Tree Group Research Primer

Thanks for your interest in doing research with me! If you haven't done so already, please send me an email (tree.doug@byu.edu) before you start the training so we can talk about the group and potential openings. I'm excited to get to know you better, and look forward to having you in the group.

This document outlines a series of tasks and tutorials that I would like you to accomplish before you start as a research assistant. This process is meant to:

- make sure you have the necessary training,
- help you get a sense of what will be expected of you,
- give you an opportunity to prove that you can do self-directed tasks, and
- ensure you have sufficient time in your schedule to do research.

I anticipate that the training will take around 40 hours (or about 4 weeks if you are working 10 hrs/week) to finish. Your time spent on the training should be mostly self-directed, but I am happy to answer your questions. In other words, seek help for problems that you have tried to solve on your own first. Then, if you are still stuck or confused, feel free to email me (tree.doug@byu.edu) or come by my office (322 Clyde Building).

The first few tasks center around getting set up with a computer and an account at the Fulton Supercomputing Lab (FSL). Then there are a series of tasks designed to help you learn some important programming skills. You won't have time to learn everything about the programming languages (especially C++), so don't worry if you feel a bit lost at this point. Finally, I have a small programming project at the end that will help you tie everything together in a hands-on way.

Please let me know if you decide at some point while doing the training that you don't have enough time for research, or that you aren't as interested as you thought you were. I will be disappointed that you won't be working with us, but it won't hurt my feelings, and I won't think less of you. (Some students are nervous to say no to a professor.) Under exceptional circumstances (such as a lengthy delay in accomplishing the training materials), I reserve the right to decide to not make you an offer to join the group.

Task 1. Keep a time sheet

In order for you to make meaningful progress on your research, you need to put in a minimum of **about 10 hours a week**. This is a significant time investment, and it is common for dedicated students to find this challenging. To help you keep this goal, I would like you to keep a time sheet while you are doing the training. Your time sheet will also help me learn about your particular skills and experience. An example time sheet is included with this document.

Before moving on from this task, you should:

- Have a time sheet, and
- Read/skim through the entire tutorial.

Task 2. Learn about HPC and get an account with FSL

Log on to https://marylou.byu.edu/ and request an account with the FSL. To complete this process you will need to describe the research project. Use the following paragraph, unless we have discussed something different.

> I will be doing research with Prof. Douglas Tree in the Chemical Engineering Department as part of his undergraduate software development team. As part of this team, I will be working on developing massively parallel software to study polymer materials and other complex fluids. We anticipate that this software will typically run on multiple CPUs or on GPUs, and take several hours or days to complete.

In addition, you need to learn a little bit about high performance computing (HPC). Watch the first two (of six) introductory videos that the FSL has provided on YouTube about HPC. Additionally, spend a few minutes reading this introduction to parallel computing from Lawrence Livermore National Lab. (But don't take too much time reading the whole thing. It is quite long.)

Before moving on from this task, you should:

- Have an active FSL account, and
- Understand basic concepts about HPC.

Task 3. Learn Unix and Choose a Local Machine.

High performance computing is overwhelmingly done on a Unix (or Linux) operating system. You will need to become familiar with Unix and the terminal. In addition, you will need to be able to connect to the FSL supercomputing clusters from a local machine of your choice.

If you are unfamiliar with Unix/Linux, read through the FSL's Unix tutorial. (Another good resource is this tutorial from the Minnesota Supercomputing Institute.) You should also spend some time practicing what you are reading about. A good way to practice is to use this online Unix terminal. It is a bit slow, but it is a useful place to start. Once you have practiced a bit, you might find this cheat sheet helpful.

Once you are familiar with Unix commands, you will need to decide what computer you will use for doing research. You will use this computer to communicate with the super-computer, for coding and to do other research tasks. You have three different options for a local computer to do your work: (1) a personal laptop, (2) a CAEDM lab computer, or (3) a computer in my student offices. (Note: As of 7/13/2018, I do not yet have a computer set up for undergraduates. If you need one, please contact me ASAP so I can do this.)

The operating system that is installed on the local machine has an impact on the software you can use to do your work. Because the super-computers run Linux, it is often convenient to use a local machine that is also running Linux. However, this is not necessary, and may not be desirable if you do not feel comfortable with it. This is a choice you will have to make for yourself; I have no preference what you use, I simply want you to be productive :).

There are CAEDM computers available with Linux, and the machine in my lab is a Linux box. If you have a personal laptop, you may decide to experiment with installing Linux alongside

Windows or in a virtual machine. (Note: there are many "flavors" of Linux that you can install. Ubuntu is a particularly user-friendly version that you may want to look up, if you decide to go this route.) Again, this is up to you.

Once you have decided what machine and operating system to use, attempt to log in to the FSL head node at ssh.fsl.byu.edu. Procedures for different operating systems are outlined below. Some additional information can be found on the FSL website.

Linux/Mac

If the operating system on your local machine is some flavor of Linux then connecting to or transferring data with the FSL clusters is straightforward via the terminal with SSH, SCP and RSYNC. (Here is a tutorial for SCP and RSYNC). Mac machines are Unix machines under the hood, and they also have a terminal. As such they can also access the FSL via SSH, SCP or RSYNC. More details about these commands can be found by a Google search.

