APPENDICES

A SINGLE VARIABLE CALCULUS

A.1 INTRODUCTION TO MATLAB PROGRAMMING

Matlab is a very powerful programming system that can be used to solve much bigger, more complicated, and more interesting problems than what we can do by hand. When most people do calculus today, they do it using computers, so learning to do calculus on a computer is an essential part of this course.

There are two places where we can run commands in Matlab. In the "Command Window" we can give Matlab commands to run one at a time. But if we want to run a series of commands in order, a program, then we should create a "Script." In the upper left corner of the screen is the "New Script" button. You can type a series of commands in a script, save this script file, and then run your program by either pressing the "Run" button, or by typing control-enter.

Variables

In Matlab variables start with letters. So I can create variables and store numbers in them with the following commands:

```
x = 4
weight = 150
a12 = 15
```

We can then do mathematics with these numbers just by typing things like

```
x + a12
a12 - weight
weight * x
a12 / x
```

1. In algebra, a=b means the same thing as b=a. However, when we are using computers, those two statements mean very different things. Run the following lines in Matlab:

```
a=2
b=4
a=b
```

   After running these lines, what is the value of a? What is the value of b?

2. Now, run the following lines in Matlab:

```
a=2
b=4
b=a
```

   After running these lines, what is the value of a? What is the value of b?

3. Write a sentence or two explaining why you got different results for #1 and for #2.

Looping

Computers are very good at doing repetitive tasks again and again. One way to do this is with a structure called a loop. One type of loop Matlab can do is called a "for" loop, and we usually use it when we know in advance the number of times we want to go through the loop. For example, suppose we create a new script in Matlab that says the following:

```
x=5
for i=1:10
   x=x+2
```

```
end
x
```

The first time through this loop we will have i = 1. The command x = x + 2 takes the current value of x, adds 2 to it, and makes this the new value of x. So we have x = 5 + 2 = 7. The second time through the loop we will have i = 2, and x = 7 + 2 = 9. This will continue until the last time through the loop we will have i = 10. As a result, the command x = x + 2 will run ten times, meaning that we start with 5 and add 2 ten times, and so in the end we find out that x = 25. Suppose we want to add up all the odd numbers from 25 to 35:

```
x=0
for i=25:2:35
   x=x+i
end
x
```

We start with x = 0. Now, the first time through the loop we have i = 25, so x = 0 + 25 = 25. The second time through the loop i is 2 more, and so we have i = 27. As a result we have x = 25 + 27 = 52. The third time through the loop, we have i = 29. As a result we have x = 52 + 29 = 81. Another type of loop that Matlab can do is called a while loop. In this case, we keep running through the loop as long as something is true. For example, take a look at the following program:

```
x=25
while (x > 1)
    x = x / 3
end
```

We start with x = 25. Then we say that this loop will continue as long as x is greater than 1. Inside the loop we replace x with x / 3. So

the first time through the loop we have x = 25 / 3 = 8.333. This is greater than 1, so we run the loop again and compute x = 8.333 / 3 = 2.777. This is greater than 1, so we run the loop again and compute x = 2.777 / 3 = 0.9259. This is not greater than 1, so the loop does not run again and the program is finished.

4. What do we get when we add up all the even numbers, starting with 50 and going to 70?
5. What do we get when we add up all the numbers 2.50 + 2.51 + 2.52 + 2.53 and going up all the way to 5.5?
6. Start with the number 275. Divide this by 2 again and again until you get a result that is less than 0.5. What do you get?
7. Start with the number 12. Multiply this by 1.25 again and again until your result is larger than 100. What do you get?

If Statements

In MATLAB we use an "if" statement to check if something is true. For example, consider the following program:

```
n = 5
if (n > 2)
n = n+6
end
n
```

This program sets n to be 5, checks to see if n is greater than 2, and if so, adds 6 to n. As a result at the end n is equal to 11. Now consider this:

```
n = 1
if (n > 2)
n = n+6
```

```
end
n
```

In this program, the if statement finds that n is not greater than 2, and so and the end of the program n is still equal to 1.

When we need an if statement to check whether or not two numbers are equal, we must use two equals signs: $= =$ This is because a single equals sign (like when we say $n = 1$) makes a variable equal a certain number. So, we can write code like the following:

```
n = 7
for i = 1:5
if (i == 3)
n = n+i
end
end
n
```

The for loop has i go through the numbers from 1 to 5. When i is equal to 3, then we add i to n, and so at the end of the program, $n = 10$.

We can check for a series of different things with the "elseif" command:

```
n = 7
for i = 1:5
if (i == 3)
n = n+i
elseif (i == 4)
n = n + 2*i
end
end
```

```
n
```

Here, we first check if i is equal to 3, and if it is, we add i to n. If i is not equal to 3, then we check if it is equal to 4, and if so, we add 2*i to n. As a result we end up with 7 + 3 + 2*4 = 18.

Now, suppose we want to check whether a particular number is a multiple of 4. To do this, we divide the number by 4, and see whether we get a whole number or if we have a decimal. To test whether or not we have a whole number, we use a MATLAB function called "floor." The floor function takes any number and rounds it down to the nearest whole number. So, for example, floor(3) = 3, floor(3.2)=3, and floor(3.9) = 3. We can check whether a particular number is a multiple of 4 by dividing our number by 4, then seeing if the floor function changes it. If we get the same thing out of the floor function that we put in, then it must have been a whole number, and so we have a multiple of 4: if(number/4 == floor(number/4))

As a result, we can add up all the multiples of 4, 5, or 6 between 1 and 9 with the following program:

```
total = 0
for i = 1:9
if( i/4 == floor(i/4) )
total = total + i
elseif( i/5 == floor(i/5) )
total = total + i
elseif( i/6 == floor(i/6) )
total = total + i
end
end
```

This program adds up 4 + 5 + 6 + 8 to get 23.

