Name: Josh Class #: 37

Engineering 11: Introduction to MATLAB
Laboratory 3 Reading Assignment
by Jeff Anderson

What is the purpose of this reading assignment?

Some of my main goals as your teacher include: helping you develop a nuanced understanding of scientific computation, inspiring you to find reasons why scientific computation matters in your life, and encouraging you to plan future coursework and professional development to become the most competent programmer you can be. In order to accomplish these goals, I need to help you get a sense of what the field of scientific computation is and how this field fits into science. These five articles that you read in Lab 3 are designed to begin your introduction. More specifically, by the end of these five articles, I hope you can do each of the following:

☐ 1. Define the three pillars of science: theory, experimentation, and computation.
☐ 2. Describe the similarities and differences between the *two* versus *three* pillars of science?
☐ 3. Make well-reasoned argument of your own about the *two* versus *three* pillars of science?
☐ 4. Describe at least three reasons why writing good code is important in science?
☐ 5. List at least three negative consequences of writing bad code?
☐ 6. Describe at least three separate safe software practices you'd like to learn more about?

What tasks do I expect you to complete in this reading assignment?

In order to complete this reading assignment, I ask you to:

☐ Spend between 90 – 180 minutes reading the five articles I have assigned in Lab 3

☐ Read these articles slowly and deeply.

☐ Fill out questions 1 – 12 on pages 2 – 9 of this worksheet.

☐ Write your responses by hand or type your response in the docx document posed on our homepage

☐ Submit your responses with your Lab 3 report

Personally, I spent about 10 hours researching, reading, and thinking about these articles. I do NOT expect you to spend this amount of time. However, please know that I have read every word of each of these documents. I also found .pdf copies of many of the references listed in each of these documents. Finally, I created an Amazon.com wishlist of all the books I plan to buy to help me be a better teacher in this class. I share a copy of a screenshot of this wish list online.

Name: Josh Class #: 37

What kind of science is Computational Science
by Rhett Allain

1. The author of this article, Rhett Allain, claims that popular opinion dictates that there are three parts of science. What, specifically, does Allain name each of these three parts? For each part of science that Allain identifies, recapitulate the corresponding definitions he provides?

- Theoretical Science: Where all new science starts. It's generally thought to involve developing hypotheses and testing them in an abstract, analytical sense, with pencil and paper computation, and using ideal models/formulas to represent the real things.
- Experimental Science: This is where theoretical science is tested by setting up experiments in the real world. Empirical results are compared with theoretical predictions, and conclusions are drawn that improve both the theory and the experiment.
- Computational Science: This is the part that many traditional scientists will say is outside of their purview. Theoretical models are tested using computational models of varying sophistication, meant to approximate, but not replace, real-world experiments and conditions. Computational science serves as a relatively resource efficient way to refine an experiment with iterative changes to the models, prior to the expensive task of setting up an experiment in the real world.

2. What do you think is Rhett Allain's main thesis in this article? What is he trying to say in this work?

I think his thesis is that the artificial separation between theoretical and computational science is an outdated fallacy. There should only be two main areas of science, model-building and real world experimentation. Both theoretical and computational science and fall under model-building. Whether you are using a pencil and paper or a computer program to derive idealized formulas for real world things, they are still just idealized, simplified away from the complexity of the physical universe. Computational/numerical science is just a supercharged tool of theoretical science, allowing a computer to do the heavy mathematical lifting for computationally intensive tasks, when it might take a human much much longer. The gold standard for science that attempts to explain the real world will always be experimenting and collecting data in the real world.

Name: Josh Class #: 37

3. List at least **TWO** of your favorite quotes from this article. Next to each quote, explain what you like about this quote and how this quote relates to your own understanding of scientific computation.

EXAMPLE: Here is one of my (Jeff's) favorite quotes from this article:

Quote from page 2: "At the most basic level, science is about building models. If you just build a model with no connection to real life, that isn't science."

