Mathematics Department, University of Massachusetts Dartmouth

High Performance Scientific Computing
MTHEAS 520 – Section 01 – Spring 2015
Project 1 – Due Thursday, Feb 19
Getting our act together

Most of the directives below assume you are working with a unix/linux-based operating system. Of course a Linux operating system is fine. Mac OSX ia Unix-based, and it is also fine. (There may be some slightly differences compared to a Linux distribution.) If you insist on using Windows, you will likely spend a large amount of time searching the internet for solutions to program and/or compilation problems. Consider yourself warned. All the machines in LArts 218 are Mac OSX machines, and so you can use those for completing these projects.

I expect most students will have access to a Mac OSX machine, and so this description will generally be OSX-skewed.

For most of this class, I will assume that you are working with a Unix-like terminal command prompt. You are, of course, free to use any GUI that you like.

There are six main tasks for this project:

1. Set up and learn version control (for this class: Git)
2. Set up C and Fortran compilers
3. Write some C and Fortran code (based on assignment below)
4. Set up and learn basics for mathematical document typesetting (for this class: LaTeX)
5. Write a report
6. Submit both the report and your code (with Git!)

1 Version control: Git

You can first test if you have Git installed by typing `git` at the terminal prompt. If you don't have Git installed, download a binary from: `http://git-scm.com/`.

Git is a *distributed* version control system. This essentially means that if various people all have copies of a repository, there is no "master" repository – from Git's point of view, all the repository copies are treated democratically. This has consequences for project workflows.

For this part of the project, all you need to do (a) install Git on your system, and (b) familiarize yourself with Git. The following are tutorials on the Internet that are useful for learning Git:

- `https://try.github.io/` – Interactive, browser-based, but short and limited on content
- `https://www.atlassian.com/git/tutorials` – More comprehensive tutorial
- `http://git-scm.com/docs/gittutorial` – The official tutorial from the documentation. (You get the same thing in a terminal with the command `gittutorial`.)

Make sure you are comfortable with the following commands and their basic usage:

- `git init`, `git clone`, `git pull`, `git push`
- `git add`, `git commit`, `git log`, `git status`

Note that Git includes a native visualization tool, `gitk`, which provides a nice GUI display of the commit history with details.

2 C and Fortran

You will need to know C or Fortran for this course. I recommend you learn C.

Here are some basic C tutorials:

- `http://www.cprogramming.com/tutorial/c-tutorial.html`
- `http://www.physics.drexel.edu/courses/Comp_Phys/General/C_basics/c_tutorial.html`

If you choose to learn Fortran, I would recommend you learn a later version of Fortran (e.g., Fortran 95). The following are some tutorials:

- `http://www.fortrantutorial.com/`
- `http://www.mrao.cam.ac.uk/~rachael/compphys/SelfStudyF95.pdf`
- `http://www-eio.upc.edu/lceio/manuals/Fortran95-manual.pdf`

You do *not* need to learn advanced features of either language at this stage. Just get the basics down: output to screen, writing to files, arithmetic, arrays, logical statements, conditional and loop statements, etc.

3 Coding assignment

Write programs to accomplish the following in *either* C or Fortran.

Calculate the value of π

One way to compute an approximate value for integrals is the Monte Carlo method: Let $f(x)$ be a real-valued function, where x is a multidimensional vector taking values in some region $R \subset \mathbb{R}^d$. Now let X_i for $i = 1, 2, \ldots$ be independent and identically distributed (iid) random variables. The random variables X_i are uniform random variables taking values in R. Then

$$\int_R f(x)\,\mathrm{d}x = \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(X_i).$$

In order to calculate the value of π, let $D \subset \mathbb{R}^2$ be the two-dimensional unit disc, i.e., $D = \left\{x \in \mathbb{R}^2 \mid x_1^2 + x_2^2 \leq 1\right\}$, and let R be the circumscribing square: $R = \left\{x \in \mathbb{R}^2 \mid |x_1| \leq 1 \text{ and } |x_2| \leq 1\right\}$. Let $f(x) \equiv \mathbb{1}_D(x)$ be the indicator function on D: The following where $\mathbb{1}_D(y)$ is an indicator function, defined as

$$\mathbb{1}_D(x) = \begin{cases} 1, & x \in D, \\ 0, & x \notin D. \end{cases}$$

Then clearly

$$\int_R f(x)\,\mathrm{d}x = \int_D 1\,\mathrm{d}x = \text{area}(D) = \pi.$$

Thus, if we take X_i as iid random variables uniformly distributed on D, the Monte Carlo approximation $\frac{1}{N}\sum_{i=1}^N \mathbb{1}_D(X_i)$ converges to π as N increases.

