Functions, Nested Loops, and Recursion

MATH-151: Mathematical Algorithms in Matlab

September 11, 2023

Why Use Functions?

- A lot of time when writing longer pieces of code, there will be repeated blocks of code. Functions allow us to make our code more readable by replacing these repeated blocks of code with a single line that tells Matlab to perform that block.
- For example, `sin(x)` informs Matlab to do a Taylor Series sum to approximate the $sin(x)$ function.
    - It is much more less cluttered to see just `sin(x)`, rather than a loop, many times.
    - Furthermore, we would see `factorial(n)` in each loop of that sum, which is another function that replaces a set of code!
- Matlab has many built-in functions to perform commonly seen computations, but it also allows us to create our own functions!

Structure of a Function

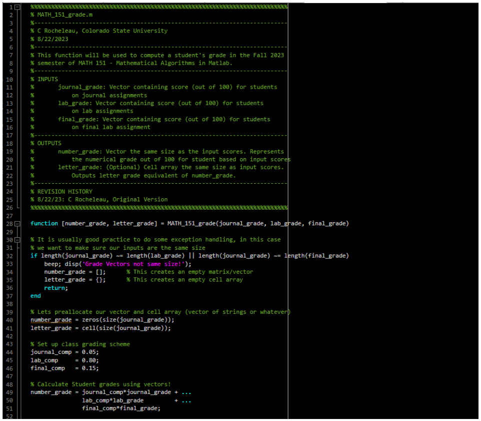
- Similar to a script a function is a .m file in Matlab however we need to inform Matlab that is is a function. So each function code needs to begin with a line like below

  ```
  function output = function_name(input)
  ```

  - function tells Matlab that we are beginning to define a function
  - `output` tells Matlab what the result of the function is, everything else will be deleted from memory when the function is completed
  - `function_name` is the name used for the function call in another piece of code telling Matlab when to run this block of code. The filename for this function should match the function name, so this file would be named function_name.m
  - `input` is the input to the function, the code should be expected to run with only the given inputs.
- It is usually good commenting practice to include a preamble describing the function's use and format of the input and output variables.

Function Example

Here is an example script for a function to calculate your grade in this class

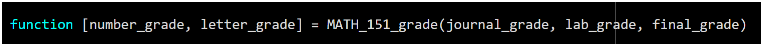


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MATH_151_grade.m
%--------------------------------------------------------------------------
% C Rocheleau, Colorado State University
% 8/22/2023
%--------------------------------------------------------------------------
% This function will be used to compute a student's grade in the Fall 2023
% semester of MATH 151 - Mathematical Algorithms in Matlab.
%--------------------------------------------------------------------------
% INPUTS
%       journal_grade: Vector containing score (out of 100) for students
%           on journal assignments
%       lab_grade: Vector containing score (out of 100) for students
%           on lab assignments
%       final_grade: Vector containing score (out of 100) for students
%           on final lab assignment
%--------------------------------------------------------------------------
% OUTPUTS
%       number_grade: Vector the same size as the input scores. Represents
%           the numerical grade out of 100 for student based on input scores
%       letter_grade: (Optional) Cell array the same size as input scores.
%           Outputs letter grade equivalent of number_grade.
%--------------------------------------------------------------------------
% REVISION HISTORY
% 8/22/23: C Rocheleau, Original Version
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [number_grade, letter_grade] = MATH_151_grade(journal_grade, lab_grade, final_grade)

% It is usually good practice to do some exception handling, in this case
% we want to make sure our inputs are the same size
if length(journal_grade) ~= length(lab_grade) || length(journal_grade) ~= length(final_grade)
    beep; disp('Grade Vectors not same size!');
    number_grade = [];      % This creates an empty matrix/vector
    letter_grade = {};      % This creates an empty cell array
    return;
end

% Lets preallocate our vector and cell array (vector of strings or whatever)
number_grade = zeros(size(journal_grade));
letter_grade = cell(size(journal_grade));

% Set up class grading scheme
journal_comp = 0.05;
lab_comp     = 0.80;
final_comp   = 0.15;

% Calculate Student grades using vectors!
number_grade = journal_comp*journal_grade + ...
               lab_comp*lab_grade         + ...
               final_comp*final_grade;

```

Wait a second, what's going on with the inputs and outputs? Can you have more than one of each?

Multiple Inputs or Outputs

- If needed, we can allow a function to have multiple outputs, as well as accept multiple inputs.
  - Be careful, the ordering is important!
- To have multiple outputs, we place them in square brackets, [...], where each output variable is separated with a comma
  `[number_grade, letter_grade]`
- Similarly, to have multiple inputs we list them in the parentheses of the function call. Again, separated by commas
  `(journal_grade, lab_grade, final_grade)`