8. If we list all of the natural numbers below 10 that are multiples of 3 or 5 we get 3, 5, 6, and 9. The sum of these multiples is 23. Write MATLAB program that will find the sum of all the multiples of 3 or 5 below 716. What is your result? (Once you have an answer, you can double check it with me to be sure youve got this right.)
9. Modify your program to find the sum of all the multiples of 4 or 6 below 1,546. What is your result?
10. Modify your program to find the sum of all the multiples of 5 or 9 below 5,291. What is your result?

A.2 DERIVATIVES AND LOOPING

Consider the function $g(x) = \cos x$. Based on the shape of the graph of this function we know what $g'(0)$ is. We can also make a numerical estimate of $g'(0)$ using a specific value of h. This wont give us exactly the right answer, but as we try smaller and smaller values of h, our estimates should get better and better, closer and closer to the correct answer.

1. What is $g'(0)$? Dont use Matlab for this one. You should be able to figure this out by yourself.
2. Using $h = 1$, estimate $g'(0)$ in Matlab.
3. To get a better estimate, lets use an interval half as big. Estimate $g'(0)$ using $h = 0.5$.
4. To get an even better estimate, lets use an interval half as big. Estimate $g'(0)$ using $h = 0.25$.

How small would h have to be in order to estimate $g'(0)$ so accurately that the absolute value of our error is smaller than 0.001? Write a Matlab program that starts by using $h = 1$ to estimate $g'(0)$. Then as long as the absolute value of the error is greater than 0.001, our program runs

a loop which cuts h in half, and makes another estimate of $g'(0)$. The loop stops running when h becomes so small that the absolute value of the error becomes less than 0.001. In Matlab we take the absolute value of a number "a" with the function:

```
abs( a )
```

As you are writing this program, think carefully about what we are comparing with 0.001 in our while statement. (Hint: We are not comparing either x or h with 0.001.)

To confirm that your program is running correctly, have it begin by making a plot of our function $g(x)$ and have your program add a secant line to this plot every time it goes through the loop, using the slope you get from each value of h. If your program is in error, the plot will probably look wrong. Choose an appropriate range of x values for your plot. Hint: In order to have a reasonable range of y values for your graph, plot $g(x)$ after your while loop. Make the cosine function the final thing we add to the graph.

When you have your program running correctly, respond to the following:

5. Put in your final plot.
6. What is our final estimate of $g'(0)$?
7. What value of h did our program use to make this final estimate?
8. How many values of h did our program have to try in order to make an estimate this accurate?

But what happens when we dont know the correct value of the derivative? Instead we can compare two different estimates made with two different values of h and see how close they are to each other.

9. Consider the function $f(x) = 3cos(72x)$. Estimate the derivative of

this function $f'(5)$ using $h_1 = 0.8$. Use the vpa command to get your answer to 8 decimal places.

10. Estimate $f'(5)$ using a value of h that is half as big: $h_2 = 0.4$.
11. What is the difference between these two estimates?

As we choose smaller and smaller values of h_1 and h_2, then our estimates of the derivative will converge on the correct value. As a result the difference between two estimates with two values of h will become closer and closer to each other.

Write a program that starts with $h_1 = 0.8$ and $h_2 = 0.4$, estimates the derivative $f'(5)$ using each of them, and then calculates the difference between our two estimates of the derivative. If the absolute value of this difference is more than 0.001, our program cuts both of our values of h in half. Then our program estimates the derivative $f'(5)$ using each of these two new values of h, and calculates the difference between our two estimates. Once again, if the absolute value of this difference is more than 0.001, our program cuts both of our values of h in half. This process continues until we get a difference that is less than 0.001.

To confirm that your program is running correctly, have it begin by making a plot of our function $f(x)$ and have your program add our two secant lines to this plot every time it goes through the loop, using the slopes you get from each value of h. If your program is in error, the plot will probably look wrong. Choose an appropriate range of x values for your plot.

12. Put in your final plot.
13. How many times does your program go through the loop?
14. What are your final two values of h?
15. What are your final two estimates of $f'(5)$?
16. What is the final difference between these two estimates?

17. Which of these two estimates of $f'(5)$ is probably more accurate?

A.3 NEWTON'S METHOD

We are learning about finding derivatives and writing equations of tangent lines. Now we are going to see how we can use tangent lines to search for the zeros of a function, using an algorithm called "Newton's Method." In other words, we want to find the values x where the function crosses the x-axis so that $f(x) = 0$.

To start Newton's method we begin with a guess for our x value, call it x_0. We find the equation of the tangent line, and solve for the place x_1 where the tangent line crosses the x-axis. This will probably be close to the value of x that were looking for. Then we find a tangent line at the point x_1, and find where it crosses the x-axis to give us point x_2. After doing this several times, our x values will usually hone in on the one were looking for.

1. For our first exercise well start with a simple function $f(x) = (x - 1)^2$. We know what the root of this equation is: For this function, what value of x makes $f(x) = 0$?
2. Now well use Newton's method to find this. Well begin with a guess of $x_0 = 2$. What is the equation of the tangent line to $f(x)$ at $x = 2$? Write it in the form: $y = f(2) + f'(2)(x2)$.
3. Now lets plot both f and your tangent line y to check your work. Plot these over the range $0 < x < 3$. (Copy this plot into your Word document.)
4. From your graph, estimate the x value where your tangent line crosses the x-axis.
5. Now solve for this point using algebra. What is this x value?
6. You can make Matlab solve this equation for you using the solve

command. If you have defined your line as y just enter:

```
solve(y,x)
```

This will find the value of x that makes $y = 0$. Does it give you the same result that you got for question 5?

7. Now we do the same process again, starting with $x = 1.5$, but this time lets make Matlab do most of the work:

```
syms x a
f(x) = (x-1)^2
fprime(x) = diff(f(x), x)
y2(x) = f(1.5) + fprime(1.5)*(x-1.5)
x2=vpa(solve(y2(x) , x), 8)
```

What is the new tangent line? Where does it cross the x-axis?