Reasoning: This quote captures the essence of the scientific method applied to real-world problems. We use experimentation and observation to make conjectures about the world and develop theory that (hopefully) describes and explains the phenomena we are studying. Then, we do further experimentation to test our theories and refine our conjectures.

I love that the author is focused on describing science as a tool that is useful in the real world. This quote captures my feeling about the argument between pure and applied mathematics. As a younger man, I studied pure math quite deeply but eventually converted to applied math because I felt my work should have deep connections to real life.

"In a differential equation, what does the term dv/dt really mean? It means that in some infinitesimally small time interval (dt), there is an change in velocity. Really, this is the slope of the v-t graph. When I model the motion of a spring with computer code, I use Δv and Δt. These are not infinitely small. However, I can make them so small that you couldn't even tell the difference.

In a derivative, we look at Δt in the limit that Δt goes to zero. For a computational model, we look at Δt in the limit that Δt gets small enough so that the output of the code agrees with real life. In both cases, the model is only useful if it agrees with real data. So, in that sense, both approaches are just as valid as the other."

Reasoning:

I liked that this passage made the connection between a fundamental theoretical/analytical idea in calculus, the derivative, and contrasted it with its computational analogue. In analytical science, we can think of a derivative as the change in output over change in input as the change in input becomes infinitesimally small, and use notational conventions to describe a problem without actually having to fundamentally represent it. When computing numerically, we can never really represent 1/infinity, so instead we must turn to experimental results from the get-go, and adjust our tiny discrete change in input to be small enough that it agrees with the real world. This is an interesting distinction when dealing with infinitesimals, and suggests that analytical is slightly more fundamental than computational in these situations (at least in the early stages of developing an experiment), since it can start from a place that is not necessarily already proven experimentally, as long as it follows accepted math conventions. Computational science is less useful until you already have experimental data to tune its necessarily-limited representations with.

"If you think of a numerical model as another tool in your belt (for theoretical models), then there isn't a good reason to exclude this method. It shouldn't be a separate course, it should just be included in all courses."

Reasoning:

This is just a painfully true statement. We need to be learning more of the skills that are important for science professionals in the modern world, less focused on analytical representations of problems alone. Analytical models are important for the early stages of learning in a field, as they help distill and generalize the ideas involved in compact form, but they are of limited use as the complexity of a system increases. We need computational methods to be more tightly integrated into science education starting sooner.

Name: Josh Class #: 37

What kind of science is Computational Science? A Rebuttal.
by Tamara Kolda

4. As described by the author, Tamara Kolda, in this article, the 2005 U.S. Presidential Committee on Computational Science provides a thorough definition of *computational science*. What is this definition? What are the three distinct elements of this definition? What is the focus of each of these three elements?

Definition: Computational science is a field spanning all domains of science, using computer programs to solve, and derive useful theory/generalizations about, problems that are much too complicated for purely analytical science to even make a dent in.

Three Distinct Elements:

- Algorithms: These are the actual sequences of computations (what we may call the programs themselves) that are useful for obtaining numerical results in specific scientific applications. They are used in theoretical science as well, but can be made much more powerful with the aid of a computer.
- Computer and information science: This is the subfield of computer science that specifically deals with developing the software systems for useful numerical scientific computation. It optimizes the specific algorithms and representation/storage of data to make them usable by scientists without specific computer science training. It becomes more important as the complexity of the problem and its associated algorithms increases, and is necessary for getting the most problem-solving power out of a given computer system.
- The computing infrastructure: This is the fundamental hardware/software technology that forms a given computer system. It allows for both of the bullet-points to exist, by constructing an adequate software operating system on top of specially designed hardware. In many cases, the computer infrastructure itself is customized and optimized to deal with highly specific, complex scientific problems.

