Laboratory Exercises for Scripting and Programming for Modeling and Simulation
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.
1 Lab 1CS 121 Computation Lab IFall 2011 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 interactive way of working with a computer is a skill that transfers to any number of similar technical computation systems, such as Mathematica, Matlab, or MathCAD.

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 and graded work on Maple TA for you to do.

1.2 Instructor-led 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 Part 1 -- overview (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.

1.4 Part 1 -- problems

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 $\frac{1}{2}$, $\frac{1}{3}$, and $\frac{1}{4}$ ? What about the sum of $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, $\frac{3}{2}$, $\frac{4}{3}$ and $\frac{5}{4}$ ? 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 $\pi$, 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 $\pi$
as the same as \( \pi \). To enter \( e \), the base of the natural logarithm, use the \( e^x \) 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 a symbol for an algebraic unknown like \( x \) or \( y \).

\[
\frac{1}{2} + \frac{1}{3} \cdot \frac{47}{2 \cdot 3 \cdot 5 \cdot 13}
\]

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

b) \( \sqrt{\ln(e^{1024})} \) (You should get 32.)

d) \( \sqrt{1 + \frac{2}{5} + \frac{3}{15 \choose 13}} \) (You should get 2.)

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

3. Powerball is a lottery that requires the player to choose six numbers. Here are the rules, as given by the Powerball site:

Powerball® is a combined large jackpot game and a cash game. Every Wednesday and Saturday night at 10:59 p.m. Eastern Time, we draw five white balls out of a drum with 59 balls and one red ball out of a drum with 39 red balls. The jackpot ... is won by matching all five white balls in any order and the red Powerball ... (http://www.powerball.com/powerball/pb_howtoplay.asp)

Determine the odds of winning the jackpot. In order to figure out the odds of winning at powerball, you need to multiply the odds of choosing the five numbers correctly, as well as choosing the powerball number correctly. For example, if the chances of picking the five numbers correctly were one in 10,000, and the chances of picking the powerball number one in 39, then the chances of doing both of them would be \( 10000 \cdot 39 = 390000 \).

You can use the "choose" function from the Maple 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 \( 2^{2^4} \). Note that \( (2^3)^4 = 8^4 = 4096 \). Why doesn't Maple give that as its answer?

5. Get Maple to reproduce this plot.

\[
\log_{10} \left( \left| \sin \left( \frac{1}{x^2 + 1} \right) \right| \right)
\]
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, the line dots and dashes, the proper horizontal and vertical ranges and labels, and correct title and caption. In order to receive full credit, you will need to everything letter-perfect.

\[(x - 1) \cdot (x - 2)^2 \cdot (x - 5) \cdot e^{-\frac{x}{10}} \rightarrow\]
1.5 Instructor-led introduction to Maple TA (20 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). Let the staff know if you have trouble logging in. 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/Fall2011 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. This will change over the year as circumstances shift.

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. After you have gotten your work graded, then you can hit the "Save and Quit" button to exit the quiz.

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. If this happens, an announcement will be made on Blackboard and the course website.

1.6 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 \(3 \cdot x + 5 = 0\).

\[3 \cdot x + 5 = 0 \quad \text{solve} \quad \begin{bmatrix} x = -\frac{5}{3} \end{bmatrix}\]

2. Find the exact solution to \(3 \cdot x^2 + 24 \cdot x + c^2 = 5\) (solve for \(x\)).

\[3 \cdot x^2 + 24 \cdot x + c^2 = 5 \quad \text{solve for } x \quad \begin{bmatrix} x = -4 + \frac{\sqrt{159 - 3c^2}}{3} \end{bmatrix} \quad \begin{bmatrix} x = -4 - \frac{\sqrt{159 - 3c^2}}{3} \end{bmatrix}\]

3. Given your answer to 2, determine values of \(c\) that make the solution for \(x\) a real number, not a complex quantity. This means that the solution for \(x\) won't involve any imaginary numbers. (Hint: experiment with using solve on an inequality. We haven't told you about this, but like many things in Maple, what should work often does. You can find inequality symbols under the Common Symbols palette. See if you can also figure out how to enter inequalities from the keyboard!)

\[159 - 3c^2 \geq 0 \quad \text{solve for } c \quad [c \leq \sqrt{53}, -\sqrt{53} \leq c]\]

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 \( t \).)

1.7 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 your 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. 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 future work won't depend on them in essential ways. Come with your questions to the Cyber Learning Center (University Crossings 147) next week during office hours and talk to the consultants there.
2 Lab 2 Cs 121 Computation Lab I Fall 2011 Directions and Problems

2.1 Lab 2 Overview

Overview

This lab practices the development of re-usable multi-step scripts to solve a problem. It assumes that you have read and understood the material in Chapters 2 and 3 of the course readings: "Getting started with Maple's Document Mode", and "Technical word processing".

Before beginning lab work, you will also see the instructor demonstrate the word processing features of Maple worksheets. You will also see a demonstration of how to assign names to parameters, and how to execute a whole script as a single block through Edit → Execute → Worksheet or Selection.

Part 1 of the lab has you apply a given script to solving several versions of a problem. You will find that constructing the first version of the script is the labor-intensive portion of the work. Solving the second and third versions of the problem should be very quick, since it involves only copying and a little editing of the numerical values used for the parameters of the problem.

Part 2 has you developing your own script and applying it. You must do three "original" things: a) figure out how to create a script that solves one version of the problem, b) identify the parameters of the problem, c) edit the script to use the parameters (if you haven't done so already), and d) apply your script to the other versions of the problem by cutting and pasting.

To develop the script for Part 2, you will need to come up with a recipe for how to get the solution as a sequence of Maple operations involving assignment, solving, plotting, and taking limits as well as work through the standard difficulties of getting the proper information and instructions into the computer. Script development is something where you should expect to succeed after trial, experimentation, and troubleshooting.

General directions for this lab

1. Form a lab team of two or three members. You should all sit on the same side of your work table. 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 overview 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.

5. Once you have finished your work, save a copy of your work (email it to yourself, or something similar), and get the staff to sign off on the verification sheet. Having the work verified means that you can demonstrate to the staff how to solve the lab problems and can get the work done. In general, showing up at the start of lab with a completed copy of the lab work will not result in verification, since that doesn't demonstrate that your team can do the work.

2.2 Instructor's demonstration of word processing, scripting

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 through 5 of the course readings before coming to lab and are already familiar with the assignment operation (what you get through "assign to name" in the clickable menu, or by entering := from the keyboard), the concept of parameters in a script. If you've worked through the examples given in chapter 5 of a script and how to build one from a problem description, you should find the work in the lab straightforward.
2.3 Part 1

In this problem, we ask you to create a worksheet to solve a version of a falling body problem. The problem gives you a formula relating elapsed time to velocity. From this formula, you can calculate information such as the terminal velocity achieved by the body, and the amount of time it takes to achieve a certain percentage of terminal velocity. The Maple worksheet you will write will set up the formula and then perform the calculations needed to provide the desired information.

