E7: Introduction to Computer Programming for Scientists and Engineers
University of California at Berkeley, Spring 2017
Instructor: Lucas A. J. Bastien

Lab Assignment 03: Branching
Version: release

Due date: Friday February 10th 2017 at 12 pm.

General instructions, guidelines, and comments:

- For each question, you will have to write and submit one or more Matlab functions. We provide a number of test cases that you can use to test your function. The fact that your function works for all test cases provided does not guarantee that it will work for all possible test cases relevant to the question. It is your responsibility to test your function thoroughly, to ensure that it will also work in situations not covered by the test cases provided. During the grading process, your function will be evaluated on a number of test cases, some of which are provided here, some of which are not.
- Submit on bCourses one m-file for each function that you have to write. The name of each file must be the name of the corresponding function, with the suffix `.m` appended to it. For example, if the name of the function is `my_function`, the name of the file that you have to submit is `my_function.m`. **Carefully check the name of each file that you submit.** Do not submit any zip file. If you re-submit a file that you have already submitted, bCourses may rename the file by adding a number to the file's name (*e.g.,* rename `my_function.m` into `my_function-01.m`). This behavior is okay and should be handled seamlessly by our grading system. Do not rename the file yourself as a response to this behavior.
- A number of optional Matlab toolboxes can be installed alongside Matlab to give it more functionality. All the functions that you have to write to complete this assignment can, however, be implemented without the use of any optional Matlab toolboxes. We encourage you to not use optional toolboxes to complete this assignment. All functions of the Matlab base installation will be available to our grading system, but functions from optional toolboxes may not. If one of your function uses a function that is not available to our grading system, you will loose all points allocated to the corresponding part of this assignment. To guarantee that you are not using a Matlab function from an optional toolbox that is not available to our grading system, use one or both of the following methods:
  - ◇ Only use functions from the base installation of Matlab.
  - ◇ Make sure that your function works on the computers of the 1109 Etcheverry Hall computer lab. All the functions available on these computers will be available to our grading system.

- For this assignment, the required submissions are:
  - ◇ `my_consumer_helper.m`
  - ◇ `my_water_quality.m`
  - ◇ `my_array_resize.m`
  - ◇ `my_sequence_id.m`
  - ◇ `my_elevator.m`

1. Coupons

In this question, you will write a function that helps customers choose the shopping coupon that yields the best discount on a purchase of their choice. There are three coupons to choose from:

1. The consumer can get a 10% discount on the value of the products that they are purchasing.
2. If the original value of these products is $200 or more, then the consumer can get $25 off their purchase.
3. If the original value of these products is $400 or more, then the consumer can get $50 off their purchase.

Only one of these coupons can be used at a time.

Write a function with the following header:

```
function [coupon, price] = my_consumer_helper(value)
```

where:

- `value` is a scalar of class `double` that represents the value (in dollars) of the products that the consumer wishes to buy.
- `coupon` is a scalar of class `double`. `coupon` can take one of the values `1`, `2`, or `3`, and indicates the coupon that yields the largest discount that the consumer can get for their purchase. The value `1` indicates the 10% discount, the value `2` indicates the $25 discount, and the value `3` indicates the $50 discount.
- `price` is a scalar of class `double` that represents the price (in dollars) that the consumer will pay after the discount from the selected coupon has been applied.

If two coupons yield the same discount, choose the coupon with the lowest number.

Test cases:

```
>> [coupon, price] = my_consumer_helper(60)
coupon =
     1
price =
    54
```

```
>> [coupon, price] = my_consumer_helper(200)
coupon =
     2
price =
   175

>> [coupon, price] = my_consumer_helper(390)
coupon =
     1
price =
   351

>> [coupon, price] = my_consumer_helper(425)
coupon =
     3
price =
   375

>> [coupon, price] = my_consumer_helper(550)
coupon =
     1
price =
   495
```

