Homework 0

BEE 4750/5750

Due: Aug 31, 2022 by 11:59pm ET

Contents

Overview **1**

Learning Objectives . . . 1

What To Submit . . . 2

Problems **2**

Problem 1: **Writing Some Basic Code** . . . 2

Problem 1.1: *Writing A Basic Function* . . . 2

Problem 1.2: *Referencing Output* . . . 2

Problem 1.3: *Making Plots* . . . 3

Problem 2: **Square root by Newton's method** . . . 3

Problem 2.1: *Justify Step 3* . . . 3

Problem 2.2: *Implement the Algorithm* . . . 3

Problem 3: **Working with Vectors and Matrices** . . . 3

Problem 3.1: *Generating Random Values* . . . 4

Problem 3.2: *Calculating a Mean* . . . 4

Problem 3.3: *Accessing Array Elements with Indices* . . . 4

Problem 3.4: *Working with Matrices* . . . 5

Problem 4: **Plotting Graph Representations of Networks** . . . 5

Problem 4.1: *Plot An Undirected Graph* . . . 5

Problem 4.2: *Plot a Directed Graph* . . . 5

Problem 4.3: *Label Edges* . . . 5

Overview

Learning Objectives

Homework 0 is intended to get you up and running with cloning the assignments using Github, installing and using Julia, and submitting the assignments. It will only be graded

based on completion, but make sure you don't run into any problems with this workflow, which will be repeated for future homework assignments.

What To Submit

Submit a single `.pdf` file to Gradecope. Make sure to tag each problem. You should include code with your submission in code blocks as needed for your solution. Similarly, figures should be captioned appropriately. You'll be walked through creating a document in this fashion in this assignment. Note that some prompts (like pre-specified code blocks) will not be included in future assignment instructions and templates, and you will need to include those as specified here. **Make sure that your final report includes a References section!**

To generate your `.pdf` locally, you'll need to have LaTeX and Julia installed on your system. Open the Julia REPL from the local working directory (or write a script to do these steps) and type:

```
using Pkg # load the Pkg package management system
# activate the package environment; the relevant files should be provided as part of the repository
Pkg.activate(".")
using Weave # load Weave.jl
Weave.weave("solution-file-name.jmd", doctype="md2pdf", template="bee4750.tpl", fig_path="figures") #
convert your .jmd file to a pdf for submission
```

If you don't want to install LaTeX locally, we've also configured the Github repositories to attempt to compile your `.jmd` file into a `.pdf` when you push commits to your repository. This may take a little bit of time (Julia and LaTeX both need to be installed on the server every time this is attempted), but if it is successful, you will end up with a `.pdf` in your repository that you can download through a `git pull` command and submit to Gradescope. If it is not successful, this doesn't necessarily mean that there's a problem; you might not be at a stage where you intended your report to be compilable.

Problems

Problem 1: Writing Some Basic Code

This problem focuses on writing basic Julia code and including and referencing that code in a `Weave.jl` report.

Problem 1.1: *Writing A Basic Function*

Write a function to compute the square of a number. Include your code in a fenced code block like that below:

```
function square_number(x)
    # your code here
    return x*x
end
```

Problem 1.2: *Referencing Output*

Write a sentence where you include a real-time calculation of 5^2 using your function. You don't need to do this by evaluating your function in real time, but you can use syntax like the

following:

```
"We can see that $5^2 =$ `j square_number(5)`."
```

Problem 1.3: *Making Plots*

Use a loop to evaluate `square_number()` over the interval $[-10, 10]$, then plot the results. This tutorial shows some plotting basics and provides links to other resources.

To include a plot in your report dynamically, you can use a code block that outputs a figure. You should definitely use the `fig_cap` chunk option, and possibly also some of the others; see the `Weave.jl` documentation. The use of the `label` chunk option lets you refer to the plot in your report using LaTeX, using `\ref{fig:square-plot}` (or whatever label). You can also just directly include figures if you've generated them separately using standard Markdown.

Problem 2: Square root by Newton's method

This problem involves implementing an algorithm: in this case, Newton's method for computing square roots. It was shamelessly copied from MIT's Introduction to Computational Thinking course.