Problem Description

1. We want to solve 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 $v = 24.61 \left( 1 - e^{-1.3t} \right)$.

(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 aerodynamic 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

$v = 27.47 \left( 1 - e^{-1.1t} \right)$.

(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 aerodynamic 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

$v = 22.47 \left( 1 - e^{-1.47t} \right)$.

(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 47.3% 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. This portion of the work is "type it in and make sure that you get the same effects as the demo example shows".

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.

<table>
<thead>
<tr>
<th>Lab 2, Problem 2.1, Version 1 Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS 121</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>(Insert Group info, section info, date info here.)</td>
</tr>
<tr>
<td>----</td>
</tr>
</tbody>
</table>

**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 \( v = 24.61 \left( 1 - e^{-1.3 t} \right) \).

(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?

**Solution**

Define the basic relationship between time and velocity. We use parameters \( a \) and \( b \) to represent the coefficients in the equation.

\[
24.61 \quad \xrightarrow{\text{assign to a name}} \quad a \quad 24.61
\]

\[
-1.3 \quad \xrightarrow{\text{assign to a name}} \quad b \quad -1.3
\]

We use the parameter \( p \) to represent the percentage of terminal velocity that we want to hit.

\[
.98 \quad \xrightarrow{\text{assign to a name}} \quad p \quad 0.98
\]

Enter the expression for velocity and assign it the name velocity

\[
a \cdot \left( 1 - e^{b \cdot t} \right) \quad \xrightarrow{\text{assign to a name}} \quad \text{velocity}
\]

Plot this expression to better understand it. We do this by entering the name of the expression for velocity, and then right-clicking on the result to order up a plot. Generate the plot through right-click \( \rightarrow \) Plots \( \rightarrow \) Plot builder so that you specify the axes labels, plot range, etc. After the plot has been generated you can change/fix any settings by right-clicking on the plot and operating the pop-up menu, which works similar to but slightly differently from the Plot Builder.
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.

$$\lim_{t \to \infty} velocity = 24.61000000$$

Assign to a name

$$terminalVelocity$$

$p$-$terminalVelocity$

$$24.11780000$$

Assign to a name

$$fractionTerminalVelocity$$

Set up the target equation that equates the fractional velocity to the velocity expression, and solve it numerically. The latter "solve" is done using the right-click $\to$ Solve $\to$ Numerically Solve.
\[ \text{fractionTerminalVelocity} = \text{velocity} \]

\[ 24.1178000 = 24.61 - 24.61 e^{-1.3 t} \quad (2.9) \]

\[ \text{solve} \]

\[ 3.009248466 \quad (2.10) \]

This value \( (2.10) \) is how many seconds it takes the falling body to attain \( p \cdot 100 = 98.00 \% \) of terminal velocity.

b) Once you have the script for Version 1 working, save the worksheet from part a) as \textit{yourNameLab2Part1-1.mw} (recall from last week that this is by doing \textbf{Save As} from the \textbf{File} menu). Then save another copy of the work under a different name, by saving the worksheet as \textit{yourNameLab2Part1-2.mw}. This allows you to do new work while retaining a copy of the original work.

Now change the parameter values in your worksheet to configure the script to solve version 2 of the problem. Edit the other textual information in the worksheet to reflect the second version.

Execute the new version. Check that it solves version 2 of the problem (how will you do that?). Save this second version in \textit{yourNameLab2Part1-2.mw}.

In order to get credit for this part, you have to be able to show to the graders that all you did to the Version 1 script from part a) was to edit the value of the parameters, and then executed the whole worksheet. You shouldn't need to retype any of the formulas, the \textit{solve} or \textit{plot} commands, etc.

c) Send your version of the result for part b) to one of your partners. They in turn should send you a copy of their file. Since this part depends on all team members being at more or less the same place, team members should help each other out to get synchronized at this point.

Once you receive your partner's worksheet, open it up on your machine and edit it so that it will solve Version 3 of the problem. Highlight the sections that you changed or fixed in red. You may need to fix the commentary as well as the code if you find spelling or grammatical mistakes, inaccuracies, or unfinished sections.

Save this script as \textit{yourNameLab2Part1-3.mw}. You can mail back the altered version to the partner you got Part1-2 from, so that they can see what changes you felt you needed to make.

\section*{2.4 Part 2}

\textbf{Part 2 Description}


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.

**Version 1**

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 "rpm", or revolutions per minute) versus amplitude.
(b) Calculate the speed of the engine (in rpm) for which the normalized amplitude is 2.

**Version 2**

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.

**Version 3**

We find that for a 1974 Mercury Marquis (black), \( c = 0.11 \), 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.

In a fresh document enter a script similar in style to that of Problem 1, to solve Version 1 of this problem. Be sure to include your name and the names of your other group members in the worksheet that you create.

Use the := operation to handle assignment of parameters in this part. You may use the "assign to a name" operation from the clickable menu to do assignment to non-parametric results, though.

One thing that you will have to think through before you start typing and clicking is how to handle the requirement that the information you is given and wanted in revolutions per minute when the formula you are given is using revolutions per second.

Another thing you will need to work on is how to establish the plotting ranges. This will not be done automatically for you since the software is not sophisticated to know what portion of the graph you'd find interesting to look at. (In other words, they haven't invented mind-reading computers yet.) We suggest experimenting with ranges until you find something satisfactory.

Save your Version 1 as myNameLab2Part2-1.mw.

Make a copy of your working script in myNameLab2Part2-2.mw and edit it to handle Version 2 of the problem. You should find that the work involved to convert the script to handle Version 2 of the problem is by editing the values of the parameters. Execute the script and check that it solves Version 2 correctly.

When you have Version 2 working, send a copy of that file to one of your lab partners, and get a copy of their version of the script from them in return. Save this as myNameLab2Part2-3.mw, and edit it to handle Version 3. Highlight the sections that you changed.
or fixed in red. You may need to fix the commentary as well as the code if you find spelling or grammatical mistakes, inaccuracies, or unfinished sections. You can mail back the altered version to the partner you got Part2-2 from, so that they can see what changes you felt you needed to make.

We are told that the normal operation of car engines leads them to operate in the 1500-2000 rpm range while cruising. Do you think that the drivers of these three cars will be satisfied with the vibration properties of their cars?

Do you know what TV shows or movies were these cars seen in? (No Maple, doesn't have a button which will answer that.)

### 2.5 Final actions (end of class)

Upload all of your work to Blackboard, or email copies to yourself and/or your partners. Before you leave, make sure that the staff has signed and accepted the verification sheet for your group so that your work is properly credited.

### 2.6 Concluding remarks

In this lab, you have learned how to create scripts that combine commentary and a sequence of computational actions. Scripting allows you to easily solve second and third versions of a problem once you have done the hard work of creating the script by studying how to the first version of the problem.