2. Assessing water quality

Water quality depends on many measurable physical and chemical parameters, including pH, turbidity, electrical conductivity (EC), and dissolved oxygen (DO) concentration. In this question, we simplify the process of assessing water quality down to assessing the water's pH and DO concentration only.

pH measures the concentration of hydrogen ions ($[H^+]$) in solution and indicates how acidic or basic a solution is. The pH scale typically ranges from 0 to 14. Low pH values are associated with acidic solutions whereas high pH values are associated with basic solutions. A pH of 7 is neutral. The pH of surface water typically falls between 6.5 and 8.5, but can vary due to precipitation and other environmental factors. Dissolved oxygen is a measure of the amount of oxygen molecules (O_2) that are dissolved in the water. Typical DO concentrations are several mg L^{-1} (milligrams of oxygen per liter of water, Nazaroff and Alvarez-Cohen, 2001, page 42). Many aquatic plants and animals rely on the dissolved form of oxygen for respiration. Low concentrations of dissolved oxygen in the water can be harmful to such organisms.

In this question, you will write a function that attributes a water quality score to a water body, given the water's pH and dissolved oxygen (DO) concentration. More precisely, write a function with the following header:

```
function [score, warning] = my_water_quality(ph, do)
```

where:

- `ph` is a scalar of class `double` that represents the pH of the water.
- `do` is a scalar of class `double` that represents the concentration of dissolved oxygen (in mg L^{-1}) in the water.
- `score` is a scalar of class `double` that represents the water quality score of the water body. More precisely, `score` should be the average of the pH score and the DO score as determined by `ph`, `do`, Table 1, and Table 2. If either `ph` is outside of the range of values described in Table 1 or `do` is outside of the range of values described in Table 2, then `score` should have the value `NaN`.
- `warning` is a scalar of class `logical`. If `score` is `NaN`, then `warning` should be true. In other cases, `warning` should be true if and only if the water is unsafe or unsuitable for the environment. In this question, we consider that the water is unsafe or unsuitable for the environment if and only if at least one of the following two conditions is met:
  - ◇ the pH of the water source is outside of the typical range for surface water, *i.e.* outside of the interval $[6.5, 8.5]$
  - ◇ the DO concentration in the water source is less than 4 mg L^{-1}

Table 1: pH score as a function of pH.

pH range	Qualitative Rating	pH score[a]
$6.5 \leqslant \text{pH} \leqslant 7.5$	Excellent	5
$7.5 < \text{pH} \leqslant 8$	Great	4
$8 < \text{pH} \leqslant 8.5$	Good	3
$6 \leqslant \text{pH} < 6.5$ or $8.5 < \text{pH} \leqslant 9$	Fair	2
$5.5 \leqslant \text{pH} < 6$ or $9 < \text{pH} \leqslant 9.5$	Poor	1
$0 \leqslant \text{pH} < 5.5$ or $9.5 < \text{pH} \leqslant 14$	Very poor	0

[a]: Scoring system adapted from the 2017 Mid-Pac Water Treatment Competition.

Table 2: DO score as a function of dissolved oxygen (DO) concentration.

DO concentration range (in mg L^{-1})	Qualitative Rating[a]	DO Score[b]
$7 \leqslant \text{DO} \leqslant 11$	"*Very good for most stream fish*"	5
$4 \leqslant \text{DO} < 7$	"*Good for many aquatic animals, low for cold water fish*"	3
$2 \leqslant \text{DO} < 4$	"*Only a few fish and aquatic insects can survive*"	1
$0 \leqslant \text{DO} < 2$	"*Not enough oxygen to support life*"	0

[a]: Qualitative ratings from: http://fosc.org/WQData/WQParameters.htm, retrieved on February 2nd 2017.
[b]: Scoring system adapted from the 2017 Mid-Pac Water Treatment Competition.

