Assignment 1
Phys 512 Computational Physics

Hans Hopkins 260919593

September 18, 2021

Q1:
a) Writing out the expansions for each term:

$$f(x+\delta) \approx f(x) + f'(x)\delta + \tfrac{1}{2}f''(x)\delta^2 + \tfrac{1}{6}f'''(x)\delta^3 + \tfrac{1}{24}f^{IV}(x)\delta^4 + \tfrac{1}{120}f^{V}(x)\delta^5$$

$$f(x-\delta) \approx f(x) - f'(x)\delta + \tfrac{1}{2}f''(x)\delta^2 - \tfrac{1}{6}f'''(x)\delta^3 + \tfrac{1}{24}f^{IV}(x)\delta^4 - \tfrac{1}{120}f^{V}(x)\delta^5$$

$$f(x+2\delta) \approx f(x) + 2f'(x)\delta + 2f''(x)\delta^2 + \tfrac{8}{6}f'''(x)\delta^3 + \tfrac{16}{24}f^{IV}(x)\delta^4 + \tfrac{32}{120}f^{V}(x)\delta^5$$

$$f(x-2\delta) \approx f(x) - 2f'(x)\delta + 2f''(x)\delta^2 - \tfrac{8}{6}f'''(x)\delta^3 + \tfrac{16}{24}f^{IV}(x)\delta^4 - \tfrac{32}{120}f^{V}(x)\delta^5$$

We want to linearly combine these to give 0f, 2f', 0f", and 0f"'. Solving the linear system gives

$$\text{deriv} \approx \frac{-\frac{1}{6}(f(x+2\delta) - f(x-2\delta)) + \frac{4}{3}(f(x+\delta) - f(x-\delta))}{?\delta}$$

We need to find what's on the bottom to make sure it actually works as a derivative approximation, and it works out to 2δ. Simplifying gives

$$\text{deriv} \approx \frac{-(f(x+2\delta) - f(x-2\delta)) + 8(f(x+\delta) - f(x-\delta))}{12\delta}$$

b) Substituting each term for its expansion and cancelling gives (noting that the fourth derivatives cancel)

$$\begin{aligned}\text{deriv} &\approx \frac{-(4f'\delta + \frac{16}{6}f'''\delta^3 + \frac{64}{120}f^{V}\delta^5) + 8(2f'\delta + \frac{2}{6}f'''\delta^3 + \frac{2}{120}f^{V}\delta^5)}{12\delta} \\ &= f' + \tfrac{2}{3}f^{V}\delta^4\end{aligned}$$

So the first order to the error from cutting off the Taylor series expansion is $\frac{2}{3}f^{V}\delta^4$.

Let ϵ be 10^-7 if the variable is a single, and 10^-15 if it's a double. Then the error from the machine precision is on the order of ϵf. Dividing that by the δ gives $\frac{\epsilon f}{\delta}$.

This is two different sources of error, so we want to minimize the variance.

$$\begin{aligned}
\text{var} &= \left(\frac{\epsilon f}{\delta}\right)^2 + \left(\frac{2}{3} f^V \delta^4\right)^2 \\
&= \frac{\epsilon^2 f^2}{\delta^2} + \frac{4}{9}(f^V)^2 \delta^8 \\
\frac{d\text{var}}{d\delta} = 0 &= -\frac{2\epsilon^2 f^2}{\delta^3} + \frac{32}{9}(f^V)^2 \delta^7 \\
\implies \delta &= \sqrt[10]{\frac{9}{16}} \left|\frac{\epsilon f}{f^V}\right|^{\frac{1}{5}}
\end{aligned}$$

The numerical factor in front is about $0.95 \approx 1$, so we'll drop it. This leaves:

$$\delta \approx \left|\frac{\epsilon f}{f^V}\right|^{\frac{1}{5}}$$

And if we assume $\frac{f}{f^V}$ is order unity,

$$\delta \approx \epsilon^{\frac{1}{5}}$$

But let's try to estimate $\frac{f}{f^V}$. Since we can afford to me more rough with this, we'll use the $f'(x) \approx \frac{f(x+\gamma) - f(x)}{\gamma}$ approximation for an arbitrary $\gamma > 0$.