Because programming is a relatively expensive activity in terms of time, reusing someone else's work is normal activity in computing. Scripts need to be written in a way that makes it easy for someone else to use it and to modify it in modest ways. Of course there will be other times where you'll have the responsibility of creating something completely on your own instead of reusing someone else's work. Just keep in mind that you're writing not only for yourself but potentially for others.
3 Lab 3 Cs 121 Computation Lab I Fall 2011 Directions and Problems

3.1 Lab 3 Overview

Overview

This lab provides more practice with scripts. It assumes that you have read and understood the material in Chapters 4 through 7 of the course readings: "Assignment", "Building Scripts", "More Sophisticated Scripting" and "Using and Defining Functions".

This time you develop scripts using the "textual style" for operation entry where everything is specified from the keyboard, rather than the "calculator style" where expressions are created by filling in slots from Palette options and operations are selected by the mouse. Being fluent in this style helps prepare the way for doing "real programming" where you specify a block of operations to be done all together in a batch. The jump to textual entry of an entire Maple operation takes some getting used to, but the greater power of expression allows you to do even more sophisticated technical problems.

Maple extends the style of mathematical functions -- f(x), g( 3, 5), etc. -- to specify not only mathematics, but also computations. Thus solve and plot are written textually as functions solve(...), and plot(...). Becoming proficient in reading and writing computer instructions in this style is a major step in learning how to program that transfers across the many different programming languages that use this style.

To add to the functional frame of mind, we also explore how you can create and name your own functions in Maple.

This lab has also introduces you to non-numeric data structures: lists and strings. Having strings allows you to compute with text rather than just with numbers. Lists are a way of collecting and accessing aggregations of results.

Before beginning lab work, you will also see the instructor demonstrate how to use these elements. This should be a recapitulation and time to clarify your own explorations with these features before lab.

In Part 1 of the lab, you apply a given script (written in textual form) to solving various versions of a problem. You then are presented with a variation of the problem and asked to figure out how to modify the script to solve the variant. This will require a bit of original thought -- it can't be handled just by changing parameter values.

In Part 2, you will study how to do data fitting -- find a formula that provides a mathematical description of measurements collected. Once you have the formula, you can use it to answer additional questions and make predictions about the situation that was measured. This part also requires the use of textual operations, as data fitting is not available under the clickable menu.

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

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 strings. "Command completion" (entering solve and then hitting the escape key) will be demonstrated as a typing shortcut. Ways of troubleshooting your way out of problems with textual entry will also be discussed. How to use convert to do unit conversion for numbers or formulas.

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.

Data-fitting

The instructor will demonstrate the data fitting facilities in Maple. Part 2 is basically "read the on-line documentation and experiment with the examples until you get them to work".

3.3 Part 1

You can estimate in a reasonably accurate way the shutter speed of a camera by taking a picture of a moving object. If the movement of the object is known, then the amount of blur is related to the shutter speed.

In this picture, we see a circular bicycle wheel rotating. The spokes of the wheel blur and sweep out an angle we can measure from the photograph.

![Photo showing motion blur caused by slow shutter speed](image)

We look at one of several photographs we have taken with various cameras and bicycles, and come up with the following measurements:

**Case A**

Wheel diameter $d = 26$ in

Bicycle forward speed $v = 15$ mph
Angle of rotation $\theta$: 10.2 degrees

We want to calculate the shutter speed $s$, which we expect to be a time in seconds, e.g. $s = .017$

means that the shutter is open for .017 seconds (17 milliseconds). The terminology of "shutter speed" does not refer to the velocity of the shutter as it moves when taking a snapshot, but rather to the length of time the shutter lets light into the camera.

1. Work out the math steps that calculate $s$ for case A.

It may not be immediately clear to you how to do this. Consider the following possible calculations. For those that you find useful, use Maple to do scratch calculations. Remember to assign useful results a name through assignment. For example, for part d) below, you might do $c := \pi*d$.

a) If $v = 15$ miles per hour, how long (in hours) would the bicycle take to travel 5 miles, if it were rolling down a road at that speed? What about 1 mile? What's the formula for the time to travel $m$ miles at that speed? Is it $m\cdot15$, $\frac{m}{15}$, or $\frac{15}{m}$?

b) If you knew the velocity in inches per second was $vips$, how long (in seconds) would the bicycle take to travel 100 inches? What's the formula for the time to travel $i$ inches at that speed? What unit of time is the travel time measured in if you compute using $vips$.

c) How can you calculate $vips$ from $v$ using the Maple convert operation? Your instructor should have demonstrated this during their overview talk.

d) What is the circumference of the bicycle wheel? Recall how to enter $\pi$ : from the Greek or Common Symbols Palettes, or textually, $\pi$. Note that $\pi$ does not refer to the mathematical constant. Maple pays attention to whether letters are upper- or lowercase. It has to be $\pi$.

You will surely have to use this quantity in your solution. Call it $c$.

e) How far does the bicycle travel if it were rolling for one wheel revolution?

f) How long does it take, in seconds, for the bicycle to travel that distance?

g) What fraction $f$ of a whole revolution is the blur angle $\theta$?

h) How long does it take for the bicycle to travel the distance which is that fraction of a revolution?

i) How is $s$ related to the answer you got for h)?

Write down on the whiteboard your steps in calculating the answer. You don't need to use all of a) through h), but some of them will probably be part of what you write down. Be prepared to provide the staff with justification that this answer is correct.

Keep it real -- what justification would you be using to convince your boss that the answer was right if you had no other group around to compare answers to? "Because Maple did it" doesn't provide good justification, since it only does what you tell it to, and you might have told it to do the wrong thing.

2. If you have not already done so, get all the steps into a Maple worksheet, and have it calculate the answer for Case A. What is your answer for $s$? Write your answer down on the whiteboard. Provide the staff with justification that this answer is correct.

3. In a fresh worksheet, organize your ideas to create a script that can do the calculation for any given diameter, speed, and angle. Put the script into standard form with the parameters first. Only the "starting information" $d,v,$ and $\theta$ are parameters here.

Your worksheet should have this general form;
CS 121 Lab 3 Shutter speed script

By ....

Section ....

Instructor: ....

Unassign any existing assignments in the session.

restart

Begin parameters

Unassign any existing assignments in the session.

restart

Velocity (in miles per hour)

\[ v := \ldots \]

Wheel diameter (in inches)

\[ d := \ldots \]

Blur angle (in degrees)

\[ \theta := \ldots \]

End parameters

Compute circumference from diameter

\[ c := \ldots \]

Compute velocity in inches per second

\[ \text{vips} := \ldots \]

etc. etc.
Compute shutter speed from .... This is our answer

8 := ....

End Script

For the last step, add a little post-calculation tidying up: use the right-clickable menu on the result for s, and apply numeric formatted (fixed) on the last result, so that it displays an appropriate number of significant digits. From Engr 101 considerations, you should be able to decide on your own how many digits that should be.

Double check that you using the correct script format, and have provided enough information for the description of each step. Your grade for this part of the lab is based in part on how closely you follow the intent to have enough information for others to understand who did the work and how you did it, and that the script is organized so that anyone will be able to use it to calculate a different version of the problem.

