# Contents

Acknowledgments ................................................................................................................................. v

1 Lab 1 CS 121 Computation Lab I Fall 2009 Directions and Problems ......................................................... 1
1.1 Overview ........................................................................................................................................ 1
1.2 Introduction to Lab 1 (20-25 minutes) ............................................................................................ 1
1.3 Problems -- Part 1 (20 minutes) ..................................................................................................... 1
1.4 Maple TA (15 minutes) .................................................................................................................. 3
    Notes on Maple TA ......................................................................................................................... 3
1.5 Problems -- Part 2 (30 minutes) ..................................................................................................... 4
1.6 Problems -- Extra Credit ............................................................................................................... 4
    Problem -- extra credit .................................................................................................................. 4
    Problem -- extra extra credit ........................................................................................................ 5
1.7 Saving your work (5 minutes) ........................................................................................................ 5
1.8 Final actions (End of class) ........................................................................................................... 5

2 Lab 2 Cs 121 Computation Lab I Fall 2009 Directions and Problems ......................................................... 7
2.1 Lab 2 Overview .......................................................................................................................... 7
    Overview ....................................................................................................................................... 7
2.2 Instructor's demonstration of exact and limited-precision arithmetic, scripting ................................. 7
2.3 Part 1 ........................................................................................................................................... 8
    Problem 1 Description .................................................................................................................. 8
2.4 Part 2 ......................................................................................................................................... 10
    Part 2 Description ....................................................................................................................... 10
2.5 Final actions (end of class) ........................................................................................................ 11

3 Lab 3 Cs 121 Computation Lab I Fall 2009 Directions and Problems ......................................................... 13
3.1 Lab 3 Overview .......................................................................................................................... 13
    Overview ..................................................................................................................................... 13
    Directions for this lab .................................................................................................................. 13
3.2 Instructor's demonstration of textual entry of commands, use of lists and sets with plotting, and definition of functions ........................................................................................................ 13
3.3 Part 1 ....................................................................................................................................... 14
    Part 1 Description ....................................................................................................................... 14
3.4 Part 2 ....................................................................................................................................... 15
    Part 2 Description ....................................................................................................................... 15
3.5 Attachment: starter script for Part 1 ............................................................................................ 17
3.6 Final actions (end of class) ........................................................................................................ 20

4 Lab 4 Cs 121 Computation Lab I Fall 2009 Directions and Problems ......................................................... 21
4.1 Lab 4 Overview .......................................................................................................................... 21
    Overview ..................................................................................................................................... 21
    Directions for this lab .................................................................................................................. 21
4.2 Instructor's demonstration of definition of functions, advanced plotting, and animation ................. 21
4.3 Introduction to the "Human Cannonball" simulation ........................................................................ 21
4.4 Problem 1 .................................................................................................................................. 25
4.5 Problem 2 .................................................................................................................................. 25
4.6 Problem 3 .................................................................................................................................. 26
4.7 Problem 4 .................................................................................................................................. 27
4.8 Final actions (end of class) ........................................................................................................ 28
4.9 Acknowledgements .................................................................................................................... 28

5 Lab 1 CS 122 Computation Lab II Winter 2010 Directions and Problems ..................................................... 29
5.1 Lab 5 Overview .......................................................................................................................... 29
    Overview ..................................................................................................................................... 29
    Pre-lab preparation ....................................................................................................................... 29
Acknowledgments

To our colleagues and families, who supported us in trail-breaking.

To our students, who learn how to work with the new and different.
Lab 1 CS 121 Computation Lab I Fall 2009

Directions and Problems

1.1 Overview

This lab introduces the use of Maple, the primary computer language used for this course. You will learn how to do simple arithmetic calculations, as well as annotated plots. A ecology management problem is introduced that can be solved with the calculational facilities introduced.

This lab also introduces Maple TA, the primary homework/quiz/exam site for the course. You will log onto Maple TA with your personal account, and taking a practice quiz. Starting next week, there will be required/graded work on Maple TA for you to do.

1.2 Introduction to Lab 1 (20-25 minutes).

The instructor will introduce themselves and present a brief overview of course, Maple, and the lab.

The lab staff will hand out verification sheets along with paper copies of these directions. In later weeks, these directions will be posted on-line and can be read from your lab computer. The verification sheets will still be passed out, to be the permanent record of your attendance and accomplishments during the lab.

1.3 Problems -- Part 1 (20 minutes)

1. Sit down with your lab partner and if you haven't previously met, introduce yourself to them. Write both of your names down on the verification sheet in the space provided.

2. All of the partners should log onto a computer, following the demo given by the instructor in the introduction.

3. Do the calculations below. Everyone should try doing the computations on their own computer. To gain more confidence that you are getting the right answer, look at what your partners are getting. Get their help if they appear to be more successful than you. Sometimes just talking about what problems you are facing may produce useful insight towards overcoming them. If there is a problem that you can't collectively resolve, call the lab staff over and get some help.

4. You are to do all of the steps below. Some of the answers should be transcribed onto the verification sheet as indicated, for grading by the staff. Have a staff member come over to sign the verification sheet for part 1. Be prepared to show your work to the staff member, and to explain how you got your answers. This is also the opportunity to clear up any questions or uncertainties you may have even after doing the work.

5. When you complete part 1, get a staff member to verify your work before moving onto part 2.

1.a) Get Maple to calculate the sum of 2+2. Presumably you will be able to tell whether or not you got the right answer pretty easily.

b) What is exact fraction you get from adding together 1/2, 1/3, and 1/4? What about the sum of 1/2, 1/3, 1/4, 3/4, 4/3 and 5/4?

Note that if you are doing a calculation that is highly similar to a previous one, cutting and pasting can save you some effort entering the second expression.

2. Use Maple to perform the following exact calculations. To enter π, you can select the letter from the Common Symbols palette on the left hand side of the Maple window (it's a few segments below the Expression palette). Note that Maple does not regard Π as the same as π. To enter e, the base of the natural logarithm, use the e' from the expression palette, or the e from the "Common Symbols" palette. Typing "e" from the keyboard unfortunately does not produce the same result -- that kind of e Maple will regard as an symbol for an algebraic unknown like x or y.

a)
b) \[ \sin\left(\frac{\pi}{3}\right) \]

c) \[ \sqrt{\ln(e^8)} \] (You should get \( 2\sqrt{2} \).)

d) \[ \sqrt{1 + \frac{2}{5} + \frac{3}{15}} \] (You should get 2.)

e) \[ \log_{55}\left(\sum_{i=0}^{10} i\right) \] (You should get 1.)

3. A state lottery allows you to pick six numbers from the numbers from 1 to 52 to win. Maple exact arithmetic to calculate the exact odds of winning. This can be done by using the "choose" function from the expression palette: \( \binom{a}{b} \) means "the number of ways you can choose b things from a things". For example, if the lottery asked you to pick three numbers from the numbers from 1 to 6, the chances of winning would be 1 out of \( \binom{6}{3} = 20 \).

4. Calculate \( 2^{14} \). Note that \( (2^4)^4 = 8^4 = 4096 \). Why doesn't Maple give that as its answer?

5. Get Maple to reproduce this plot: \[ \log_{10}\left(\sin\left(\frac{1}{x^2 + 1}\right)\right) \rightarrow \]

6. You should use the right-click->plots->Plot Builder menu to specify things such as the plot range, the plot color, etc. Get Maple to reproduce this plot exactly, including the color and the proper horizontal and vertical ranges, and the title with the proper font size and style.
1.4 Maple TA (15 minutes)

1. The instructor will give a brief demo of how to use Maple TA, including how to log in, and how to take simple quizzes. (5 minutes)

2. Take Maple TA quiz 0. (10 minutes)

Notes on Maple TA

1. Maple TA is a quiz-administration system running separately from Blackboard Vista and Drexel One. Your userid should initially be your Drexel One userid (e.g. egk23) and the password should be your Drexel student ID number (e.g. 10096739). Note that the password is probably Drexel One password. You can change your Maple TA password after you log in.

2. The address for Maple TA will be given in class. Links to it will also appear on the class web site www.cs.drexel.edu/cs121/Fall2009 as well as the class site on Blackboard Vista, under "Maple TA".

3. After logging onto Maple TA, you need to select the correct class, and then the correct test to take. Usually your choices will be limited, but the choices may change during the term depending on the need.

4. After you have finished answering all the questions, you should hit the "Grade" button so that your score is recorded. If you don't do this this, Maple TA will record your answers but you will receive no credit for your work because your recorded score will remain at 0.

5. If you encounter any technical difficulties, you should contact the course staff by visiting the Cyber Learning Center (University Crossings 147) or on-line in the Blackboard class discussion group. If you have questions about the grading of an Maple TA assignment, you should contact your section instructor (the person listed in the schedule of courses).

6. The quiz server will only handle 150 simultaneous users and will turn away the excess, so don't wait until the last moment to take the quiz. You will be given credit for only that part of the quiz that you finish before the deadline. 7. If there is a catastrophic system failure, the deadline will be adjusted. An announcement will be made on Blackboard and the course website.
1.5 Problems -- Part 2 (30 minutes)

Complete part 1 problems if you haven't finished. Then work on part 2 of Lab. Get verification.

1. Find the exact solution to $3x + 5 = 0$.

2. Find the exact solution to $a \cdot x^2 + b \cdot x + c = 5$ (solve for $x$).

3. From Anton, Calculus, 8th edition, ch. 1 review exercises, problem 37, p. 99. A breeding group of 20 bighorn sheep is released in a protected area in Colorado. It is expected that with careful management the number of sheep, $N$, after $t$ years will be given by the formula:

$$N = \frac{220}{1 + 10 \cdot (0.83)^t}$$

and that the sheep population will be able to maintain itself without further supervision once the population reaches a size of 80.

(a) Graph $N$ versus $t$. (b) How many years must the state of Colorado maintain a program to care for the sheep? Show your work.

(c) How many bighorn sheep can the environment in the protected area support? (hint: examine the graph for large values of $t$.)

1.6 Problems -- Extra Credit

Only do these problems if you are far ahead of everyone else. You can get a little extra credit for them.

Problem -- extra credit

Do this for Lab extra credit if you have finished parts 1 and 2 with plenty of time to spare. In general, doing extra credit will bump your score for this lab beyond 100%. The final grade for the course includes the weighted sum of lab grades for this course, so by doing extra credit you will be smoothing out any deficiencies you may be incurring in other parts of the course such as the exam or the quizzes. Unless stated otherwise, extra credit is available only if done in the student's assigned lab session, and is not available in make-up labs. It may be possible to get partial extra credit even if you do not completely finish the extra credit work.

Explore a mathematical phenomenon jotting notes and observations as you go. In a fresh Maple document, show the trail of your computation and your written explanation of what you found and how you found it. Here's the problem. Consider the following sequence of expressions:

\[
(1.1) \quad \frac{1}{2}
\]

\[
(1.2) \quad \frac{1}{2 + \frac{1}{2}}
\]

\[
(1.3) \quad \frac{1}{2 + \frac{1}{2 + \frac{1}{2}}}
\]

What is the next number in the sequence?

