of the position or velocity? How is this related to your last answer in the previous problem?) How is this information different than that which would be available from plots of $x(t)$ or $x'(t)$? (Hint: this is the similar as in problem (2).)

Lab 13. Of Birds and War—An Intro to Systems of Differential Equations

We began this course with the question "what is a differential equation," and the related question "what is the solution to a differential equation." In this lab, we consider in a similar vein the solutions to systems of differential equations—though, having weeks of experience in the subject already, we may have some insight into this to begin with!

So, without further adieu, let us consider the *second order*, *linear* system of *constant coefficient* differential equations for the population of a large flightless bird and the voracious predators that prey upon it. If we call the population of birds $d(t)$ and that of predators $w(t)$ (for wolves?), we might arrive at the system of equations

$$\begin{aligned} d'(t) &= \alpha d(t) - \beta w(t) \\ w'(t) &= \delta d(t) + \gamma w(t), \end{aligned} \tag{1}$$

where α, β, δ and γ are positive constants. This says that the bird population increases at a rate proportional to the number of birds present, and decreases proportional to the number of wolves—and that the number of wolves increases proportionally with the number of both birds and wolves.

Problem 1: What types of functions do we expect in the solutions to linear differential equations? Do you expect this to be the case for a system such as this? How many constants do you expect to see in the solution to the system (1)? Why?

Problem 2: Suppose that the parameters in the system (1) are $\alpha = .1$, $\beta = .2$, $\delta = .04$, and $\gamma = .3$. Use the palette button `DSolve[{■==□, □==□}, {□,□}, □]` (in the "Basic Calculations" palette, under "▹ Calculus" "▹ Differential Equations") to solve the two equations for $d(t)$ and $w(t)$ (the `{□,□}`) with independent variable t (the last `□`).

Problem 3: To better see what solutions you are getting, you may want to use *Mathematica*'s Simplify command to simplify them as you extract the solutions from the rule that DSolve gives you. If your solution does not consist of the types of functions with the number of constants that you predicted in (1), explain the differences.

a. What do you notice about the constants in the solutions that you obtain—in particular, do they appear in both $d(t)$ and $w(t)$? How are they related to the different functions in the solutions?

Problem 4: Suppose that the initial number of birds is 10 (measured in, of course, bird-population-units) and the initial number of wolves is 1 (wolf-population-unit). Is this the right number of initial conditions for the number of constants in the solutions? Find the constants by applying these initial conditions. (You may want to use *Mathematica*'s Solve command—e.g., the command `Solve[{a x[t] + b y[t] == 3, 2 a x[t] - b y[t] == 5}/.t->0, {a, b}]` solves the equations $ax(0)+by(0) = 3$, $2ax(0)-by(0) = 5$ for a and b—explain how the replacement operator `/.` works here!) Then plot the solution you've found. What does it portend for the bird population $d(t)$?

Problem 5: Solve the system (1) with parameter values $\alpha = .1$, $\beta = .2$, $\delta = .05$, and $\gamma = .3$. What functions make up the solutions now? Do they correspond with your predictions in problem (1)? Are your observations about the constants in your solution that you made in problem (3a) applicable here? Explain any differences you may see.

Next, we'll consider a different differential equation. F. W. Lanchester proposed† that armies fighting a conventional war will inflict casualties on an opposing army at a rate proportional to the number of soldiers in the army (he originally considered air combat in WWI). Thus, if x is the number of soldiers in the first army and y the number of soldiers in the second, $x(t)$ and $y(t)$ satisfy the system of differential equations

† see, for example, Giordano and Weir, *A First Course in Mathematical Modeling*, Brooks/Cole (1985)

$$\frac{dx}{dt} = -ay$$
$$\frac{dy}{dt} = -bx, \tag{2}$$

where a and b are positive constants.

Problem 6: Use *Mathematica*'s DSolve command to solve this system of differential equations. Confirm that you obtain the types of solutions that you expect, and that the constants in the solutions are behaving consistently with your observations from before. Then assume that $a = .05$ and $b = .01$[‡] and find $x(t)$ and $y(t)$ for the initial conditions $(x(0), y(0)) = (54000, 21500)$.