5. What do you think is Tamara Kolda's main thesis in this article? What is she trying to say in this work?

Her thesis is meant as a direct response to Allain's thesis. She disagrees with him on his point that theoretical and computational science are two sides of the same model-building coin, and separate from experimentation. She emphasizes that computational/numerical science is actually much more important than theoretical/analytical science for most of modern complex problems being actively studied. Analytical methods are hopelessly slow and inefficient for making a dent in these problems, without significant use of computer algorithms, as they involve either unfathomably vast data sets, or mathematical statements that are analytically unsolvable (like most differential equations). She also notes that in many cases, real-world experiments are impossible to carry out with current technology and available resources, and so, in these situations, much of the "experimentation" can only be done using computational models that reasonably represent reality. So she believes that computational science is an overarching and increasingly vital discipline for all of modern science.

Name: Josh Class #: 37

6. List at least **TWO** of your favorite quotes from this article. Next to each quote, explain what you like about this quote and how this quote relates to your own understanding of scientific computation.

EXAMPLE: Here is one of my (Jeff's) favorite quotes from this article:

Quote from page 2: "Going from application area to computational results requires domain expertise, mathematical modeling, numerical analysis, algorithm development, software implementation, program execution, validation and visualization of the results. CSE involves all of this. Computer models are ultimately discrete approximations of continuous phenomena."

Reasoning: This quote reminds me of Lloyd Trefethen's definition of Numerical Analysis. I also love that this quote captures the essence of multidisciplinary work. The challenge and beauty of modern day scientific computation is that scientists who want (or need) to write code will do a lot better if they master a relevant set of mathematical and computational tools. One of the reasons I became a teacher is because I think our school system (K12, Undergraduate, and Graduate-level) does a very poor job of helping students learn these really important skills. I want one of my professional contributions to be a meaningful re-design of lower-division mathematics curriculum to integrate foundational ideas in CSE into our classes in a way that is meaningful for each and every student.

"Designing, analysing and validating computational approaches that handle these issues is another field beyond the realms of experimental data and theoretical models. More generally, the three elements of computational science — algorithms, computer and information science and the computing infrastructure — call on many research concepts that are distinct from those used in the generation of experimental data and the development of theoretical models."

Reasoning: She makes the good point that it is important to keep computational science as its own field with its own dedicated experts, because it does require a very specialized skill set that we cannot reasonably expect all other scientists to be well-versed in, on top of the skills required of their specific field. This is important as a reality check, since any one person can only become an expert in so many things during their lifetime. So computational science should be respected as a highly technical "meta-discipline" that keeps the wheels of all modern science turning.

"The 2003 DOE Report on A Science-based Case for Large-scale Simulation (SCaLeS) makes a similar case, arguing that breakthroughs in both theory and experiment rely on advances in computational science. Consider the theory of fluid dynamics, radiations transport, and weather prediction, where models are large-scale and nonlinear, making computation the "only truly systematic means of making progress.""

Reasoning: I like that this section gives some specific examples of modern science fields in which theoretical and experimental advances have only been made as a result of the awesome number-crunching power of computational science. It is not useful or productive for anyone to approach these problems without highly optimized computer models, when a calculation that would take a team of scientists using pencil and paper many lifetimes can be accomplished by a super-computer in a matter of minutes. This is especially true for applications that demand prompt results to be useful, such as weather prediction.

Name: Josh Class #: 37

Science has only two legs
by Moshe Y. Vardi

7. What do you think is Moshe Vardi's main thesis in this article? How does Vardi think computation relates to theory and experimentation?

His thesis is that it is misguided to label computational science as something separate from "normal" science. He believes that there are only two fundamental pillars of science, theory and experimentation, and mathematical computation has always been necessary in both pillars, in order to gain any useful results. The only difference is that now computers are required to in this computation, as the theory and experimental models become more and more complex. He doesn't think this is anything fundamentally new, just a modern extension of the same, and so says that computation should be considered an integrated part of both theory and experimentation.