This gives:

$$\begin{aligned}
f^V(x) &\approx \frac{f^{IV}(x+\gamma) - f^{IV}(x)}{\gamma} \\
&\approx \frac{f'''(x+2\gamma) - f'''(x+\gamma) - (f'''(x+\gamma) - f'''(x))}{\gamma^2} \\
&= \frac{f'''(x+2\gamma) - 2f'''(x+\gamma) + f'''(x)}{\gamma^2} \\
&\approx \frac{1}{\gamma^3}\left(f''(x+3\gamma) - f''(x+2\gamma) - 2(f''(x+2\gamma) - f''(x+\gamma))\right. \\
&\qquad \left. + f''(x+\gamma) - f''(x)\right) \\
&= \frac{1}{\gamma^3}\left(f''(x+3\gamma) - 3f''(x+2\gamma) + 3f''(x+\gamma) - f''(x)\right) \\
&\approx \frac{1}{\gamma^4}(f'(x+4\gamma) - f'(x+3\gamma) - 3(f'(x+3\gamma) - f'(x+2\gamma)) \\
&\qquad + 3(f'(x+2\gamma) - f'(x+\gamma)) - (f'(x+\gamma) - f'(x)))
\end{aligned}$$

Continued:

$$
\begin{aligned}
&= \frac{1}{\gamma^4}(f'(x+4\gamma) - 4f'(x+3\gamma) + 6f'(x+2\gamma) - 4f'(x+\gamma) \\
&\quad + f'(x)) \\
&\approx \frac{1}{\gamma^5}(f(x+5\gamma) - f(x+4\gamma) - 4(f(x+4\gamma) - f(x+3\gamma)) \\
&\quad + 6(f(x+3\gamma) - f(x+2\gamma)) - 4(f(x+2\gamma) - f(x+\gamma)) \\
&\quad + f(x+\gamma) - f(x)) \\
&= \frac{1}{\gamma^5}(f(x+5\gamma) - 5f(x+4\gamma) + 10f(x+3\gamma) - 10f(x+2\gamma) \\
&\quad + 5f(x+\gamma) - f(x))
\end{aligned}
$$

That hopefully gives a good estimate for f^V. That $\frac{1}{\gamma^5}$ is a big problem though. I guess I'll set $\gamma = 0.01$ because I don't want to deal with hitting 10^{-15}.

Testing it on $e^{0.01x}$, it seems pretty bad. I think it's hitting some major roundoff problems. But maybe δ can be off by a few orders of magnitude.

The plan to show that the δ is valid is to compute the derivatives for points in a specific range, and then sum up all the differences, and then compare that to the same process with different δs. I'll choose 100 points and work in the range of [-3,3] completely arbitrarily. I'm working in double precision, so $\epsilon = 10^{-15}$.

Well I ran my code (the file called asst1_Q1.py) and it looks good. Here's the results:

For e^x:

Tested δ	Sum of differences
100δ	0.0010598653688042511
δ	5.250776896614795e-11
0.01δ	1.6174582923023095e-09

For $e^{(0.01x)}$:

Tested δ	Sum of differences
100δ	3.664540008246897e-10
δ	3.5596872671739277e-13
0.01δ	3.7544517841880953e-11

In both cases, the guessed δ (middle row) produces the least difference from the actual derivative.

Q2:

This one's mostly done in code. I do have to estimate the third derivative though.

$$\begin{aligned} f'''(x) &\approx \frac{f''(x+dx) - f''(x)}{dx} \\ &\approx \frac{f(x+2dx) + f(x) - 2f(x+dx) - (f(x+dx) + f(x-dx) - 2f(x))}{dx^3} \\ &= \frac{f(x+2dx) - 3f(x+dx) + 3f(x) - f(x-dx)}{dx^3} \end{aligned}$$

I used this result from class for guessing dx:

$$dx = \left(\frac{\epsilon f}{f'''}\right)^{\frac{1}{3}}$$

I ran the code and it plots the real derivative directly over the numerical derivative, so that's good.