What is the next one after that?

Can you predict what the next one after that is without computing it first? Write down your prediction and your reasons for it.
Problem -- extra extra credit

If you've finished the extra credit problem, this one may take a little longer to do. It is meant for people who already know something about recursive programming or recurrence relations -- or who are superaccurate typists. What is the 47th number in the sequence? (We aren't interested in approximations, we want the exact fraction.) If you've finished the extra credit problem in the lab, then come back in two weeks with the answer to this problem and an explanation of how you can with just a little more computer time and no more typing get almost any number in the sequence, and you will get three points extra extra credit.

You will get five points extra extra credit if you use Maple to get the 47th number, or any other number in the sequence.

1.7 Saving your work (5 minutes)

1. The instructor will demo how to save a Maple worksheet file, and how to upload the work to Blackboard (occasionally required for some labs).

2. Save your work into a .mw file. The resulting file should show up on your Desktop, although it depends on your computer's notion of current working directory. If you have problems finding the file on the Desktop and your partners can't help you, call over a staff member. After saving the file, upload a copy of the file to Blackboard so that you can refer to it later on. (Most public computers at Drexel automatically wipe out all files created during a student session after the student logs out.) Ask a lab staff member for a demo of this if they haven't done it already. You can also send yourself a copy via email as an attachment. This is good for those who want to remember how they did things, or wish to look at the worksheet again after lab. If you upload the file to Blackboard, you can download it to your home computer from there.

1.8 Final actions (End of class)

1. Before you leave, get the staff to grade, sign, and collect the verification sheet. You don't get credit for the lab unless they have a score recorded for you in a signed verification sheet that they have at the end of lab. You may leave the lab after you do this. You can get partial credit for the lab if a portion of your work is verified.

2. Final grades for the course will be curved if necessary, so don't fret excessively if you don't finish but it looks like others are in the same shape. However, you should try to learn the material you don't complete in lab so that you can pass the quizzes and be ready for the next lab. Computer work at this introductory level introduces a lot of ideas and concepts that appear pervasively in subsequent work. The plus side is that you'll probably see next time more of what you worked on this time, so you'll have another chance to practice and improve. The down side is that you can't ignore tough details and hope that they won't matter much.
2 Lab 2 Cs 121 Computation Lab I Fall 2009
Directions and Problems

2.1 Lab 2 Overview

Overview

This lab practices the development of re-usable multi-step scripts to solve a problem.

Before beginning lab work, you will also see the instructor demonstrate elements of floating point and exact arithmetic.

As explained by Chapter 3 of the course readings, the way Maple (and most other systems that use "calculator" numbers) works with floating point numbers is a bit different from what you learn in mathematics courses.

Part 1 of the lab has you apply a given script to solving various versions of a problem. Part 2 has you developing your own script and applying it. In the latter, you must do three "original" things: a) identify the parameters of the problem, b) develop a solution, and then c) apply your script to the other versions of the problem by cutting and pasting.

Directions for this lab

1. Find a lab partner. You need not use the same partner as last time. Group members should introduce themselves to each other if they haven't already met.

2. Listen to the instructor's overview of this week's new concepts and Maple features.

3. Get your copy of the verification sheet and check out which parts of the work will be verified.

4. Work on part 1 (30 minutes), and part 2 (50 minutes). We would like to see everyone end up with individual copies of the solution scripts. However, it may be more efficient this time to work as a pair at a single computer, and distribute the scripts to both partners via email or other file exchange at the end.

2.2 Instructor's demonstration of exact and limited-precision arithmetic, scripting

Limited-precision and exact numbers, arithmetic

The instructor will demonstrate:

the difference between the way exact and limited-precision numbers look.

the difference in the way Maple does arithmetic with exact numbers, and with limited-precision numbers

what happens when you exercise common commands such as solve with floating point (limited-precision) and exact numbers

Scripting and script-building

After this, the instructor will quickly review the work required in the scripting portion of the lab. It is expected that you will have read Chapters 3 and 4 of the course readings before coming to lab and are already familiar with the assignment (=) operation, the concept of parameters in a script. If you've worked through the examples given in chapter 4 of a script and how to build one from a problem description, you should find the work in the lab straightforward.
Problem 1 Description

1. Consider the following three versions of a problem:

Version 1
A package of medical supplies is dropped from a helicopter straight down by parachute into a remote area. The velocity \( v \) (in feet per second) of the package \( t \) seconds after it is released is given by the equation \( v = 24.61(1 - e^{-1.3 t}) \). (a) Graph \( v \) versus \( t \).

(b) What is the horizontal asymptote of the graph? (c) How long does it take for the package to reach 98% of its terminal velocity?

Version 2
A different package (with a different aerodynamical configuration) is dropped from a helicopter. The velocity \( v \) (in feet per second) of the package \( t \) seconds after it is released is given by the equation \( v = 27.47(1 - e^{-1.1 t}) \).

(a) Graph \( v \) versus \( t \).

(b) What is the horizontal asymptote of the graph? (c) How long does it take for the package to reach 87.5% of its terminal velocity?

Version 3
A different package (with a different aerodynamical configuration) is dropped from a helicopter. The velocity \( v \) (in feet per second) of the package \( t \) seconds after it is released is given by the equation \( v = 22.47(1 - e^{-1.47 t}) \).

(a) Graph \( v \) versus \( t \).

(b) What is the horizontal asymptote of the graph? (c) How long does it take for the package to reach 47.47% of its terminal velocity?

a) Do File -> New -> Document to get a fresh blank Maple worksheet. At the top of the document, insert the names of your group members, your lab section, your lab instructor's name, and the date/time. Then enter the following sequence of commands to solve version 1 of the problem. In some cases you'll have to figure out what you do with the right-click menu to get the effect.

For verification on this part, you should be able to identify the parameters of the problem (and explain why they, and not other variables in the script are parameters. Save this script as Lab2Part2.1.1.mw.
Enter the equation and assign it to the name eqn.

\[ eqn := v = a \left(1 - e^{b t}\right) \]

Plot this expression to better understand it. You will have to fiddle with the plot builder (not shown) in order to get the axis and title labeling set up correctly.

The horizontal asymptote is the limit as \( t \) goes to infinity of the right hand side of the equation. Don't worry too much if the mathematical notation for getting the asymptote seems unfamiliar; the important thing to note is that Maple can figure it out if you learn how to fill in the "lim" template from the expression palette.

\[ terminalVelocity := \lim_{t \to \infty} eqn \]

Set up the target equation that equates the fractional velocity to the velocity expression, and solve it numerically.

\[ fracTV = \text{fracTV} \]

\[ 24.11780000 = 24.61 - 24.61 e^{-1.3 t} \]
Evidently it takes \((1.319) = 3.009248466\) seconds to attain \(\frac{p}{100} = 98.00\%\) of terminal velocity.

b) Once you have gotten version 1 of the script running successfully, copy the script into a new document (File -> New -> Document). Change the parameter values to configure the script to solve version 2 of the problem. Execute the new document. Check that it solves version 2 of the problem (how will you do that?). Save this script as Lab2Part2.1.2.mw. (Note: there appears to be a bug in the "execute selection" operation that causes extra unwanted plots to be generated sometimes. However, it's easy enough to delete those after you get the solution.). In order to get credit for this part, you have to be able to show to the graders that all you did to the script from part a) was to edit the value of the parameters, and then executed the whole worksheet. You shouldn't need to modify any of the formulas, the solve or plot commands, etc. That is, the only lines of the script that you are permitted to alter are the initial lines establishing the values of the parameters.

c) Do it again for Version 3 of the problem. Save this script as Lab2Part2.1.3.mw.

2.4 Part 2

Part 2 Description

(From Hodge and Luck, "Using Computation Software Root Solvers", *Computers in Education Journal*, American Society for Engineering Education, No. 2, vol 18, June 2009, p. 81-92.) The normalized amplitude \(A\), of the vibration of a door panel of an automobile is found to be

\[
A = \frac{1}{\sqrt{\left(1 - \left(\frac{\omega}{\Omega_f}\right)^2\right)^2 + c^2 \left(\frac{\omega}{\Omega_f}\right)^2}}
\]

where \(c\) is a measured constant that depends on the car, \(\omega\) is the number of revolutions per second of the motor, and \(\Omega_f\) is the measured frequency of vibration of the door panel in cycles per second.

Consider the following three versions of the problem.

<table>
<thead>
<tr>
<th>Version 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>We find that for a 2009 Camaro (yellow, of course), (c = 0.15), and (\Omega_f = 20) Hz. (a) Display a reasonable graph of engine speed (in revolutions per minute) versus amplitude. (b) Calculate the speed of the engine (in rpm) for which the normalized amplitude is 2.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>We find that for a 2003 Mini Cooper, (c = 0.18), and (\Omega_f = 25) Hz. (a) Display a reasonable graph of engine speed (in revolutions per minute) versus amplitude. (b) Calculate the speed of the engine for which the normalized amplitude is 2.7.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>We find that for a 1984 Ferrari 308 (red), (c = 0.11), and (\Omega_f = 15) Hz. (a) Display a reasonable graph of engine speed (in revolutions per minute) versus amplitude. (b) Calculate the speed of the engine for which the normalized amplitude is 1.5.</td>
</tr>
</tbody>
</table>

In a fresh document enter a script similar in style to that of Problem 2.1, to solve Version 1 of this problem. name this document Lab2Part2.2.1.mw. Another thing to think through is how to get the answer in revolutions per minute rather than revolutions per second. You will need to establish the plotting limits by making some trial plots until you get a range that seems to be in the right ballpark.

Once you have that working, create two more documents that solve Versions 2 and 3 of the problem, called Lab2.Part2.2.2.mw and Lab2.Part2.2.3.mw.
Why were these particular cars chosen for the example?

2.5 Final actions (end of class)

Upload all of your work to Blackboard, or email copies to yourself and/or your partners.
3 Lab 3 Cs 121 Computation Lab I Fall 2009
Directions and Problems

3.1 Lab 3 Overview

Overview

This lab introduces more features of Maple that can be used for scripts that compute solutions to technical problems: lists, textual specification of commands to solve and plot, the use of functions including user-defined functions. It also introduces a data-fitting feature.

Before beginning lab work, you will also see the instructor demonstrate how to use these elements. The jump to textual entry of an entire Maple operation takes some getting used to, but the flexibility and power of expression is needed to do sophisticated technical problems.

Part 1 of the lab has you apply a given script to solving various versions of a problem. You then are presented with a variation of the problem and asked to modify the script to solve the variant. The original work here is to determine how to solve the variant given the solution technique presented by the script in the original version, and then

In the latter, you must do three "original" things: a) identify the parameters of the problem, b) develop a solution, and then c) apply your script to the other versions of the problem by cutting and pasting.

Directions for this lab

1. Find a lab partner. You need not use the same partner as last time. Group members should introduce themselves to each other if they haven't already met.