Problem 7: Plot the solutions that you found against time t. What outcome is predicted for the battle between the two armies? When is one army victorious? How many troops do they have left at that time? (In this model, $x(t)$ represents the number of American troops and $y(t)$ the number of Japanese—is the result consistent with history?)

Problem 8: Finally, make a general assertion about the form that you expect solutions to a second order system of linear differential equations,

$$x'(t) = ax(t) + by(t)$$
$$y'(t) = cx(t) + dy(t)$$

to have.

a. Are there any functions that you would expect to see in the solutions to the system that you haven't already seen?

b. Comment on the effect you might expect nonlinearity or variable coefficients to have on what you are able to say in response to the questions above.

[‡] As is estimated for the battle of Iwo Jima, when x is interpreted as the number of American and y the number of Japanese troops (see M. Braun, *Differential Equations and their Applications*, 2nd ed.; New York: Springer and Verlag, 1975)

Lab 14. MegaSystems—An Investigation of Nonlinear Species Competition

Suppose that we are trying to model populations x and y of two species that compete for a common resource. A simple logistic equation models exponential growth limited by available resources for each population, giving

$$\frac{dz}{dt} = \alpha z(1 - \frac{z}{\lambda})$$

(where z might be either of x or y). Adding competition means that either population will decrease as either population increases, with the decrease tending to zero as either population dies off. This might lead us to the system of equations for x and y

$$\begin{aligned} \frac{dx}{dt} =& \alpha_1 x(1 - \frac{x}{\lambda_1}) - \delta_1 xy \\ \frac{dy}{dt} =& \alpha_2 y(1 - \frac{y}{\lambda_2}) - \delta_2 xy, \end{aligned} \tag{1}$$

where the parameters $\alpha_{1,2}$, $\lambda_{1,2}$ and $\delta_{1,2}$ are determined by the nature of the species and the environment. We use this system in this lab as a vehicle to introduce the investigation of a nonlinear system of differential equations—for the purposes of this investigation, we take $\alpha_1 = \lambda_1 = \delta_2 = 2$, $\alpha_2 = \lambda_2 = 3$, and $\delta_1 = 1$.

Problem 1: Before beginning with any high powered analysis, stop for a moment and decide if we can solve the system (1) (with the indicated values for the parameters) analytically. How or why not?

a. What are the steady states of the system? Have you found solutions to the system? (Does this contradict your conclusion in (1)?)

Problem 2: Use *Mathematica*'s PlotVectorField command to draw a direction field over an appropriate range of x and y values (think about what values are reasonable before running the command—and remember to first load the PlotField package with the command `Needs["Graphics‘PlotField‘"]`, from the LocalMath palette). Note that the PlotVectorField command takes two arguments—the x and y components of the system.†

a. Based on your direction field, are the steady state solutions that you found in (1) stable? Are they attractors?

b. Also from the direction field, predict what will happen to the initial condition $(x(0), y(0)) = (0.3, 0.1)$.

What we've managed in these first problems is a qualitative analysis of some of the system's behavior, based on only the direction field (which, hypothetically, we could draw by hand, thereby obtaining at least a qualitative analytical analyis). However, as we saw in the last problem, it is not always straightforward to tell what the system is doing from just the direction field, and so we turn next to the solutions of linear systems (that we can solve analytically) to try see what the behavior of the full system is.

Problem 3: Find linear systems that model the behavior of the system (1) around each of the steady states that you found in the first problem—by linearizing the system (1) around each equilibrium point. (Recall that to do this, we let $x = x_e + u$ and $y = y_e + v$ for $|u, v| << 1$, plug in, and drop terms that are nonlinear in u and v.)

a. Solve each of these linear systems. Based on your solution, identify which of the steady states are stable and which are not.

† *Recall that in the past we've used the PlotVectorField command for first order differential equations* $P'(t) = f(t, P(t))$ *by using the command* `PlotVectorField[{1,f[t,P]},...]`. *This is the same as using* `PlotVectorField[{f[x,y],g[x,y]},...]` *to graph the direction field for a system of equations* $x'(t) = f(x(t), y(t))$, $y'(t) = g(x(t), y(t))$*—think of the one equation for* $P(t)$ *as a system of equations* $t' = 1$, $P' = f(t, P)$.