We expect that what you will have is a script with the values of the parameters set to calculate the solution for Case A. Check that it's the same as your computed before in step 2. Save the worksheet as YourNameLab3-A.

4. Swap copies of your worksheet with one of your lab partners or some one nearby. Edit it appropriately and save it as YourNameLab3-B.mw. Edit and re-execute the worksheet to handle the following case:

Case B

Wheel diameter d = 25.5 in
Bicycle forward speed v = 18 mph
Angle of rotation Î¸: 12 degrees

Assuming that the author wrote the worksheet well, it should take you about 30 seconds to do this after you open up the worksheet. Save this as YourNameLab3-B.

5. In this step, you don't need to build a script, but you do need to do the work in a fresh worksheet. Do New-> Document Mode and then save the blank worksheet as MyLab3-C.mw

If you have been following the derivation of s closely, you can realize that you can build a computation that combines all the operations. We're going to build a function that encapsulates all the operations. We will name the function shutterSpeed (rather than "f" or "g", as a math text book might. It will take three arguments values for v, d, and θ. The result of the function will be the shutter speed, in seconds, as the scripts did.

Function definitions were discussed in chapter 7 of the chapter readings, as well as the subject of a quizlet question or two.

In a fresh worksheet, type in this function definition:

\[
\text{shutterSpeed} := (v, d, \theta) \rightarrow \text{evalf}((d \cdot \pi / \text{convert}(v, \text{units}, \text{miles} / \text{hour}, \text{inches} / \text{second})) * \theta / 360)
\]

(As mentioned in chapter 9 of the course readings, evalf, is the textual way of doing approximation. By default you get a ten digit approximation.)
a) Compute shutterSpeed(15, 26, 10.2) and compare your result to your script's operation for case A. They should be the same.

b) Do the same for Case B.

c) If you enter the symbol v1 instead of a number for v, shutterSpeed(v1, 25, 10.2) returns a formula involving v1 rather than a number. Try it. This is the formula for shutter speed for an arbitrary velocity v1, a wheel diameter of 25 inches, and a blur angle of 10.2 degrees. If you make all of the arguments to shutterSpeed symbolic, e.g. shutterSpeed(v1, d1, a1), what does the result that you get back mean?

d) Save your work for this part.

3.4 Part 2

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 \( \left( x_1, y_1 \right), \left( x_2, y_2 \right), \ldots, \left( x_n, y_n \right) \) 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 produces the following data for pressure \( P \) versus temperature \( T \) with the volume held constants:

<table>
<thead>
<tr>
<th>&quot;P (kilopascals)&quot;</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>&quot;T (deg Celsius)&quot;</td>
<td>0</td>
<td>20.1</td>
<td>39.8</td>
<td>60.0</td>
<td>79.9</td>
<td>100.3</td>
</tr>
</tbody>
</table>

We want to find values \( a \) and \( b \) so that the line described by \( a*t + b \) 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: \( \text{pData} := [134.2, 142.5, 155.0, 159.8, 171.1, 184.2] \). Similarly, create a second list and assign it to tData.

3. Produce a point plot with Maple using the techniques discussed in chapter 6 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 your 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 Final actions (end of class)

Email (or put on a flash drive) copies of your work to yourself and/or your partner so that you have it for future reference and use.

3.6 Conclusion

In this lab, you have gotten further practice at creating scripts in Maple. We have expanded the repertoire of objects to include lists, which allow us to maintain aggregations of values in an easy-to-access fashion, and character strings, which allow computations with textual information. We have done a little data fitting, producing a formula describing a curve that approximately describes a collection of numerical measurements. We have practiced with point plotting, and "multi-plotting" (where multiple curves appear together on a single set of axes). In preparation for programming, we have begun to practice with entry of operations using text rather than mouse clicks and menu items.
4 Lab 4 Cs 121 Computation Lab I Fall 2011 Directions and Problems

4.1 Lab 4 Overview

Overview

The lab explores the problems of finding artillery trajectories that satisfy certain requirements and constraints. It assumes that you have read and understood the material in Chapters 8 and 9 of the course readings: "Programming with Functions" and "Visualization, Modelling and Simulation".

These computations use a mathematical model that describe how key values and properties change over time. It uses more sophisticated plotting and animation more rapidly achieve understanding essential to design verification or modification. As you will see, animation can lead to more rapid understanding compared to large tables of numbers or pages of graphs.

Before beginning lab work, you will 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.

To prepare for this lab beforehand

1. Read Chapters 8 and 9 of the course readings. One way to better understand the material is to learn how to reproduce the examples in the text on your own computer.

2. Study these lab directions and the lecture notes for this lab, posted on the course web site.

3. Take the pre-lab quizlet.

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 the four problems of the lab (80 minutes). You should create a separate worksheet for each problem, named as described in the directions below. Your work should use only textual versions of Maple operations. You will not receive credit for answers that invoke operations through the clickable interface. You will not receive credit for solutions that are crammed together in a single worksheet.

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.

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

\[
x(t) = V_0 x \cdot t \quad \text{and} \quad y(t) = y_0 + V_0 y \cdot t - \frac{1}{2} gt^2
\]

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

\[
v_0 := 100
\]

\[
100
\]

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

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

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

\[
v0x := \cos(\text{alpha}) \cdot v0
\]

\[
50 \sqrt{2}
\]

\[
v0y := \sin(\text{alpha}) \cdot v0
\]

\[
50 \sqrt{2}
\]

\[
g := 32
\]

\[
32
\]

\[
y0 := 0
\]

\[
0
\]

\[
x0 := 0
\]

\[
0
\]

\[
\text{xpos := } (t) \rightarrow x0 + v0x \cdot t
\]
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.

\[
\text{solve}(\text{ypos}(t) = 0, t)\]

\[
\begin{align*}
0, \quad &\frac{25}{8} \sqrt{2} \\
\end{align*}
\]

(4.10)

Not surprisingly, one of the times is \( t=0 \) (the start). We can get the time we want by daisy-chaining \textit{solve} (which gives a sequence of two roots) and the \textit{max} function.

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

\[
\begin{align*}
\frac{25}{8} \sqrt{2} \\
\end{align*}
\]

(4.11)

We daisy-chain the function that calculates \( x \) position with the approximation function \textit{evalf}. By default, \textit{evalf} computes ten decimal digits accuracy.

\[
\text{distance} := \text{evalf}(\text{xpos}(\text{flightTime}))\]

\[
312.5000000
\]

(4.12)

Evidently Blammo travels 312.5 feet.
Now, if we plot from \( t=0 \) to \( \text{flightTime} \), we should see the whole plot:

\[
\text{plot}([\text{xpos}(t), \text{ypos}(t), t = 0 \ldots \text{flightTime}], \text{labels} = ["feet", "feet"])
\]

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 \(t=0..\text{flightTime}\).

\[
drawBlammo := (t) \rightarrow \text{plot}([\text{xpos}(t), \text{ypos}(t)], \text{style} = \text{point}, \text{color} = \text{"red"})
\]

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

\[
\text{animate(drawBlammo, [t], t = 0 .. \text{flightTime})}
\]

Maple has other useful functions to help us:

\texttt{maximize} will find the largest value that a function attains. For example,

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

\[
78.12500000
\]

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.