Test cases:

```
>> [score, warning] = my_water_quality(7, 4)
score =
     4
warning =
  logical
   0

>> [score, warning] = my_water_quality(6.2, 3)
score =
    1.5000
warning =
  logical
   1

>> [score, warning] = my_water_quality(8.5, 2)
score =
     2
warning =
  logical
   1

>> [score, warning] = my_water_quality(15, 10)
score =
   NaN
warning =
  logical
   1
```

3. Array resizing

Resizing an array in Matlab is a costly operation. Having your computer code to frequently resize arrays can increase its execution time significantly. In many applications, a better approach is to create an "empty" array of the desired size from the beginning, and to update the values in this array as necessary. Here, we call an "empty" array an array that contains only zeros. There are still applications where it is necessary to resize arrays despite the cost of this operation. Resizing an array by adding only a few values at a time is often not the most efficient approach, as the array will likely be resized many times. An example of an alternative approach is to double the size of the array every time the array must be resized.

In this question, you will write a function that resizes arrays in different fashions. More precisely, write a function with the following header:

```
function array_out = my_array_resize(array_in, dimension)
```

where:

- `array_in` is an m by n array of class `double` to be resized.

- `dimension` is a character string that should take one of the three following values (in a case-insensitive manner):
  - ◇ `'row'`. In this case, the array `array_in` should be resized by appending m rows of zeros to the bottom of the array.
  - ◇ `'col'`. In this case, the array `array_in` should be resized by appending n columns of zeros to the right of the array.
  - ◇ `'both'`. In this case, the array `array_in` should be resized by appending m rows of zeros to the bottom of the array and n columns of zeros to the right of the array.
- `array_out` is the resized array. Its class should be `double`.

The case (*i.e.* lower-case versus upper-case letters) of the input argument `dimension` should not matter. For example, your function should produce the same output whether `dimension` has the value `'row'`, `'ROW'`, `'Row'`, or `'RoW'`. This requirement applies to all allowed values of `dimension` (`'row'`, `'col'`, and `'both'`).

If the value of `dimension` is neither of the allowed values, then `array_out` should be the same array as `array_in`. In other words, do not resize the array if the value of `dimension` is neither of the allowed values. You can assume that `dimension` will always be a character string.

Test cases:

```
>> my_array_resize([2, 3, 5; 4, 6, 7], 'row')
ans =
     2     3     5
     4     6     7
     0     0     0
     0     0     0

>> my_array_resize([2, 3, 5; 4, 6, 7], 'col')
ans =
     2     3     5     0     0     0
     4     6     7     0     0     0

>> my_array_resize([2, 3, 5; 4, 6, 7], 'both')
ans =
     2     3     5     0     0     0
     4     6     7     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0

>> my_array_resize([2, 3, 5; 4, 6, 7], 'row and col')
ans =
     2     3     5
     4     6     7
```

4. Sequences

Consider three sequences (U_n), (V_n), and (W_n) of integers defined recursively for any $n \geqslant 2$ by:

$$U_n = U_{n-1} + U_{n-2} \tag{1}$$
$$V_n = V_{n-1} - V_{n-2} \tag{2}$$
$$W_n = W_{n-1} \times W_{n-2} \tag{3}$$

In this question, you will write a function that, given three numbers, identifies which of these three sequences these numbers follow, and outputs the next term in that sequence. More precisely, write a function with the following header:

```
function [next_term] = my_sequence_id(a, b, c)
```

where:

- `a`, `b`, and `c` are scalars of class `double` that represent integers.
- `next_term` is a scalar of class `double` (see below for description).

If `a`, `b`, and `c` (in this order) cannot be three consecutive terms of any of the three sequences (U_n), (V_n), and (W_n), then `next_term` should have the value `NaN`. If `a`, `b`, and `c` (in this order) can be three consecutive terms of one of the three sequences (U_n), (V_n), and (W_n), then `next_term` should be the next term in that sequence. If there are multiple distinct choices for `next_term`, then `next_term` should have the value `NaN`. For example, if `a`, `b`, and `c` have the values `2`, `2`, and `4` (respectively), potential values for `next_term` are `6` according to sequence (U_n) and `8` according to sequence (W_n). In this example, `next_term` should have the value `NaN`. If `a`, `b`, and `c` can be three consecutive terms of more than one of the sequences but the corresponding next terms are all equal, then `next_term` should have the value of this next term. For example, the values 0, 0, and 0 can be three consecutive terms of any of the three sequences, but the next term in the sequence is 0 in each case. In this example, `next_term` should have the value `0`.