8. Make a plot showing $f(x)$, the first tangent line y, and the second tangent line $y2$, again using the range $0 < x < 3$.
9. Take another step, now starting with $x = 1.25$. What is the new tangent line? Where does it cross the x-axis?
10. Is each step taking you closer to the correct answer? Predict: What do you think the next value of x will be?
11. Now make Matlab take that next step. What is the new tangent line? Where does it cross the x-axis?
12. Add the new tangent line to your plot.
13. All right, lets do something more interesting: Suppose you want to know where the polynomial $g(x) = \frac{5}{4}x^4 - \frac{29}{2}x^3 + 48x^2 - \frac{35}{2}x - 40$ is equal to zero. Plot this polynomial, and find a range that will allow you to estimate the locations where this crosses the x-axis. How many zeros are there? Include your plot and your estimates in your Word document.

14. Now lets find these values with Newton's Method. Lets choose $x = 1$ as our first guess. We can make Matlab do Newton's method all in one step if we use the commands:

```
x0 = 1
gprime(x)=diff(g(x), x)
x1 = solve( g(x0)+ gprime(x0)*(x-x0) , x)
```

What result do you get?

15. Use the vpa command to change your result into a decimal.

16. We want to apply Newton's Method until we find the zero with a good deal of accuracy. One possibility is to work through these iterations manually, changing the values yourself each time and stopping when you see that your result doesnt change. Another way to do this is with a loop. The loop will let us tell Matlab to run Newton's Method over and over again. Lets automate Newton's Method with "while" loop command, using the same starting point that we began working with in #14. Rather than calling the variables x0, x1, and so on, we really only need to worry about two: xold and xnew. xold will be the input for the iteration, xnew will be the output, and then we will reset xold to have the value of xnew for the next iteration.

    If our first guess is $x = 1$, then we need to initialize xold with the value 1. So type

```
xold=1
```

    We also want to count the number of iterations we're doing, so let's initialize a counter:

```
count=0
```

    We will have to choose an error bound, and then redo all of this

(including resetting xold and count) in the "while" statement. Lets start with an error bound of 0.0000005. We want to run the loop as long as xnew and xold are different by more than this value.

Create this program. In your lab report, record the sequence of values you got for xnew. How many iterations did it take before that value did not change? (Remember we actually need to see it stay at the same value, so we always end up doing one more iteration than we really need, so count that confirmation iteration in your answer.)

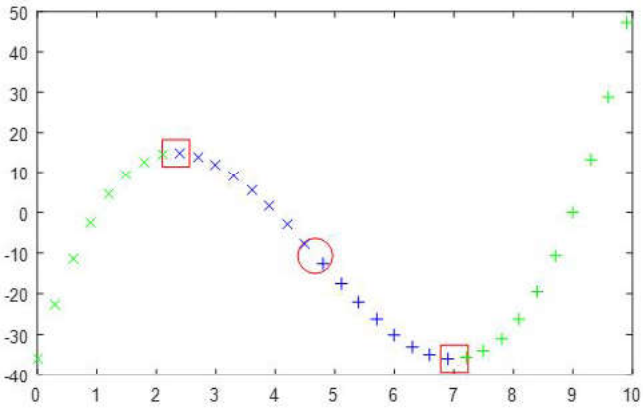
17. As you should have found above, this function crosses the x-axis twice. In order to find the other solution of this equation, you need to start with an x value closer to the other crossing point. Try different starting x values until you can find the other root of this equation. Give me your starting value, your final result, and tell me the number of steps it took to find the zero.
18. Apply Newton's method to the function g for each starting value listed below. For each one, explain what is strange about the result and why that happened.
    (a) x0 = 3
    (b) x0 = 3.5

A.4 PLOTTING DERIVATIVE INFORMATION

In this lab we will use Matlab to create plots that display lots of information:

1. Using different colors where the function is increasing (green) or decreasing (blue)
2. Using different symbols where the function is concave up (+) or concave down (x)

3. Putting special symbols at critical points (squares) and inflection points (circles)



Learning the Commands

We can do all of these things using the "plot" command, rather than the "ezplot" command. The "plot" command plots specific coordinates, rather than symbolically defined functions. For example we can plot a single point at $x = 3, y = 4$ with:

```
plot(3,4,'ob')
```

Here, the option 'ob' tells Matlab that we want to plot a circle that is blue in color.

Suppose we start by defining a function with:

```
syms x
f(x) = x^3 - 14*x^2 + 49*x - 36
```

We could plot a single point on this function at $x = 3$ as a green square with:

```
plot(3,f(3),'sg')
```

We can plot a set of points as red + signs at x coordinates going from 0 to 10 in steps of 0.3 with the following for-loop:

```
for( x=0:.3:10)
    plot(x,f(x),'+r')
    hold on
end
```

To see what other colors and symbols we can use, search for "linespec" in the Matlab help window (upper right of the command window), or google: matlab plot linespec

To plot different symbols and colors in different places, we will use Matlabs "if" command, which only runs a set of lines if a particular statement is true. For example, we could plot green x symbols where $y \geq -10$ and yellow diamonds at other points:

```
for( x=0:.3:10)
    if(f(x) >= -10)
        plot(x,f(x),'xg')
    else
        plot(x,f(x),'dy')
    end
end
```

We can even use "if" commands inside of other "if" commands. For example, we could plot green x symbols where $y > 10$, blue squares where $-10 \leq y \leq 10$, and yellow diamonds where $y < -10$:

```
for( x=0:.3:10)
    if(f(x) >= -10)
        if(f(x) > 10)
            plot(x,f(x),'xg')
```

```
        else
            plot(x,f(x),'sb')
        end
    else
        plot(x,f(x),'dy')
    end
end
```

If we want to plot special symbols at special points, we can use Matlabs solve command to locate these points for us. We can make these symbols very large by setting the "MarkerSize" inside the plot command. For example, we could plot size 20 magenta asterisks at points where a function crosses the x axis:

```
syms x
xz = solve(f(x))
plot(xz,f(xz),'*m','MarkerSize',20)
```

Note that if we have been using x to represent specific numbers, like we do in a for-loop, then we must turn x back into a symbolic variable with "syms x" before we can use the "solve" command to work with it algebraically.