2. Listen to the instructor's overview of this week's new concepts and Maple features.

3. Get your copy of the verification sheet and check out which parts of the work will be verified.

4. Work on part 1 (30 minutes), and part 2 (40 minutes). We would like to see everyone end up with individual copies of the solution scripts. However, it may be more efficient this time to work as a pair at a single computer, and distribute the scripts to both partners via email or other file exchange at the end.

3.2 Instructor's demonstration of textual entry of commands, use of lists and sets with plotting, and definition of functions

Textual entry of Maple commands, lists, sets, and plots

The instructor will demonstrate:

How to invoke solve and plot with textual entry of the function, including the use of lists and sets. Ways of troubleshooting your way out of problems with textual entry will also be demonstrated.

Use of functions and defining your own functions

The instructor will review the available functions in Maple, where to read more about them, and how to define your own. Use of a user-defined function in a script will also be shown.
3.3 Part 1

Part 1 Description
From notes on Time Constants, ENGR 101 Fall 2009 (week 3)

A capacitor connected to a battery charges according to the following formula

\[ V(t) = T[i] + (T[a] - T[i]) \left( 1 - e^{-k \cdot t} \right) \]

where \( T[i] \) is the starting voltage, \( T[a] \) is the final voltage, and \( k \) is the reciprocal of the time constant. The voltage is measured in volts, the time in seconds.

Here is a problem that can be solved by a Maple script:

Problem A, Version 1
For a particular capacitor/battery set up, we find \( T[i]=63 \), \( T[a]=266 \), \( k=0.09 \).

(a) What is the value of \( \tau \), the time constant?
(b) What is the voltage after 20 seconds?
(c) What is the voltage after 40 seconds?
(d) How long does it take the voltage to reach 256 volts? Plot the voltage curve and a horizontal line whose intersection gives a visualization of when this happens
(e) What percentage of the difference in voltage has the capacitor been charged to after \( \tau \) seconds?

Directions, Part 1

1. Download the file Lab3Part1Script.mw and open it.
2. Do Edit->Execute->Worksheet and execute the worksheet. Verify that it's working correctly by comparing the output to what you believe is the correct answer.
3. Modify the worksheet to run the following version of the problem:

Problem A, Version 2
For a particular capacitor/battery set up, we find \( T[i]=37 \), \( T[a]=251 \), \( k=0.0075 \).

(a) What is the value of \( \tau \), the time constant?
(b) What is the voltage after 60 seconds?
(c) What is the voltage after 120 seconds?
(d) How long does it take the voltage to reach 220 volts? Plot the voltage curve and a horizontal line whose intersection gives a visualization of when this happens
(e) What percentage of the difference in voltage has the capacitor been charged to after \( \tau \) seconds?

After you make your changes to the worksheet's parameters, you can do Edit->Execute->Worksheet to run the whole thing again. Verify that the worksheet is still working plausibly.

4. Here is a similar but different problem. Figure out how to solve the problem, and then modify the previous script to solve this problem. You will need to do more than change the parameter values, you will have to change the instructions that occur in the script. The modifications revolve around figuring out what you want to solve for \( k \). That in turn will cause you to change which parameters the script has. Because of the great similarity between the problems, it's easier to edit a copy of your original script than to type things in all over again.
This happens all the time in programming -- once you have something successful, you run into variations that require reprogramming rather than just re-execution of a fixed script.

### Problem B, Version 1

For a particular capacitor/battery set up, we find $T[i]=37$, $T[a]=251$.

(a) We find that after 25 seconds, the voltage has risen from its initial value to 72.0 volts. What is the value of $k$?

(b) What is the value of $\tau$, the time constant?

(c) What is the voltage after 40 seconds?

(d) What is the voltage after 60 seconds?

(e) How long does it take the voltage to reach 220 volts? Plot the voltage curve and a horizontal line whose intersection gives a visualization of when this happens.

(f) What percentage of the difference in voltage has the capacitor been charged to after $\tau$ seconds?

5. Change the title/header of the script to indicate that it solves a different problem. Change individual lines of the commentary within the script where you have altered actions. Save a copy of this modified script as Lab3Part1AnswerB1.mw on the desktop. Send a copy of it to yourself and to your lab partner through email.

6. Verify that you've done an adequate job of parameterizing by using your script from (4) to solve this version of the problem.

### Problem B, Version 2

For a particular capacitor/battery set up, we find $T[i]=30$, $T[a]=274$.

(a) We find that after 30 seconds, the voltage has risen from its initial value to 101.0 volts. What is the value of $k$?

(b) What is the value of $\tau$, the time constant?

(c) What is the voltage after 40 seconds?

(d) What is the voltage after 60 seconds?

(e) How long does it take the voltage to reach 220 volts? Plot the voltage curve and a horizontal line whose intersection gives a visualization of when this happens.

(f) What percentage of the difference in voltage has the capacitor been charged to after $\tau$ seconds?

7. Save this version of the script as Lab3Part1AnswerB2.mw on the desktop. Send a copy of it to yourself and to your lab partner through email.

### 3.4 Part 2

#### Part 2 Description

Sometimes rather than plotting points of a function, we are given data points taken from measurements and want to find a function that would produce them. One additional issue is that the data is typically precisely accurate, there is experimental error in making the measurements. So we are satisfied if the function we derive is "reasonably close" to the data points rather than passing exactly through them. This is called the data fitting problem.

Typically rather than searching through all possible functions to find the best fit, we look for good candidates from a particular class of functions. One class are the linear functions: all functions $g(x) = ax + b$ for some values $a$ and $b$.

The data fitting problem becomes that of finding good values of $a$ and $b$.

There are several techniques for doing data fitting. One of the more popular is called least squares data fitting.
The data points \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\) can of course be split into two separate lists of values, \(x\) written as \([x_1, \ldots, x_n]\) and \(y\) written as \([y_1, \ldots, y_n]\).

Problem C, Version 1

(From Anton, Calculus 8th ed., p. 1007)

If a gas is cooled with its volume held constant, then it follows from the ideal gas law in physics that its pressure drops proportionally to the drop in temperature. The temperature, that, in theory, corresponds to a pressure of zero is called absolute zero. Suppose that an experiment procues the following data for pressure \(P\) versus temperature \(T\) with the volume held constants:

<table>
<thead>
<tr>
<th>(P) (kilopascals)</th>
<th>134.2</th>
<th>142.5</th>
<th>155.0</th>
<th>159.8</th>
<th>171.1</th>
<th>184.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T) (deg Celsius)</td>
<td>0</td>
<td>20.1</td>
<td>39.8</td>
<td>60.0</td>
<td>79.9</td>
<td>103.3</td>
</tr>
</tbody>
</table>

We want to find values \(a\) and \(b\) so that the line described by \(aT + b\) does a good job of representing the data. Then we will use the formula we get for \(P\) to answer some questions.

1. Create a fresh Maple session through File->New->Document mode.

2. Enter two lists. Call the first list \(pData\) and assign it the numbers found in the first row of the above table: \(pData := [134.2, 142.5, 155.0, 159.8, 171.1, 184.2]\). Similarly, create a second list and assign it to \(tData\).

3. Produce a point plot with Maple using the techniques discussed in chapter 5 of the course readings. Make the plot blue.

4. Look up the data fitting facility in maple by starting up Maple help and looking up "least squares". Find the examples given in the documentation page on CurveFitting[LeastSquares] and see one that will help you do data fitting using \(pData\), and \(tData\). Produce a formula for the line. Notes: (a) You will have to experiment in order to get things to work. Start by copying and pasting the instructions from the examples and getting them to work as advertised in your own worksheet. Then try substituting \(pData\) and \(tData\) for the values in the example.

(b) The "with(CurveFitting):" operation needs to be done before you can do any of the other lines in the examples.

(c) You don't have to use "v" as the variable in the curve fitting formula. It makes more sense to use "T".

(d) In the work to come, it helps to give the formula produced by the curve fitting a name, through assignment.

5. Plot the line you got from (d). Make the line blue.

6. Here's a trick to do a quick multi-plot that's not documented in the chapter readings. (a) copy the point plot to the bottom of the worksheet. (b) copy the formula plot. (c) click on the copy of the point plot. (d) right-click and select "paste". It works for a one-of plot but it doesn't lend itself to scripting very easily. The combined plot should show the line passing close by most of the data points. If it doesn't this is an indication that something is wrong.

7. Once you have gotten a least squares formula, answer the following questions:

(a) Based on the formula, get Maple to estimate the pressure when the temperature is 120 degrees Celsius by evaluating the expression at \(T=120\). (The eval operation is handy here).

(b) Produce an estimate for absolute zero (where pressure is zero) by solving an equation involving this formula. What is your estimate? Look up the actual value of absolute zero on the Internet and compare it with your estimate from this "virtual experiment". Include to calculated answer in a textual explanation of what you are doing, similar to the way that the target voltage was mentioned in the script for Part 1.
8. Save your worksheet for part 2 as Lab3Part2Solution.mw and mail copies of it to yourself and your lab partner. Be sure to put the names of your team on the worksheet for easy identification.

9. Re do this problem with Tools->Assistants->Curve Fitting. You will want to select "least squares" as the technique for fitting, not splines or interpolation. Be prepared to show your worksheet and the solution with the assistant to the staff for grading. Which way was easier for you to do?

3.5  Attachment: starter script for Part 1

Script for Lab 3, Part 1

CS 121 Computation Lab I

Fall 2009

This script was run by: (fill in here).

This script solves the following problem:

A capacitor connected to a battery charges according to the following formula

\[ T[i] := 63 \]

where \( T[i] \) is the starting voltage, \( T[a] \) is the final voltage, and \( k \) is the reciprocal of the time constant. The voltage is measured in volts, the time in seconds.

Problem A, Version 1

For a particular capacitor/battery set up, we find

\( T[i]=63, T[a]=266, k=0.09. \)

(a) What is the value of \( \tau \), the time constant?

(b) What is the voltage after 20 seconds?

(c) What is the voltage after 40 seconds?

(d) How long does it take the voltage to reach 256 volts? Plot the voltage curve and a horizontal line whose intersection gives a visualization of when this happens

(e) What percentage of the difference in voltage has the capacitor been charged to after \( \tau \) seconds?

Start of parameters

\[
63
\]

\[
T[a] := 266; \quad (3.1)
\]

\[
266
\]

\[
k := 0.09 \quad (3.2)
\]

\[
0.09
\]

\[
tf := 20 \quad (3.3)
\]
Define a function that describes the voltage. This is not a parameter because it doesn't change between versions of the problem.

\[ t \rightarrow T[i] + (T[a] - T[i]) \cdot (1 - \exp(-k \cdot t)) \]

Test the function at \( t=0 \) -- should be initial voltage.

\[ \tau := \frac{1}{k} \]

According to the notes, the time constant is the reciprocal of \( k \).

Since we have defined \( V \), all we have to do is evaluate \( V \) at \( t=20 \) and \( t=40 \).