Problem 4: Identify from your solutions in (3) the eigenvectors associated with the linear system modeling the system at each steady state. Do they correspond to the directions in your direction field that you expect—to answer this, print out a copy of the direction field and carefully sketch in the eigenlines for the eigenvectors. Then add arrows along the eigenlines that show the direction trajectories will travel for each steady state. Finally, complete the local phase portraits by adding trajectories in the vicinity of each steady state.

a. Explain for two of the steady states how you knew to draw the trajectories that don't lie exactly along the eigenvectors—in particular, how are the rates of contraction or expansion given by the exponentials reflected in the trajectories that you drew?

Problem 5: Use the local analyses and direction field plot that you did in (4) to finish your description of the system (draw representative trajectories in the phase plane that show how the system will behave for different initial conditions). What will become of each of the initial conditions $(x(0), y(0)) = (0.3, 0.1)$, (0.1,0.1), and (0.1,0.3)? Describe in each case the trajectory that you expect to see (or sketch it on the graph that you printed out, being careful to label it so that it is clear which it is).

a. Solve the full system (1) (using NDSolve) with each of these initial conditions and verify by plotting them on top of the direction field plot that your predictions in (5) are correct.

b. Is it possible for an initial condition in the domain $0 \leq x \leq 1$, $0 \leq y \leq 1$ to converge to an final state other than (2,0) or (0,3)? Explain.

The curve on which a solution such as you (should have?) discussed in (4b)—which separates those initial conditions that converge to different attractors—is called a **separatrix**, for more-or-less obvious reasons. Thus we have now considered two types of lines or curves related to the solution of systems such as this—the **eigenlines** (which are parallel to the eigenvectors, and extend through the equilibrium points), and the **separatrix** (which is generally not going to be a line). The first determine the directions of the local behavior of the system near select points, and the second helps determine the global behavior of the system.

One other curve that describes the behavior of the system is a **nullcline**, which is a curve along which one of the variables in the system has a zero derivative. There is thus an x-nullcline (or nullclines), along which $x'(t) = 0$, and a y-nullcline(s), along which $y'(t) = 0$.

Problem 6: Find the x- and y-nullclines for the system (1).

a. Explain what the nullclines say about any trajectory that crosses them.

b. Show the x and y nullclines with the direction field for the system and verify that your expectation in (a) is correct (what directions do trajectories move in as they cross the different nullclines?). The nullclines in this case form triangles in the "upper-left" and "lower-right" portions of your plot. Explain by referring to the nullclines and the direction that solutions must therefore be traveling in each region what you can say about the destination of any trajectory in each region.

All of these methods give us a general qualitative picture of the behavior of a nonlinear system. This is often the best we can do without solving for a large number of initial conditions and plotting them all together to try and get a reasonable picture of the phase plane. And often the insight gained by doing the (analytical) qualitative analysis provides us with valuable insight on the behavior of the system that even repeated numerical solution cannot provide.

Problem 7: Write out the steps that you might take in obtaining a overall picture of the behavior of a nonlinear system of differential equations. Include reference to each of the different methods that we considered in this lab.

Appendix A. Mathematica Review/Tutorial

The following is not a complete overview of everything that you will want to know about *Mathematica*, but is a short review of most of the important points that will be useful to remember. Some of these are re-introduced in the course of one or more of the labs in this book. While this overview doesn't assume that you know anything about *Mathematica* (it actually starts from the very beginning), it doesn't cover anything in much depth, giving you only one or two examples of the point being made before moving on. *Mathematica* is interesting in that it is quite complex, but that there are a relatively small set of rules that one needs to know to be able to figure out most of the rest of the complexity. So the primary objective of this tutorial is to cover the basic rules in a reasonably concise manner. The rules themselves are summarized at the end of this section. So, without further discussion, let's start at...

A.1. The Very Beginning

When you start *Mathematica* without specifying a file, at least two things appear on your screen: a menu bar across the top of the screen, and a notebook window (imaginatively titled "Untitled-1"). (If you start *Mathematica* by opening a *Mathematica* file, the "Untitled-1" notebook is replaced by that which you opened.) In addition, some palettes may appear—these are push-button interfaces for common functions in *Mathematica*.