Windows

Windows does not natively connect to Linux/Unix systems without additional software. Connecting via ssh can be done with a program called Putty. You will need to follow the instructions at the Putty website to install it on windows and to connect to the FSL.

Unfortunately, Putty cannot do file transfers. A common FTP client for transferring files is FileZilla. Again, you will need to follow the instructions at the website to install FileZilla and to learn to use it to transfer files.

You have two other potential options for both ssh and FTP commands. Bitvise is a putty-like program that combines SSH and FTP commands into a single software package. I have never used it, but many of the students seem to prefer it. Details on installation and use can be found on its website.

Finally, Windows has recently begun a project to incorporate a native Linux shell. Again, I have never tried this route, but it seems promising. Details regarding installation and use can be found at this website. Connecting to a cluster and file transfers using this shell can be done using SSH, SCP and RSYNC like in Linux.

SSH Keys

It can be annoying to repetitively enter your password when copying files to and from a cluster. It is possible to set up a secure key between your machine and FSL (and as you will see later, a Git server) so that you don't have to enter your password every time you log in.

This tutorial shows you how to create an SSH key, and this one shows you how to set up password-less log-ins.

Before moving on from this task, you should:

- Have chosen a local machine and operating system
- Be able to log on to/log off from the FSL
- Be able to create, move, and delete files and directories on the cluster
- Be able to copy files and directories to/from your local machine to the cluster
- Set up SSH keys as desired

Task 4. Learn Bash, Slurm and Choose a Text Editor

Once you have chosen a local machine and you can log into the FSL head node, you need to learn some basic Bash scripting and how to submit jobs to the FSL queue.

Before you can write any Bash or Slurm commands however, you need to choose a text editor. A text editor is a program like Notepad, Gedit, Emacs or Vim (among many others) where you can type code. A text editor will be one of your main research tools. A good text editor (i.e. *not* MS Notepad) will have features such as syntax highlighting that will make it easier to write readable code. I use Vim. It has many useful features, and is available from the command line on practically every Linux machine (here is a link to a decent vim tutorial). However, there is some overhead associated with learning vim. Like the choice of machine and operating system, you should chose a text editor that is the most productive for you.

Additionally, you may want to consider an integrated development environment (IDE). IDEs are text editors coupled with debugging tools. I don't use one, but some people really like them. Examples include MS Visual Studio (Windows), XCode (Mac) and Eclipse (any).

Bash is the default language for most Unix command prompts. You can write *scripts* which are files that contain many bash commands, which you can use to automate routine tasks. Go through this bash tutorial. Don't just read it. Practice what you are learning with your text editor and a terminal on either your local machine or on FSL.

Once you feel reasonably comfortable writing bash scripts, you need to learn how to run them on the supercomputer's compute nodes. Scripts are submitted as jobs to a scheduler, and are put in a queue before being run. The scheduler being used at the FSL is called Slurm. Watch the four remaining introductory videos on the FSL website which talk about Slurm. Additionally, read FSL's documentation about Slurm. It is pretty terse, so take a look at LLNL's documentation (user manual, quick start guide) as well. Another useful tool is FSL's Slurm script generator, which may help you generate an initial script. Make sure you practice submitting scripts to the scheduler in addition to reading the documentation.

Finally, you may want to spend a little bit of extra time reading about environment modules (even more info is here). Modules are introduced in one of the videos, and they allow you to access already-installed software on the cluster. A list of available software modules is given at this FSL site.

Before moving on from this task, you should:

- Have chosen a text editor you feel comfortable with,
- Be able to write a basic bash script for automating tasks,
- Be able to submit scripts to the scheduler using Slurm commands, and
- Be able to load and unload environment modules on an FSL node.

Task 5. Learn Python Basics

Python is an interpreted programming language that is very useful for doing quick programming tasks like pre- or post-processing data and making plots. You may have already used Python if you have taken Ch En 263. Regardless, go through this tutorial by Scott Shell at UCSB on the basics of scientific programming with Python.

Python also has a robust plotting module called matplotlib. Work through the matplotlib tutorial to learn about how to make plots using Python. Again, practice what you are reading about. (If you have already taken Ch En 263, you will obviously move faster through this material than you otherwise might.) Practice will be especially effective if you try and write the codes from memory without looking at the tutorial. It will be harder to do, but you will remember the material better. For additional information on the Python language, see the official documentation on the web.

As you will see in the tutorials, you can practice your code in the interactive shell that comes with python in Linux or as scripts that you execute in the terminal. Additionally, you may want to install iPython for a more powerful interactive shell.

Spyder is an IDE-style alternative to the traditional terminal environment for programming in Python. If you have used Matlab before, you will find that Spyder gives you an very similar look and feel to the Matlab IDE. Spyder can be installed in Windows, Mac or Linux.

As I mentioned in the introduction, Python is a large and complex programming language. Don't worry if you don't master every aspect of the language. Instead, focus on the basics outlined below.