To find how long it takes to reach the target voltage, we evaluate \( V \) at the symbol \( t \), and equate it to the target voltage. Solving that equation for \( t \) will give us the answer.

\[ 256 = 266 - 203 \cdot e^{-0.09 \cdot t} \]

The target voltage is reached in \( 33.45134318 \approx 1 - (1 - .632)^{10} \) seconds.
To do a plot which gives the same kind of answer, plot V and a horizontal line (a "constant function") whose value is the target voltage. The intersection of the two lines indicates where the voltage reaches the target. For the range, we pick ten times the time constant, which should be enough to see the exponential curve go into its "almost flat" phase. After all, when t is equal to ten times the time constant, V(t) should achieve \[0.9999544511 = \text{plot}([V(t), v\text{Target}], t = 0..10\cdot\tau, \text{labels} = ["time (in seconds)", "voltage"])\] of the final voltage T[a].

\[
\text{vdiff} := T[a] - T[i]
\]

\[v\text{gain} := V(\tau) - T[i]\]

\[\text{percentGain} := \frac{v\text{gain}}{\text{vdiff}}\]

End of script
3.6 Final actions (end of class)

Upload all of your work to Blackboard, or email copies to yourself and/or your partners.
4 Lab 4 Cs 121 Computation Lab I Fall 2009
Directions and Problems

4.1 Lab 4 Overview

Overview

This lab uses more sophisticated plotting and animation to explore a situation where computational simulation -- figuring out how things change over time -- can help understand and design.

Before beginning lab work, you will also see the instructor demonstrate how to use function definition, display, and animate which are key operations in the day's lab. You will be asked to write scripts using only textual versions of Maple operations. You will not receive credit for answers that use the clickable interface for operations.

The lab explores a problem of a "human cannonball" through animating motion after being shot out of a cannon.

Directions for this lab

1. Find a lab partner. You need not use the same partner as last time. Group members should introduce themselves to each other if they haven't already met.

2. Listen to the instructor's overview of this week's new concepts and Maple features.

3. Get your copy of the verification sheet and check out which parts of the work will be verified.

4. Work on Part 1 (80 minutes). We would like to see everyone end up with individual copies of the solution scripts. However, it may be more efficient this time to work as a pair at a single computer, and distribute the scripts to both partners via email or other file exchange at the end. Your work should use only textual versions of Maple operations. You will not receive credit for answers that invoke operations through the clickable interface. However, you may use the clickable interface for entering expressions (e.g. square roots) or symbolic constants (e.g. π).

4.2 Instructor's demonstration of definition of functions, advanced plotting, and animation

The instructor will demonstrate: function definition, function daisy chaining, display, paramplot, and animate. In order to prepare for this lab, you need to read Chapters 7 and 8 of the chapter text available on-line. One way to better understand the material is to reproduce the examples in the text yourself. After you have done this, you can take the pre-lab quizlet which gives you course credit for doing the lab prep.

4.3 Introduction to the "Human Cannonball" simulation

The following problem comes from the book Calculus: Early Transcendentals, 7th edition, by Howard Anton, Irl Bivens, and Stephen Davis, pages 462-465 (module created by John Rickert and Howard Anton): "Blammo the Human Cannonball will be fired from a cannon and hopes to land in a small net at the opposite end of the circus arena. Your job as Blammo's manager is to do the mathematical calculations that will allow Blammo to perform his death-defying act safely. The methods that you will use are from the field of ballistics (the study of projectile motion)."
In this problem you will compute the equations of motion for Blammo traveling in the plane and use these equations to simulate the motion of Blammo flying towards the net. The equations of motion in the plane are similar to those that were derived and used in first tutorial; however, in this case you must track both the $x$ and $y$ coordinates of the object. Prior to shooting Blammo from the cannon, you will have to specify the angle of elevation of the cannon and the initial speed of Blammo exiting the cannon. Based on these parameters, and the distance between the cannon and the net, you need to determine whether the Blammo hits the net or not. We will initially assume that there is no resistance from the air as Blammo travels and that the only force acting on Blammo is gravity, which only affects the $y$ coordinate.

Consider an elevation angle of $\alpha$ degrees and an initial speed of $V_0$. In the triangle below, the cannon is located at point A, the angle of elevation is the angle CAB and the length of the side AC is equal to the initial speed $V_0$. The initial velocity in the $x$ direction is the length of the side AB and is equal to $V_{0x} = V_0 \cos(\alpha)$. (Remember, SOHCAHTOA, $\cos\alpha = \frac{\text{adjacent}}{\text{hypotenuse}}$.) The initial velocity in the $y$ direction is the length of the side BC and is equal to $V_{0y} = V_0 \sin(\alpha)$.

The position of the cannonball is given by $(x(t), y(t))$, which provides the coordinates of the cannonball at time $t$. The equations of motion, as derived in most elementary physics texts, can be found to be

$$x(t) = V_{0x} t \quad \text{and} \quad y(t) = y_0 + V_{0y} t - \frac{1}{2} gt^2$$
where \( g = 32 \frac{ft}{sec^2} = 9.8 \frac{m}{sec^2} \), depending on the units used. \( y_0 \) is the initial position of the object (if we are launching from the ground \( y_0 = 0 \)). We will use these equations for the following problem.

Let's work in the English FPS (foot/pound/second) system of units.

If we shoot Blammo off at 100 feet per second at an angle of 45 degrees, we find that

\[ v_0 \theta = 100 \]

We can develop a plot, using the paramplot feature of Maple's plots package:

\[ \alpha := convert(45\text{-degrees}, \text{radians}) \]

\[ \frac{1}{4} \pi \]

\[ v_0x := \cos(\alpha) \cdot v_0 \]

\[ 50 \sqrt{2} \]

\[ v_0y := \sin(\alpha) \cdot v_0 \]

\[ 50 \sqrt{2} \]

\[ g := 32 \]

\[ 32 \]

\[ y_0 := 0 \]

\[ 0 \]

\[ x_0 := 0 \]

\[ 0 \]

\[ xpos := (t) \rightarrow x_0 + v_0x \cdot t \]

\[ t \rightarrow x_0 + v_0x \cdot t \]

\[ ypos := (t) \rightarrow y_0 + v_0y \cdot t - \frac{1}{2} g \cdot t^2 \]

\[ t \rightarrow y_0 + v_0y t - \frac{1}{2} g t^2 \]

\[ \text{plot}([xpos(t), ypos(t), t = 0..3], labels = ["feet","feet"], title = "Flight of Blammo") \]
We can see that after three seconds Blammo is still in mid-flight, having already reached his apex.

We can solve an equation to find out at what times $t$ Blammo is on the ground.

\[
solve(ypos(t) = 0, t)\]

\[
0, \frac{25}{8} \sqrt{2}
\]  

(4.10)

Not surprisingly, one of the times is $t=0$ (the start). We can get the second time by using the \texttt{max} function and feeding it the sequence produced by \texttt{solve}:

\[
flightTime := \max(solve(ypos(t) = 0, t))
\]

\[
\frac{25}{8} \sqrt{2}
\]  

(4.11)

\[
distance := evaf(xpos(flightTime))
\]

\[
312.5000000
\]  

(4.12)

Evidently Blammo travels 312.5 feet.

Now, if we plot from $t=0$ to flightTime, we should see the whole plot:

\[
plot([xpos(t), ypos(t), t = 0 .. flightTime], labels = ["feet", "feet"])
\]

If we want to produce an animation of Blammo flying through the air, we need to create a function that creates a plot with a shape located at $(\text{xpos}(t), \text{ypos}(t))$ for any given time $t$, and then gives it to the animate function for $t=0..\text{flight-Time}$. 
Maple has other useful functions to help us:
maximize will find the largest value that a function attains. For example,

\[ \text{evalf(maximize(ypos(t)))} \]

\[ 78.12500000 \]  

This means that Blammo reaches a maximum height of 78 1/8 feet. We could figure this out through calculus and/or remembering enough high school analytic geometry about parabolas, but Maple knows how to do these things without further programming on our part.

### 4.4 Problem 1

Suppose we shoot Blammo out of the cannon at an initial velocity of 110 feet per second, at an angle of 50 degrees. Read in the script Lab4StarterScript. Use the computational machinery to determine how high and how far Blammo travels. Play the animation to confirm that it is consistent with the numbers computed, as well as the parameter plot.

### 4.5 Problem 2

We want to have a different shape for Blammo. Consult the on-line documentation for plottools and look up the `disk` and `pieslice` functions. Choose one, and replace the point plot with a red or blue object of your choice. Assume that Blammo is roughly six feet tall.

To develop your code, begin with the starter script and change the plotting portions so that they produce a shape in the correct position rather than a point plot. Then change the function being given to animate to draw the shape rather than the point. Produce an animation for when Blammo is launched at a speed of 50 feet/second, at an angle of 35 degrees.

You may notice that the shape looks more squashed than it ought to be. This is because `animate` is not using the same scaling for the horizontal and vertical axes. To correct this, add the option `scaling=constrained`, as illustrated in the various examples in Chapter 8 of the readings.

Save your work as Lab4Problem2.mw, to show to the grader.
4.6 Problem 3

Suppose we have a tent that's 200 feet long and 50 feet high. We buy a standard explosive charge from a manufacturer that will shoot Blammo out at a velocity of 82 feet per second. Use the computational machinery to determine what angles the gun may be set up to have Blammo safely fly through the air without running into the walls of the tent.

Directions

Open up the starter script again and modify it so that it eliminates the animation but adds bounding lines to the parameter plot, so that you can see whether Blammo’s trajectory will exceed the boundaries of the tent.

To do this, create a function that uses display with some green dotted lines. Plotting this will quickly establish whether Blammo exceeds the boundaries. You don't need to produce an animation.

For example,

\[
\text{\texttt{with(\texttt{plottools}) :}}
\]

\[
p1 := \text{\texttt{line([0, 50], [200, 50], linestyle = "dash", color = "green")}}
\]

\[
\quad \text{\texttt{CURVES([\{0., 50.\}, [200., 50.\}], COLOUR(\texttt{RGB}, 0., 1.00000000, 0.), LINESTYLE(3))}} \tag{4.15}
\]

\[
p2 := \text{\texttt{line([200, 0], [200, 50], color = "green", linestyle = "dash")}}
\]

\[
\quad \text{\texttt{CURVES([\{200., 0.\}, [200., 50.\}], COLOUR(\texttt{RGB}, 0., 1.00000000, 0.), LINESTYLE(3))}} \tag{4.16}
\]

If we display them together we get a plot that looks like this:

\[
\text{\texttt{with(\texttt{plots}) :}}
\]

\[
\text{\texttt{display([p1, p2])}}
\]

\[
\text{\texttt{PLOT(...)}} \tag{4.17}
\]

What you want to do is to generate the parameter plot and assign it to p3. Then doing \texttt{display([p1,p2,p3])} should result in a picture that looks something like this.
This diagram shows quickly that the flight path goes quickly beyond the boundaries of the tent when we launch Blammo at 100 feet per second at 45 degrees. But you will have different results for 10 feet per second.