1. It's useful to have the "BasicInput" palette available, so if you don't have a little window with buttons showing common mathematical functions such as exponentials, fractions, derivatives and integrals, select `[File]` from the menu bar, slide the mouse down to `[Palettes]`, and over to `[BasicInput]`. Then click on the notebook window to be sure that the "focus" is on that window (*i.e.*, the top bar on the window is highlighted in a different color or with non-grayed-out horizontal lines).
2. Notice that there is a *horizontal black line* at the top of the notebook window. This tells you that *Mathematica* expects you to give it input, and where it will insert your input. You can get a horizontal black line by clicking anywhere in the notebook (though this only makes sense after you have some work in the notebook!).
3. Execute a simple command such as $2 + 3$: type "`2+3`" (without the quotes, of course) and then press the shift and return keys together (or press the enter key on the numeric keypad of the keyboard). The first time you run a command in *Mathematica* it will wait for a moment as it starts the *Mathematica Kernel*, which is the part of *Mathematica* that actually does all of the mathematics that you have it work out. When you are running *Mathematica* there are therefore two programs running simultaneously: the *Front End*, which controls the notebook in which you are working, and the *Kernel*, which does all of the mathematics. The kernel is also that part of *Mathematica* that remembers all of the mathematics that you've done, so if you stop and restart the kernel (*e.g.*, by stopping and restarting *Mathematica*) *Mathematica* will have forgotten all you've done in the notebook—in particular, it will have forgotten all definitions that you have made (intentionally or not!).
4. Note that many of the functions and objects that you will want to put into your *Mathematica* notebooks exist on palette buttons, which can save you much pain and agony from typing them out and then finding that you have a typographical error (*Mathematica* is very picky about spelling and use of braces, parentheses, and brackets). Practice using the palette buttons by finding $\int_0^5 (3x+5)\,dx$ and $\partial_x(\sqrt[3]{12x^2 - 15x^{-1}})$ (what are each of these finding, really?). You need to enter something for each of the boxes in the template that the palette button pastes into your notebook. If one of the boxes is dark, that is where you will start putting input if you start typing—to move between boxes either press the tab key or click on the next box with the mouse.

A.2. Parentheses, Brackets, and Braces (squiggly brackets)

Mathematica is very picky about what parentheses ("`( )`"), brackets ("[ ]") and braces ("{ }") are used for. In short: parentheses are used for grouping algebraic terms, brackets are used for functions and list access, and braces are used for lists. Let's explore this next.

1. Evaluate the expression $4(2+3)$: type "`4(2+3)`" and press shift and return. You could of course evaluate this as $4 \cdot 2 + 4 \cdot 3$: type "`4*2+4*3`" and run that. Then run "`4 2+4 3`" and check that you get the same thing—but notice that if you leave out the spaces: "`42+43`", you get a rather different answer. *The*

same is true with variables: type "`a=2; b=3; c=4;`" and run that (notice that the semicolon suppresses *Mathematica*'s output, but that you get a horizontal line back so you know that *Mathematica* is done). Then run each of "`c(a+b)`", "`c*a+c*b`", "`c a+c b`", and "`ca+cb`". Notice how the results are different. *If you leave out spaces between letters in an expression in Mathematica, it thinks that you are referring to a new variable.*

2. Now let's use some functions: evaluate $4(\sin(\pi/4)+\cos(\pi/3))$ by entering "`4(Sin[π/4]+Cos[π/3])`" (π is on the "BasicInput" palette). Notice that we use brackets to indicate the argument of the functions—and that *functions defined by Mathematica are Capitalized*—and parentheses to group the terms.
3. One more function—suppose you wanted to have the result of your last calculation multiplied out. *Mathematica* has a function called "Expand" which does just that, and its argument is the thing to expand. So type "`Expand[%]`". This says "apply the Expand function to the very last output you generated." We could get the same thing by typing "`Expand[4(Sin[π/4]+Cos[π/3])]`", because the last output that we got was from multiplying the sines and cosines.
4. Finally, let's define a list: enter "`alist={6,28,67,7,1,68}`". This has defined "alist" to be a list of the six numbers that you put in. To access an element of the list, use double brackets: *e.g.*, enter "`alist[[3]]`" and notice what you get out.
5. Lists show up in a lot of function arguments. For example, the "Plot" function in *Mathematica*: it's a function, so we know we will type "`Plot[ ]`" to use it. The arguments that go in the brackets are the function to plot, and the variable and range to plot over—so to plot $\sin(x)$ as a function of x for x between 0 and 2π, we enter "`Plot[Sin[x],{x,0,2π}]`"—the variable and range are specified in a list.