Before moving on from this task, you should:

- Be able to write a "hello world" code in Python that runs on an FSL machine,
- Be able to make and save a plot, and
- Be able to use variables, loops, if/else statements and read and write from files.

Task 6. Learn C++ Basics

C++ is a very powerful computer programming language that can be used to write large, efficient programs. With this power comes added complexity. Don't worry about mastering all of C++ at once. Instead, focus on the basics: a "hello world" code, variables, loops, functions, classes, and file I/O.

With that in mind, go through this tutorial at cplusplus.com. Unlike Python, C++ is a compiled language. So, to practice you will need to learn about how to compile your code into an executable. The MSI has a short tutorial about compiling that should help. You should be able to use either the GNU or Intel compilers on the FSL clusters.

As C++ programs become larger, they are typically divided into many different files. Compiling all of these files by hand can be cumbersome. The *make* utility was designed to overcome this problem, and makefiles are commonly encountered when programming in C++. Take a look at this tutorial and this tutorial to learn a little bit more about makefiles.

There are many books that give more information on C++. You might consider going to the library to check out a book to help you get started. StackOverflow has compiled a very useful list of references that will help you guide your search. (As an aside, StackOverflow is a great place to go when you have programming questions. Cplusplus.com is also a useful reference.) There are many more useful documents that you may also consider reading, including this article on debugging, and this introduction to the Standard Template Library.

Before moving on from this task, you should:

- Be able to write a “hello world” code in C++ that compiles and runs on a FSL machine,
- Be able to use variables, loops, if/else statements, functions, classes and file I/O,
- Know what a makefile is, and
- Spend a little bit of time exploring some more advanced C++ concepts.

Task 7. Learn Git Basics and Get a Bitbucket account

It can be difficult to keep track of different versions of a code when it is shared among many authors. Version control systems (VCS) were written as a solution to this problem. *Git* is a popular and useful VCS that we use in the group. Bitbucket is a web hosting service for Git repositories that allows us to easily share code.

Read the first three chapters of the Pro Git book to learn the basics of Git. (Atlassian, the company that runs Bitbucket, also has some useful tutorials.) Again practice is important. You can practice by making a local repository containing your C++ and Python codes that you have written during previous tasks.

Once you have a local repository, and have mastered the basics of committing and branching, go to Bitbucket.org and set up an account. Bitbucket offers free accounts to academic users, so **please use your official BYU email address when signing up to Bitbucket**.

Once you have an account, you will need to do two things to get Bitbucket to track your local repository: (i) set up ssh and (ii) push your local repo to bitbucket.

Before moving on from this task, you should:

- Know how to set up a git repository, how to commit, and how to create, delete and merge branches,
- Have a Bitbucket account,
- Be able to push from your local repo to your Bitbucket account and pull from Bitbucket to your local repo.

Task 8. Solve an ODE with Explicit Euler

You should now be able to access the FSL, understand basic Unix, write bash scripts, submit jobs using Slurm, write code in Python and C++, and use Git! To tie this all together, I want you to do a short C++ project, run it on a cluster, and produce a plot with Matplotlib. This will help you put everything you have learned together, and see if there is anything you still don’t understand.

You are going to write a code to solve a linear, first-order differential equation,

$$\frac{dx}{dt} = -3x, \qquad (1)$$
$$x(0) = 1$$

which has the analytical solution,

$$x(t) = e^{-3t}. \qquad (2)$$

The numerical method you will use is called the Explicit Euler method. It is very straightforward. Assume that the derivative on the left-hand side of Eq. 1 is discrete,

$$\frac{x_{i+1} - x_i}{t_{i+1} - t_i} = -3x_i. \qquad (3)$$

If this is so, then if we know x_0 (which we do), then we can solve for x_1 (and all of the others), assuming we pick a discrete time step $\Delta t = t_{i+1} - t_i$,

$$x_{i+1} = (1 - 3\Delta t)x_i. \tag{4}$$

With that background in place, do the following project.

1. Create a directory called "training\" in your FSL home directory.
2. Write a C++ code to solve the differential equation above. Put the code in a new directory: "training\src\".
   a) The code should read in a text file (params.dat) where the parameters of the time integration (time step, number of steps) are read from file.
   b) The code should write an output file (output.dat) with t_i and x_i in separate columns.
3. Create a Git repository to track your changes as you write the C++ code. Make regular commits and push them to Bitbucket as you work.
4. After you are sure the code works, compile the code and place the executable in a directory called "training\bin\"
5. Write a Slurm script to run the executable as a batch job. Solve the equation for $t \in [0, 9]$ for several different values of the time step parameter, $\Delta t = 10^{-z}, z \in \{-6, -5, ..., 0\}$. Store the resulting input and output files in "training\data\".
6. Write a Python script to plot the seven data sets. Calculate the error between the numerical solution you have generated and the analytical solution. Save the script and the plot in "training\plt\".

Task 9. Show Me Your Work

Congratulations! You have completed the training and have learned a lot about high performance computing. Your last task is to come talk to me and show me your work.

It is likely that you have had questions and struggles along the way, and we will have talked some already. Either way, make an appointment to come talk to me in my office. You can show me what you have accomplished, and we will get started on the research project I have in mind for you.