Test cases:

```
>> next_term = my_sequence_id(1, 1, 2)
next_term =
     3

>> next_term = my_sequence_id(3, 7, 21)
next_term =
   147

>> next_term = my_sequence_id(2, 2, 4)
next_term =
```

```
   NaN

>> next_term = my_sequence_id(3, 5, 2)
next_term =
    -3
```

5. Elevators

Consider a building that features two elevators (elevator 1 and elevator 2) next to each other. On each floor of the building, a single button can be used to call an elevator. Let us call "caller" a person who calls an elevator by pressing this button.

In this question, you will write a function that determines which elevator should be sent to the caller when the caller presses the button to call an elevator. More precisely, write a function with the following header:

```
function [elevator] = my_elevator(location_caller, ...
                                  location_one, destination_one, ...
                                  location_two, destination_two)
```

where:

- `location_caller` is a scalar of class `double`. `location_caller` is an integer that represents the floor where the caller is located.
- `location_one` is a scalar of class `double`. `location_one` is an integer that represents the location of elevator 1 at the moment when the caller presses the button to request an elevator.
- `destination_one` is a scalar of class `double`. `destination_one` has the value `NaN` if elevator 1 is not moving. If elevator 1 is moving, then `destination_one` is an integer that represents the floor toward which elevator 1 is currently traveling to. If elevator 1 is moving, then you can assume that its current location and its destination are different.
- `location_two` is similar to `location_one` but for elevator 2 instead of elevator 1.
- `destination_two` is similar to `destination_one` but for elevator 2 instead of elevator 1.
- `elevator` is a scalar of class `double` that can take one of two values:
  - ◇ `1` if elevator 1 should be sent to the caller.
  - ◇ `2` if elevator 2 should be sent to the caller.

Notes:

- Negative integers, positive integers, and zero are all valid values for floor levels. Negative numbers may represent, for example, underground levels.
- You can assume that elevators travel instantaneously from one floor to the next. In other words, elevators are never located in between floors, and the elevators' locations (as specified by `location_one` and `location_two`) will always be specified as integers. These

integers represent the floors at which the elevators are located.

Use the conditions listed below to determine which elevator should be sent to the caller. Check the first condition first. If one of the elevators satisfies this first condition but the other elevator does not, send the former elevator. If neither or both elevators satisfy the first condition, check the second condition. If one of the elevators satisfies this second condition but the other elevator does not, send the former elevator. If neither or both elevators satisfy the second condition, check the third condition, and so on. If both elevators satisfy all the conditions, send elevator 1. If both elevators satisfy none of the conditions, send elevator 1.

Conditions:

1. The elevator is located at the same floor as the caller.
2. The elevator is moving toward the caller.
3. The elevator is not moving.
4. Both elevators are moving, but the distance between the destination of this elevator and the caller is smaller than the distance between the destination of the other elevator and the caller.
5. The distance between this elevator and the caller is smaller than the distance between the other elevator and the caller.

Test cases:

```
>> elevator = my_elevator(2, 1, -1, 2, 5)
elevator =
     2

>> elevator = my_elevator(2, -2, 4, 1, 0)
elevator =
     1

>> elevator = my_elevator(5, -2, NaN, 1, NaN)
elevator =
     2

>> elevator = my_elevator(0, -1, -5, 2, 10)
elevator =
     1

>> elevator = my_elevator(0, 3, 5, 2, NaN)
elevator =
     2

>> elevator = my_elevator(10, 12, 10, 8, 10)
elevator =
     1
```

6. References

Nazaroff, W. W. and L. Alvarez-Cohen. 2001. *Environmental Engineering Science*, John Wiley & Sons, Inc.