A.3. Functions

We've already seen a couple of *Mathematica*'s built-in functions, so next let's look at how we can define our own functions.

1. First, let's define the function $f(x) = x^2$. We know it's a function (think: brackets) named f, so we expect to use something like "`f[ ]`". In fact, we enter "`f[x_]:=x^2`". *Note: We use an underscore after the independent variable on the left-hand side, but not the right.* The underscore tells *Mathematica* that we will be plugging in values for x later. Test your function by evaluating "`f[3]`".
2. There are two subtleties to the definition we just made. The first is the underscore, the second is the ":=". To see what this does, try defining two functions $g(x) = x^2 + a$ and $h(x) = x^2 + a$, by entering "`g[x_]:=x^2 + a`" and "`h[x_]=x^2 + a`" (leave out the ":" before the "=" in the second one). Recall that we defined $a = 2$ earlier. Evaluate $g(3)$ and $h(3)$. Now redefine $a = 3$ and find what $g(3)$ and $h(3)$ are. *The ":=" means that if any parameters in the definition change the function changes accordingly—without it, the function is defined to have the value of the parameter (if any) that exists before defining the function.* Check this by evaluating "`g[x]`" and "`h[x]`" to see what they are.
3. Notice that we inherited the value of a that we had defined earlier. Because of this, it's a good idea to "Clear" variables that you are no longer using after you are finished with them—or, clear the variables before you start using them. Clear the values of a, b, and c with the command "`Clear[a,b,c]`".
4. We've said this before, but it bears repeating: *All of Mathematica's built-in functions are capitalized, and all functions use square brackets to delimit their arguments.*
5. Many of *Mathematica*'s build-in functions allow you to specify "Options" to modify the behavior of the function. For example, for the "Plot" function, we can specify the vertical range to plot with the "PlotRange" option. Options are entered as the last arguments of the function, and are separated by commas: *e.g.*, execute "`Plot[Sin[x],{x,0,2π}, PlotRange ->{-2,2}]`".

A.4. Loading Packages and Files

Most of the common functions and tools that you will need when running *Mathematica* are built-in, but there are a few things that are in other "packages" that come with *Mathematica*. In order to use these things, we have to run a command to load the package for the current *Mathematica* kernel—at which point the kernel has that information and you can use it until you leave and restart *Mathematica* or the kernel.

1. One common set of information that it's handy to have available are named colors. These are in a "Colors" package in *Mathematica*, which is found in a "context" called "Graphics." So, to use named

colors we first run the command "`Needs["Graphics‘Colors‘"]`". *This is on the LocalMath palette—either use the palette button there or be very careful to get the right apostrophes in the command.* Then we can plot in color—try "Plot[Sin[x],{x,0,2π, PlotStyle− >Chartreuse]". (Hey! That's another Option!)

2. What happens if you accidentally try to use a color or a function that is in a package before you load the package? In this case, *Mathematica* will either give you the message "Unknown Graphics primitive <colorname> encountered" (and plot in black—for the case where a color is being used) or just echo back the function that you typed (if you are trying to use a function). This means that *Mathematica* now "knows" that the <colorname> or function should exist—so that if you try and run the "Needs" command it will have a conflicting definition error and the Needs command will not work. To get around this, run the command "`Remove[<usedname>]`", where <usedname> is the color or function name that you used earlier. Then run the "Needs" command.†

A.5. Substitution

Many of *Mathematica*'s functions return rules, or sets of rules, that tell what values to substitute into an expression to obtain a desired result. A "rule" is anything of the form "`something -> somethingelse`", and says "substitute `somethingelse` for all occurrences of `something`." The operator to apply a rule is "`/.`"—a slash followed by a period. So, let's see how this works.