Q3:

Immediately I notice that around the voltage of 1.13, the slope seems to change dramatically. That might cause a bit of an issue.

All the points are pretty close together, and there's not a lot of extreme changes between points, so a polynomial connecting the points is probably fine. There is a bit of curve, so I won't just connect points with a line. I'll do a cubic fit for each point. (Except for the ones close to the edges where there's not enough data points for cubic, in which case I'll do linear.)

I ran the code and it gave this:

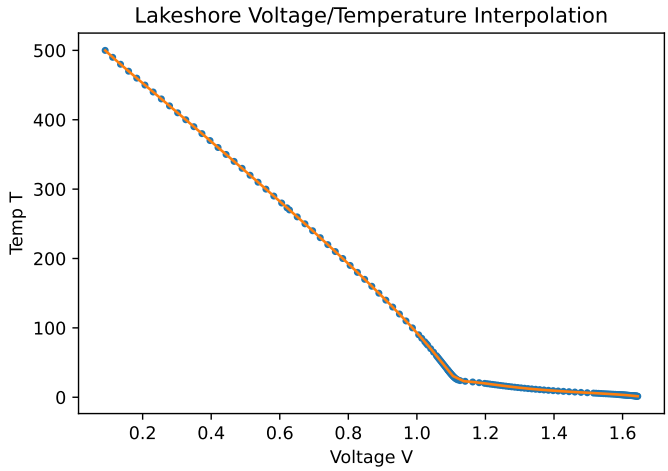

Figure 1: The blue points are the lakeshore data points and the orange line is interpolating a linspace. The interpolation matches the points pretty well. Even the harsh corner bit seems okay (from far back anyway).

The next part is to figure out the error. To do this, I'm going to use the "second method" from the tutorial in bootstrap_interp.py because it looks like it works.

So I'm choosing a bunch of subsets of original data, doing my interpolation on those, and then seeing how much those results vary.

The big issue with this method is that it introduces two different new parameters, the number of times I resample and the percent of points that we randomly choose in each resample, and both of those seem to change our error estimate. Ideally in an error estimate we would want the error to converge onto something as the parameters get better, but of course if we choose to keep 100% of points, we would always keep the same answer, and our error would just be 0. Choosing to keep 80% of points each time provides a good order of magnitude estimate I think. The other problem with it is that it's quite a bit slower to find the errors than to do the interpolation once. If I was writing this code for real, I would add an option to not compute the errors.

Q4:

To compare the fits, I found the difference for 50 points in the interval then added up the absolute values of those differences. Here are the results:

For cos:

Type of fit	Sum of differences
Polynomial fit	0.002273628366990371
Spline	0.0018463375656032543
Rational fit	3.2034001642652423e-06

For lorentzian:

Type of fit	Sum of differences
Polynomial fit	2.808543495991708
Spline	2.788298436087891
Rational fit	126.49128443876721

In both cases the spline and polynomial fit are about the same. The rational fit is really good for cosine, but it's terrible for the Lorentzian.

The error for the rational fit of the Lorentzian should be 0 theoretically. Since if you set $P(x) = 1$ and $q(x) = 1 + x^2$ you have exactly the Lorentzian. In my case for the above numbers, I had a higher n and m, and the fit wasn't good at all. I printed out the determinant of the matrix inside the rational fit, and it's like 10^{-38}, which is not good, since it gets inverted in the next line.

And when I tried it with pinv, it gives the error as 2.79, which is about the order of error as the other fits.

The last step before inverting the matrix is

$$y = \begin{bmatrix} X & -yX' \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix}$$

If the 0th order terms wasn't fixed to be 1, then this would look like

$$y = \frac{1}{a} \begin{bmatrix} X & -yX' \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix}$$

Which leaves an extra degree of freedom.

More generally, if there's more ms and ns than are required, they also create extra degrees of freedom, which manifest as linearly dependent lines in the matrix. As I saw, the matrix becomes next to singular, so it's inaccurate when inverted. pinv just sets the eigenvalues to 0 when inverted, so the extra degrees of freedom are ignored.