```
function [number_grade, letter_grade] = MATH_151_grade(journal_grade, lab_grade, final_grade)
```

- Inputs may be treated as optional in the function, or outputs may be optional when calling the function, for example
  `number_grade = MATH_151_grade(100,100,100)`
  will only output the number grade

Subfunctions

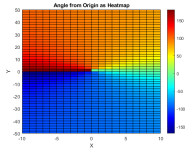
- Sometimes we will have a repeated block of code inside a function
    - That sounds like a job for a function!
- When needed, we can put multiple functions in the same file, if the primary function needs additional function calls.
    - For example, my simulation's physics model function had a subfunction for computing the amount of force each thruster created, rather than copy the same block of code for each of the 4 thrusters!
- Creating a subfunction is just like creating a function, the only difference is that we need to make sure that the functions are given `end` statements, so Matlab knows when one function ends and the other begins.

We'll see an example of this shortly, but while we're talking about functions inside functions...

Loops in Loops

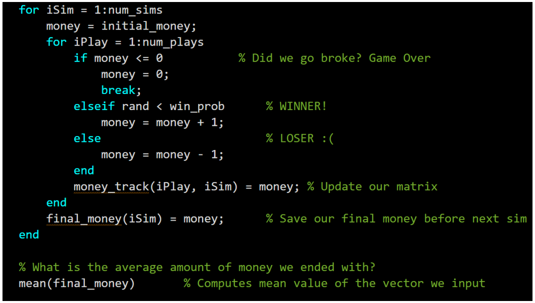
- In many real-world cases, we deal with multivariable functions. In those cases, we don't just have a single x that we need to iterate across, we also have a y that varies as well.
- To calculate all combinations of values from x and y we need to use a **nested loop**
  - This means that we have a loop inside of a loop
  - We hold the x value constant and iterate across all the y values, then we move onto the next x value and repeat.
- Suppose we have a grid of points and want to find the angle to each of them from the origin.

```
for iX = 1:length(x)
    for iY = 1:length(y)
        angles(iY,iX) = atan2d(y(iY), x(iX));
    end
end
```



Data Matrices

- This can also be used for Data Matrices. Each row is a different variable, and each column is a different subject.
- We can use a nested loop to perform our Gambler's Ruin game many times and collect statistics!

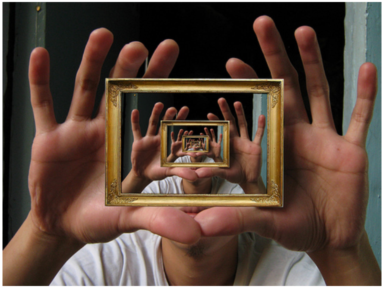


```
for iSim = 1:num_sims
    money = initial_money;
    for iPlay = 1:num_plays
        if money <= 0           % Did we go broke? Game Over
            money = 0;
            break;
        elseif rand < win_prob      % WINNER!
            money = money + 1;
        else                        % LOSER :(
            money = money - 1;
        end
        money_track(iPlay, iSim) = money; % Update our matrix
    end
    final_money(iSim) = money;      % Save our final money before next sim
end

% What is the average amount of money we ended with?
mean(final_money)       % Computes mean value of the vector we input
```

- In this game we had an average final money count of $5.31

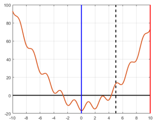
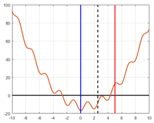
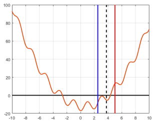
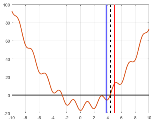
Recursive Relationships

- In some algorithms we will run into each step will be a tiny version of the algorithm itself. For example, when doing $n!$ we will need to find $(n-1)!$, which means we'll need to find $(n-2)!$, and so on.
- In these cases, rather than writing a loop, we can create a **recursive function**
    - This is a function that calls on itself
    - Each time we call the function, we will get closer to some base case
    - The answer will "flow up" through all the previous function calls to give us our desired result in the end.



Bisection Method

- This is an algorithm for finding the root, $f(x) = 0$ of an equation
- Suppose we know $f(a)$ is negative and $f(b)$ is positive. If f is a continuous function, is must pass through 0 between a and b.
- So we cut it in half and try $f(m)$, where $m = \frac{a+b}{2}$
  - If it has the same sign as $f(a)$, m becomes our new a
  - If it has the same sign as $f(b)$, m becomes our new b
- This gives us a new, smaller possible range, so we do Bisection Method on that range
- We continue until $f(m)$ is sufficiently small.


100
80
60
40
20
0
-20
-10 -8 -6 -4 -2 0 2 4 6 8 10

100
80
60
40
20
0
-20
-10 -8 -6 -4 -2 0 2 4 6 8 10

100
80
60
40
20
0
-20
-10 -8 -6 -4 -2 0 2 4 6 8 10

100
80
60
40
20
0
-20
-10 -8 -6 -4 -2 0 2 4 6 8 10