Lab Assignment #9
Pointer and Data Structures 1
Due 4/24/2020 at 11:59pm on bCourses

The assignment will be graded by an autograder which will check the outputs of your functions. For your functions to be scored properly, it is important that the names of your classes, properties and functions **exactly match** the names specified in the problem statements, input and output variables to each function are in the **correct order** (i.e. use the given function header exactly), and each function performs exactly what it is asked to. Instructions for submitting your assignment are included at the end of this document.

Skeleton code is provided to you on bCourses for wordList. You are to fill in the are specified by comments with the appropriate code. **DO NOT MODIFY** any code that is given in the skeleton code or you may lose points. For the rest of the problems, you will write your code from scratch.

Suppress all non-required outputs.

You may have noticed that the format of this assignment document is different from the previous ones. We recommend you to read through it before starting to work.

Useful Resources

We are now moving into the last part of this course: data structures. This assignment (and the next) touches upon what is called objected-oriented programming. We won't learn much of it - if you are interested you can take CS61B - but it can still be a bit challenging. Those resources are collected to help you start this assignment.

- MATLAB guides on Classes. This is a large family of documentations.
- Particularly useful pages include how to Create a Simple Class and Class Components.
- This introduces class properties with different levels of accessibility (public, private, etc.).
- Structure array, also known as struct. (It is not actually an array.)

1 Ordinary differential equations

1.1 Euler's method

function [y] = myEuler(f,tspan,y0,N)		
Input	Type	Description
f	function_handle	A function handle to the function $f(t, y)$
tspan	1x2 double	The range of time values $[t_0, t_1]$.
y0	1x1 double	The value of y at $t = t_0$
N	1x1 double	The number of timesteps to use.
Output		
y	1x(N+1)double	The value of y at each timestep from time $t = t_0$ to time $t = t_1$

Details

In this question, you will calculate the value of y from $t = t_0$ to $t = t_1$ with the ODE $\frac{dy}{dt} = f(t, y)$ and initial value y_0, using the forward Euler method with N equal time steps.

1.2 Solving compartmentalized model with MATLAB's ode45

`[S, I, R, fig] = myCompartment(S_init, I_init, R_init, tf, mu, lambda)`		
Input	Type	Description
`S_init`	`1x1 double`	The initial number of people who are susceptible
`I_init`	`1x1 double`	The initial number of people who are infectious
`R_Init`	`1x1 double`	The initial number of people who are recovered
`tf`	`1x1 double`	The final time t_f
`mu`	`1x1 double`	The value μ
`lambda`	`1x1 double`	The value λ
Output		
`S`	`1x(N+1)double`	The set of number of people who are susceptible from $t = 0$ to $t = t_f$
`I`	`1x(N+1)double`	The set of number of people who are infectious from $t = 0$ to $t = t_f$
`R`	`1x(N+1)double`	The set of number of people who are recovered from $t = 0$ to $t = t_f$
`fig`	`1x1 matlab.ui.Figure`	The figure plotting `S`, `I` and `R` over time

Details

The field of epidemiology uses models called compartmental models. In compartmental models, the population is divided into compartments, with the assumption that every individual in the same compartment has the same characteristics: either susceptible, infectious, and recovered (SIR). The SIR model is one of the simplest compartmental models, illustrated in the figure below. The model consists of three compartments: S for the number of susceptible, I for the number of infectious, and R for the number of recovered, deceased or immune individuals. The SIR model is reasonably predictive for infectious diseases, and can be applied to the current COVID-19 pandemic.

The arrows in the following figure represents people transformed from 'Susceptible' to 'Infectious' and people transform from 'Infectious' to 'Recovered'. Between 'Susceptible' and 'Infectious', the transition rate is $1 - \mu$, which means that at time t, the derivative of the number of people transform from 'Susceptible' to 'Infectious' over time is $(1-\mu) \cdot S(t)$. Between 'Infectious' and 'Recovered', the transition rate is $1 - \lambda$, which means at the time t, the derivative of the number of people transform from 'Infectious'

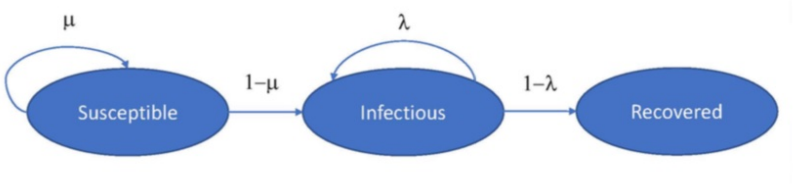

Figure 1: SIR Model

to 'Recovered' over time is $(1-\lambda)\cdot I(t)$.

