Assignment 1: From Derivatives to Finite Differences

Due Feb 3, 2024. Submit both written codes and writen answers and figures as pdf in the directory `/projectnb/ec526/students/yourloginname/HW1` on your SCC account by Jan 29, 11:59PM.

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, the graphing program in some way. The basics UNIX tool bf gnuplot is find but soon we will use Mathematica for great graphics and symbolic abilities for exact equation (easy way to check algebra and do amazing graphics). In this class, there will make sure everyone's laptop is functional and everyone has access to the Shared Computer Cluster.

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 plotting with gnuplot.

That said, modern software practices take advantage of a (vast) variety of high level languages as well. You also may use a bit of two symbolic or interpreted languages, Mathematica or Python respectively, because of the power they have to prototype, 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 `floats` and 64 bit `doubles`, 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 or even 8 bit floats for some

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 $\boldsymbol{\pi}$ and $\boldsymbol{e}$ as a special token since there are no finite bit representations! See for more about these issues: https://en.wikipedia.org/wiki/Floating_point. This has lots of interesting history and examples to show how tricky floating point arithmetic really is! Read for fun.

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:

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

Two other methods are the backward difference

$$\Delta_{-h} f(x) = \widetilde{\Delta}_h f(x) = \frac{f(x)-f(x-h)}{h} \tag{4}$$

and average or 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 Written Exercise

Computer Engineering is link between mathematics (logic) and hardware (physics). The kind of math stuff you need is very practical part often submerged in math courses in too much formalism. You learn this language by speaking it. To make this link in the current exercise, first know how you expand a function in a Taylor series for small h. Most often the quadratic approximation is enough,

$$f(x+h) \simeq f(x) + hf'(x) + (h^2)/2f''(x) + \cdots \tag{5}$$

for *small* h.

More generally we can write more and more term (if you care!)

$$f(x+h) \simeq f(x) + h\frac{df(x)}{dx} + \frac{h^2}{2!}\frac{d^2f(x)}{dx^2} + \frac{h^3}{3!}\frac{d^3f(x)}{dx^3} + + \frac{h^4}{4!}\frac{d^4f(x)}{dx^4} + O(h^5) \tag{6}$$

This means the error using 3 term scales like h^4 or using 4 term is h^5 etc.

2.1 Part 1: Averaging forward and backward Differences

Calculate using *pencil and paper* the following expressions,

$$\begin{aligned} a = \Delta_h f(x) &= [f(x+h) - f(x)]/h = ? + O(?) \\ b = \widetilde{\Delta}_h f(x) &= [f(x) - f(x-h)]/h = ? + O(?) \\ (a+b)/2 = (1/2)[f(x+h) &- f(x-h)]/h = ? + O(?) \\ A = \Delta_{2h} f(x) &= [f(x+2h) - f(x)]/(2h) = ? + O(?) \\ B = \widetilde{\Delta}_{2h} f(x) &= [f(x) - f(x-2h)]/(2h) = ? + O(?) \end{aligned} \tag{7}$$

using the Taylor series expression above in powers of h. Note the third expression in Eq. 7 is just the average of the first two. This is the *central difference* which cancels all odd h powers in the expansion (or odd terms for the approximate derivative.)

HINT: Do this the easy way! The quadratic approximation is enough for this.

2.2 Part 2: Double averaging

For extra credit combine the 4 expression of the derivative a, b, A, B with magic weights to reduce the error to h^5.

$$2a/3 + 2b/3 - (A/6 + B/6) = \frac{df(x)}{dx} + O(h^4) \tag{8}$$

This a clever *average* of two central difference! Do the algebra to get the magic formula in Eq. 8. It is ok (as I did) to use Mathematica or some symbolic program to get the actual expression for the error term to be $(h^4/30)\frac{d^5f(x)}{dx^5}$! It is smart to *cheat* but also good to do it out my hand to really understand what you are doing and to exercise your brain.

3 Programming Exercises

3.1 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 derivative $\cos(x)$ at $x = 1$ in each column.

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

using namespace std;

double f(double x) {
  return sin(x);
}

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

double forward_diff(double x, double h) {
  return (f(x+h)-f(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 `findiff.cpp`. You can compile it with:

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

3.2 Part 2: 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`. This is a basic tool and with scripts you can design useful general for plotting and fitting.