Write C and Fortran programs that compute this Monte Carlo approximation to π for $N = 10^k$ for $k = 2, 3, 4, 5$. Note that if X_i is a uniformly distributed on $R = [-1, 1]^2$, then its x and y coordinates are uniform scalar random variables on the domain $[-1, 1]$. Thus, you will need to find out how to generate uniform random variables in C and Fortran.

Approximate the Mandelbrot set

Let $z = x + iy$ be a complex number, with $i = \sqrt{-1}$. Define $p_a(z) = z^2 + a$. The *Mandelbrot set* is the collection of values $a \in \mathbb{C}$ such that

$$\lim_{n\to\infty} p_a^{(n)}(0) \neq \infty,$$

where ∞ is taken as "complex" infinity, and a nonexistent limit is equivalent to the limit not being ∞. Let M be the Mandelbrot set. (You can verify yourself directly that $a = 0 \in M$ so that M is nonempty.)

To understand how we can approximate M, consider first the known fact that if $a \in M$, then $|p_a^{(n)}(0)| \leq 2$ for *any* $n \geq 0$. Thus, if $|p_a^{(n)}(0)| > 2$ for any n, then the value a does not lie in n. Secondly, we cannot actually compute the limit as $n \to \infty$ in finite time, so we content ourselves with the assumption that, for some large N,

$$p_a^{(N)}(0) \approx \lim_{n\to\infty} p_a^{(n)}(0),$$

where of course this is a bit nonsensical if we choose an a such that the limit doesn't exist. However, combining the first and second points, we can test whether some a lies in the Mandelbrot set via the following algorithm: select a large N, set $z_0 = 0$, and recursively define $z_n = p_a(z_{n-1})$ for $n = 1, \ldots, N$. If, at some n, we test and find $|z_n| > 2$, we conclude that $a \notin M$. On the other hand, if we reach z_N with $|z_N| \leq 2$, we conclude that $a \in M$. (This latter conclusion is possibly false, hence this is only an approximation.)

Write programs to (a) compute points that fall within the Mandelbrot set, and (b) approximate the area of M. You may compute points in the set by choosing a large N (say 1000), creating an equidistant grid on $[-2.5, 1] \times [-1, 1] \subset \mathbb{C}$, and using the algorithm from the previous paragraph to test if each point fall within M. Note that you may also choose to construct a color plot of Mandelbrot set; to do so, you associate a color (i.e., a number) with each point in the equidistant grid. The way in which the color is chosen is usually via some count of number of iterations required to escape, or distance to the boundary of M, etc. You may find such a discussion on the Wikipedia page.

`https://en.wikipedia.org/wiki/Mandelbrot_set`

4 Typesetting with LaTeX

LaTeX is a typesetting language for documents. It is most often used in technical disciplines for its ability to display languages, but is used even in non-technical fields. In this class, you will write report documents for each project and submit these to me.

In order to use LaTeX, you must first install it on your system; for Mac OSX, the MacTeX package is a relatively painless way to do so: `https://tug.org/mactex`. This installs command-line tools for you to use. However, most people prefer a GUI interface. There are many operating-system-specific tools for this. E.g., on OSX, TeXShop are two such TeXworks packages.

In order to generate plots from the programs you have written, you have several options:

- Open data files in Matlab (these data files should be generated from your C/Fortran code)
- Open data files in Python. OSX comes with its own versions of NumPy and Matplotlib; if you are using your own machine, you may want to consider a custom install: `http://matplotlib.org/faq/installing_faq.html#os-x-notes`. This is not necessarily, but will allow you to more easily update in the future.
- Use another third-party but free plotting software (e.g., ParaView, gnuplot, etc.)

5 Write a report

Write a report (in LaTeX) that describes the algorithms from Section 3 and displays your results. Include details such as computational runtime (this will depend on your machine), plots, and any observations you have about the algorithms. Do *not* include C or Fortran source code in your report. I will already have your source code. (See the "Submit" section below.)

6 Submit

You must submit (1) your source code (C and Fortran), and (2) your LaTeX source code (*not* the dvi/ps/pdf output) via a Git repository.

Instructions for submission are given at
`http://www.math.umassd.edu/~anarayan/eas520/pdfs/git-instructions.pdf`

Submit to the remote repository `john-smith-project-1`, replacing `john-smith` with your first+last name.