1. Did you clear your definitions of a, b, and c? If not, clear them now with the command "`Clear[a,b,c]`".
2. Now, let's apply the rule "a→3" to the expression $a + 5$: enter "`a+5 /. a -> 3`". Try entering "`a(b+c) /. {b -> 2a, c -> 3a}`" to show that you can apply a list of rules to an expression.

A.6. Iterators

We will find on occasion that it is useful to do something a large number of times without having to type it separately each of those times. There are two built-in *Mathematica* commands that make this easy: "Do" and "Table." We'll look at those briefly next.

1. Suppose that you want to define a function in *Mathematica* that is only given for integer values, *e.g.*,

$x =$	1	2	3	4	5	6	7	8
$f(x) =$	6	28	67	7	1	68	35	39

We can do this in *Mathematica* by defining a list of output values and then using the "Do" function to assign those, in turn, to the function. First, define the list of output values with the command "`outlist = {6,28,67,7,1,68,35,39}`". Next, use a "Do" command to do the assignment $f(i) = outlist[[i]]$ for $1 \le i \le 8$: enter "`Do[f[i] = outlist[[i]],{i,1,8}]`". Check that your function returns the right values for $f(3)$ and $f(5)$.

2. The "Table" command builds a list of objects, but is otherwise similar to the "Do" function. To make a list of the squares of the integers from 1 to 100, we could just start typing: "`biglist = {1, 4, 9, 16, ...}`", but that gets old fast. We could also use the "Table" command: enter "`Table[i^2,{i,1,100}]`". Much easier!

† If this doesn't work, a more drastic solution is to quit and restart the kernel—by selecting [Kernel] [Quit Kernel] [Local] and then executing another command in the notebook—but this means that *Mathematica* will forget about all of the work that you have done up to that point.

- ▷ Press [shift] [return] to evaluate a command in *Mathematica*.
- ▷ Use the Palettes under [File] [Palettes]. Clicking on a box in a pasted palette button allows entry of text. Pressing [tab] moves to the next box.
- ▷ You can enter something new in *Mathematica* when the notebook has a horizontal line across the page. To get a horizontal line, click at or below the point you want to enter text.
- ▷ `?FunctionName` tells about a function & its arguments. Also see [Help] [Find in Help].
- ▷ Be careful to **leave spaces between variables and function names, or use ***: `x Sin[x]` or `x*Sin[x]`.
- ▷ Parentheses and brackets have specific, reserved uses in *Mathematica*: `[ ]` are used for **function arguments**: `Sin[x]`. `( )` are used to **group terms**: `(3 x + 2) (5 x - 8)`. `{ }` are used to **specify lists**: `Plot[Sin[x], {x, 0, π}]`.
- ▷ *Mathematica*'s internal functions are Capitalized: `Sin[x]`.
- ▷ To apply a rule, use `/.`: `xsol = x[t]/.{x[t] -> InterpolatingFunction[t]{0,5}}`
- ▷ Change the cell style with [Format] [Style] before you start typing to enter text, subtitles, etc. If you want to change the style of a cell, click on the blue cell marker at the right margin, then [Format] [Style].
- ▷ If you save a *Mathematica* notebook and then open it again later all of your work will be there, but you'll have to re-execute the commands to be able to use their results.

Basic Input

$\partial_{\square}\blacksquare$ $\sqrt{\blacksquare}$ $\frac{\blacksquare}{\square}$ $\int \blacksquare\, d\square$

$\blacksquare_{\square}$ $\begin{pmatrix} \square & \square \\ \square & \square \end{pmatrix}$ $\int_{\square}^{\square} \blacksquare\, d\square$ $\blacksquare^{\square}$

Basic Calc

- **Graphics**
  ```
  Plot[■, {□,□,□}]
  Plot[{■,□}, {□,□,□}]
  ListPlot[■]
  ParametricPlot[{■,□},
      {□,□,□}]
  ```
- **Calculus**
  - Diff'l Eqns
    ```
    DSolve[■==□,□,□]
    DSolve[{■==□,□==□}, □},□,□]
    NDSolve[■==□,□,{□,□,□}]
    ```
  - N Root Find
    ```
    FindRoot[■==□,{□,□}]
    ```
- **Algebra**
  - Solving Eqns
    ```
    Solve[■==□,□]
    NSolve[■==□,□]
    ```
  - Polyn Manip
    ```
    Simplify[■]
    ```
  - Complex #s
    ```
    ComplexExpand[■]
    ```
- **Lists/Matr**
  - Create L&M
    ```
    Table[■,{□,□,□}]
    Table[■,{□,□,□,□}]
    ```
  - Matrix Op
    ```
    ■.□
    Det[■]
    Inverse[■]
    Transpose[■]
    LinearSolve[■,□]
    LinearSolve[■]
    ```