The Laboratory Assignment

1. Write a Matlab program (a script) that begins by defining our function, clearing the figure, and telling it to hold on to points as we plot them:

   ```
   syms x
   f(x) = x^3 - 14*x^2 + 49*x  36
   clf
   hold on
   ```

Then, your script must do items 1, 2, and 3 listed at the beginning of the lab, going from 0 to 10 in steps of 0.3. Your program should produce a plot identical to the one at the beginning of the lab. Hint: To do item 3, it can be helpful to compute the derivatives of the function at the beginning of your program.

Next, make slight modifications of your code to create similar plots of the following functions:

2. $f(x) = 3e^(-x^2)$ going from -3 to 3 in steps of 0.2.
3. $f(x) = -x^4 + 2.2x^3 + 11.4x^2 - 12.6x$ going from -2.5 to 4.5 in steps of 0.25.
4. $f(x) = x^2e^(-x)$ going from 0 to 7 in steps of 0.1.

For each of these, please turn in both your plots and your programs.

A.5 RIEMANN SUMS

We have been studying the derivative, which tells us how quickly a function is changing, giving us the rate of change of a function. A closely related problem is to figure out the total amount of something, given its rate of change.

Heres a problem where this is important: During a time of scarce water, a town wants to monitor how quickly its water supply is being used, so they send someone down to measure the rate that water is flowing out of the reservoir throughout a day. We want to use this data to estimate the total amount of water that flowed out of the reservoir during this day. If the rate that water flowed was a constant (say 3000 gallons per hour), we could simply multiply this by 24 hours to get the total amount of water (3,000 gallons per hour x 24 hours = 72,000 gallons). Instead we will use Riemann Sums to estimate the total amount of water that flowed out of the reservoir during this 24 hour period.

The flow rates of the water out of the dam are well modeled by the following function for $0 \leq t \leq 24$:

$$f(t) = 100t^{1.5} \sin(0.2618t) + 10,000$$

Here the input to the function t is the number of hours since 10am, and the output from the function is in gallons per hour.

1. What units do the numbers 0.2618 and 10,000 have? What period of oscillation does the number 0.2618 correspond to?
2. Plot this function over a full day.
3. At what rate is water flowing out of the reservoir at 6pm?
4. If water flowed at this rate for the full 24 hour period, what would be the total amount of water that left the reservoir?
5. Estimate from the graph: At what time is the water flowing out of the reservoir at the slowest rate?
6. What is this minimum flow rate?
7. If water flowed at this minimum rate for the full 24 hour period, what would be the total amount of water that left the reservoir?
8. Now, write a program in Matlab that breaks the day into n=4 intervals and computes a left Riemann sum to estimate the total amount of water that flowed out of the reservoir over this 24 hour period.
9. Modify your program so that it breaks the day into n=4 intervals and computes a right Riemann sum to estimate the total amount of water that flowed out of the reservoir over this 24 hour period.
10. Average your two previous estimates in order to make a better estimate.

When we average the left sum and the right sum, we call this the trapezoid rule. We can do this directly if in interval we average the rate

at the left side and the right side before we multiply by the width of each interval. Modify your program to do this, and confirm that you get the same result as you get when you average your left and right sums.

11. Modify your program to estimate the total amount of water that flows out of the reservoir with the trapezoid rule using n=8 intervals.
12. Which of these two estimates do you expect will be more accurate the one based on n=4 or based on n=8? Explain your thinking.
13. Modify your program to estimate the total amount of water that flows out of the reservoir with the trapezoid rule using n=18 intervals.
14. Now, explore increasing the number of intervals until your result appears to converge to a final correct answer. Make a table listing all the values of n that you try, and each accompanying estimate.

B YOUTUBE PLAYLISTS

We have used YouTube to supplement programming instruction so that we can quickly get new students up to speed as well as remind students of skills that they learned in the past. We also use these videos as pre-lab activities so they are walking into the lab explorations ready to go. The following brief list points the interested reader to our relevant YouTube Playlists.

- MATLAB and LaTeX: https://www.youtube.com/watch?v=D21bj-Elli0&list=PLftKiHShKwSMnIbkq8nB-HNrlBXsyu04B
- Calculus: https://www.youtube.com/watch?v=aBYSq exMzUg&list=PLftKiHShKwSOpsOGvFldir0MIiPs6mdrc
- Multi-Variable Calculus: https://www.youtube.com/watch?v=Cr1oQ ukRu3Y&list=PLftKiHShKwSMPtP84j3yBoJ2tptOiRnZf

- Differential Equations and Linear Algebra: https://www.youtube.com/watch?v=8eSFdrsLQIo&list=PLftKiHShKwSPL6oZ_HMwQoOMmxqRBbd1O

C MULTI-VARIABLE CALCULUS

C.1 LAGRANGE MULTIPLIERS

In this lab we have the students solve a constrained optimization problem with Lagrange multipliers.

Problem Statement:

The spread of an aerosol in the atmosphere by a combination of diffusion and wind can be modeled by a function of three variables. If the coordinates are chosen so that the aerosol is released from the origin and the wind is blowing in the positive x-direction with velocity v_0. After a period of time, the concentration of the aerosol reaches a steady state independent of time at a point (x, y, z) that is given by[1]

$$S(x, y, z) = \frac{Q}{2\pi v_0 \sigma_y \sigma_z x^{2-n}} e^{-(y^2/\sigma_y^2 + z^2/\sigma_z^2)/(2x^{2-n})},$$

where Q is the rate of release of the chemical and σ_y, σ_z, and n are empirically determined constants. Distances are measured in centimeters, velocity in centimeters per second, and concentration in parts per cubic meter. The values of $\sigma_y = 0.4$, $\sigma_z = 0.2$, and $n = 0.25$ give good results for wind velocities less than $5m/s$ ($v_0 = 500cm/s$) and greater than $1m/s$ ($v_0 = 100cm/s$). Among other applications, this model has been used to model the diffusion of insect pheromones. Pheromones are "odor" chemicals that are released by animals for chemical communication within a species. For example, a single female gypsy moth, Q is on the order of 3×10^{13} particles per cm^3. Given that male gypsy moths

[1] This model is designed only to work for a limited range of velocities. Clearly, if $v_0 \to 0$ then this model given a non-physical (infinite) steady state concentration.

can detect as as few as 100 particles per cm^3, let us investigate the range over which a male gypsy moth can detect a female gypsy moth.