8. This is the third article you've read that describes a debate between thinking of science as having two pillars (theory and practice) versus of thinking of science as having three pillars (theory, practice, and computation). What do you think: does science have two pillars or three pillars? Explain your reasoning.

I agree with Vardi that there are only really two distinct pillars of science, theory and experimentation, and that computational science is just a super powerful extension of the by-hand calculations that classical science was composed of. It makes sense to keep computational science a separate professional field, for the purpose of training experts in all the specialized knowledge it entails. I think it is very analogous to how math and its experts are viewed in relation to the sciences. They provide the general framework and tools that allow anything to get done in a specific application (there'd be no chemistry without algebra, no physics without calculus). The only difference is that computational science has the added complexity and intricacy of using computers as tools and getting the most out of their finite resources. In many cases it requires designing computer systems from the ground up for a specific application, and this demands engineering and computer science knowledge far beyond the scope expected of the average scientist.

Name: Josh Class #: 37

9. List at least **TWO** of your favorite quotes from this article. Next to each quote, explain what you like about this quote and how this quote relates to your own understanding of scientific computation.

EXAMPLE: Here is one of my (Jeff's) favorite quotes from this article:

"At the same time, *computational thinking*… thoroughly pervades both legs. Computation is the universal enabler of science, supporting both theory and experimentation. Today the two legs of science are thoroughly computational!"

Reasoning: I love thinking of computation as part of both theory and experimentation. Personally, I don't like to think of computation as a third leg of science. Instead, I prefer to think of computation as a necessary component of every great scientists arsenal of tools. I think of computation like I think of lab equipment, or the ability to read and write. This quote captures my thinking on this quite well.

On the other hand, I recognize why we are having the debate. If we think of computation as a third pillar, our education system is much more likely to take this field seriously (which we do not do very well at the moment). Could you imagine getting a degree in college degree in chemistry or physics without ever doing work in a laboratory. Hopefully, that idea seems ludicrous. Yet, we allow tens of thousands of students to graduate from college with science degrees without ever having taken classes that specialize in scientific computation. Moreover, most introductory computer language classes focus on syntax rather than major themes of software development and computation. This approach has to change. In my opinion, we should be teaching coding in every single lower-division science course we teach in the college system. I hope to be part of a generation of educators that designs and implements such a change. My first goal is to get this done at Foothill college.

"The nature of the theories has also changed. Maxwell's Equations constitute an elegantly simple model of reality. There is no analogue, however, of Maxwell's Equations in climate science. The theory in climate science is a highly complex computational model. The only way to apply the theory is via computation. While previous scientific theories were typically framed as mathematical models, today's theories are often framed as computational models. "

Reasoning: I thought this analogy was interesting. We are so used to thinking of science being expressed in terms of closed-form generalized math formulas, but this is no longer the reality. For these situations, some knowledge of computer science and specific algorithms is likely necessary to even start to grasp the gist of a model. It is an emerging scientific language that all scientists should have at least some training in, to be able to continue contributing to the conversation.

"Experimentation typically implies carrying out measurements, and the analysis of these measurements has always been computational. Again, what has changed is the scale. The Compact Muon Solenoid experiment at CERN's Large Hadron Collider generates 40 terabytes of raw data per second, a volume one cannot hope to store and process."

Reasoning: A nice concrete example that makes the limits of scientific computation without computer systems very clear. No one could ever hope to work with all that data without a specialized computer system to separate the signal of interest from the numerical noise. All the big important problems of our time are like this, so we need training in these new methods to be able to contribute.

Name: Josh Class #: 37

Error: why scientific programming does not compute
by Zeeya Merali

10. What problem(s) does Zeeya Merali describe in this article?

Scientists are increasingly required to produce computer programs to assist them in their research, but according to the article, less than half of scientists have formal training in and good understanding of best practices in computer programming, and so many of them are currently poor programmers. This is a major problem, since so much of modern science relies on code to produce its results, and these results find use in many real world applications. So, on top of invalidating scientific results, buggy programs can lead to major malfunctions in the real world, wasting resources and potentially causing harm to individuals or societies.