This is a "use the script and interpret the results" problem. You have to learn how to use the features in the script, but you don't have to modify any code or write new code.

4.5 Problem 2

This problem requires you to modify code. To cleanly separate your work from problem 1, save a copy of the starter script in a different file, YourNameLab4Problem2.mw, then begin the Problem 2 worksheet to solve this problem.

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, modify the plotting instructions 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 9 of the readings.

Save your work as YourNameLab4Problem2.mw, to show to the grader.

This file should contain the solution to problem 2 ONLY. Having several solutions (possibly incorrect, and possibly interfering with each other) all in the same worksheet will just create more problems for you to have to solve.

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,

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

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

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

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

\[
\text{CURVES}([ [200., 0.], [200., 50.]], \text{COLOUR}(\text{RGB}, 0., 1.00000000, 0.), \text{LINESTYLE}(3))
\] (4.16)
If we display them together we get a plot that looks like this:

```python
with(plots):
    display([p1, p2])
```

What you want to do is to generate the parameter plot and assign it to p3. Then doing `display([p1, p2, p3])` should result in a picture that looks something like this.

```python
display([p1, p2, p3])
```
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 YourNameLab3Problem3.mw.

4.7 Problem 4

Exchange the script from Problem 2 with one of your partners. You will now use their script to solve another problem.

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 drawNet := (d, w) -> 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)

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 Conclusion

In this lab, you have used a computerized mathematical model of artillery trajectories to solve problems having to do with a human cannonball. One of the advantages of using the model is that it's not necessary to spend as much money on test firings and replacement human cannonballs while you try to find what will work safely. You have extended your repertoire to include more elaborate user-defined functions that can be used to generate animated plots. You have learned how to plot objects besides lines and points. You have gotten more practice at using and modifying scripts.

4.10 Acknowledgements

This exercise was developed with the help of Dr. Jeremy Johnson, Dr. Fred Chapman, and Mr. Ryan Walls.

4.11 Attachment: Lab4StarterScript

CS 121
Lab 4

Starting Script for Blammo

This script takes as parameters the initial velocity v0 of Blammo, and initial position (x0,y0). It shows a plot of Blammo's path through the air, and calculates the maximum horizontal and vertical distance achieved during his trip. It also produces a computer-generated animation of him flying through the air.

Start of Parameters

Starting velocity

\[ v0 := 100 \]  

Starting positions

\[ x0 := 0 \]  

\[ y0 := 0 \]

Firing angle of gun

\[ angle := 45 \]
End of Parameters

Calculate the angle in radians

\[
\alpha := \text{convert}(\text{angle} \cdot \text{degrees}, \text{radians})
\]

\[
\frac{1}{4} \pi
\]  
(4.21)

Split the starting velocity into x- and y- components.

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

\[
50 \sqrt{2}
\]  
(4.22)

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

\[
50 \sqrt{2}
\]  
(4.23)

Establish the gravitational constant (in English Units, its 32 feet per second, squared).

\[g := 32\]  
(4.24)

Define the functions for x and y position

\[
x_{\text{pos}} := (t) \rightarrow x_0 + v_0x \cdot t
\]

\[t \rightarrow x_0 + v_0x \cdot t\]  
(4.25)

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

\[t \rightarrow y_0 + v_0y \cdot t - \frac{1}{2} \cdot g \cdot t^2\]  
(4.26)

Determine how long Blammo will be in the air (assuming he doesn't hit the wall of the circus tent, the man on the flying trapeze, etc.). We can get the time we want by daisy-chaining \texttt{solve} (which gives a sequence of two roots) and the \texttt{max} function.

\[
\text{flightTime} := \text{max}(	ext{solve}(y_{\text{pos}}(t) = 0, t))
\]

\[
\frac{25}{8} \sqrt{2}
\]  
(4.27)

Calculate the horizontal distance Blammo travels. We daisy-chain the function that calculates x position with the approximation function \texttt{evalf}. By default, \texttt{evalf} computes ten decimal digits accuracy.

\[
\text{distanceTraveled} := \text{evalf}(x_{\text{pos}}(\text{flightTime}))
\]

\[
312.5000000
\]  
(4.28)
Determine the highest vertical position.

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

78.12500000

(4.29)

Use a parameter plot to show the whole trajectory.

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

Create a function that draws Blammo at a moment of time, \( t \)

\[ \text{drawBlammo := (t) \rightarrow plot([xpos(t)], [ypos(t)], style = point, color = "red")} \]

\[ \text{t \rightarrow plot([xpos(t)], [ypos(t)], style = plottools:-point, color = "red")} \]

(4.30)

Create an animation of Blammo flying through the air.
with plots:

animate(drawBlammo, [t], t = 0..flightTime)
Overview

At this point, we've written scripts to do mathematical calculations and visualizations -- plots and animations -- of them. Some of the calculations involved mathematical models of physical situations. We found that that parameterizing scripts gave them greater flexibility, making it easy to reuse them to solve variations of a basic situation.

In the first part of this lab, we introduce the code edit region of Maple that facilitates the construction and execution of textually-entered scripts. The Maple clickable interface that you learned originally is good for rapidly doing a "one of" calculation, as long as it involves a sequence of one-line calculations available from the clickable menu. However, when you get into more elaborate and length segments of code, it's more convenient to be able to deal with multi-line sequences of operations in one block of text as code edit regions allow.

Besides mathematical modeling, another important application of computer programming is control of devices. In this lab we introduce a Maple package that supports control of a simplified model of a car. Rather than controlling a physical car, the programming controls a "virtual car" whose movements are shown in an animation constructed when you run the simulation. The programming for the control is similar to the NXT robots used in ENGR 102 -- but you can certainly learn to control the car without any knowledge of what goes on in that course. The simulator world does not have the same difficulties with wheel slippage, sensor error, and motor imprecision that a real robot would have. This makes it easier to develop and debug program for more complicated situations. A control program for a real car in the same situation would be yet more complicated because it would also have to deal with the imperfections of real world robots. Building a more realistic simulation is feasible but we have left those details because getting good control of a car even in the idealized world of this simulator is hard enough for a short student lab.

This lab presents an experience with controller programming, using an API (application programming interface). It gives you a chance to practice your software development skills by controlling a car simulator that behaves suspiciously like a NXT robot. However, since it's a simulation, you don't have to spend a lot of time with mechanical breakdowns or resets.

A brief overview of autonomous car control

The Defense Advanced Research Projects Agency DARPA a few years ago sponsored a contest
for "autonomous vehicles" -- cars that drive themselves. There were no humans inside the cars, but there is a "chase car" nearby with an emergency cut-off switch just in case. The driving range for the contests in successive years became increasingly more difficult, from open countryside, to urban traffic navigation. The winning team from Carnegie Mellon in 2007 won $2,000,000 for placing first. You can read about the 2007 contest at http://www.darpa.mil/GRANDCHALLENGE/.

