COMP 116
February 15th, 2012

Assignment #4: Functions & Conditionals, due February 27th

Submit your published PDF along with a zip file containing all your code on Sakai.
In your submission, you must include:

- A publishable MATLAB file named **Assignment4.m** that demonstrates your solutions to problems #1 and #2 as described below.
- A published, readable PDF of the Assignment4.m script that summarizes your functions, graphs, and test cases.
- All MATLAB files (*.m) you created to solve this assignment, including any individual MATLAB function or script files, which should be included in a single .zip file.

Make sure you submit everything for this assignment by the end of the day on Feb 27th.

Problem #1: Gas Tank Volume

Background: A *piecewise function* is a function that is broken into multiple pieces or intervals. Each interval is covered by a different function over that interval.
The `sign` function is a good example of such a piecewise function.
Mathematically the **sign** function works like →

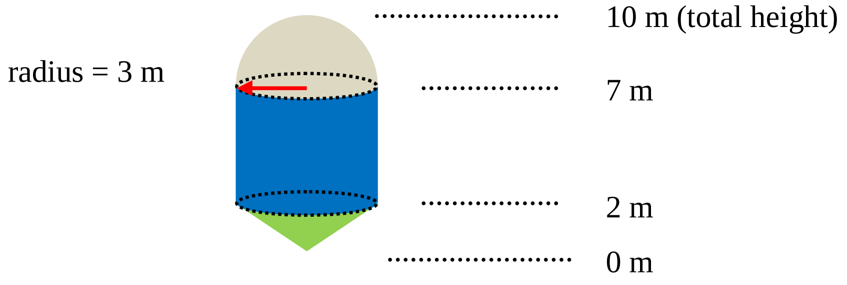
$$sign(x) = \begin{cases} -1, if\ x < 0 \\ 0, if\ x = 0 \\ +1, if\ x > 0 \end{cases}$$

We could implement this function like this ↓

```
function [result] = my_sign( x )
% returns {-1,0,+1} depending if x is negative, zero, or positive.
if x < 0
   result = -1;   % negative x, sign(x) = -1
elseif x > 0
   result = +1;   % positive x, sign(x) = +1
else              % by process of elimination ...
   result = 0;    % x must be zero, sign(x) = 0
end % end the "if" statement
end % end the function declaration
```

Note: the last two "end" comments were included for clarity. Please do **not** comment "end" statements in your code. If your "if" statements become so long that it's difficult to keep track of which section you're in when you reach an "end" statement, you should probably consider reorganizing your code to make the "if" statements shorter and more readable, either by extracting sections of code into sub-functions or by reconsidering the process you use to compute your result.

Problem Statement:
A gas station has a gasoline tank whose structure consists of 3 parts stacked vertically: The upper part is a hemisphere of radius 3 meters. The middle part is a right circular cylinder of radius 3 meters, and height 5 meters. The lowest part is an upside down right circular cone of radius 3 meters and height 2 meters.



We would like to measure the volume of the gas tank **in liters** as a function of the current height, where current height is measured in meters. Assume that the tank is stored vertically, the bottom of the tank is at height zero, and the very top of the tank is at 10 meters. Assume that there are **1000 liters per cubic meter of volume**. The density of real gasoline actually varies slightly based on temperature. For the purpose of this problem, we will ignore any such complications.

Part 1A: Volume of the Gas Tank

Create a piecewise function to compute the volume of the gas tank **in liters** as a function of the current height in meters. This function should return correct values for any real-number input. The way to do this is to divide the range of all possible inputs for the current height (i.e. all real numbers) into 5 intervals: heights which fall below the bottom of the tank (any non-positive height), heights which fall within the bottom part of the tank (the inverted cone), the middle part of the tank (the cylinder), the top part of the tank (the hemisphere), and heights above the top of the tank (greater than the total height of the tank).

Declare your function like this:

```
function [liters] = GasTankVolume( currHeight )
   % your code goes here . . .
end
```

Tip: It might be useful to create helper functions that compute the volume of a truncated hemi-sphere, the volume of a right circular cylinder, and the volume of a right circular cone. Make sure to name these functions clearly – it will make the code that uses them much more readable.

