Assignment 4

1. This exercise reviews some aspects of Taylor approximations of functions of more than one variable. it shows that the idea of *directional derivative* allows you to find multi-variate Taylor approximations using one variable calculus (which "everyone" understands).

   (a) Let $f(t)$ be a scalar-valued function. We say that $f(t) = O(t^p)$ as $t \to 0$ if there is an $\epsilon > 0$ and a C so that
   $$f(t) \leq C\, t^p \quad \text{if } |t| \leq \epsilon \; .$$
   If p is not an integer and t might be negative, the corresponding thing might be $f(t) = O(|t|^p)$. If f might be negative, the corresponding thing might be $|f(t)| = O(|t|^p)$. The important thing is always $\epsilon > 0$ and C. One of the one variable Taylor remainder formulas is
   $$f(x+t) = f(x) + tf'(x) + \cdots + \frac{t^n}{n!} f^{(n)}(x) + \frac{t^{n+1}}{(n+1)!} f^{(n+1)}(\xi) \; .$$
   The unknown ξ is somewhere between x and $x+t$ (even if $t<0$). The hypothesis is that the derivative $f^{(n+1)}(y)$ is a continuous function of y for y between x and $x+t$. A basic theorem of mathematical analysis is that if $h(y)$ is a continuous function of y on a finite closed[1] interval, then h is bounded (there is a C with $|h(y)| \leq C$ for all y in the closed interval). Show that if the derivative $f^{(n+1)}(y)$ is continuous on the closed interval $[a,b]$, then the order n Taylor series approximation error satisfies
   $$\left| f(x+t) - \left(f(x) + tf'(x) + \cdots + \frac{t^n}{n!} f^{(n)}(x) \right) \right| = O(|t|^{n+1}) \; . \tag{1}$$
   Assume that $x \in (a,b)$ (meaning that $a < x < b$) with endpoints excluded).

   (b) Show that if $f(t) = O(t)$ and $g(t) = O(t)$ then $h(t) = f(t)g(t)$ has $f(t)g(t) = O(t^2)$. Find the general version of this for $|f(t)| = O(|t|^p)$ and $|g(t)| = O(|t|^q)$. *Explanation.* This is an exercise in working with the definition of "big Oh". You have to show that there is an ϵ and a C. You can use the fact that both f and g have their own ϵ and C, but that these might be different, and that the C for fg depends on the C for f and the (probably different) C for g.

[1] An interval is *closed* if it includes its endpoints. As an example, $h(y) = \frac{1}{y}$ is continuous on the *open* interval $(0,1)$, which is the set of points $0 < y < 1$ (endpoints 0 and 1 excluded) but is not continuous on the closed interval $[0,1]$ because it is not continuous when $y = 0$. The function $h(y) = \frac{1}{y}$ is not bounded on $(0,1)$.

(c) Let x have n components: $x = (x_1, \cdots, x_n)$. Let $f(x)$ be so that all the partial derivatives in the formulas we get exist and are continuous functions of x. The gradient of f is the vector

$$g = \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} .$$

Let v be an n component vector. The *directional derivative* in the direction v is

$$\frac{d}{dt} f(x + vt) \Big|_{t=0} .$$

Show that

$$\frac{d}{dt} f(x + vt) \Big|_{t=0} = v^T \nabla f(x) .$$

The right side of this is often written $v \cdot \nabla f$ or $\nabla_v f$. Use this and Exercise 1 to show that

$$f(x + vt) = f(x) + t g^T v + O(t^2) .$$

(d) The *Hessian* matrix is the $n \times n$ matrix of second partial derivatives

$$H_{jk} = \frac{\partial^2 f}{\partial x_j \partial x_k} .$$

It is a theorem of calculus that H is symmetric. Show that

$$\frac{d^2}{dt^2} f(x + vt) \Big|_{t=0} = v^T H v .$$

Use this and Exercise 1 to show that

$$f(x + vt) = f(x) + t g^T v + \frac{t^2}{2} v^T H v + O(t^3) .$$

(e) A matrix H is *positive definite* if there is a $\lambda_{\min}$ so that

$$v^T H v \geq \lambda_{\min} \|v\|^2 .$$

[We will see that there is such a $\lambda_{\min} > 0$ if $v^T H v > 0$ whenever $v \neq 0$.] Suppose $\nabla f(x_*) = 0$ and $H(x_*)$ is positive definite, show that x_* is a *local minimizer* in the sense that there is an $\epsilon > 0$ so that $f(x) > f(x_*)$ if $x \neq x_*$ and $\|x - x_*\| \leq \epsilon$.