Use this as an idea to modify your existing scripts to handle this problem. When you find a range of angles that works, keep the execution from the largest angle that works. Save your work as Lab3Problem3.

**4.7 Problem 4**

We would like to add a net into our simulation, that Blammo can land in safely. Change the script from Problem 2 to add a new function \( \text{drawNet} := (d, w) \to \) a green line centered at \((d,0)\) with total width \(w\). This means that the line extends from \((d-w/2,0)\) to \((d+w/2,0)\).

Modify your animation function so that every frame displays not only the position of Blammo but also the net. For example, here is a frame of an animation we created that has Blammo (a red disk), flying through the air towards the net.
Through trial and error, find angles and initial velocities that solve the following problems. Try to get Blammo's center point to land as close as you can conveniently arrange to the center of the net.

(a) **Distance to net = 100 feet. Size of net = 10 feet.** Try 70 feet/second and 30 degrees initially. Team members should try various values of v0 and the angle to make Blammo land in the net. Be prepared to play the animation of the successful shot for the grader. Note that you can vary both the velocity and the angle, so it doesn't have to be just a patient variation of just velocity or just the angle to find a solution. Once you find a solution, see if you can find another solution with an initial velocity 10% faster, and a different angle.

(b) **Distance to net = 200 feet. Size of net = 5 feet.** Record the velocity and angle that you found that worked.

(c) **Distance to net = 500 meters. Size of net = 3 meters.** Record how many tries it took you to find a solution that worked. In order to solve this problem, you will have to figure out how to handle metric values for the distances. However you find it, at the end of the script be sure to express the initial velocity in meters/second instead of feet per second.

### 4.8 Final actions (end of class)

Upload copies all of your work to Blackboard, or email copies to yourself and/or your partners. Be sure to get credit for doing this on the verification sheet before you leave.

### 4.9 Acknowledgements

This exercise was developed with Dr. Jeremy Johnson, Dr. Fred Chapman, and Mr. Ryan Walls.
5 Lab 1 CS 122 Computation Lab II Winter 2010
Directions and Problems

5.1 Lab 5 Overview

Overview

This lab has you developing a user-defined function from a "word specification" and applying it to solve a problem. The work involved includes determining how to translate a "human-friendly" description of a problem into the syntax of Maple, entering it into Maple, and troubleshooting what you enter and figuring out how to get it to work. This exercise makes the transition to the style more usual with programming work -- most of the directions are given in English, with the programmer handling the task of figuring out how to convert the intentions into the syntax of the programming language. The programmer is not left totally to their own devices -- usually there are some parts inherited from other workers and given as "use this", and while pieces can be straightforwardly lifted and edited from tutorial examples. However, the problem-solving and mental English-to-programming language translation abilities are the key skills for programmers.

Also, we will learn how to utilize Maple's code edit region feature to segregate and execute user developed scripts.

Pre-lab preparation

1. Reading: review chapters 1-8 from cs 121. Read chapter 9 (new material).

2. Practice creating a code edit region (chapter 9), and replicate the operation of an example given in the readings, or one of your own devising.

3. Take the pre-lab quizlet 1 at the CS 122 Maple TA web site. The deadline for doing quizlet 1 will be through the end of the first lab (week 2 of the quarter). This is an exception for the typical due date for quizlets. The pre-lab quizlets for Labs 2-4 will be available only in the week before the lab.

Directions for this lab

1. Form a lab group of two or three people. You are not allowed to work on your own without obtaining prior permission from the lab instructor. Group members should introduce themselves to each other if they haven't already met.

2. Listen to the instructor's overview of this week's new concepts and Maple features.

3. Get your copy of the verification sheet and check out which parts of the work will be verified.

4. Work on Part 1 (20 minutes). We would like to see everyone end up with individual copies of the solution scripts. However, it may be more efficient this time to work as a pair at a single computer, and distribute the scripts to both partners via email or other file exchange at the end. Your work should use only textual versions of Maple operations. Avoid using the clickable interface to perform calculations. You will not receive credit for answers that invoke operations such as solve through the clickable interface. However, you may use the clickable interface for entering expressions (e.g. square roots) or symbolic constants (e.g. π).

5. Work on Part 2 (20 minutes).

6. Work on Part 3 (40 minutes)
5.2 Part 1 -- function design (20 minutes)

This is an exercise to practice your ability to design and implement a user-defined function from a specification.

Design and implement a user-defined function that, when given two plot structures $p1$, and $p2$, a string $t$, and a list of strings $labels$, displays the two plots together with the specified title $t$, and labels. Give this function a mnemonic name that suggests what it does and test it.

For example, if had a Danish boss and she told you to call your function $toGraf$, then the following script would draw a single plot that displays two graphs together.

```maple
plotOne := plot(sin(x),x=0..10);
plotTwo := plot([1,2,3],[1,5,6], color="DodgerBlue");
toGraf(plotOne,PlotTwo, "Two plots", ["time", "temperature (in Celsius)"]);"
```

Of course, if you were doing this for your own satisfaction, or to satisfy the graders for this course, you wouldn't use a Danish mnemonic (even if you are Danish -- but most of the people reading your code aren't). In order to do Part 1, you should review the material on user-defined functions and the $display$ function in chapters 7 and 8 of the course readings from last term. You should discover how to get the $display$ function to display a title and labels through a combination of reading the on-line documentation and experimentation.

5.3 Part 2 -- working with code regions (20 minutes)

The Human Cannonball (Blammo) problem treated in Lab 4 of CS 121 used a mathematical model that did not take into account air resistance. In Part 3, we will use a more realistic mathematical model that takes air resistance into account, and compare the results of the two models. To prepare for this exercise, Part to will familiarize us with Maple's code edit region feature, which will facilitate the assignment in Part 3.

In the original model, the functions for $x$ and $y$ position were given by:

$$\begin{align*}
    xpos &:= (t) \rightarrow x0 + v0 \cdot \cos(\theta) \cdot t \\
    ypos &:= (t) \rightarrow y0 + v0 \cdot \sin(\theta) \cdot t - \frac{1}{2} \cdot g \cdot t^2
\end{align*}$$

where $t$ is time (in seconds), $v0$ is the initial velocity, $\theta$ the initial angle of the cannon (in radians), and $x0$, and $y0$ the initial horizontal and vertical position. and $g$ is the Earth's gravitational constant (32 feet/sec$^2$, or 9.8 meters/sec$^2$).

(a) Download and open the worksheet CS122Lab1Starter.mw It contains a script for running a version of the original "no air resistance" model of Blammo's flight. Note that the script is an a code edit region and it has comments and print statements. Execute the script (see Chapter 9). Observe how the results are printed out both as a result of executing an instruction, or through a print/printf instruction.

(b) Edit the code region to rerun this starter script with an initial position of (0,0), $\theta=35$ degrees, and $v0= 40$ meters/second. Your computation and results should be displayed in metric units, so you should figure out how to adjust the computation to do this. In the sample output below, we have suppressed most of the execution trace by changing the semi-colons to colons.

<table>
<thead>
<tr>
<th>Code region output with most output suppressed with colons, initial firing angle of 45 degrees and firing velocity of 40 meters per second.</th>
</tr>
</thead>
<tbody>
<tr>
<td>45 degrees converts to 0.785398 radians.</td>
</tr>
</tbody>
</table>
With an initial velocity of 40.000000 meters/sec and firing angle of 0.785398 radians, Blammo lands 1.632653e+02 meters away, reaching a maximum height of 40.816327 meters.

\[
\begin{align*}
10 \\
\quad t = x_0 + v_0 t \\
\quad t = y_0 + v_0 t - \frac{1}{2} g t^2 \\
\quad 20 \sqrt{2} \\
\quad 20 \sqrt{2}
\end{align*}
\]

If you run into trouble making these changes, part of your work will be to practice your trouble shooting skills. If there are Error messages, see if they match one of those discussed in Chapter 9 (or earlier chapters). If they match, you should try to fix them yourself. If there are no error messages but the results are not correct, employ the "comment trick" explained in the troubleshooting sections of Chapter 9 of the readings to identify which line is the first one that causes problems. If you can't figure out anything to do after two minutes, then ask for help. This "think for a few minutes before seeking help" will boost a crucial skill in troubleshooting -- your ability to connect symptoms with causes, and your development of strategies that work for you in finding the cause of problems.

(c) In the example output, we have used colons to suppress much of the execution trace, leaving only the information we really want to see from the correct program. Try to do this yourself after you are certain that things are working correctly.
5.4 Part 3 -- A new model, comparing two models (40 minutes)

In a model that takes wind resistance into account, the functions for $x$ and $y$ position can be described by:

Horizontal position is given by \[ x(t) = \frac{m v_0 x}{b} \left( -1 + e^{\frac{bt}{m}} \right) \]

Vertical position is given by \[ y(t) = -\frac{mg}{b^2} - \frac{b}{m} vt + \frac{b}{m} v_0 \] 

where in addition to the symbols described previously, we have $m$ the mass of Blammo (60 kg), and $b=10$, the constant due to wind resistance. 

Note that the original model that we worked with in Part 1 and the new model are talking about the same situation -- shooting Blammo out of a cannon. However, the formulae in the new model are more complicated, indicating that they are handling details (the effect of wind resistance) that were ignored in the formulae for the first model. Since all models choose to leave some details of reality out of the mathematical description, engineers and scientists often then do computations comparing the results of the models to understand whether the extra mathematics is producing better or less accurate results, or whether the two produce results that are about the same. While we don't have a "human cannonball lab" to measure what actually happens when people are fired out of a cannon, we can compare the results of the two models through computation.

(a) First, let's get the new model working. Start with a copy of what you did in Part 2. Modify the code edit region so that it uses the new formulae, and plots the trajectory for the new model. Plots with wind resistance should look something like this:

![Blammo's flight with wind resistance](image)

(b) Take the code outline file CS122Lab1Starter2.mw. It contains an outline for a script that combines the computation of the trajectories under the original and wind resistance models, and compares the result. The net result will be a plot of both trajectories. Proceed to develop the outline incrementally, as described in Chapter 9 of the readings. If you don't proceed incrementally, and enter the code for the entire computation before testing any piece of it, you will spend...
a long time tracking down where the mistakes are occurring. The final output of your script should be a plot that has both trajectories plotted together, similar to the following:

Comparison of trajectories: launch angle = 45 degrees, $v_0 = 40.000000$ meters/sec

(c) Repeat the analysis for a variety of weights (other than 60 kg) for Blammo. Describe the impact of his weight on the overall trajectory when wind resistance is taken into account.
6 Lab 2 CS 122 Computation Lab II Winter 2010
Directions and Problems

6.1 Overview

There are two parts to this lab. Part 1 has you creating a series of user-defined functions to draw things. User-defined functions provide a way to provide short-cuts to typing in Maple commands that can be customized to the particular use by giving different values for the parameters. Daisy-chaining function together also provides another way to develop code incrementally. Pieces can be tested and debugged individually and then connected together, like Lego.