Comments:
Most of your code for this assignment will reside in functions. Each function should be well-commented and well-structured such that someone who knows the problem description but not necessarily the approach you used (i.e. the grader) can follow your logic and clearly see that your solution is correct.

In Assignment4.m (your main publishable script file), write a brief, high-level description in MATLAB comments for how your **`GasTankVolume`** function works. Also include the function names and brief (1-to-2 sentence) descriptions of each helper function you created to help solve this problem. The declarations of the helper functions should also be well-commented, but the comments in Assignment4.m should only explain the general purpose of each such helper function in simple English.

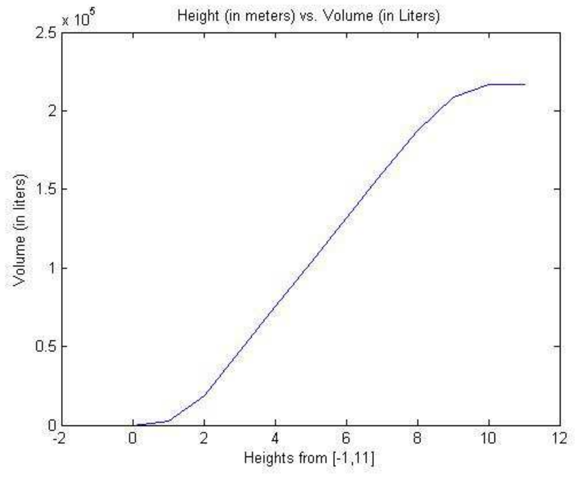
For example, consider a function that computes a geometric mean, GeomMean.

High-level description (in Assignment4.m):	**Low-level description (in the actual function definition)**:
The function GeomMean computes the geometric mean of a set of numbers. We use this function to...	[nthRoot] = GeomMean(V) The function GeomMean expects a single vector of numbers 'V' as an argument. It finds the product of all N numbers in the vector and returns the Nth root of the product.

Part 1B: Plotting Volumes from Current Height

In Assignment4.m, create a 1D vector of current height values in the range [-1,11] meters using step size increments of a full meter. Generate a corresponding volume vector by calling your **`GasTankVolume`** function on each scalar element in the 1D height vector. Plot the computed volume (in liters) against the current height (in meters).

Your resulting plot should look like the following



Note: Conditional logic statements like (`if` and `switch`) don't work as you might expect with vectors and matrices: every element in the vector or matrix must return true for the logical expression as a whole to be considered true (and for that part of the 'if' statement to execute). This means that you must call your **`GasTankVolume`** one element at a time (1 scalar argument at a time) instead of all at once using a 1D vector (or 2D matrix) to get it to behave the way you would expect. One way to accommodate for this is to use a looping structure like '`while`' or '`for`':

```
volumes = zeros( 1, length( heights ) );
for idx = 1:length( volumes )
      volumes(idx) = GasTankVolume( heights(idx) );
end
```

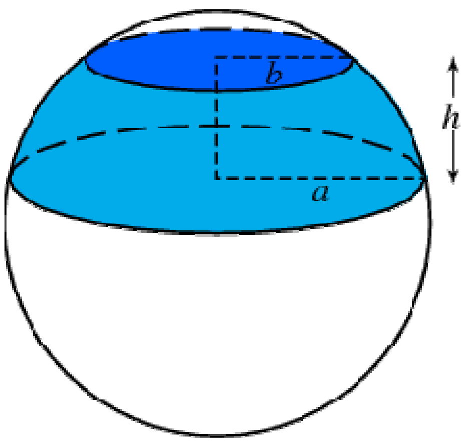
MATLAB is actually quite slow (compared to other programming languages) when it comes to executing 'if' statements and loops like 'for' and 'while', but not so slow that you'll notice on this assignment. Since MATLAB has so much built-in functionality for working with arrays and matrices, the truly correct way to do this would be to rewrite the **`GasTankVolume`** function to take a matrix of heights and return a matrix of the same size containing the corresponding volumes. However, doing this correctly is needlessly tricky at this stage – please only implement **`GasTankVolume`** for scalars and use the loop provided above.