The DARPA challenge spurred significant progress. Google continued this work, unveiling its self-navigating car in 2010. There are You Tube videos (e.g. http://www.youtube.com/watch?v=d0NyutzjS8&noredirect=1, http://www.youtube.com/watch?v=-nYhKD8leAg&feature=related) of the Google car driving in traffic.

Since good driving depends on road, vehicle, and traffic conditions, it's clear that any autonomous vehicle could not function on a fixed strategy. In other words, it would have to carry on-board some computer programming that would look at and respond to variable conditions.

In this lab, we look at some aspects of the programming involved in a simple simulation of an autonomous vehicle. We introduce and use the concepts of procedure, Application Programming Interface (API), and control loop to guide the car. In the next lab, we will introduce the concept of conditional execution to extend our programming to handle more complicated scenarios.
Scenes from the DARPA autonomous vehicle race in 2007

The Google self-driving Prius
Pre-lab preparation

1. Reading: chapters 10 and 11. Review older chapters 1-9 and labs from cs 121 as needed.
2. Take the pre-lab quizlet 1 at the CS 122 Maple TA web site. You should do quizlet 1 before lab to be prepared for the lab activities.

Part 0 -- Prelab prep

Problem A - Learn how to enter a short script into a code edit region and execute it

To do this work, it will be helpful to have read "Code edit regions: executing a series of actions at once" in Chapter 10 of the course readings and have it handy as you do the work.

Do the following:

1. Open a Maple worksheet
2. Create a code edit region by

Insert -> Code Edit Region

Checkpoint: What you should have should look like this:

3. Change size of region by
   Right click -> Component Properties -> change to 800 x 200 pixels

Checkpoint: What you should have should look like this:
4. Enter some code (enter this code exactly)

```plaintext
#This is a proc to compute the hypotenuse of a triangle with sides s1 and s2.

Hyp := proc(s1,s2)
local hypotenuse;
hypotenuse := sqrt(s1^2 + s2^2);
return hypotenuse;
end proc;

# Now, execute this proc
Hyp(7, 24);
```

Checkpoint: What you should have should look like this:
Notice the elements of a proc (read over comments)
Hyp := proc(s1, s2)  
local hypotenuse;  
hypotenuse := sqrt(s1^2 + s2^2);  
return hypotenuse; 
end proc;

# Now, execute this proc
Hyp(7, 24);

5. Execute the code. You can do this (assuming that the flashing cursor is located in the code edit region box, by typing command-E (control-E on Windows), or by clicking on the "!" icon in the Maple toolbar.

Checkpoint: What you should have should look like this:

Hyp := proc(s1, s2)
local hypotenuse;
hypotenuse := sqrt(s1^2 + s2^2);
return hypotenuse;
end proc;

# Now, execute this proc
Hyp(7, 24);
local hypotenuse;
hypotenuse := sqrt(s1^2 + s2^2); return hypotenuse
end proc

6. Collapse the code region by clicking on the region and then Right-click -> Collapse code region. After that, you should see:

Hyp := proc(s1,s2)

Clicking on the code region icon will re-execute the script.

Problem A deliverables and outcomes

Have copies of your executed worksheet files ready to show and demo to the staff in order to get credit for doing this part of the pre-lab. Your team should be prepared to explain verbally or demonstrate some of the following things:

a) How do you create a code edit region in a worksheet?
b) How do you adjust the width and height of the text box region? What are reasonable settings for a region so that you can get work done?
c) How do you "iconify" the region? What happens when you click on the icon? How do you "un-iconify" a region?
d) How do you execute all of the lines in a code edit region? What is the keyboard shortcut for doing so?
e) What is an execution trace and what kinds of information can you get from it?

Problem B - "while" loop introduction - using "while" loop and variables as counters

1. Create a code edit region and re-size it as you proceed so that the entire segment of code is visible in the CER window.

2. Enter and execute the following code segment. Be sure to carefully review the code along with the associated comments:
# this code adds a list of numeric elements

AddList := [1, 2, 3, 4, 5];
ListSize := nops(AddList);  # nops is a Maple function to determine the number of elements in a list

ListSum := 0;  # keep an accumulated value of the sum as new elements are added in
count := 0;    # keep track of the number of elements already added to the sum

while (count < ListSize) do
    count := count + 1;  # increment the counter here so that the correct element is accessed next
    ListSum := ListSum + AddList[count];
end do;  # end of while loop - all elements accumulated

print("Sum of list elements ", ListSum);

3. **Problem B deliverables and outcomes**: Answer the following questions:

   a. What is the sum of the elements as produced by the program?
   b. What happens when we change the semi-colon after the "end do" statement to a colon?
   c. What is the effect of initializing the count to 1 instead of 0?
   d. If the count was initialized to 1, what else in the program would need to change in order to obtain the correct answer?

Show the modified code and result to the grader.

**Problem C -- learn how to run a car simulator program in a code region and interpret the results**

First download and expand the zip file CS122Lab1.zip from the course web site.

CarSimulator.hdb
CarSimulator.mla

CS122Lab1StarterB1.mw
CS122Lab1StarterB2.mw

CS122Lab1Problem4-Tutorial.mw

CS122Lab1Problem1.mw
CS122Lab1Problem3.mw
CS122Lab1Problem4.mw
CS122Lab1Problem5.mw
When working on car simulator problems, you should begin work by double clicking on one of the lab .mw files to start up the simulator. This will have the effect of causing Maple to automatically look in the right place for the CarSimulator library files. If you have trouble doing this, ask for help.

Let's see a quick demo of the simulator.

(a) **Open and execute** the worksheet named CS122Lab1StarterB1.mw. Within the worksheet, you should see lines of code that:

1. Establish initial values for Maple variables used in the simulation. This initialization must be each time you want to use the simulator in order to set up the car "arena" and set the initial position of the car.
2. Setting up a target for the car to end up at (it's optional).
3. Establishing the limits of the "playing field" for the simulator. You can visualize this as a wall of specified dimensions that the car will run into if it goes too far.
4. Writing a Maple procedure that describes actions that the car will take. Note that the sample program, run1a, contains a few actions from the car simulator package move and turn.

Note that the car is pinkish when it is running but turns gold when it finishes. This is what it does when it stops next to a target square (blue). The car should automatically stop when it is pointing at and next to a target.

(b) **Open and execute** the worksheet named CS122Lab1StarterB2.mw. This worksheet explores the state table, which explains the position and condition of the car at points of time.

**Problem C deliverables and outcomes**

Have copies of your executed worksheet files ready to show and demo to the staff in order to get credit for doing this part of the pre-lab. Your team should be prepared to explain verbally or demonstrate some of the following things:

a) What is the Maple name of the package for the Car Simulator?
b) What does the command with(CarSimulator): do?
c) What is the color of a "wall" square? What is the color of a target square?
d) What does direction of the arrow drawn at each step for the car indicate?
e) Describe what you would see if the car was originally located at (3,2) and was told to move 3 squares backwards. How could you tell whether the car was moving forwards or backwards?
f) What is the name of the operation in the car simulator package that
   - produces the animation of the car moving around?
   - runs the simulation of the car with a control program?
g) What is the purpose of the argument given to the run operation?
Lab Exercises

For these problems, you should continue to work with the unzipped files for this lab.