Part 2 is our first experience with iteration -- getting a computation of many steps performed by getting the computer to repeat a short segment of code. Rather than being strictly repetitive, the code is written so that it does something different each repetition, leading to an interesting cumulative result. The code you will be adding to uses for and while to control the repetition.

6.2 Pre-lab preparation

1. Reading: chapters 10 and 11 (new material). Review older chapters and labs as needed.

2. Practice creating loops (chapter 11) in a code edit region, getting a few of the examples to work on your own computer. See whether you understand how to modify them to do slightly different things.

3. Take the pre-lab quizlet 2 at the CS 122 Maple TA web site. The deadline for doing quizlet 2 will be before the start of lab week, as was the case for most quizlets in CS 121.

6.3 Problems -- Part 1 (45 minutes)

Problem 1.1

A function for drawing a red box

We are going to develop a function that draws a box of any specified size and location. We do so incrementally rather than doing all the coding at once, so we can test each piece at a time.

a) Open Lab2StarterPart1-1.mw, which should be contained among the downloadable files for this lab. It contains a code region that contains part of the code for drawing a box whose right hand corner is at (0,0). Read about the line function in the plottools package to figure out how this function works. Then complete the code region to draw a box that looks like this.
Once you understand the basic actions that draw a box, we can move onto incorporating this into a function.

b) After you have been successful at drawing the square, add where indicated the definition a function called drawBoxA. After it is defined, drawBoxA can be invoked to draw a red box whose height and length are specified as numbers provided as arguments to the function. Here's a brief recapitulation of how to design and implement a function: 1. First describe and name the inputs (also called parameters) to the function. You can give them arbitrary names, but typically names suggestive of the purpose of the inputs is chosen. For example, for drawBoxA, there should be two inputs. We could name them height and width, or h and w, depending on our bent. These names do not have to do anything with any variables that we are using in our session. They are names of placeholders. For example, in mathematical functions, they may describe a function as "f(x)", but there is no expectation that the only way to use the function is to use x. You could talk about f(5) or f(a), or f(y+1). The initial "x" is just the placeholder name you are giving for first input to f. 2. Next describe the result or output of the function based on the inputs. For drawBoxA, you could say that this function should produce as a result "a plot structure that is a box whose left bottom corner is at (0,0) and has width w and height h". 3. The next step is to write the code that produces the result, using the names for the placeholders you have decided on. The coding of the function definition in Maple is always given in the general form: function name := ( argument, argument , ... argument) -> expression involving the argument. For the case of drawBoxA the function definition could look like this: drawBoxA := (height, length) -> display([ line(...) , line(...) , line(...) , line(...) ]) The expression to the left of the arrow does create a plot structure, using display's ability to take four plot structures (the lines) in a list and draw them all together. We expect to see the parameters height and length to show up in the expressions for the individual lines. If we chose different names for the parameters:

drawBoxA := (h,l) -> display([ line(...) , line(...) , line(...) , line(...) ])

then we would expect the innards of the line expressions to include mention of h and l in the appropriate way.

Either way should define the same function that does exactly the same thing when it is used. The function definition does not make any displaying occur right away. That only happens when the function is invoked by writing an expression that provides actual values for the arguments. For example, if we had already entered the code to define drawBoxB, then on a later line we could enter

drawBoxA(5, 6) and we would expect a plot structure to be created that was that of a red box with height 5 and length 6. After you have defined drawBoxA, uncomment the first test of drawBoxA at the bottom of the code region and re-execute the region. In addition to the original box, you should now see another, smaller, box drawn.

c) Uncomment more of the tests of drawBoxA and see that they also work as they should. If not, then fix your problems.
d) Create another function `drawBoxB` that takes five arguments -- the length, width, the \((x,y)\) coordinates of the bottom left hand corner and a string that describes the color (see Maple's on-line help for `colornames` for a complete list). Uncomment the first test for `drawBoxB` and get it to work, then run the rest of the tests.

e) Once you have convinced yourself that your definition of `drawBoxB` works, write more code that uses `drawBoxB` and `display` to draw the following pictures. Using `drawBoxB` to draw all the boxes should be a lot more convenient than copying a lot of code that includes multiple lines.

**Art project 1**

![Art project 1](image1)

**Art project 2**

![Art project 2](image2)

**Art project 3**

![Art project 3](image3)
In the next lab we'll take this a little further by simulating a particle bouncing around in the box.

### 6.4 Problems -- Part 2 (45 minutes)

A chemical reaction involves four chemicals, A, X, Y, and B. B is the product, A is an initial "ingredient", X is a catalyst, and Y is an intermediate result. The reaction rates are moles/second.

<table>
<thead>
<tr>
<th>Reaction step</th>
<th>Reaction</th>
<th>Contribution to reaction</th>
</tr>
</thead>
</table>
| 1             | $A + X \rightarrow 2X$ | $\frac{d[A]}{dt} = -k_1 \cdot [A] \cdot [X]$  
$\frac{d[X]}{dt} = k_1 \cdot [A] \cdot [X]$ |
| 2             | $X + Y \rightarrow 2Y$ | $\frac{d[X]}{dt} = -k_2 \cdot [X] \cdot [Y]$  
$\frac{d[Y]}{dt} = k_2 \cdot [X] \cdot [Y]$ |
| 3             | $Y \rightarrow B$ | $\frac{d[Y]}{dt} = -k_3 \cdot [Y]$  
$\frac{d[B]}{dt} = k_3 \cdot [Y]$ |

We can approximate what happens in a process driven by this reaction through a computer script. To set things up, we do initialization that

a) Defines initial concentrations of the four chemicals.
b) Gives values for the constants \( k_1 \), \( k_2 \), and \( k_3 \). Typically, we would find values for the constants by looking them up in a handbook or by determining it through experimentation and observation in the Chem lab.

The simulation would then establish four variables \( A, X, Y, \) and \( B \) that are initialized to the initial concentrations. Then it would conduct a loop of \( n \) time steps. Each time step would establish the most recent values of the four concentrations as an add-on to the previous values, using the rules:

\[
\begin{align*}
\text{new } A &= \text{previous } A - k_1 \times \text{previous } A \times \text{previous } X \\
\text{new } X &= \text{previous } X + k_1 \times \text{previous } A \times \text{previous } X - k_2 \times \text{previous } X \times \text{previous } Y \\
\text{new } Y &= \text{previous } Y + k_2 \times \text{previous } X \times \text{previous } Y - k_3 \times \text{previous } Y \\
\text{new } B &= \text{previous } B + k_3 \times \text{previous } Y
\end{align*}
\]

Note that we need eight variables \( A, \text{newA}, X, \text{newX}, Y, \text{newY}, B, \) and \( \text{newB} \) because we need the previous values around while we do a full round of computations of the new values.

**Problem 2.1**

Open *Problem2-1Starter.mw*. Execute the script.

Complete the script so that it successfully updates \( A, X, Y, \) and \( B \) and plots the concentration of \( A \). Once you have things working, change the number of time steps so that \( n = 50 \) rather than 10. You should see something like this:

![Graph showing concentration over time](image)

**Problem 2.2**
Extend the script so that it simultaneously plots the concentrations of A, X, Y, and B. What eventually happens? chemical eventually has the highest concentration? How many simulation steps does it take for this to become true? To do this, you should create tables for X, Y, and B (you could call them xTab, yTab, and bTab) and plot them together with the plot for A. Recall that the display function takes a list of plots, so what is needed is for you to fill out the list that currently has only the plot of aTab, with similar plots for the other chemicals.

### 6.5 Final actions (end of class)

Upload copies all of your work to Blackboard, or email copies to yourself and/or your partners. Be sure to get credit for doing this on the verification sheet before you leave. If you cannot complete the work in the lab period, talk to the instructor before you leave about whether you can get credit for anything beyond what you finished.
7 Lab 3 CS 122 Computation Lab II Winter 2010
Directions and Problems

7.1 Overview

There are two parts to this lab which gives you experience in conditional execution, where different statements are executed depending on what is true at any particular point in time. Some of you who have taken Engineering 102 already have seen conditional execution in programming the robots, where the value of sensor input is used to conditionally cause the robot to turn one way or the other, or to stop and back up.

Part 1 revisits Blammo, asking you to use a while loop to automatically find the angle to fire him at in order to hit a target. This will give you practice at looping under conditional control. Part 2 asks you to build a simulation of a particle rolling around a box. This is our first experience with conditional branching -- "if statements". Each time step of the simulation involves movement of the particle according to its present velocity and position. Occasionally the particle hits the wall of the box, which causes it to rebound, changing the direction and possibly the speed. This happens under control of a while loop that has if statements in it to handle the possibilities of different cases of bouncing or no bouncing.

7.2 Pre-lab preparation

1. Reading: chapters 12 and 13 (new material). Review older chapters and labs as needed. Note that this lab expects you to remember your experience with the Blammo code last used in Lab 1/CS122 and Lab 4/CS121, and to have a copy handy so that you can re-use it.

2. Practice creating while loops (chapter 12) in a code edit region, getting a few of the examples to work on your own computer. See whether you understand how to modify them to do slightly different things. Gain a similar practical understanding of if statements (Chapter 13) in a similar way.

3. Take the pre-lab quizlet 3 at the CS 122 Maple TA web site. The deadline for doing quizlet 3 will be before the start of lab week.

7.3 Problems -- Part 1 (45 minutes)

In this part, we see how to modify our Blammo code so that it can answer the following question: Blammo is shot out of his cannon at 40 m/sec. What angle should the cannon be aimed at in order for him to land approximately 60 meters away in a net that's 5 meters wide?

**Problem 1.1**

First, we'll practice with for loops. We assume that Blammo is launched at 40 meters per sec and use the wind resistance model to calculate where he lands. As before, we know that his mass is 60 kg and the wind resistance constant $b=10$.

What we want to do in this problem is to print a table of distances as we vary the angle from 30 degrees to 70 degrees. We want the script output to look like this:

```
Angle = 30 degrees, time= 3.703112 seconds, distance = 95.721287, max height = 16.710341
Angle = 31 degrees, time= 3.804913 seconds, distance = 96.609279, max height = 17.636593
Angle = 32 degrees, time= 3.905204 seconds, distance = 97.370883, max height = 18.573115
Angle = 33 degrees, time= 4.003975 seconds, distance = 98.008304, max height = 19.518697
Angle = 34 degrees, time= 4.101217 seconds, distance = 98.523747, max height = 20.472140
Angle = 35 degrees, time= 4.196919 seconds, distance = 98.919422, max height = 21.432256
Angle = 36 degrees, time= 4.291074 seconds, distance = 99.197540, max height = 22.397871
Angle = 37 degrees, time= 4.383671 seconds, distance = 99.360313, max height = 23.367824
Angle = 38 degrees, time= 4.474703 seconds, distance = 99.409952, max height = 24.340967
```

etc. etc.
Upload the Blammo code in CS122Lab3Part11Starter.mw. This is slightly modified from what you did in the previous lab, to take into account model corrections and other improvements.