LocalMath

```
Do[■,{□,□,□}]
Eigensystem[■]
Needs["■‘□‘"]
Needs["Graphics‘Colors‘"]
Needs["Local‘calculus‘"]
LHSum[■,{□,□,□,□}]
RHSum[■,{□,□,□,□}]
```

```
Needs["Graphics‘PlotField‘"]
PlotVectorField[{1,■},
  {□,□,□},{□,□,□},Axes->True]
Show[■,□]
```

Appendix B. Local Mathematica Files

The labs in this book assume the existence of one nonstandard (*i.e.*, not distributed with *Mathematica*) palette, the “LocalMath” palette, with some standard and nonstandard functions. The nonstandard functions are in a local package called “diffeq.m”. This appendix describes the functions on the LocalMath palette. Because *Mathematica* is used with both calculus and higher level courses at Nebraska Wesleyan, it includes some calculus and general use buttons as well as those specifically used by differential equations in this lab book.

B.1. Functions on the LocalMath palette

The functions on the LocalMath palette are:

```
Do[■, {□,□,□}]
Do[■, {□,□,□,□}]
Needs["■‘□‘"]
Needs["Graphics‘Colors‘"]
Show[■]

Needs["Local‘calculus‘"]
LHSum[■,{□,□,□,□}]
RHSum[■,{□,□,□,□}]

Needs["Local‘diffeq‘"]
ODESolve[{■==□,□==square}, □, {□,□,□}]
ODEMSolve[■==□, {□,□,□,□}, □, {□,□,□}]
Needs["Graphics‘PlotField‘"]
PlotVectorField[{1,■}, {□,□,□}, {□,□,□}, Axes->True]

Eigensystem[■]
TraditionalForm[■]
Needs["LinearAlgebra‘MatrixManipulation‘"]
AppendRows[■]
```

The LocalMath palette file should be installed in the “SystemFiles/FrontEnd/Palettes” directory in the “Mathematica” installation directory, and may be downloaded from

`http://www.mathcs.nebrwesleyan.edu/delabs/`

B.2. Description of nonstandard functions

The nonstandard differential equations functions on the palette are found in the “diffeq.m” package file located in a directory called “Local” in the “Applications” subdirectory of the “AddOns” directory of the *Mathematica* 3.0 installation. These are described below, and the package file shown in the following section.

- `ODESolve[{`$\boxed{1}$`==`$\boxed{2}$`,`$\boxed{3}$`==`$\boxed{4}$`}, `$\boxed{5}$`, {`$\boxed{6}$`,`$\boxed{7}$`,`$\boxed{8}$`}]`: Numerically solve (with NDSolve) the differential equation(s) $\boxed{1}$==$\boxed{2}$ with initial condition(s) $\boxed{3}$==$\boxed{4}$ for the dependent variable(s) $\boxed{5}$ and independent variable $\boxed{6}$ between $\boxed{7}$ and $\boxed{8}$, and plot the solution curve. ODESolve will accept Plot options. Examples:

```
ODESolve[{y'[x]==1-y[x]^2, y[0]==3}, y[x], {x,0,5}]
ODESolve[{y''[x]==1-y[x]^2, y[0]==1, y'[0]==-1}, y[x], {x,0,5}, PlotStyle->Red]
```