Problem 1 -- Get the car to move in a variety of directions

Open the worksheet named CS122Lab1Problem1.mw Solve the problem in it, which asks you to write a program to move in a variety of directions, as specified in the work sheet. To get points for this part, you should be able to demonstrate the program.

Problem 2 -- Understand more about the CarSimulator package

The CarSimulator package includes some help pages on the various features of the package. Activate Maple on-line Help. In the "Table of Contents" pane on the left hand side you should see a folder for the CarSimulator. (Note: CarSimulator help should be available after you have done Part 0, because some of the actions in Part 0 tell Maple to look in the current directory for help. If not, open again one of the Tutorial worksheets from Part 0 and then activate Maple help again.)

CarSimulator documentation displayed in Maple on-line help
The documentation for this was written by course staff and included in the CarSimulator.hdb file that you downloaded with the other lab files. Use this Help segment as you encounter car simulation features which need explanation or clarification.

**Problem 3 -- Write a program to get the car to move in a Square**

Solve the programming problem in CS122Lab1Problem3.mw. You should write a program that *exactly duplicates* the behavior of the sample animation.

This problem just has the car moving in a square whose side is 6 units. Note that movement begins Eastward from the starting position.
Problem 4 -- Write a program to get the car to move under a wall, detect a gap and proceed towards a target

In this problem, we present another scenario. Your mission is to write a program that will negotiate any of the various versions of the scenario. In order to get credit, the same program should work for all of the scenarios. In other words, you should be able to copy and paste your solution for the first version into the code edit region that executes the second version, and it should work there without alteration.

(a) Open, inspect, and execute the scenario and code presented in CS122Lab1Problem4-Tutorial.mw. This shows the car moving forward until it bumps into the right wall, turns around and proceeds until it bumps in to the left wall. Notice that you can execute the same program in a scenario where the walls are in different locations and the car still turns/stops when it hits te right/left walls respectively. You should be able to answer these questions:

(i) Describe, in your own words, what actions are being repeated by the while.
(ii) Describe in your own words, what actions are necessary as initialization before the repetition of the while gets started
(iii) Describe in your own words, the condition checked by the while loop. When will the loop repeat? When will the loop stop?

Write these descriptions on the whiteboard.

(b) Open and study the scenario in the code presented in CS122Lab1Problem4.mw. Complete the coding so that your control function can handle any of the sample scenarios.

Idea: move forward until you encounter a gap in the overhead barrier. Then turn right and move until you encounter a goal.

When you have solved the problem and can get the same code to solve both executions (with different wall lengths and target positions), have it graded by the staff.

Problem 5 -- Write a program to detect the nth gap in an overhead wall

Solve the following scenario. As with Problem 4, your mission is to write a program that will negotiate any of the various versions of the scenario. In order to get credit, the same program should work for all of the scenarios. In other words, you should be able to copy and paste your solution for the first version into the code edit region that executes the second version, and it should work there without alteration.

Open the code in CS122Lab2Problem5.mw, follow the instructions and solve the problem. You will notice that this problem is an extension of Problem 4, except that you need to find the nth (not 1st) gap in the wall before turning right and proceeding towards the target. This additional complexity causes you to use nested "while" loops (the outer loop moves you from gap to gap, while the inner loop moves you into and then across a specific gap. A counter variable will be
needed to keep track of the number of gaps and special logic will need to be included before you proceed to the final target to turn the car around and move back to the nth gap (if you overshot this gap).

Afterword: Comparison of NXT and Maple as programming languages for control

NXT for Lego robots and the CarSimulator package in Maple present different approaches to writing computer programs that control a device. Mindstorms' visual interface allows for "wired together" actions configured by point-and-click menu selection that move a robot. The Maple CarSimulator uses textual specification for actions and configuration. Repetition (handled by "for" in Maple) and conditional decision-making (handled by if or while) are present in both languages as a way of expressing control. A significant difference is that the car simulator takes a "move one unit at a time" approach to motion whereas control of a real robot typically engages motion or a motor continuously until another command disengages the action.

The textual interface tends to be preferred by programmers for larger control programs. It is usually faster to enter control programs (or to modify them) through textual entry than through clicking -- assuming that you are reasonably proficient at typing and meeting the syntactic exactitude required when communicating in a programming language. It also tends to be easier to do transformations of textual programs than visual programs. For example, if you decided that you wanted to modify a procedure so that all the turns to the right are replaced by turns to the left, you could find and modify all relevant instances of turn textual search and replacement. Doing a similar transformation in a visual program can often be much more laborious.

The Maple CarSimulator does not control a real robotic car. Nevertheless it and other software control simulators have their use in engineering applications because it's usually much faster to develop ideas and principles using simulator results, than it is to set up and test a physical device. For example, in a single lab period, you've been able to get in many more test runs with the simulator than were possible with running a physical robot. Even if the simulation is not perfectly realistic, there are typically many problems that can be worked out using them. Once the program is ported from the simulator world to a robot, more development will need to be done to adjust the simulator solution into a solution that works with the real robot. However, the overall development time should be less.

Textually-entered languages such as Java, C or Python can be used to program NXT robots. Maple could be as well. For the reasons we have mentioned, extended projects with robots such as solving and writing the solution to Sudoku puzzles (http://www.gizmowatch.com/entry/lego-mindstorms-nxt-robot-tailored-to-solve-sudoku-puzzles/) or manipulating and solving a Rubic's Cube (http://www.youtube.com/watch?v=3Q0vEG2Gt4) would be more quickly developed in such languages rather than NXT visual language.
Overview

In this lab, we continue to expand our repertoire of program control by looking at what we can do with \texttt{if} and \texttt{for}. We've already seen the use of conditions for continuing or terminating repetition. \texttt{if} addresses situations which select between two possibilities, each of which is intended to be done only once. \texttt{for} addresses situations that are repetitions that uses counter (index) variables to help control repetition. \texttt{for} is like \texttt{while}, with the counter variable initialized and updated automatically. It can simplify certain situations where there the number of times a repetition should happen is known before the looping starts.

We will continue to utilize the car simulator API from Lab 1 to introduce these new control statements.

Pre-lab preparation

1. Reading: chapters 12 and 13. Review previous chapters as needed.
2. View the on-line video that demonstrates usage of \texttt{if} and \texttt{for}. You can find the video at the course web site among the course files for this lab. Before lab, imitate the examples given in the video, to see that you can enter code into a code edit region and execute it.
3. Take the pre-lab quizlet 2 at the CS 122 Maple TA web site. You should do quizlet 2 before its deadline to be prepared for the lab activities.