We intend to do produce the table by modifying the Blammo calculation code using the following code outline:

```plaintext
#initialize parameters for calculation, except for angle
for angle from 30 to 70 do
  #calculate distance
  #print a line of the table
end do;
```

Execute the code region, being careful that you are getting the expected results in the execution trace. You might do incremental development by first getting the "new" code to execute without giving errors, and then adding in the portion of the old code that does initialization, getting that to work, and then adding in the contents of the loop body.

Once you get the script to print out the table, you should be able to answer the question posed at the beginning of the problem.

**Problem 1.2**

Now rather than trying all the angles from 30 to 70, we'll try to find the smallest angle that will allow us to hit a target approximately 90 meters away. We develop another two parameters and change the looping control:

```plaintext
targetDistance := 90;
tol := 2.5; #Since Blammo is six feet tall, if he lands within 2.5 feet of the target he'll be okay.
...
for angle from 30 to 75 while condition do
  ...
end do;
```

You should invent a condition should stop the repetition when the distance computed is within $tol$ meters of the target-Distance.

One issue that the condition will mention distanceWind, but before you execute the loop once, distanceWind will not have a value yet. The computer won't be able to decide if the condition is true or not and you will get an error Error, cannot determine if this expression is true or false: ..... You can fix this by initializing distanceWind with arbitrary value of infinity in the code before the loop. This will allow the loop to work correctly the first time. After that distanceWind will be given a properly computed value, so the loop will work correctly the second and subsequent times.

**Problem 1.3**

Once you have your code set up, answer the following question:

Suppose Blammo is being launched at a speed of 90 meters per second. We want to Blammo to land in a net 250 meters away. Find an angle that, taking wind resistance into account, will allow Blammo to hit the target. If there is more than one solution, find them all and explain which one is the best "show biz solution".

Feel free to alter your script so that it makes it easier to solve this problem.
Problem 1.4

We can create a movie of the trajectories by drawing a plot each trip through the table, and putting it into a table. At the end of the loop, we can convert the table into a list and use display to make it into a movie, that looks like this:

7.4 Problems -- Part 2 (45 minutes)

In this part we are going to use while and for loops to run an simulation of a particle moving in a box. The approach taken is that we will give you model code that does one of the simple animations. You will then modify the code to do similar but different things.

Problem 2.1

Creating a moving particle inside the box, with a bounce

Open the file CS122Lab3Part2Starter1.mw. Read the code. At one point there is an empty spot for the drawBoxB function that you built in Lab 2. Paste this into the code region and execute it. You will see an execution trace, and at the end an animation (movie). Run the animation and see what happens.

a) Change the script so that it does not print during execution by putting a colon instead of a semi-colon at the end of the loop block.

b) Change the limit of the number of time steps to be 50 rather than 20. You should see the particle move a bit further. What happens if you set the number of time steps to be 200?

c) Save a copy of your for this problem as CS122Lab3Problem2-1.

Problem 2.2

Save another copy of your work as CS122Lab3Problem2-2. We will start altering this copy to handle problem 2.2.

Your task in this problem is to extend the script to make it handle a bounce off of both the eastern and western walls. At the particle velocity set in the script, this should happen no later than 150 time steps.

In order to handle the western bounce, you should work out and understand the math used in the formula being used to calculate the location of the particle after the bounce after the eastern wall. Here is an explanation of the eastern wall bounce:
If the particle is within the box at \((x, y)\) and would move to \((x + \Delta x, y)\), it will hit the wall if \(x < WID \leq x + \Delta x\). Since \((x + \Delta x, y)\) is outside the box, it would bounce at \((WID, y)\). If a bounce occurs with "perfect rebound" as we will assume here, the \(x\)-velocity is reversed and becomes \(-\Delta x\). The \(y\) velocity stays the same (and would do so for any bounce into the eastern wall even if the \(y\) velocity were non-zero).

Figure 1 Particle hitting eastern wall

<table>
<thead>
<tr>
<th>Eastern wall running from (WID,0) to (WID,LEN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Particle would have travelled to ((x + \Delta x, y)) if there were no wall. This is a distance of (x + \Delta x - WID) beyond the wall.</td>
</tr>
<tr>
<td>Starting point of particle is ((x, y)).</td>
</tr>
<tr>
<td>Particle hits wall at ((WID, y)).</td>
</tr>
</tbody>
</table>

This means that the travel back from the wall during the time step will be the amount beyond \((WID, y)\) that the particle would have traveled, but in the opposite direction. Thus during a time step that has a bounce off the eastern wall we would break things into three phases: a) The particle travels between \(x\) and \(WID\). b) The particle hits the wall, causing the \(x\) velocity to reverse itself. c) The particle bounces and travels a distance \((x + \Delta x - WID)\) more away from the wall, but in the western direction. This is the same amount as it would have traveled during the remainder of the time period if the wall had not been there. This means that the final location of the particle is \((WID - (x + \Delta x - WID), y)\) = \((2 \cdot WID - x - \Delta x, y)\). That is, the \(x\)-coordinate of the location of the particle at the end of the time step when the bounce occurred is \(2 \cdot WID - x - \Delta x\).

Figure 2 Particle rebound from eastern wall
Your job is to figure out the analogous formula for the bounce off the western wall (writing on a whiteboard and getting your teammates to agree is a good idea here -- it may be hard to keep it all in your head). Then alter the script so that instead of

\[
\text{if (delx} > 0 \text{ and ptpos[1]} \geq \text{ WID) } \# \text{bounce East}
\]

\[
\text{then}
\]

\[
xwallPos := \text{WID};
\]

\[
\text{ptpos} := [2*\text{xwallPos} - \text{ptpos[1]}, \text{ptpos[2]}];
\]

\[
delix := -\text{delix};
\]

\[
dely := \text{dely};
\]

\[
\text{end if;}
\]

it has, as another case, the test the bounce into the western wall.

\[
\text{if (delx} > 0 \text{ and ptpos[1]} \geq \text{ WID) } \# \text{bounce East}
\]

\[
\text{then}
\]

\[
xwallPos := \text{WID};
\]

\[
\text{ptpos} := [2*\text{xwallPos} - \text{ptpos[1]}, \text{ptpos[2]}];
\]

\[
delix := -\text{delix};
\]

\[
dely := \text{dely};
\]

\[
\text{elif condition } \# \text{bounce West}
\]

\[
\text{then}
\]

\[
\text{updating xwallPos, ptpos, delx, dely for a western wall bounce}
\]
To test this combined proposition, you can change the initial conditions in the script to test the western bounce first. Set
\[
\text{pos0} := [1, 1];
\]
\[
\text{delx} := -0.1;
\]
\[
\text{dely} := 0.1;
\]
and run for 50 time steps. You should see the particle move westwards and bounce off the western wall, then stop.

Once you have that working, set the script to run for 150 time steps. You should see the particle bounce off of both walls. If you want to play, set it for 250 time steps and you should see multiple bounces.

Your code will be inspected for this problem. You will be expected to following standard indentation and commenting style for the code that you write.

We are following the "incremental development" approach, in that you have code that has a "bounce of eastern wall" working and you are trying to extend it to handle another case.

**Problem 2.3**

The next step is to extend your answer in Problem 2.2 to handle north-south bounces as well. You should test this by setting
\[
\text{pos0} := [1, 9];
\]
\[
\text{delx} := 0.0;
\]
\[
\text{dely} := 0.1;
\]
and running the script for 150 time steps. To extend the script, you will have to figure out the bounce formulas for when the particle hits north or south. You can extend the script by using the following idea for a code outline for that portion of the code:

```
# check for bounce east/west and update ptpos, delx, dely if a bounce occurs (include existing if statement)
# check for bounce north/south and update ptpos, delx, dely if a bounce occurs (add an analogous if statement)
```

**Problem 2.4**

We now get some pay off for constructing the script script using parameters LEN, WID, delx, and dely instead of just using numbers in the code. We can change the scenario just by changing the lines that define the values for these parameters, rather than trying to hunt down every 10 and 0.1, and 0.0 and editing them.

Take your script from problem 2.2 and change the following lines:
\[
\text{pos0} := [8, 9]; \# initial position of point.
\]
\[
\text{ptpos} := \text{pos0}; \# initialize ptpos
\]
\[
\text{delx} := 0.1;
\]
\[
\text{dely} := 0.1;
\]
(The second line doesn't change but we include it because the other lines do.).

Rerun the script and you should see the particle moving in a diagonal direction and bouncing off of walls.
LEN := 10:

as well as

pos0 := [1,1]; # initial position of point.

ptpos := pos0; #initialize ptpos

delx := 0.2;

dely := 0.1;

Experiment with other values of the parameters and show the grader something else interesting of your creation.

7.5 Final actions (end of class)

Upload copies all of your work to Blackboard, or email copies to yourself and/or your partners. Be sure to get credit for doing this on the verification sheet before you leave. If you cannot complete the work in the lab period, talk to the instructor before you leave about whether you can get credit for anything beyond what you finished.
8  Lab 4 CS 122 Computation Lab II Winter 2010

Directions and Problems

8.1 Overview

There are two parts to this lab which gives you experience in conditional execution, where different statements are executed depending on what is true at any particular point in time. Some of you who have taken Engineering 102 already have seen conditional execution in programming the robots, where the value of sensor input is used to conditionally cause the robot to turn one way or the other, or to stop and back up. Part 1 asks you to build a simulation of a particle rolling around a box. This is our first experience with conditional branching -- “if statements”. Each time step of the simulation involves movement of the particle according to its present velocity and position. Occasionally the particle hits the wall of the box, which causes it to rebound, changing the direction and possibly the speed. This happens under control of a while loop that has if statements in it to handle the possibilities of different cases of bouncing or no bouncing.

Part 2 asks you to build a simple simulation from specifications and a modest code outline. You will be expected to supply the rest.

8.2 Pre-lab preparation

1. Reading: chapters 13 and 14. Review older chapters and labs as needed. Note that this lab expects you to remember your experience with the box and particle code in Labs 2 and 3 as well as the Blammo code of Lab 4 of CS 121.

2. Take the pre-lab quizlet 4 at the CS 122 Maple TA web site. The deadline for doing quizlet 4 will be before the start of lab week.

3. Become familiar with the code development steps of Parts 1 and 2 of this lab. To do this, you should identify what you can use from chapter readings, quiz problems, and prior labs. It is unlikely that you can finish coding if you begin from a standing start at the lab -- bring your ideas and prototypes ahead of time.

8.3 Problems -- Part 1 (45 minutes)

In this part we are going to use while and for loops to run an simulation of a particle moving in a box. The approach taken is that we will give you model code that does one of the simple animations. You will then modify the code to do similar but different things.

Problem 1.1

Retrieve a copy of your work from Part 2 of Lab 3. We will build on it for this part.

To start, create a new copy of this as CS122Lab4Problem1-1. We will start altering this copy to handle problem 1.2.

Your task in this problem is to extend the script to make it handle a bounce off of both the eastern and western walls. At the particle velocity set in the script, this should happen no later than 150 time steps.