Given λ and μ at time t, we can build the following ODE to represent the number of people that are susceptible $S(t)$, the number of people that are infectious $I(t)$ and the number of people who have recovered.

$$\frac{dS(t)}{dt} = -(1-\mu)\cdot S(t)$$

$$\frac{dI(t)}{dt} = (1-\mu)\cdot S(t) - (1-\lambda)\cdot I(t)$$

$$\frac{dR(t)}{dt} = (1-\lambda)\cdot I(t)$$

Your function should take the inputs given and solve this system of ODEs using MATLAB's `ode45` function. It should then plot number of people who are susceptible, infectious, and recovered.

- In order to used MATLAB's `ode45` to solve a system of ODEs, you must write a function of the form $f(t, y)$ that takes a column vector of variables and produces a column vector of derivatives.
- Again, use `fig=figure;` **before** you do any plotting in order to create the figure handle. All plotting commands will then be applied to the most recent figure by default.
- Your figure should contain the **title** and **axis labels** as shown in the following figure.

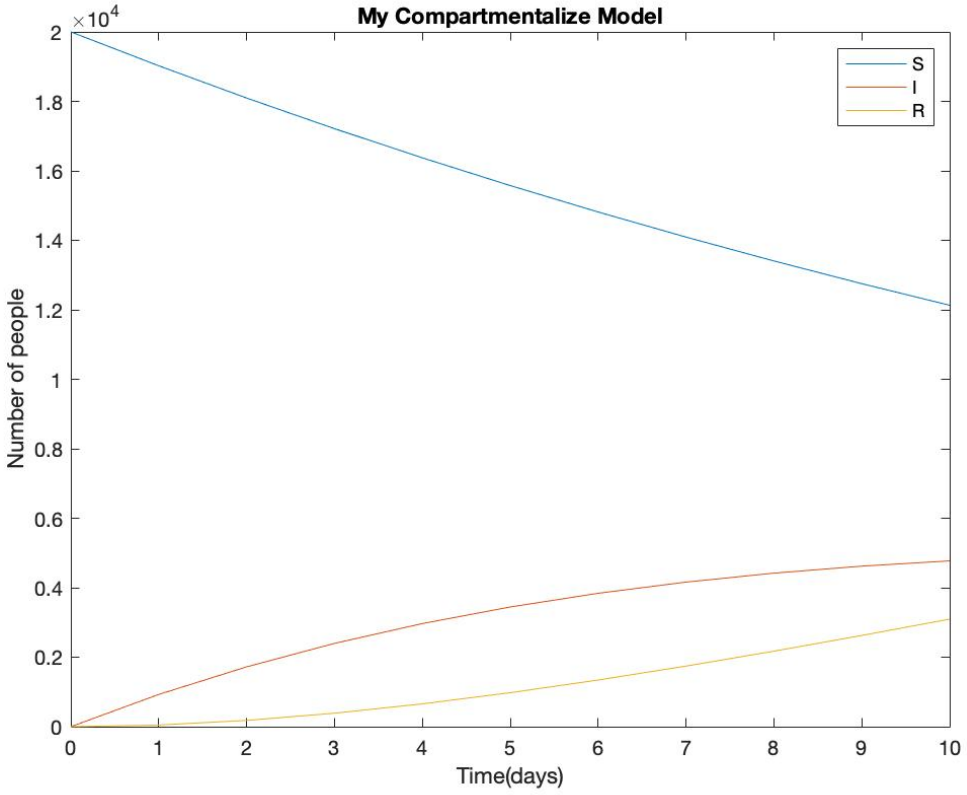

Figure 2: Sample figure from tester

2 Pointers

In this section we will practice using pointers in MATLAB. We begin by writing a very simple pointer class.