Of courses there are many plotting and data analysis programs. You may want to use others. Personally I am trying to interface with Mathematica. But in the high performance computing environment dumping data in to simple files and using a `gnuplot` is sometime useful because it alway there. Using the output file you generated with C 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:

$$\frac{|\text{approximate} - \text{exact}|}{|\text{exact}|} \tag{9}$$

Nice to set x and y labels on your graph, and titles for each curve so you know what is plotted.

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!

3.3 Submitting Your Assignment

Both the written part of first assignment as pdf, source code and the plots are due Thursday Feb 29, 11:59PM. All should be posited in your CCS account `/projectnb/paralg/LoginName/HW1`

- The code must compile from a Makefile. Only have source code and the Makefile – later we may have input files as well.
- **Do NOT include a compiled executable!** in `HW1`

- Make a subdirection `/projectnb/paralg/LoginName/HW1/doc` for all the written solution, figure or other putput files.

This will insure that all codes will compile and run in a uniform high performance environment. On GitHub there will be a Makefile that does the compilation automatically.

4 Things to try if you are bored in these strange times!

Extra credit is really not required and there is no strict due date! Here are some suggestions to explore further the ideas. Playing around with code is the best way to learn, you many find some surprises and it is fun. If you find cool or strange results you might show the in class or maybe get start on thinking about project.

Extra Credit:: Once you have a program, it is good to see what else you can do. Try functions, $10^{-6}x + 10^{20}$ and e^{-10x} at $x = 1$. You can do more if you get hooked. 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 $x = 0.0001$. Your don't have to pass this in, but you can brag about in class for virtual extra credit and show the result in class. Discussion variation of the assignments is part of classroom activities.

Extra Extra Credit: For simple powers it is possible to use simple math to avoid the limit of 0/0. For example

$$\begin{aligned}
\Delta_h 1 &= 0 \\
\Delta_h x &= \frac{x+h-x}{h} = x \\
\Delta_h x^2 &= \frac{(x+h)^2 - x^2}{h} = \frac{x^2 + 2xh + h^2 - x^2}{h} = 2x + h \\
\Delta_h x^3 &= \frac{(x+h)^3 - x^2}{h} = \frac{x^3 + 3x^2h + 3xh^2 + h^3 - x^3}{h} = 3x^2 + 3xh + h^2
\end{aligned} \tag{10}$$

Of course you can do this with any power using the binomial theorem:

$$(x+h)^n = \sum_{k=0}^{n} \frac{n!}{k!(n-k)!} x^{n-k} h^k \tag{11}$$

. Subtraction drop the first term and leads with $nx^{n-1}h$. Nice. Therefore any function as Taylor series. Apply this to a simple approximation of $\sin(x)$ for small x

$$f(x) = x - x^3/6 \tag{12}$$

Now use the tricks above to compute $\Delta_n f(x)$ using the tricks above and get a result that is better and better a $h \to$ with no round off problem! Math works.

A Software Tools: Nothing to Pass in!

Here is some random advice and suggestions. They probably **not** upto date. Things keep changing all the time– that is why google has taken over for any documentation. `Also PLEASE exchange question and complaints on Slack so we can crowd source the answer.`

Of course you may have better ways to do that you like. That is ok! You will want to be able to do your work on your laptop. For this you may want to have a unix environment and the default `gnuplot` at least. All unix tools also exist at the SCC and remote acess is now pretty darn good.

A.0.1 Part I: 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.

Amazing Interactive Tutorials: C++

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

A.0.2 Part II: 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 look 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.0.3 Part III: 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/services/cccs/desktop/distribution/mathsci/mathematica/student/

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

After installing Mathematica, you can go through the following quick tutorials BUT I will give out a few simple examples. To me this a better way to learn the minimal that is useful. Mathematica under `help` has a `Wolframe Document` tab with live scripts for more than any one wants. Here are some tutorial which are basically the same as `Wolframe Document` .

- 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

Also I find the googling subjects including `Mathemtica` in the search gives nice brief dicussions of lots of interesting things.