The algorithm is as follows:

Given $x > 0$, the desired output is $\sqrt{x}$.

1. Take a guess `a`
2. Divide `x` by `a`
3. Update `a` to be the average of `x/a` and `a`,
4. Repeat until `x/a` is close enough to `x`.
5. Return `a` as the square root.

Problem 2.1: *Justify Step 3*

Why must $\sqrt{x}$ lie between x/a and a, as in Step 3 of the above algorithm? Take advantage of any math typesetting you might need using LaTeX.

Problem 2.2: *Implement the Algorithm*

Implement the above algorithm in Julia. Notice that Step 4 requires some interpretation for "close enough"; this is usually done by including an additional parameter specifying an error tolerance.

Test your code by outputting $\sqrt{2}$.

Problem 3: Working with Vectors and Matrices

We won't be doing much (if any) linear algebra in this class, but vectors and matrices are useful data structures, so let's see how to use them.

Problem 3.1: ***Generating Random Values***

Make a random vector of length 20 using the `rand()` function. *Note*: We'll see later in the course how to use `Distributions.jl` to sample from particular distributions; using `rand()` in this way just samples from a uniform distribution over the unit interval $[0, 1]$.

Problem 3.2: ***Calculating a Mean***

Implement your own `mean()` function to calculate the mean of a vector using a `for` loop and the random vector from above. Then write another function `demean()` which subtracts the mean from every element of the vector using your `mean()` function.

> **NOTE: Mutation**
>
> Julia has particular conventions around functions which *mutate* the input data. What do we mean by mutation? A function can operate on data by modifying the original vector, or by creating a new vector. For example, in this problem, you might write the following function:

```
function demean(vect)
    ... # some other code might be needed here
    for i in 1:length(vect)
       vect[i] -= mean(vect)
    end
end
```

> Given how Julia handles memory and passing arguments (the details of which are not essential), this would mutate the original array and change the underlying data. In general, this is undesirable behavior unless high levels of code optimization are required (which they won't generally be for this course, and rule #1 of coding is not to prematurely optimize code), as the original data then cannot be reused.
>
> Instead, try to work with copies of data, particularly when they aren't really large. This is always done when you use the `=` operator; *e.g.* `x = 2*x`.
>
> If you write a function which does mutate the original data, the convention in Julia is to append an exclamation mark (`!`) to its name to indicate that it is doing so. So `sort(x)` returns a sorted copy of `x`, while `sort!(x)` sorts `x` in place. In other words, our function above should have been called `demean!(vect)`.

Problem 3.3: ***Accessing Array Elements with Indices***

Create a vector of 10 elements, where the center 6 elements are equal to 1 and the others are equal to 0. *Remember that indexing in Julia starts with 1, not 0.*

Problem 3.4: ***Working with Matrices***

Using the `rand()` function, create a random 5x5 matrix. Then subtract the mean of each column from that column.

Problem 4: Plotting Graph Representations of Networks

This problem introduces you to `GraphRecipes.jl` (see the documentation), which provides tools for plotting networks as connected graphs. This may be useful in future assignments and projects for plotting system components and flows. We won't explicitly do this in this assignment, but you can change colors and shapes for the nodes to indicate different types of components.

A key component in these plots is the *neighbor matrix* A. A should be symmetric for undirected graphs (graphs where the edges are simple connections with no "starting" or "ending" node): $A_{ij} = A_{ji}$ to indicate that nodes i and j are mutually connected. For a directed graph, where an edge starts at i and ends at j, $A_{ij} = 1$ but A_{ji} is not necessarily 1 unless that edge is bidirectional (flows can occur in both directions).

Problem 4.1: ***Plot An Undirected Graph***

Plot an undirected graph with 5 nodes, labeled by index. Specify your graph's neighbor matrix using Julia matrix notation, *e.g.*:

```
A = [...;
     ...;
     ...;
     ...;
     ...
    ]
```

Problem 4.2: ***Plot a Directed Graph***

Plot a directed graph with 5 nodes, labeled by index.

Problem 4.3: ***Label Edges***

For your directed graph from Problem 4.2, label each edge with a string expressing a quantity.