Note: The cone is inverted, so you also might need to compute the current radius of the truncated cone from the maximum height, current height, and base cone radius.

Note: The volume V of a spherical segment with two parallel planar faces can be computed as follows:

$$V = \frac{1}{6}\pi h(3a^2 + 3b^2 + h^2)$$

Where *a* and *b* are the radii of each of the parallel plane faces and *h* corresponds to the height of the spherical segment. One of the radius values *a* is fixed at 3 meters. The other radius *b* can be computed from the actual sphere radius '*r*' and the height *h* of the segment.



Note: Piecewise functions don't have to be written using just one "if" statement. Sometimes, it may make your code cleaner to break up the calculation of the result into multiple "if" statements. Consider the following function:

$$f(x) = \begin{cases} x, if\ x < 0 \\ 2x, if\ 0 \leq x < 3 \\ 3x - 3, if\ x \geq 3 \end{cases}$$

One way to implement this function is:

```
function [result] = f1( x )
if x < 0
   result = x;
elseif 0 <= x && x < 3 % Note: must make each comparison separately
   result = 2 * x;
else
   result = 3 * x - 3;
end
end
```

However, we can see that if each piece of the piecewise function somehow "builds" on the previous part, we can be a little clever and do the following:

```
function [result] = f2( x )
result = x; % All results have at least one "x" in them
if x > 0
   % All results with x greater than 0 have at least "2x" in them,
   % so add "x" to the current result to get x + x == 2 * x
   result = result + x;
end

if x >= 3 % Note: this is a new 'if' statement, separate from above
   % If x >= 3, then x > 0, which means the above "if" statement must
   % have executed, so "result" currently contains 2 * x. To get
   % 3 * x – 3, we therefore only need to add x – 3 to result!
   result = result + x - 3;
end
end
```

Problem #2: Parking Rates at SLC Airport

Most major airports have lots for both long-term and short-term parking. The cost to park depends on which lot you pick to park, and how long you stay. The following parking rate structure is from the Salt Lake International Airport as of Sept 28th, 2009. (The pay structure has changed slightly since then, but is still much the same: http://www.slcairport.com/parking.asp.) We'll be using the old values, from 2009.

Economy Parking (Long Term)

- The first hour is $0.00
- The second hour is $2.00
- Every additional hour is $1.00
- Daily maximum is $7.00
- Weekly maximum is $49.00

Note: Any fractional left-over parts of an hour should be charged as if you stayed the whole extra hour. So 1.2 hours rounds up to 2 hours, and costs $2.

Hourly / Daily Parking (Short Term)

- The first 30 minutes is $0.00
- 31-50 minutes is $2.00
- Every additional 20 minutes is $1.00
- Daily maximum is $28.00

Note: Any left-over minutes larger than zero and less than 20 should be charged as if you stayed the entire 20-minute period (as long as you've already stayed more than 50 minutes).

Part 2A: Long Term Parking

Write a function for correctly computing the ***long term parking rate*** subject to the rules above. Create your function according to the following declaration:

```
function [rate] = LongTerm( days, hours )
```

Unlike the function defined in Part 1, you should disallow certain inputs to this function:

- The `LongTerm` function should call `error` with a descriptive message for invalid input values of 'days' and 'hours'.
  - Days should be in the range [0, 180]. That is, we assume that it is *impossible* to park for a negative number of days and *unreasonable* to leave your car in long term parking for more than 6 months.
  - Hours should be in the range [0, 24). That is, every value between zero inclusive and 24 exclusive (so 23.999 is a legal value for 'hours', but not 24).

Note: Once **`error`** is called, none of the remaining code in a function will be run until the function is called again, so calls to **`error`** should probably only occur within "if" statements.