- `ODEMSolve[`$\boxed{1}$`==`$\boxed{2}$`, {`$\boxed{3}$`,`$\boxed{4}$`,`$\boxed{5}$`,`$\boxed{6}$`}, `$\boxed{7}$`, {`$\boxed{8}$`,`$\boxed{9}$`,`$\boxed{10}$`}]`: Numerically solve (with NDSolve) the differential equation(s) $\boxed{1}$==$\boxed{2}$ to obtain solutions with each of the sequence of initial conditions $\boxed{3}$==$\boxed{4}$, $\boxed{3}$==$\boxed{4}$+$\boxed{6}$, $\boxed{3}$==$\boxed{4}$+2$\boxed{6}$,... $\boxed{3}$==$\boxed{5}$ for the dependent variable $\boxed{7}$ and independent variable $\boxed{8}$ between $\boxed{9}$ and $\boxed{10}$. All solution curves are shown together. ODEMSolve will also accept Plot options. Examples:

```
ODEMSolve[y'[x]==1-y[x]^2, {y[0],1,2,.2}, y[x], {x,0,3}, PlotRange->All]
ODEMSolve[{y''[x]==1-y[x]^2, y[0]==1}, {y'[0],-1,1,.2}, y[x], {x,0,5},
   PlotStyle->Table[RGBColor[1-.1*i,0,.1*i],{i,0,10}]]
```

B.3. diffeq.m package file

The diffeq.m package file may also be downloaded from
`http://www.mathcs.nebrwesleyan.edu/delabs/`

The package file is

```
(*^
*)
(* :Title:  diffeq.m *)
(* :Author:  Gavin LaRose *)
(* :Summary:  Differential Equations Functions *)
BeginPackage["Local`diffeq`"]
ODESolve::usage="ODESolve[derivs==stuff, y[x0]==otherstuff,..., y[x], x,x0,x1,
plotargs->vals] calculates the solution to the differential
equation 'derivs = stuff' with initial conditions 'y[x0] =
otherstuff,...', finding y[x] for x between x0 and x1, and plots it.
The number of initial conditions supplied must equal the order of the
differential equation.
   Example:  ODESolve[y''[x] == -y'[x] - (y[x])^2, y[0] == 1.5, y'[0] == 0,
      y[x], x,0,5, PlotStyle->RGBColor[0,0,1]];"
ODEMSolve::usage="ODEMSolve[y'[x]==f[x,y], y[x0],y0,y1,ystep, y[x],
x,x0,x1, plotopt->vals] finds the solution to the first-order
differential equation y'[x] = f[x,y] for each of the initial
conditions y[x0] = y0, y[x0] = y0 + ystep,..., y[x0] = y1.  All
solutions are obtained for x0 <= x <= x1, and are plotted.  If ystep
is omitted, it is assumed to be 1.
   Example:  ODEMSolve[y'[x] == x y[x] - (y[x])^2, y[0],0,5, y[x], x,0,4,
      PlotRange->0,6]"
Begin["`Private`"]
ODESolve[equations_,y_,xvals_,args___]:= Module[solnrule,ysoln,
   solnrule = NDSolve[equations,y,xvals,MaxSteps->100000];
   ysoln = y /.  solnrule[[1]];
   Plot[ysoln,xvals,args]
]
ODEMSolve[equation_,icvals_,y_,xvals_,args___]:=
   Module[icvar,icstart,icstop,icinc,numsoln,iceqn,i,eqns,solnrule,ysoln,
      allsolns,
      icvar = icvals[[1]];
      icstart = icvals[[2]];
      icstop = icvals[[3]];
      If[Length[icvals] > 3,
         icinc = icvals[[4]],
         icinc = 1];
      numsoln = Round[(icstop-icstart)/icinc]+1;
      Do[iceqn = icvar == icstart + (i-1)*icinc;
         eqns = Flatten[equation,iceqn];
         solnrule = NDSolve[eqns,y,xvals,MaxSteps->100000];
         ysoln[i]= y/.solnrule[[1]], i,1,numsoln];
      allsolns = Table[ysoln[i],i,1,numsoln];
      Plot[Evaluate[allsolns],xvals,args]
]
End[]
EndPackage[]
(*
^*)
```

NWU Differential Equations Lab Manual

Gavin LaRose
Nebraska Wesleyan University
Spring 2000

These lab materials were developed as part of an Instrumentation, Laboratory and Instruction (ILI) grant given to the Nebraska Wesleyan University math department in 1995 by the National Science Foundation (Grant #DUE–9551794). Most were originally written in Spring 1996, with substantial revisions and additions taking place in 1997, 1998 and 1999.