Scientists need to become better versed in best practice computing standards, and/or recruit computer science experts to their teams to help them. Well-written, tested and documented scientific code is important for checking results and enabling experiment transparency and reproducibility, making scientists and their work accountable to the broader scientific community and the public. This is necessary for these scientists to maintain credibility in the modern science landscape.

11. How are these problems related to your life as a student and future course work you plan to complete? Pretend that you were serious about training yourself to use scientific computation in your future career. What are some suggestions that Merali mentions that you would like to learn more about. Why do you feel you might value these suggestions.

I know that coding will be an important skill for my EE degree, since complex electronic systems generally involve a computer, or at least a processor (like Arduino), in some capacity, for monitoring the state of the system and/or data collection. I am currently taking my third C++ class at Foothill (CS 2C) and so have had the privilege of spending a decent amount of time in a structured setting learning about computer science theory and best practices. As the author mentions, breaking a big wall of code down into many smaller chunks (a.k.a. functions) that each do one thing is very important, and significantly reduces headaches. Especially when you have a specific task that you will want to repeat over and over, writing a function for it has many benefits: a well-named function in place of many lines of code improves readability; extensive function testing over many different use cases teases out all possible bugs, and makes the source of those bugs clear; reusing well-tested functions and drawing from the same source code repository reduces the possibility of future bugs and mistakes.

Git and GitHub are vital tools for program version control, particularly for any project where a group of people are contributing on separate computers. They avoid the nightmare of code not syncing between computers, and its separate parts becoming an incompatible mess. I have learned the basics of these, but know that they are deep tools that I have a lot more to learn about.

One thing mentioned that I have not had much experience with is group programming strategies. My courses have all been online with solo projects only, so I have not yet coded along with another person, or received the in-person tutelage of an expert. I would like to seek out such opportunities.

Name: Josh Class #: 37

12. List at least **TWO** of your favorite quotes from this article. Next to each quote, explain what you like about this quote and how this quote relates to your own understanding of scientific computation.

EXAMPLE: Here is one of my (Jeff's) favorite quotes from this article:

"The level of effort and skills need to keep up aren't in the wheelhouse of the average scientist. As a general rule, researchers do not test or document programs rigorously, and they rarely release their codes, making it almost impossible to reproduce and verify published results generally by scientific software, say computer scientists."

Reasoning: When I read this quote between the lines, I see huge potential for my students. Researchers are some of the most well trained people on earth. Usually these people have been through 4 years of undergraduate studies plus at least 5 more years of a masters and PhD program. And yet, the ability to code well is a rare and valuable skill. Thus, if my students train themselves to code effectively, they will have an uncommon skill compared with most other researchers. Any student that takes this practice seriously can then leverage this skill to solve real-world problems and improve the practice of science at the institution of their choosing. This quote makes me want to work harder as a teacher to empower my students!

"Some software developers have found ways to combat the growth of monster code. One example is the Visualization Toolkit, an open-source, freely available software system for three-dimensional computer graphics. People can modify the software as they wish, and it is rerun each night on every computing platform that supports it, with the results published on the web. The process ensures that the software will work the same way on different systems."

Reasoning: This example is basically the gold standard for transparent, healthy code. The more systems code is tested on and the more eyes that review it, the more reliable it becomes. It is necessary to keep the source code centralized and to carefully account for and test all changes to it, so that no one mistaken or malevolent actor can ruin it. Once a good version control system is in place, having the whole internet vet your code constitutes an incredible service. The complexity of computer programs, and their interactions with different hardware and software systems leads to a vast number of ways that things can go wrong, and no one person can ever hope to test all possible configurations for a given program. So these systems outsource that job to the world, to the benefit of the coder and all those who might use the code in the future.