Testing your Long Term Function:
Come up with 3 reasonable test cases to verify your code is working properly. A test is a specific combination of legal values for 'days' and 'hours' for which you can work out the ***expected*** answer by hand, and then verify that **`LongTerm`** computes the answer correctly when given these inputs. Include your test cases in Assignment4.m like this: (but choose your own values!)

```
% Test case 1: a stay of 27 days, 3 hours should result in
% an expected rate of $192
testRate1 = LongTerm( 27, 3 );
expectedRate1 = 192;
if (testRate1 ~= expectedRate1)
    error('Error: computed rate didn't match expected rate');
end
% Other test cases . . .
```

Note: If your test cases don't work, this implies that your code may contain a bug, or you calculated the expected answer incorrectly. You should try using the debugger to step through your code when trying to figure out what is going wrong.

Part 2B: Short term parking

Write a function for computing the ***short term parking rate*** subject to the rules above. Create your function according to the following declaration:

```
function [rate] = ShortTerm( days, hours, minutes )
```

- Similar to the **`LongTerm`** function, you should do simple validation on the inputs 'days', 'hours', and 'minutes' and call **`error`** with a descriptive message when you determine that any of the inputs is invalid.
- Minutes should be in the range [0, 60).
- Hours should be in the range [0, 24).
- Days should be in the range [0, 60]. That is, we assume it is ***unreasonable*** to leave your car in short term parking for more than 2 months.

Testing Your Short Term Function:
Come up with 3 test cases for your **`ShortTerm`** function to help verify your code is working properly. Include your test cases in Assignment4.m in a format similar to those in Part 2A.

Part C: Robustness & User Input

In this part, you will write two functions: one that transforms illegal days-hours-minutes combinations into legal ones, and another that takes (potentially illegal) input from the user and determines the rates for both long term and short term parking.

Create the following function definition:

```
function [validDays, validHours, validMinutes] = ValidateTime( days, hours, minutes )
```

This function takes values for days, hours, and minutes, and translates them into more appropriate values, or outputs an error message:

- *Invalid inputs:* If any input is negative or empty (that is, an empty matrix), an error message should be generated by calling `error`.
- *Irregular inputs:* Inputs which are too large for their respective ranges should be handled by converting them into a more valid form:
  - *Example 1:* If you called this function with 5 days, 73 hours, and 66 minutes, it should convert this input into a more understandable form by setting validDays to 8, validHours to 2, and validMinutes to 6.
  - *Example 2:* If you called this function with 2.75 days, 1.33 hours, and 13 minutes, it should convert this into a more regular format, with validDays set to 2, validHours set to 19, and validMinutes set to 32.8.

You should implement this function by converting the input days, hours, and minutes into 'totalMinutes', and then calculating validDays, validHours, and validMinutes from totalMinutes.

Next, write a simple function to get the length of the stay from the user using the `input` function. Three inputs are required: *days*, *hours*, and *minutes*. You can call `input` 3 times to retrieve each value individually, or just once to get a vector of length 3. The number of days, hours, and minutes should be validated (by calling ValidateTime) and then the validated time values should be used to compute the total rate that the user would be charged for stays in both the long term and short term parking lots. When calculating long term parking, you should add the number of minutes as a fractional value into the number of hours (so 2 hours and 6 minutes is 2 + 6/60 = 2.1 hours)

Since this function gets its inputs from the user and prints its outputs to the console, it doesn't need any parameters and doesn't return any values. Declare this function like this:

```
function CalcParkingRates()
```

Below is an example of what could be outputted on a sample run of `CalcParkingRates`. **NOTE**: the output that *asks* the user for input is omitted here.

You entered 5 days, 73 hours, 66 minutes.
Staying in Long Term Parking for 8 days, and 2.1 hours will cost you $59.
Staying in Short Term Parking for 8 days, 2 hours, 6 minutes will cost you $230.

