Assignment 1: Floating Point Numbers
due: Jan 29, 2018 – 11:59 PM

GOAL: The main purpose of this exercise is to make sure everyone has access to the necessary computation and software tools for this class: a C compiler, a Python interpreter, the graphing program gnuplot, and for future use the symbolic and numeric evaluator Mathematica (though we won't be using that in this exercise). In this class, there will be help to make sure everyone's system is functional.

1 Background

1.1 Numerical Calculations

LANGUAGES & BUILT IN DATA FORMATS: The primary language at present used for high performance numerical calculations is C or C++. The standard environment is the Unix or Linux operating system. For this reason, C and Unix tools will be emphasized. We will introduce some common tools such as Makefiles and gnuplot.

That said, modern software practices take advantage of a (vast) variety of high level languages as well. We will use a bit of two interpreted/symbolic languages, Mathematica and Python, because of the power they have to develop, test, and visualize simple algorithms. Little prior knowledge except familiarity with C will be assumed.

In large scale computing all data must be *represented* somehow—for numerical computing, this is often as a *floating point* number. Floats don't cover every possible real number (there are a lot of them between negative and positive infinity, after all). They can't even represent the fraction 1/3 exactly. Nonetheless, they can express a vast expanse of numbers both in terms of precision as well as the orders of magnitude they span.

As you'll learn in this exercise, round off error, stability, and accuracy will always be issues you should be aware of. As a starting point, you should be aware of how floating points are represented. Each language has some standard built in data formats. Two common ones are 32 bit `float`s and 64 bit `double`s, in C lingo. To get an idea of how these formats work on a bit-by-bit level, give a look at https://en.wikipedia.org/wiki/Single-precision_floating-point_format and https://en.wikipedia.org/wiki/Double-precision_floating-point_format.

You'll notice we're not shy about hopping off to the web to supplement the information we've put in this document and will discuss in class, and we expect you to do the same when you need to. **Searching the web is part of this course.**

As a last remark before we hop into some math, bear in mind that data types keep evolving. **Big Data** applications (deep learning!) are now using **smaller** 16 bit floats for a lot of applications. Why? Images often use 8 bit integer RGB formats. Some very demanding high precision science and engineering applications use 128 bit floats (quad precession). There are lots of tricks in code.

Symbolic codes represent some numbers like π and e as a special token since there are no finite bit representations! See for more about these issues: https://en.wikipedia.org/wiki/Floating_point.

1.2 Finite Differences

A common operation in calculus is the *derivative*: the local slope of a function. With pencil and paper, it's straightforward to evaluate derivative analytically for well known functions. For example:

$$\frac{d}{dx}\sin(x)\bigg|_{x=x_0} = \cos(x_0) \tag{1}$$

On a computer, however, it's a bit of a non-trivial exercise to perform the analytic derivative (you'd need a text parser, you'd need to encode implementations of many functions... there's a reason there are only a few very powerful analytic tools, such as Mathematica, that handle this). In numerical work the standard method is to approximate the limit,

$$\frac{df(x)}{dx} \equiv \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2}$$

with a finite but small enough difference h. The good news is this is completely general... the bad news is this is only an approximation and it is prone to errors. Due to round off, it's dangerous for h to get close to zero. $0/0$ is an ill-defined quantity!

One approximation of a derivative is the *forward finite difference*, which should look familiar:

$$D_h^+ f(x) = \frac{f(x+h) - f(x)}{h} \tag{3}$$

Two other methods are the backward difference and the central difference. The point of this exercise is to implement different types of differences, as well as test the effect of the step size h.

2 Programming Exercises

2.1 Part 1: Simple C++ Exercise

For this first exercise we've included the shell of a program below; it's your job to fill in the missing bits. The purpose of this program is to look at the forward, backward, and central difference of the function $\sin(x)$ at the point $x = 1$ as a function of the step size h. You should also print the exact dervative $\cos(x)$ at $x = 1$ in each column.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;
```

```cpp
double function(double x) {
  return sin(x);
}

double derivative(double x) {
  return cos(x);
}

double forward_diff(double x, double h) {
  return (function(x+h)-function(x))/h;
}

double backward_diff(double x, double h) {
  // return the backward difference.
}

double central_diff(double x, double h) {
  // return the central difference.
}

int main(int argc, char** argv)
{
  double h;
  const double x = 1.0;

  // Set the output precision to 15 decimal places (the default is 6)!