In order to handle the western bounce, you should work out and understand the math used in the formula being used to calculate the location of the particle after the bounce after the eastern wall. Here is an explanation of the eastern wall bounce:

If the particle is within the box at \((x, y)\) and would move to \((x + \Delta x, y)\), it will hit the wall if \(x < WID \leq x + \Delta x\). Since \((x + \Delta x, y)\) is outside the box, it would bounce at \((WID, y)\). If a bounce occurs with “perfect rebound” as we will assume here, the \(x\)-velocity is reversed and becomes \(-\Delta x\). The \(y\) velocity stays the same (and would do so for any bounce into the eastern wall even if the \(y\) velocity were non-zero).
This means that the travel back from the wall during the time step will be the amount beyond \((WID, y)\) that the particle would have traveled, but in the opposite direction. Thus during a time step that has a bounce off the eastern wall we would break things into three phases: a) The particle travels between \(x\) and \(WID\). b) The particle hits the wall, causing the \(x\) velocity to reverse itself. c) The particle bounces and travels a distance \((x + \Delta x - WID)\) more away from the wall, but in the western direction. This is the same amount as it would have traveled during the remainder of the time period if the wall had not been there. This means that the final location of the particle is \((WID - (x + \Delta x - WID), y)) = (2 \cdot WID - x - \Delta x, y)\). That is, the \(x\)-coordinate of the location of the particle at the end of the time step when the bounce occurred is \(2 \cdot WID - x - \Delta x\).
Your job is to figure out the analogous formula for the bounce off the western wall (writing on a whiteboard and getting your teammates to agree is a good idea here -- it may be hard to keep it all in your head). Then alter the script so that instead of

```cpp
if (delx>0 and ptpos[1]>= WID) #bounce East
then
  xwallPos := WID;
  delx := -delx;
  dely := dely;
end if;
```

it has, as another case, the test the bounce into the western wall.

```cpp
if (delx>0 and ptpos[1]>= WID) #bounce East
then
  xwallPos := WID;
  delx := -delx;
  dely := dely;
elseif condition #bounce West
```
then

updating xwallPos, ptpos, delx, dely for a western wall bounce

end if;

To test this combined proposition, you can change the initial conditions in the script to test the western bounce first. Set

pos0 := \([1,1]\);
delx := -0.1;
dely := 0.1;

and run for 50 time steps. You should see the particle move westwards and bounce off the western wall, then stop. Once you have that working, set the script to run for 150 time steps. You should see the particle bounce off of both walls. If you want to play, set it for 250 time steps and you should see multiple bounces.

Your code will be inspected for this problem. You will be expected to following standard indentation and commenting style for the code that you write.

We are following the "incremental development" approach, in that you have code that has a "bounce of eastern wall" working and you are trying to extend it to handle another case.

**Problem 1.2**

The next step is to extend your answer in Problem 1.1 to handle north-south bounces as well. You should test this by setting

pos0 := \([1,9]\);
delx := 0.0;
dely := 0.1;

and running the script for 150 time steps. To extend the script, you will have to figure out the bounce formulas for when the particle hits north or south. You can extend the script by using the following idea for a code outline for that portion of the code:

#check for bounce east/west and update ptpos, delx, dely if a bounce occurs (include existing if statement) #check for bounce north/south and update ptpos, delx, dely if a bounce occurs (add an analogous if statement)

**Problem 1.3**

We now get some pay off for constructing the script script using parameters LEN, WID, delx, and dely instead of just using numbers in the code. We can change the scenario just by changing the lines that define the values for these parameters, rather than trying to hunt down every 10 and 0.1, and 0.0 and editing them.

Take your script from problem 2.2 and change the following lines:

pos0 := \([8, 9]\); # initial position of point.

ptpos := pos0; #initialize ptpos
delx := 0.1;
dely := 0.1;

(The second line doesn't change but we include it because the other lines do.).
Rerun the script and you should see the particle moving in a diagonal direction and bouncing off of walls.

\[
\text{WID} := 2;
\]
\[
\text{LEN} := 10:
\]
as well as
\[
\text{pos0} := [1,1]; \# \text{initial position of point.}
\]
\[
\text{ptpos} := \text{pos0}; \# \text{initialize ptpos}
\]
\[
\text{delx} := 0.2;
\]
\[
\text{dely} := 0.1;
\]

Experiment with other values of the parameters and show the grader something else interesting of your creation.

### 8.4 Problems -- Part 2 (45 minutes)

We launch a rubber ball up in the air, at a velocity of 100 m/sec and an angle of 45 degrees. Each time the bubble hits the ground, it rebounds upwards with a velocity that is only a fraction of the downwards velocity. In addition, the horizontal motion slows down each time due to friction between the ball and the ground.

We want to create a time-step simulation similar to that of part 1. The x and y positions will be updated in a loop. While the x velocity is constant (as it was in the particle-in-a-box simulation), the y velocity will be changing all the time due to the influence of gravity. Nevertheless, the overall structure of the program will look similar to that of part 1.

#### Problem 2.1

Recall that the Blammo model in Lab 2 without air resistance, we used the following functions for position and velocity:

\[
\text{velx}(t) = v0x
\]
\[
\text{vely}(t) = v0y - g \cdot t
\]
\[
\text{xpos}(t) = x0 + t \cdot \text{velx}(t)
\]
\[
\text{ypos}(t) = y0 + v0y \cdot t - \frac{g \cdot t^2}{2}
\]

From this, we calculate
\[
\theta = \text{angle in radians}
\]
\[
v0x = v0 \cdot \cos(\theta)
\]
\[
v0y = v0 \cdot \sin(\theta)
\]

We will update the movement of the ball in small time steps, in the way we did with Part 1. This contrasts to what we did with Blammo, where we had a formula for the entire movement and just evaluated it for various values of \( t \).

First, we will just get a few time steps to work with this way of doing the simulation. In later parts of this problem, we will add in bouncing and the collection of summary statistics.

The simulation revolves around the values of the following variables:
dt: the amount of time between steps of the simulation. We will set it to .1 for this Part, although after you get the program to work you can change this value and see how that affects things. In a calculus mentality, making the value of dt smaller and smaller should produce a better and better approximation to real-life, where changes in velocity are instantaneous and constantly happening. In our simulation, the changes in velocity occur only every dt seconds.

We use the following variables to keep track of the position and velocity of the ball.

- $x_p$: $x$ position of Blammo at the current time
- $y_p$: $y$ position of Blammo at the current time
- $x_v$: horizontal velocity of Blammo at the current time
- $y_v$: vertical velocity of Blammo at the current time

Each time step, we do the following

1. Store current values of $x_p$ and $y_p$ in the tables $xpos[i]$ and $ypos[i]$.
2. Calculate new values for $x_p$, $x_v$, $y_p$, and $y_v$. Store these into $newxp$, $newxv$, $newyp$, and $newyv$ respectively.

These values can be calculated as:

- New $x$ position is $x_p + dt \times x_v$ (present position plus the horizontal velocity times the amount of time of the time step)
  \[
  newxp := x_p + dt \times x_v;
  \]
- New $y$ position is approximately $y_p + dt \times y_v$
  \[
  newyp := y_p + dt \times y_v;
  \]
- New $x$ velocity is $x_v$ (no change unless there's a bounce)
  \[
  newxv := x_v;
  \]
- New $y$ velocity is $y_v - g \times dt$
  \[
  newyv := y_v - g \times dt;
  \]

Open up Lab4Part2-1Starter.mw, which has most of this code already written. Fill in the parameters so that the simulation runs ten time steps, and then stops. You should see a plot that looks like the beginning of a Blammo-like trajectory.
Once you have this working, make it run for 100 or 200 time steps. You should see the trajectory go back down, but keep on going even after it reaches the ground. In the next part, we will but more features into the simulation to get the bounces to happen.

Save a copy of your work for this part as Lab4Part2-1.mw

**Problem 2.2**

Now we will detect the bubble hitting the ground and calculate the rebound.

For a bouncing model, we will use two new parameters,

\[ R = \text{the coefficient of restitution} \]  
that describes the ratio of the rebound speed to the collision speed. We will take \( R = 0.6 \).

\[ \eta = \text{the coefficient of friction} \]  
that describes the ratio of the pre-impact horizontal speed, to the post-impact horizontal speed. Basically, the forward motion "erodes" a bit with each impact. We will take \( \eta = 0.5 \). (The Greek letter \( \eta \) is pronounced "nu", by the way.)

Save a fresh copy of your work from 2.1 as Lab4Part2-2.mw and then modify it as follows:

a) In the initialization section of the simulation, assign \( R \) and \( \eta \) the values described in the model description of part 1 if you haven't already written this code.  
b) We will now install the bounce code, which will be contained within an if statement inside the loop, just as we did in Part 1.

Right after you compute newxp, newyp, newxv, and newyv in the simulation loop, check to see if newyp is at or below ground level. If it is, then do the following: change newyv to be \(-R\times\text{newyv}\). This will cause the velocity to switch directions, but to be a factor of \( R \) less. Don't forget the minus sign or else the velocity will not reverse in direction!  
change newxv to be \( \eta\times\text{newxv} \). This will cause the horizontal velocity to slow due to friction, but to continue in the same direction as before. There is no minus sign here.

change newyv to be \(-\text{newyv}\). In other words, the altitude achieved at the end of the time step is the same as the bubble would have traveled under ground level if there had been no rebound.

This is not exactly correct (challenge: what is the mathematically correct adjustment?) but should be close enough to produce realistic results with small values of \( dt \).

When you run the simulation, you should now see a bounce occur. If you run it long enough, you should see several bounces.

Save a copy of your work.

**Problem 2.3**

Now save a copy of your work as LabPart2-3.mw and make further changes:

a) Introduce an accumulation variable totalTime. Initialize it to zero before the simulation loop, then add \( dt \) to it each trip through the loop. Use it to print out the total number of steps for the bouncing, along with the total horizontal distance traveled (approximately). The print statement needs to occur only after the loop is finished, as summary data, but when debugging your code you may want to put print statements inside the loop to check that the updates to the accumulation variable are happening correctly. You can then comment out or delete the debugging prints when you are satisfied that things are working.

b) Introduce an additional variable numBounces. Set it up so that it is initialized to zero and then incremented every time a bounce occurs. Print out the number of bounces after the simulation is over.  
c) Modify your loop control so that the simulation runs until a specified number of bounces (say, 6) occurs regardless of how many time steps it takes. The grader will grade this segment by telling you how many bounces they want to see.
d) In addition to a plot, make a movie of the bouncing happening as happened in the original Blammo work. You should use `display(listOfFrames, insequence=true, scaling=constrained)` rather than the `animate` function to do this.

Save your work.

8.5 Final actions (end of class)

Upload copies all of your work to Blackboard, or email copies to yourself and/or your partners. Be sure to get credit for doing this on the verification sheet before you leave. If you cannot complete the work in the lab period, talk to the instructor before you leave about whether you can get credit for anything beyond what you finished.