How would you use the method of Lagrange multipliers to detect the maximum distance downwind from a female gypsy moth over which a male gypsy moth can detect a female gypsy moth? In other words, how would you set up the optimization problem

$$\text{maximize: distance from female,} \quad \text{subject to: } S(x, y, z) = 100$$

using Lagrange multipliers? Write down the mathematical formulation and provide a thorough explanation describing what the Lagrange multiplier problem is solving. Implement the problem in MATLAB to find the maximum distance downwind over which a male gypsy moth can detect a female when the wind is blowing between 100 cm/s and 500 cm/s. You can safely make this problem simpler by assuming that the male is on the ground (setting z to 0). This removes 1 dimension from the problem and should give MATLAB's `vpasolve` command a fighting chance of finding solutions to your Lagrange multiplier system.

When you write your solution provide a plot showing the relationship between wind velocity (on the x axis) and the maximum separation distance (on the y axis). These values should be large and should cover many orders of magnitude, so be sure that your plot is readable with proper labels, titles, and thorough discussion.

C.2 VECTOR FIELDS

In this lab we have students write MATLAB code to generate visualizations of parameterized curves and vector fields.

1. **Plotting Parameterized Curves in MATLAB:**

   MATLAB is great for plotting parameterized curves. The following

instructions will give a MATLAB plot of the parameterized curve

$$\begin{cases} x(t) = 3\sin(t) \\ y(t) = 2\cos(t) \end{cases} \quad \text{for} \quad 0 \leq t \leq 2\pi$$

Follow these steps to get a plot:

- Define the numerical variable t on the domain $t \in [0, 2\pi]$. (Use a small step.)

```
t = 0:0.01:2*pi;
```

- Define the functions $x(t) = 3\sin(t)$ and $y(t) = 2\cos(t)$ using this numerical domain.

```
x = 3*sin(t);
y = 2*cos(t);
```

- To plot this function simply type `plot(x,y)`.
  this works because you've defined both x and y in terms of the parameter t. You should now see the resulting ellipse.
- Now we want to animate the parameterization. This requires the computer science concept of looping. Watch the following YouTube video to animate this parameterized curve.
  http://youtu.be/2viem03vXWI

  You don't need to write anything in your lab report for this problem.

2. **Creativity With 2D Parameterized Curves!**

Create parameterized curves that exhibit the behaviors below. There are many answers for some of these. Be sure to properly label all of your plots with obvious titles and describe your thought process for each problem in your lab writeup.

(a) A line that starts at the point $(1, 2)$ and ends at the point $(5, 7)$. Hint: Use the parameter $0 \leq t \leq 1$.

(b) A parameterized curve that starts at the point $(1, 2)$ and ends at the point $(5, 7)$ but does not go in a straight line.

(c) A parameterization of a circle with radius 3 and center $(2, 4)$ so that the circle is completely traced out on $t \in [0, 1]$.

(d) A spiral that converges in to the origin on the interval $0 \leq t \leq 2\pi$.

(e) Create a parameterized curve that describes the path a rider travels on a double ferris wheel (see https://www.youtube.com/watch?v=2DV4hN0c8WU Describe your thought process in writing this function.
Hint #1: put the main axis of the ferris wheel at the origin.
Hint #2: first model the motion of the axis of the small wheel, then use that as the center for the motion of the rider.
Hint #3: you get to pick the angular velocities.

3. **Vector Fields:**

We've plotted vector fields before, but most of the ones that we encountered thus far came from gradients. Now we'll define the vector field directly and plot it. You don't need to write this problem up in your lab report.

Let's get a plot of the vector field

$$\boldsymbol{F}(x, y) = \langle x\,,\, y^2 \rangle$$

(a) First define a mesh grid for $(x, y) \in [-1, 1] \times [-1, 1]$ (do you remember how to do the meshgrid command ... see lab 1!)

(b) The `quiver` command gives you the vector field. the syntax is
`quiver(xstart , ystart , xdirection , ydirection)`
for our problem, the vector field will be `quiver(x,y,x,y.^2)`

where `x` and `y` come from your `meshgrid` command.

Hint: You may want to use the `axis` command to make the plot look a bit nicer:

```
axis([xmin , xmax , ymin , ymax])
```

(c) You can think of a vector field in several ways. the three most common are

- The direction of a moving fluid
- The force field induced by a magnet (or by gravity)
- The potential lines in an electric field

In either case, you can think of the arrows as pointing to where a particle might move if you were to drop it into the vector field.

Using these analogies, discuss how a particle would behave if you dropped it into the vector field .

4. **Creativity with 2D Vector Fields!**

Create 2D vector fields that exhibit the behaviors below. There are many possible answers to these questions so be creative, label your plots clearly, and be sure to describe your thought processes. Some of these will take some experimentation!

(a) A vector field that mimics the earth's gravitational pull (google newton's law of universal gravitation if you have forgotten your physics).

Hint: Make the earth the origin and be sure to follow the inverse square law for gravitational fields!

(b) A vector field that appears to be a pan of water being stirred counterclockwise (as viewed from above).

(c) A vector field that mimics a swirling drain as viewed from above.

(d) A vector field on the domain $[-5,5]\times[-5,5]$ for which the flow does not penetrate these boundaries. For example, when $x=5$ or $x=-5$, the $\boldsymbol{i}$ component is zero.