  // Loop over 17 values of h: 1 to 10^(-16).
  for (h = /*...*/; h /*...*/; h *= /*...*/)
  {
    // Print h, the forward, backward, and central difference of sin(x) at 1,
    // as well as the exact derivative cos(x) at 1.
    // Should you use spaces or tabs as delimiters? Does it matter?
    cout << /*...*/ << cos(1.0) << "\n";
  }
  return 0;
}
```

Don't be afraid to search online for any information you don't know! I'm not good at programming, I'm good at Googling and I'm good at debugging. You should name your C++ program `asgn1_findiff.cpp`. You can compile it with:

```
g++ -O2 asgn1_findiff.cpp -o asgn1_findiff
```

2.2 Part 2: Finite Differences, Python

Once you've written this program in C++, your next task is to rewrite it in Python! You should name your Python program `asgn1_findiff.py`. You should verify that the outputs agree exactly. If you save the output of each program to file (this is done with the character `>` on the bash command line), you can quickly verify the outputs agree with the bash program `diff`.

I'm currently learning Python myself from Codecademy. I'm developing my own scripts through a simple text editor, but for a more integrated development environment, you can look at Spypder: https://pythonhosted.org/spyder/. If you want an even more complete IDE, give a look at Anaconda: https://www.continuum.io/anaconda-overview. Stick with Python 3.5.

2.3 Part 3: Plotting using gnuplot

You have all of this data, now what? To visualize how the finite differences for different h compares with the analytic derivative, we can plot the data using the program `gnuplot`. Using the output file you generated with C or with Python (which should be equivalent!), plot the relative error in the forward, backward, and central difference as a function of h. This is similar to what is being plotted on the right hand side of Fig. 3 in the Lecture notes. As a reminder, the relative error is defined as:

$$\frac{|\text{approximate} - \text{exact}|}{|\text{exact}|} \tag{4}$$

Don't forget to set x and y labels on your graph, and titles for each curve in the key.

By default `gnuplot` will output to the screen. You'll want to submit an image at the end of the day; the commands `set terminal` and `set output` will be helpful in this regard! As an FYI: while it's best to play with making plots in the `gnuplot` terminal, it can get annoying to do everything there! `gnuplot` can just run a script file:

```
gnuplot -e "load \"[scriptname].gp\""
```

Where you should replace `[scriptname]` with, well, the name of your plotting script!

> **Extra Credit& Extra Fun:** Once you have a program, it is good to see what else you can do. If you want to see a function that is difficult to numerically approximate, try the derivative of $\sin(1/x)$, which is exactly $-\frac{\cos(1/x)}{x^2}$, at some point close to zero, say $x = 0.0001$. Don't pass this in, but you can brag about in class for virtual extra credit.

2.4 Submitting Your Assignment

This first assignment is due at 11:59 pm on Monday, January 30. Please e-mail a **tarball** containing the assignment to the class e-mail, bualghpc@gmail.com. Include your name in the tarball filename.

If you're not familiar with tar, here's a sample instruction that perhaps I would use:

```
tar -cvf evan_weinberg_asgn1.tar asgn1_findiff.cpp asgn1_findiff.py asgn1_findiff.pdf
```

You may want to include other files (such as your gnuplot plotting script, though it's not required). **Do NOT include a compiled executable!** That's a dangerous, unsafe practice to get into.

A   Software Tools: Nothing to Pass in!

A.0.1   Part I: Accessing BU Computing

Make sure you can access BU's "Linux Virtual Lab" by following the instructions here:

http://www.bu.edu/tech/services/support/desktop/computer-labs/unix/

These Linux machines aren't for running long calculations, but they are useful for small, interactive jobs. (I think any job will get killed after 15 minutes—don't quote me on that.) These machines also allow access to Mathematica.

Access to BU Computing is not a substitute for installing Mathematica and standard Unix compilers on your own machine, as described below.

A.0.2   Part II: Making sure you have a C++ compiler

In this class, we'll be using the standard compiler "g++". If you have a Mac or a Linux install, g++ may exist already. Try running the command:

```
which g++
```

from the terminal. On my machine, it returns:

```
/usr/bin/g++
```

But your mileage may vary. If it returns nothing, it means you don't have g++ installed, which you should go do! I'd be surprised if it wasn't installed, though.

If you're on Windows, you'll need to install Cygwin, which even I struggled with—other options are dual-booting, or a much better idea is installing Linux in a virtual box! If you're interested in that but not brave, send me (Evan) an e-mail at `weinbe2@bu.edu` and I'll help you out!

To make sure you understand compiling without an IDE (Integrated Development Environment), follow the quick tutorial here: https://en.wikibooks.org/wiki/C%2B%2B_Programming/Examples/Hello_world

Amazing Interactive Tutorias: C++

http://www.tutorialspoint.com/cplusplus/index.htm

A.0.3 Part Iii: Installing gnuplot

You may have gnuplot already installed on your machine. You can test this the same way we tested for g++:

```
which gnuplot
```

If it returns a path, you have gnuplot installed! If not, use your favorite package manager to install it. I'm an Ubuntu user, so I had to run:

```
sudo apt-get install gnuplot
```

If you're on a different distribution, you'll probably need to use `yum`, or some GUI tool. On Mac OS X, an optional package manager is Brew: http://brew.sh/, which will help you out.

By looking around on stackoverflow, I found a sample brew install command:

```
brew install gnuplot --wx --cairo --pdf --with-x --tutorial
```

Which will let you output PDFs as well as to the screen (that's the whole `with-x` and `wx`), I imagine. If you get stuck, let us know!

To test out gnuplot in OS X or Linux, run:

```
gnuplot
```

from the terminal. This will put you in an interactive gnuplot terminal. A few useful commands:

```
# Hashes aren't for twitter, they're for comments in gnuplot!
plot sin(x) # plot the sine function
f(x) = cos(x) # assign a function
plot sin(x), f(x) # plot two functions at once.
set xrange [0:2] # change the x axis.
set yrange [-2:2] # change the y axis range.
replot # update the plot with your new axis.
set yrange [-5:-2] # change the y axis range again.
replot # you won't see anything! So do...
reset # ... because you've messed up!
set xrange [-1:1]
plot x*sin(1/x) # This will lootk really bad!
set samples 1000 # sample the function more frequently.
replot # it should look a lot better now
exit # and we're done!
```

Your will want to save a figure from time to time. In this case before you exit add in these instructions.

```
set term postscript color   #one option that gives a .ps figure.
set output "myfigure.ps"    #whatever you want to name it
```

```
replot                       #send it to the output
set term x11                 #return to interactive view.
                             #On linux, you may need ''wxt'' instead of x11.
```

A.0.4 Part IV: Installing Mathematica

As students, you can luckily install Mathematica on your own computer without much pain. Follow this link and install Mathematica:

http://www.bu.edu/tech/support/desktop/distribution/mathsci/mathematica/student/step-1/

We've tested this on both Windows and Mac OS X. Mathematica will also work on standard Linux distros, we've just never tried installing it there ourselves—please try and let us know asap if you have issues.

After installing Mathematica, you should go through the following quick tutorials. They cover very simple topics, such as plotting, differentiation, and integration. The differentiation and integration articles go into much deeper mathematical detail than you'll need in this class! Just gleam out how to take a simple derivative and perform a simple integral. Don't let the word "Hessian" scare you.

- Plotting functions: https://reference.wolfram.com/language/tutorial/BasicPlotting.html

- Plotting data: https://reference.wolfram.com/language/howto/PlotData.html

- Differentiation: https://reference.wolfram.com/language/tutorial/Differentiation.html

- Integration: https://reference.wolfram.com/language/tutorial/Integration.html

We will suggest further reading as the need arises!

A.0.5 Part IV: Installing Anaconda for Python

You can get Anaconda distribution of python at

https://www.continuum.io/downloads

Get the one for Python 3.5 for your computer(s).

https://docs.python.org/2/tutorial/index.html

Amazing Interactive Tutorials:

https://docs.python.org/2/tutorial/index.html

http://www.tutorialspoint.com/python3/index.htm

http://docs.python-guide.org/en/latest/writing/style/#zen-of-python