"Science administrators also need to value programming skills more highly, says David Gavaghan, a computational biologist at the University of Oxford, UK. "There needs to be a real shift in mindset away from worrying about how to get published in Nature and towards thinking about how to reward work that will be useful to the wider community."'

Reasoning: Programming is an important general skill for an informed citizen of the 21st century, just like math. This is even more true in the sciences, where scientists are increasingly expected to produce code in the course of their work, and the results of that code can shape public policy and critical systems. So this skill must be actively developed and championed by the scientific powers that be. Though in the past, simpler science could be carried out by individuals, most modern science cannot; it requires collaborative effort. If a culture of prideful competition or one-up-manship persists, this will hinder progress. Open-source, transparent, well-tested scientific code is a major key to making ideal scientific collaboration a reality.

Name: Josh Class #: 37

Best Practices for Scientific Computing
by Greg Wilson et al.

Best Practices for Scientific Computing Cheat Sheet:

- "Software is just another kind of experimental apparatus and should be built, checked, and used as carefully as any physical apparatus."

1. Write programs for people, not computers.
    a. A program should not require its readers to hold more than a handful of facts in memory at once.
    b. Make names consistent, distinctive, and meaningful.
        i. pick between CamelCaseNaming and pothole_case_naming, and stick with choice
    c. Make code style and formatting consistent.
2. Let the computer do the work, to avoid errors associated with mindless repetition
    a. Make the computer repeat tasks.
    b. Save recent commands in a file for re-use.
        i. Use command line interface with "command history" function
    c. Use a build tool to automate workflows.
        i. Similar to programming language compilers and linkers
        ii. Try Make software (http://www.gnu.org/software/make)
        iii. Make note of: unique identifiers and version numbers for raw data records; unique identifiers and version numbers for programs and libraries; the values of parameters used to generate any given output; and the names and version numbers of programs (however small) used to generate those outputs.
3. Make incremental changes.
    a. Work in small steps with frequent feedback and course correction.
    b. Use a version control system. Git in command line, and/or GitHub for friendly interface.
    c. Put everything that has been created manually in version control.
        i. use separate archive system for binary files (photos/audio), and no need to save automatically generated files
4. Don't repeat yourself (or others).
    a. Every piece of data must have a single authoritative representation in the system.
        i. Math constants should be defined exactly once; only one master version of raw data files should be used; every location from which data has been collected should be given a unique "address"
    b. Modularize code rather than copying and pasting. Use Functions and Classes!
    c. Re-use code instead of rewriting it. Don't reinvent the wheel, there's no time for/money in that!
5. Plan for mistakes.
    a. Add assertions to programs to check their operation.
        i. Assertions will print some debug output iff the condition you want to be true is false, at the point of the error.
    b. Use an off-the-shelf unit testing library. Look into unit tests for C++ and MATLAB.
    c. Turn bugs into test cases (or the inputs that trigger bugs, once they're corrected)
    d. Use a symbolic debugger. Breakpoints are best friends!
6. Optimize software only after it works correctly.
    a. Use a profiler to identify bottlenecks.
    b. Write code in the highest-level language possible.
        i. "Most programmers write roughly the same number of lines of code per unit time regardless of the language they use"

   ii. switch to lower-level, faster language like C only when needed
   iii. high-level languages are good for rapid prototyping. Once they work, they can be used for sanity checking a newer version in more complex, lower level code
7. Document design and purpose, not mechanics.
   a. Document interfaces and reasons, not implementations.
   b. Refactor code in preference to explaining how it works.
      i. If it doesn't absolutely need to be complicated, make it simpler. That reduces the likelihood of bugs.
   c. Embed the documentation for a piece of software in that software.
8. Collaborate.
   a. Use pre-merge code reviews.
      i. Having a peer review code before it has been committed to a shared version control repository makes sure it gets done
   b. Use pair programming when bringing someone new up to speed and when tackling particularly tricky problems.
   c. Use an issue tracking tool.
      i. like a group to-do list for a program