2.1 Define a pointer class

The first task is to write a class that records a person's name and their student ID number. Objects of that class are then called by reference.
The class is named `sid` and should have two properties: `name` and `number`. `name` will be given as a string and `number` will be an integer. Both properties should be empty when a new object of the class is created.

2.2 Write function in the class

In this class, you will write a function.

Function: `changeNumber`

Input: `newNum`, a new number representing the student ID

Output: None

This function should change the number of the student ID number to `newNum`.

2.3 Identity a pointer class object or a struct

Now you will be given an input that is either an object of class `sid` or a struct. The struct will have the same fields as the class properties: `name` and `number`. You are to write a function to identify whether the input is of class `sid` or a struct.

function result = pointerVerify(sid1)		
Input	Type	Description
sid1	sid or struct	A student ID
Output		
result	int	1 if sid1 is a pointer; 2 if sid1 is a struct

Restriction: You are not allowed to use the MATLAB built-in function `class`, `whos`, or anything that would explicitly tell you an object's class.

Hint: what is the main difference between calling by reference and calling by value?

3 Word List

We have learned the most basic version of the linked list structure in class: the **single** linked list. In this problem, you will implement one of its variations: the **double** linked list, which we will use to store words in alphanumeric order and also track how often they appear. We will begin by defining a simple word class and a word list class. Then we will implement a few functions that will describe the data contained in the list.

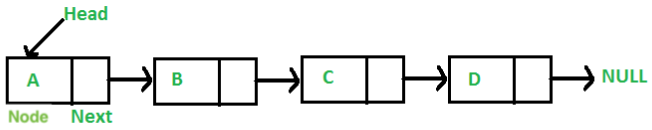

Figure 3: Single Linked List

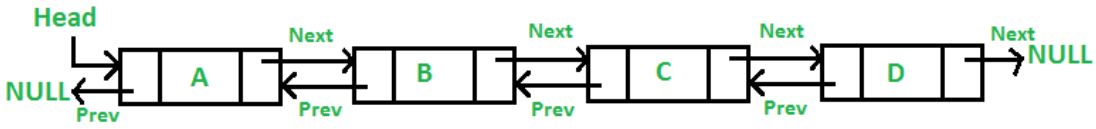

Figure 4: Double Linked List

3.1 Define the `myWordNode` class

`myWordNode` is a pointer class with properties `word`, `occurrences`, `prev` and `next`. `word` is the unique identifier for the node, and for the purposes of this problem, it will be a char array. `occurrences` holds additional information about the node and its data type can be flexible. However, for the purposes of this problem, it will be the number of times an individual word has been added to the list.

The most distinctive feature of a linked list is that each node has a pointer to the next node in the list (next node, `next`). In a double linked list, a node will also have a pointer to the previous node (previous node, `prev`). If a node does not have a connecting node, then that property should point to an empty node.
Hint: `myWordNode.empty` creates an empty node.

Define the class with the given properties above and no functions. A new node is initialized as empty `word`, `occurrences`, `prev` and `next`.

3.2 Define the `wordList` class

`wordList` will be a pointer class. This is the main class for our word list that utilizes a double linked list structure. First, define a `wordList` class with properties `head` and `tail` that point to

the first and the last nodes of this list respectively. These properties will be protected. The reason behind creating a protected property is preventing the manipulation of the properties by the user. Protected properties can only be changed by the functions within the class, but still allow the user to view them.

`wordList` also has two private properties, `uniqueWords` and `totalWords`, which are used to track the number of nodes in the list and the total number of words that have been added to the list. Private properties cannot be seen by the user. `totalWords` will be updated with each add or remove call, while `uniqueWords` will only change if the added word is not currently in the list and when a node is removed from the list.

A new instance of `wordList` is initialized as an empty list: it has empty `head` and `tail` and 0 for both `totalWords` and `uniqueWords`.

We will add additional functions to `wordList` in the following steps.