5. **3D Vector Fields:**

   As a final task, turn all of the vector fields from problem 4 into 3D vector fields. Try to make them as physically realistic as possible. Be sure to explain your though processes.

D DIFFERENTIAL EQUATIONS AND LINEAR ALGEBRA

D.1 LORENZ SYSTEM

In this lab we have students create stand-alone function for Euler and Runge-Kutta numerical solvers. Students use these functions to solve systems of differential equations and to explore the Lorenz attractor.

1. Go back to Lab #2 and remind yourself how the 1D Euler and Runge-Kutta codes work. Go through the code with your lab partner in great details. Dissect every line so you know what it does.
2. **Euler's Method in 2D:** We wish to (numerically) solve

$$\frac{d\boldsymbol{x}}{dt}=\begin{pmatrix}f(t,x,y)\\ g(t,x,y)\end{pmatrix}\quad\text{subject to}\quad \boldsymbol{x}_0=\begin{pmatrix}x_0\\ y_0\end{pmatrix}$$

   where f and g are continuous functions. Euler's method simply approximates the slope, so we want to make approximations of x and y based on

$$x_{n+1}\approx x_n+\Delta t f(t_n,x_n,y_n)$$
$$y_{n+1}\approx y_n+\Delta t g(t_n,x_n,y_n)$$

   Create a new function in MATLAB called `MyEuler2D`. Copy your 1D code from Lab #2 and make all of the necessary changes.

3. **Runge-Kutta in 2D:** Now we are going to make Runge-Kutta solver that works in 2D. The Runge-Kutta 2D algorithm uses the following:

$$
\begin{aligned}
k_1 &= f(t_n, x_n, y_n) \\
h_1 &= g(t_n, x_n, y_n) \\
k_2 &= f(t_n + \frac{\Delta t}{2}\,,\, x_n + \frac{\Delta t}{2}k_1\,,\, y_n + \frac{\Delta t}{2}h_1) \\
h_2 &= g(t_n + \frac{\Delta t}{2}\,,\, x_n + \frac{\Delta t}{2}k_1\,,\, y_n + \frac{\Delta t}{2}h_1) \\
k_3 &= f(t_n + \frac{\Delta t}{2}\,,\, x_n + \frac{\Delta t}{2}k_2\,,\, y_n + \frac{\Delta t}{2}h_2) \\
h_3 &= g(t_n + \frac{\Delta t}{2}\,,\, x_n + \frac{\Delta t}{2}k_2\,,\, y_n + \frac{\Delta t}{2}h_2) \\
k_4 &= f(t_n + \Delta t\,,\, x_n + \Delta t k_3\,,\, y_n + \Delta t h_3) \\
h_4 &= g(t_n + \Delta t\,,\, x_n + \Delta t k_3\,,\, y_n + \Delta t h_3)
\end{aligned}
$$

Finally culminating with the weighted sums

$$
\begin{aligned}
x_{n+1} &= x_n + \frac{\Delta t}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\
y_{n+1} &= y_n + \frac{\Delta t}{6}\left(h_1 + 2h_2 + 2h_3 + h_4\right).
\end{aligned}
$$

Create a new function in MATLAB called `MyRungeKutta2D`. Copy your 2D code from Lab #2 and make all of the necessary changes.

4. Consider the following system of differential equations:

$$
\begin{cases}
x' &= y \\
y' &= -x + (1 - x^2)y
\end{cases}
\qquad \text{with} \quad x(0) = 1 \text{ and } y(0) = 2.
$$

This is a non-linear system so we don't have any analytic techniques to solve it!

   (a) Use the `pplane` applet linked from Moodle to get a phase plane diagram for the solution starting at $x = 1$ and $y = 2$.

(b) Use your `MyEuler2D` and `MyRungeKutta2D` codes to solve the system of differential equations numerically.

(c) Put plots of your Euler and Runge-Kutta solutions on top of each other. What do you observe about the Euler solutions as compared to the Runge-Kutta solutions?

5. **Some Animation:** Your job in this problem will be to create a MATLAB function called `DrawPath2D`. This code will be a modification to your Runge-Kutta code so start by making a copy of the code and giving it the proper file name and function name (no spaces in file or function names!!). Modify your code so that it plots a large point at the new iteration and leaves a small trail behind so you can see the trajectory through time. Your code should animate over two plots. The left-hand plot should have time on the x axis and both $x(t)$ and $y(t)$ plotted as functions of t. The right-hand plot should have x on the x-axis and y on the y-axis and trace out the phase portrait. Test your code on the system from the previous problem. (Ask your teacher to show his animation so you know what you're aiming for!)

6. In the 1960's the following system of differential equations was developed as a model for atmospheric convection:

$$\begin{cases} x' &= \sigma(y-x) \\ y' &= x(\rho - z) - y \\ z' &= xy - \beta z \end{cases} . \tag{1}$$

The equations are called the **Lorenz Equations** after the person who discovered them. An interesting set of parameters is $\sigma = 10, \beta = 8/3$, and $\rho = 28$.

(a) Create a new function `MyRungeKutta3D` that adapts your `MyRungeKutta2D` code to handle the third dimension. Solve this system of differential equations using the initial conditions $x(0) = 0.5$, $y(0) =$

0.2, and $z(0) = 10$. (Be sure to use LOTS of points in your solutions)

(b) Create a function called `DrawPath3D` by modifying your `DrawPath2D` and `MyRungeKutta3D` codes. Use your function to animate your solution to (a). The left-hand plot should show the three functions evolved over time and the right-hand plot should show the 3D phase portrait (be sure to use `plot3` instead of `plot`).

(c) Change the initial conditions to $x(0) = 0.51$, $y(0) = 0.21$, and $z(0) = 10.5$ and create a new 3D portrait. What effect does a small change in the initial conditions have on the solution?
Suggestion: Try plotting both solutions on the same 3D phase graph using different colors.
Once you have a qualitative way to talk about the error, get RK solutions for both sets of initial conditions and plot the *pythogorean distance* between the two solutions as a function of time (time on the x axis and distance on the y axis). Make any further observations about the behavior of this system.