Part 0 -- Prelab prep

A. Introducing the "for" loop

Sometimes, we know exactly how many times a loop should be executed prior to running the loop. In this case, using a "for" loop control statement instead of the conditional "while" loop construct can be a more efficient programming approach. The following example illustrates the "while" and "for" loop alternatives for solving the same problem.
The code edit region below demonstrates this comparison for adding the numeric elements of a Maple list and reporting their sum:
# 1st, define the list of integers to add together
AddList:= [1, 2, 3, 4, 5];

# obtain the number of elements in this list using Maple's "nops" function
ListSize:= nops(AddList);

# Using a while loop, solve this problem
WhileListSum := 0;  # stores the cumulative sum of the elements
Count := 0;    # Current count of number of elements processed
while (Count < ListSize) do
Count := Count + 1;
WhileListSum := WhileListSum + AddList[Count];
end do:

print("Sum of list using while loop logic :", WhileListSum);

#################################################################
#  Now solve the same problem using a "for" loop
# Note that a "for" loop based approach is appropriate since we know the
# number of list elements (= ListSize) prior to entering the loop
ForListSum := 0;
for count from 1 to ListSize do
    ForListSum := ForListSum + AddList[count];
end do:

print("Sum of list using for loop logic :", ForListSum);
Note that the "for" loop based solution does not require any manual tracking of the "count" of elements. This simplifies the logic for the solution algorithm.

**B. Introducing the "if" statement**

The "if" statement allows a program to selectively execute code, based on whether or not a condition is true (or false). The following pair of examples demonstrate 2 of the basic versions of the "if" statement construct.

1. Simple "if" - action to be executed only if condition is "true" (no action if "false")
# From a list of people's ages, determine the number of people who are 21 years old or older.

```plaintext
AgeList := [25, 15, 88, 11, 96, 21, 30];
Count21Plus := 0;
for i from 1 to nops(AgeList) do
    if(AgeList[i] >= 21) then
        Count21Plus := Count21Plus + 1;
    end if;
end do;
print(Count21Plus, "people are 21 years old or older");
```

[25, 15, 88, 11, 96, 21, 30]
5, "people are 21 years old or older"

2. "if-else" construct - different action based on condition being "true" or "false"
# Print a list of GPAs and include the note "Dean's List" if greater than 3.5

GpaList := [3.2, 4.0, 3.5];
for i from 1 to nops(GpaList) do
    grade := GpaList[i];
    if (grade <= 3.5) then
        print("GPA :", grade);
    else
        print("GPA :", grade, "  Deans List");
    end if;
end do:

C. Using "for" loops and "if" statements to simplify a Car Simulation example

In CS122 Lab 1 (problem 5), we created code to drive the car to a target placed above the nth gap in an overhead wall. Since we were confined to using "while" loops (we did not yet know about "for" loops and "if" statements), we were forced to construct a rather inelegant algorithm that might have looked something like this:
GapCount := 0;

# ngap is passed to the proc and represents the gap above which the
target resides

while (GapCount < ngap) do  # if you are at a gap which is not the nth gap
    while (isTouching(Pi/2, 'bt')) do  # you are moving towards a gap
        move(1);
    end do;

    # at this point, you are at a gap

        move(1);  # move 1 more block past the gap to the start of the next wall
        GapCount := GapCount + 1;
    end do;  # end of processing for current gap. Search for next one.

    # At this point, we are 1 block beyond the nth gap. Must turn around and go
    # back to the nth gap and re-orinet the car.

    turn(Pi);  # do a 180
    move(1);  # move back to the nth gap
    turn(Pi/2)  # make a left turn, so that car is now aimed toward the target

    # move up until you reach the target from below
Please notice that this (inelegant) solution, constructed using only (nested) while loops and counters, requires us to include rather awkward logic at the end to correct the overshoot and re-position the car. Also, notice the need to manually manage a gap counter variable.

A more well constructed algorithm is shown below to accomplish the same navigation by introducing "for" and "if" control statements.
# ngap is passed to the proc and represents the gap above which the
# target resides

# since we know that we need to run following "while" loop code exactly ngap
# times, we can use a "for" loop to accomplish this

for GapCount from 1 to ngap do
    while (isTouching(Pi/2, 'bt')) do  # you are moving towards a gap
        move(1);
    end do;

    # at this point, you are at a gap
    if (GapCount < ngap) then  # use an "if" to avoid the extra move at the
        move(1);
    end if;
end do;  # end of processing for current gap. Search for next one.

# At this point, we are exactly at the nth gap. No need to turn around and
# back to the nth gap and re-orient the car.

turn(-Pi/2);  # make a right turn towards the target

# move up until you reach the target from below

---

Compare this version of the code to the version above.

a. Note the elimination of manually maintained counters. The "for" loop automatically handles this activity
b. Note also how the "if" statement selectively supresses the move beyond the gap for the target gap, thus enabling the direct "right turn" at the final gap towards the target.

You might want to run this version of the code using the files from Lab 1, problem 5 to confirm it's correctness.

# Lab Exercises

For these problems, you should continue to work with the unzipped files for this lab. The zip file is located in the lab materials section of the course web site along side the Lab 2 description link.

## Problem 1 - Moving the car in a figure 8

Problem: Open and study the scenario in the code presented in CS122Lab2Problem1.mw. With the car at this starting position, create code to have it move in a figure 8 consisting of 2 squares whose sides are both 5 units in length.

Use nested "for" loops to trace the figure 8 path.

This exercise is similar to the week 1 engr102 robot lab assignment of tracing a variety of shapes. One difference is that the Car Simulator API is restricted to turns in increments of 90 degrees (cannot simulate a curved turn).

## Problem 2 - Moving to and turning at designated intersections

Problem: Open and study the code and setup in CS122Lab2Problem2.mw. It shows a grid of streets and intersections with the car postioned at a starting point. Develop code to move the car forward to the 1st 4 way intersection, turn left and proceed to a target.

The key concept to employ is to determine when the car is at a 4 way intersection. Using the "isTouching" function, the car will be at a 4 way intersection when this function returns "false" for all 4 angles (0, Pi/2, Pi and -Pi/2) when testing a specific car position.

## Problem 3 - Moving on a diagonal
The Car Simulator API turns only at 90 degree angles (ie. it cannot execute a 45 or other non right angle turn). If a target object is diagonally across from the car, move the car in a "straight line" to the target.

Problem: Open the code in CS122Lab2Problem3.mw and develop code to diagonally move across the grid to the target.

This example is similar to the engr102 robot lab requirement of moving directly to the appropriate (nuclear or waste) dump site.

---

**Problem 4 - Moving around a barrier to a target placed above the barrier**

Problem: Open the code in CS122Lab2Problem4.mw. Move up until you touch a wall. Then move left around the wall and back to the middle above the wall. Finally, proceed to a target which is directly (vertically) above the original position of the car.

This problem mimics the engr102 robot lab exercise in which a touch sensor notices the barrier and then action is taken to travel around it and finally straighten out again to move forward to a target.