3.3 Report totalWords and uniqueWords

For this section, you will create two method functions. These functions will allow the user to view the private properties.
`getUnique`: outputs the number of unique words in the wordList. This also acts as the length of the wordList.
`getTotal`: outputs the number of total words that have been added to the wordList.

3.4 Add and remove an element

`add` inserts a node to the `wordList` in alphanumeric order. If the node `word` already exists, it increments the `occurrences` by 1. It also makes the necessary updates to the `getUnique` and `getTotal` properties. Also, you are to verify the class of all input nodes are identical.

For example, if you add the 5 words "the cat and the dog" to `wordList`, your `wordList` would contain 4 nodes and the node with `word` 'the' would have `occurrences` = 2 while `occurrences` for the other 3 nodes would be 1.

This has been done for you already. However, you should spend some time creating nodes and adding them to the list to see how the properties of the list change to familiarize yourself with the structure.

You will implement the following method function in `wordList` with O(n) complexity:
`removeWord`: remove a node based on the given input word. `removeWord` should also output the removed node. This node should have no pointers remaining. Remember to make the necessary updates to the `getUnique` and `getTotal` properties. If a node with the input word does not exist within the `wordList`, then output must be `'This word is not in the list'`.

NOTE: Pay attention to the changes in linking relationships. You will have to adjust `prev` and `next` of the affected nodes appropriately.

Optional: An advantage of double linked list is that can start from either the `head` or `tail`, whichever is closer. For example, if the node you are trying to add or remove has `word` = `'zebra'`, you may want to start from the `tail`, while a node with `word` = `'apple'`, you would want to start from the `head`. Try it out if you are interested! (not for credit)

3.5 Word retrieval

`retrieveWord`: returns the node that contains the given word. The pointers in the output node should be empty. If the node does not appear, your output should be `'This word does ... not appear in the list'`. This function will be useful for the upcoming functions and will be implemented with **O(n)** complexity.

3.6 Print list of `word` that occur in the `wordList`

`printList`: outputs a 1 x uniqueWords cell array. Each cell in the cell array should contain a char array that holds a unique `word`. The cell array should be ordered based on the order they appear in the list, in other words, ordered alphanumerically. For example, if the `wordList` was created from the phrase "the cat and the dog", then `wordList` would contain 4 nodes and the output for this function would be

```
1×4 cell array
  {'and'}    {'cat'}    {'dog'}    {'the'}
```

3.7 Word frequencies

`wordFrequency`: will output a 1 x `uniqueWords` double array containing the `occurrences` of nodes in the order they appear in the `wordList` if no input word is provided. If an input word is provided, the function should return a 1 x 1 double containing the `occurrences` of the given word.

`relativeFreq`: will output a 1 x uniqueWords double array containing the relative frequencies of the `occurrences` of nodes in the order they appear in the `wordList` if no word input is provided. If input is provided, the function should return a 1 x 1 double containing the relative frequency of the given word. The relative frequency should be taken with respect to the `totalWords`.
Hint: **`nargin`** will be used in these two functions.

For the example phrase used above, the `wordFrequency` and `relativeFreq` would appear as the following:

```
>> example.wordFrequency
ans =
     1     1     1     2
>> example.relativeFreq
ans =
    0.2000    0.2000    0.2000    0.4000
```

Testers

The tester for this assignment contains many things you should think about for your data structures; however, it is not exhaustive. We give an extensive and thorough set of test cases for this assignment. Each section of the testers have brief headers; however, we are unable to provide detailed descriptions of each test case. If you get an error from a tester, you must locate that particular test case and diagnose the cause. You will not ask on Piazza "Why the tester gives me an error?"

Submission

This homework is a long one and please follow the submission instructions when submitting it!

All classes must have the exact same names as those specified in the respective sections. All functions must have the exact the same names and input parameters as those specified in the respective sections. All file names must also follow either the specifications or MATLAB's naming rules.

The following files are to be submitted for this assignment:

- `myEuler.m`
- `myCompartment.m`
- `sid.m`
- `pointerVerify.m`
- `myWordNode.m`
- `wordList.m`