Display text output using the **`fprintf`** command. To move to the next line, print the newline character '\n'. So to print "Hi" and then move to the next line of text, run fprintf('Hi\n');

Testing (on your own):
Call **`CalcParkingRates`** several times and try entering different values for days, hours, and minutes. Deliberately see if you can break your program. Try invalid inputs (negative numbers, very large numbers, empty answers, etc.). The program should give appropriate error messages in these cases. Try irregular combinations (1 day, 121 hours, 2017 minutes). The program should convert irregular combinations into regular combinations that will work when you call your long term / short term functions. The goal is for you to find and fix as many problems as soon as possible, and to make your program as unbreakable as possible, so that when you're done testing it, you're fairly confident that your code is correct.

Part D: Plotting Parking Rates

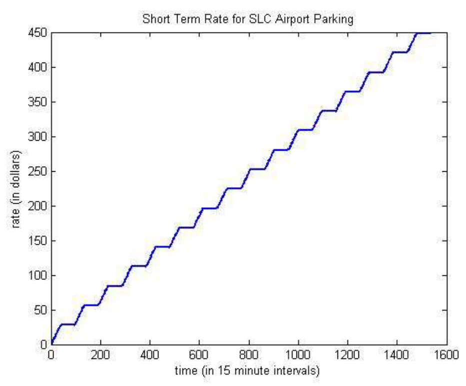
Create an input time vector with 3 rows for days, hours, and minutes that represents increasing time in 15 minute increments starting from a 0 minute stay up through 15 total days. This vector should have 1536 columns all together (16 days including day zero * 24 hours per day * 4 15-minute increments per hour). In Assignment4.m, compute the short term and long term rates in an output rate vector with two rows (row 1 = short term rate, row 2 = long term rate).

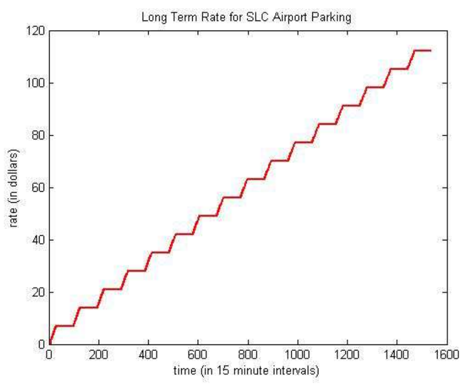
Create two plots. One plot should be of time vs. rate for short term parking. The other plot should be of time vs. rate for long term parking.

Note: Again, conditional statements work best with scalars and don't work so well with vectors or matrices. You can use the following looping structure to work around this problem by calling your functions one element at a time.

```
% Assume time is a 3 x 1536 matrix: [days; hours; minutes]
rates = zeros( 2, length( time ) );
for idx = 1:length( time )
    days = time( 1, idx );
    hours = time( 2, idx );
    minutes = time( 3, idx );
    rates( 1, idx ) = ShortTerm( days, hours, minutes );
    rates( 2, idx ) = LongTerm( days, hours + minutes/60);
end
% Code for plotting the rates . . .
```

Here is an example of what these plots look like:





Tips

You might find the functions **`min`**, **`max`**, **`floor`**, **`ceil`**, and **`mod`** useful for several parts of this assignment.

You should think about how you will solve this problem up front before starting to code up your solution. It may be helpful to write down your approach to solving this problem in simple English before you try to turn that into actual MATLAB code.

Try to use "divide and conquer": break the original problem into simpler, more manageable sub-problems. This process can be helpful when programs get bigger. Feel free to create small helper functions to accomplish simple tasks.

Here is a complete list of the things you must submit for this assignment

- Assignment4.m script
  - High-level explanation of how GasTankVolume works, and names and descriptions of helper functions (if any)
  - Plot of tank height vs. tank volume
  - 3 test cases for LongTerm
  - 3 test cases for ShortTerm
  - Plots of short term and long term parking rates
- Published PDF of Assignment4.m
- A .zip file containing all MATLAB files you created for this assignment. All functions should have comments that explain how they work.
  - GasTankVolume
  - LongTerm
  - ShortTerm
  - ValidateTime
  - CalcParkingRates
  - Any other helper functions you defined