Table of contents

This document provides guidance for submitting high-quality problem set (and project) solutions. This guidance is based on general good practices for scientific communication, reproducible research, and software development.

General presentation

- Simply presenting code or derivations is not sufficient.
- Briefly describe the overall goal or strategy before providing code/derivations.
- As needed describe what the pieces of your code/derivation are doing to make it easier for a reader to follow the steps.

Coding practice

- Minimize (or eliminate) use of global variables.
- Break down work into core tasks and develop small, modular, self-contained functions (or class methods) to carry out those tasks.
- Don't repeat code. As needed refactor code to create new functions (or class methods).
- Functions and classes should be "weakly coupled", interacting via their interfaces and not by having to know the internals of how they work.
- Use data structures appropriate for the computations that need to be done.
- Don't hard code 'magic' numbers. Assign such numbers to variables with clear names, e.g., `speedOfLight = 3e8`.
- Provide reasonable default arguments to functions (or class methods) when possible.
- Provide tests (including unit tests) when requested (this is good general practice but we won't require it in all cases).
- Avoid overly complicated syntax – try to use the clearest syntax you can to solve the problem.
- In terms of speed, don't worry about it too much so long as the code finishes real-world tasks in a reasonable amount of time. When optimizing, focus on the parts of the code that are the bottlenecks.
- Use functions already available in the language rather than recreating yourself.

Code style

- Follow a consistent style. For example (not required) the tidyverse style guide or its offshoot Google R style.
- Use informative variable and function names and have a consistent naming style.
- Use whitespace (spaces, newlines) and parentheses to make the structure of the code easy to understand and the individual syntax pieces clear.
- Use consistent indentation to make the structure of the code easy to understand.
- Provide comments that give the goal of a given piece of code and why it does things, but don't use comments to restate what the code does when it should be obvious from reading the code.
  - Provide summaries for blocks of code.

– For particularly complicated syntax, say what a given piece of code does.