D.2 COMPUTER ANIMATIONS

In this lab we introduce students to basic 2D and 3D graphics using linear transformations. The end result of the lab is a collection of animations that can be played to show their programming proficiency. A student's solution can be found here: https://www.youtube.com/watch?v=l-wmMQ8Kwcw&list=PLftKiHShKwSPL6oZ_HMwQoOMmxqRBbd1O&index=23

Two-Dimensional Computer Graphics

Viewing matrices as linear transformations is an important part of the utility of linear algebra. A **linear transformation** is a special type of function that acts on the elements of a vector space. Every linear

transformation T has the following properties:

$$T(\boldsymbol{u}_1 + \boldsymbol{u}_2) = T(\boldsymbol{u}_1) + T(\boldsymbol{u}_2) \tag{2}$$

$$T(c\boldsymbol{u}) = cT(\boldsymbol{u}) \tag{3}$$

where $\boldsymbol{u}, \boldsymbol{u}_1, \boldsymbol{u}_2$ are vectors in a vector space V and c is a scalar. If the vector space V contains elements of $\mathbb{R}^n$ then it can be shown that for every linear transformation T on V there is a matrix A where $T(\boldsymbol{u}) = A\boldsymbol{u}$. Hence, matrix multiplication is just a linear transformation of vectors.

In this lab we will see how several linear transformations can be combined to create simple computer generated animations. During the course of this lab, I will ask you to create several different functions to apply a variety of different linear transformations.

We can generate images of simple geometric shapes in `MATLAB`by defining their vertices and using `MATLAB`to "connect the dots" for us. For instance, to draw a simple triangle in two dimensions, we could use the following code:

```
myTriangle=[0 0; 0 1; 1 0; 0 0]';
plot(myTriangle(1,:),myTriangle(2,:));
axis([-2,2,-2,2]);
```

1. Create a set of points `myShape` for a convex polygon with one vertex at the origin.

If we want to make the triangle larger, we can apply a *scaling* linear transformation to each of the points that define the triangle.

Scaling Matrices in Two Dimensions: A *scaling* transformation in two dimensions acts to elongate each vector in $\mathbb{R}^2$. More precisely, given a vector $\vec{x} = \langle x, y \rangle \in \mathbb{R}^2$, we define a transformation $S_{a,b}(\vec{x}) = \langle ax, by \rangle$. We can show (and you should be able to prove) that such a

linear transformation can be attained using a matrix

$$S = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}.$$

2. Create a new function called `Scale2.m`. This function should take as input a two real values a and b and should return a 2×2 scaling matrix that scales the x component by a and the y component by b. For example, to scale our triangle by a factor of 2 in the x direction and 3 in the y direction, you should be able to use

```
s = Scale2(2,3);
newTriangle = s{*}myTriangle;
plot(newTriangle(1,:),newTriangle(2,:));
axis([-2,4,-2,6]);
```

3. In your master script, use your `Scale2` function to plot your shape scaled by

   (a) a factor of 2 in both the x and y directions

   (b) a factor of $\frac{1}{2}$ in the x direction and 3 in the y direction.

4. In your master script, write a loop to animate the scaling of your object by the function $f(\theta) = \cos\theta$ on the interval $[0, 2\pi]$. (that is, for each step, you should call `Scale2`$(\cos\theta, \cos\theta)$.)

If we want to rotate the triangle (about the origin), we can apply a *rotation* linear transformation.

Rotation Matrices in Two Dimensions: A *rotation* transformation in two dimensions acts to rotate each vector in $\mathbb{R}^2$ by a fixed angle. To produce a counter-clockwise rotation by θ radians, we can show (and you should be able to verify) that the transformation matrix must have

the form

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

5. Create a new function called `Rotation2.m`. This function should take as input a single real value θ and should return a 2×2 rotation matrix for θ.
6. In your master script, use your `Rotation2` function to plot your shape rotated by
   (a) $\frac{\pi}{4}$ radians counter-clockwise about the origin
   (b) $\frac{\pi}{3}$ radians clockwise about the origin.
7. In your master script, write a loop to animate the rotation of your object one time about the origin.

Homogeneous Coordinates

Scaling and rotation matrices are examples of linear transformations. Another fundamental aspect of computer graphics is *translation*. Unfortunately, translation is (strictly speaking) not a linear transformation. We *can*, however, still use matrix multiplication to represent translation– we just need to use special coordinates.

Homogeneous Coordinates: Each point $(x, y) \in \mathbb{R}^2$ can be identified with the point $(x, y, 1)$ in $\mathbb{R}^3$ that lies one unit above the xy-plane. We say that (x, y) has *homogeneous coordinates* $(x, y, 1)$. Using these homogeneous coordinates, we can now define a translation matrix.

The translation of a point (x, y) to $(x + h, y + k)$ can be computed using homogeneous coordinates and a 3×3 matrix:

$$\begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + h \\ y + k \\ 1 \end{bmatrix}.$$

Rotation, Translation, and Scaling using Homogeneous Coordinates:

To rotate a vector $\vec{x}$ by θ radians counter-clockwise about the origin, we use the homogeneous coordinate rotation matrix

$$R\vec{x} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \vec{x}.$$

To scale a vector $\vec{x}$ by a in the x-direction and b in the y-direction, we use the homogeneous coordinate scaling matrix

$$S\vec{x} = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} \vec{x}.$$

To translate a vector $\vec{x}$ by h units in the x-direction and k units in the y-direction, we use the homogeneous coordinate translation matrix

$$T\vec{x} = \begin{bmatrix} 1 & 0 & h \\ 0 & 1 & k \\ 0 & 0 & 1 \end{bmatrix} \vec{x}$$

Combining Transformations:

We may combine several transformations in sequence by repeated matrix multiplication. For instance, if we wish to apply a rotation R followed by a translation T to a vector $\vec{x}$, we could compute

$$TR\vec{x}.$$

Be careful–the order of operations matters here!

8. Create new functions `Scale2h.m`, `Rotate2h.m`, and `Tranlsate2h.m` that create 3×3 matrices for scaling, rotation, and translation, respectively.

9. When we rotated your shape in parts 5 and 6 you should have noticed that the rotation was about the origin and not the center of the shape. To make the shape rotate about its center we compose two operations: (1) translate the shape to the origin, (2) rotate the shape, (3) translate the shape back to it's original coordinates. Create a loop to animate the rotation of your shape once about its center.
10. OK–let's put it all together now. Create a loop that animates your figure by rotating it once about its center while scaling it in both directions by the function $f(\theta) = 2-\cos(\theta)$ (over the interval $[0, 2\pi]$) and translating it from the origin to the point $(2\pi, 2\pi)$. (Hint: Try doing each action individually first. Then you can combine them by multiplying the appropriate matrices together).

Homogeneous Coordinates in Three Dimensions: Each point $(x, y, z) \in \mathbb{R}^3$ can be identified with the point $(x, y, z, 1)$ in $\mathbb{R}^4$ that lies one unit away from xyz-space. (It's really hard to visualize this, though...). We say that (x, y, z) has *homogeneous coordinates* $(x, y, z, 1)$. As we did in the two-dimensional case, we can now define our linear transformations in terms of the homogeneous coordinates.

Rotation, Translation, and Scaling in Three Dimensions using Homogeneous Coordinates:

To rotate a vector $\vec{x}$ by θ radians counter-clockwise about the z-axis, we use the homogeneous coordinate rotation matrix

$$R_z\vec{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{x}.$$

To rotate a vector $\vec{x}$ by θ radians counter-clockwise about the y-axis, we

use the homogeneous coordinate rotation matrix

$$R_y\vec{x} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{x}.$$

To rotate a vector $\vec{x}$ by θ radians counter-clockwise about the x-axis, we use the homogeneous coordinate rotation matrix

$$R_x\vec{x} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{x}.$$

A general rotation about an arbitrary axis can be written as a product of the rotations about each axis:

$$R = R_z R_y R_x.$$

To scale a vector $\vec{x}$ by a in the x-direction, b in the y-direction, and c in the z-direction, we use the homogeneous coordinate scaling matrix

$$S\vec{x} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{x}.$$

To translate a vector $\vec{x}$ by h units in the x-direction, k units in the y-direction, and l units in the z-direction, we use the homogeneous coordinate translation matrix

$$T\vec{x} = \begin{bmatrix} 1 & 0 & 0 & h \\ 0 & 1 & 0 & k \\ 0 & 0 & 1 & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \vec{x}$$

Combining Transformations:

We may combine several transformations in sequence by repeated matrix multiplication. For instance, if we wish to apply a rotation R followed by a translation T to a vector $\vec{x}$, we could compute

$$TR\vec{x}.$$

Be careful–the order of operations matters here!

11. Create new functions `Scale3h.m`, `Rotate3hx.m`, `Rotate3hy.m`, `Rotate3hy.m`, and `Tranlsate3h.m` that create 4×4 matrices for scaling, rotation, and translation, respectively. Test your functions on the cube with vertices at the origin, (1,0,0), (0,1,0), and (0,0,1).

```
myCube = [0 0 0 1; 0 0 1 1; 0 1 1 1; 0 1 0 1;
          0 0 0 1; 1 0 0 1; 1 1 0 1; 0 1 0 1;
          0 0 0 1; 0 0 1 1; 1 0 1 1; 1 0 0 1;
          1 1 0 1; 1 1 1 1; 0 1 1 1; 0 0 1 1;
          1 0 1 1; 1 1 1 1; 1 1 0 1; 0 1 0 1;
          0 1 1 1; 1 1 1 1; 1 0 1 1; 1 0 0 1;
          1 1 0 1]';
```

12. Create an animation that demonstrates your functions. Your animation should, in sequence, show your cube:

    (a) rotate once about the x-axis;

    (b) rotate once about the y-axis;

    (c) rotate once about the z-axis;

    (d) rotate once about its center (using any axis you like);

    (e) scale (in all directions) by the function $f(\theta) = 2 - \cos\theta$;

    (f) translate around in a circle defined parametrically by $(\sin\theta, 1 - \cos\theta, 0)$.

13. Create an animation of your own choosing. You may choose to use any shape and apply any transformations you like. The coolest animation will get a bonus of 5 points on their overall lab grade. Have fun with it!

E UPPER LEVEL

E.1 DIFFERENCE EQUATIONS AND COMPLEX NUMBERS

Consider the difference equation $z_{n+1} = z_n^2 + z_0$ that starts with a complex number z_0. Let us define the set of complex numbers S as the set of all z_0 such that $\lim_{n\to\infty} |z_n|$ is bounded.

1. Demonstrate whether or not $z_0 = -1$ is a member of this set.
2. Demonstrate whether or not $z_0 = \frac{1}{3}$ is a member of this set.
3. Demonstrate whether or not $z_0 = i$ is a member of this set.
4. Demonstrate whether or not $z_0 = -1 + \frac{1}{4}i$ is a member of this set.
5. Prove that if $|z_0| \leq \frac{1}{4}$ then $z_0 \in S$.
6. Prove that if $|z_0| > 2$ then $z_0 \notin S$.
7. Prove that if z_0 is in the set, then $\bar{z}_0$ is also in the set.
8. Now, write a program in Matlab that applies this difference equation to a Cartesian grid of points in the complex plane representing many values of z_0. Have your program plot out the points based on the logarithm of the number of iterations it takes for the magnitude of each z_0 to exceed 2. I recommend creating a Cartesian grid of points in the complex plane with the help of the meshgrid command, applying this difference equation with a for loop, and counting the number of cycles each point has a magnitude less than 2 by using the Boolean: count = count + (abs(z) < 2). The logarithm of this

matrix can then be plotted with the imagesc command. Please turn in the code, the m file for your Matlab program.