2. The idea of directional derivatives can be applied to functions of a matrix, either matrix-valued functions such as $f(A) = A^{-1}$ or scalar-valued functions[2] such as $f(A) = \det(A)$. Much of linear algebra perturbation theory

[2]In math, a *scalar* is a number or the entry of a 1×1 matrix. In physics, a scalar is a one-component quantity that transforms in a specific way under changes of coordinates. We are using the math definition here.

can be derived in this way. A simpler and equivalent form is to suppose that A is a differentiable function of t and calculate derivatives using the chain rule and the product rule. It is convenient to use a dot to represent differentiation with respect to t, as

$$\dot{A} = \frac{d}{dt} A(t) \ .$$

The entries of $\dot{A}$ are the derivatives of the entries of $A(t)$.

(a) Show that if $A(t)$ and $B(t)$ are differentiable functions of t, then

$$\frac{d}{dt} \left[A(t)B(t) \right] = \dot{A}B + A\dot{B} \ .$$

(b) Assume that A^{-1} is a differentiable function of t. Show that

$$\frac{d}{dt} A^{-1}(t) = -A^{-1}(t)\dot{A}(t)A^{-1}(t) \ .$$

Give an example to show that this is not equal to either of

$$-A^{-2}(t)\dot{A}(t) \quad \text{or} \quad -\dot{A}(t)A^{-2}(t) \ .$$

Here (and often in the future) we use the notation

$$A^{-2} = \left(A^{-1}\right)^2 \ .$$

(c) We say that $B(t) = O(t^p)$ if all the entries of B are order t^p. Show that

$$A(t) = A(0) + t\dot{A}(0) + O(t^2) \ .$$

Don't worry if this seems easy. It is. We will write A_0 for $A(0)$.

(d) Derive the formula

$$\frac{d}{dt} \det(A) = \det(A_0) \operatorname{Tr}(A_0^{-1}\dot{A}) \ . \tag{2}$$

Some facts:

- $\det(AB) = \det(A)\det(B)$.
- $\operatorname{Tr}(B)$ is the sum of the diagonal entries of B. $\operatorname{Tr}(AB) = \operatorname{Tr}(BA)$ (so the order on the right of (2) does not matter).
- $\det(I + tB) = 1 + t\operatorname{Tr}(B) + O(t^2)$.

Remark. $\operatorname{Tr}(A) = \sum \lambda_j(A)$ (the trace is the sum of the eigenvalues). The eigenvalues $\lambda_j(t)$ do not have to be differentiable functions of t even if A is. Since the differentiable function is a sum of non-differentiable functions, the singularities (places where $\lambda_j(t)$ are not differentiable) must cancel in some way.

(e) Suppose $H(t)$ is a family of positive definite symmetric matrices and a differentiable function of t. Suppose $H(t) = L(t)L^T(t)$ and L is lower triangular and a differentiable function of t. Find an formula/algorithm that uses L and $\dot{H}$ to compute $\dot{L}$. This is perturbation theory for the Cholesky decomposition.

3. A linear combination of decaying exponentials is

$$f(t) = \sum_{k=1}^{n} a_k e^{-\lambda_j t} \ . \tag{3}$$

Suppose there are measurements

$$Y_j = f(t_j) + \epsilon_j \ .$$

Here, the observed value Y_j differs from the true value $f(t_j)$ by measurement error ϵ_j (common notation in statistical estimation problems). We write $f(t, \lambda, a)$ to describe the function (3) even if the numbers $\lambda_1, \cdots, \lambda_n$ and $a_1, \cdots, a_n$) are not the values that generated the data. We "abuse notation" to write $a = (a_1, \cdots, a_n)$, etc. A *sum of squares* measure of *goodness of fit* is

$$u(\lambda, a) = \sum_{j=1}^{n} \left(Y_j - f(t_j, \lambda, a)\right)^2 \ .$$

Least squares parameter estimation means finding the n rates λ_k^* and the n amplitudes a_k^* to minimize this function of $2n$ variables

$$(\lambda^*, a^*) = \text{arg min } u(\lambda, a) \ .$$

We will return to this optimization problem in later assignments.

This exercise involves reading from a data file, then using the data to evaluate u and its derivatives.

This exercise uses *fake data*.[3] The data are in three files, one ascii file `[name].py` that gives the values of m and n and the names of two *binary* files, one with λ and a, the other with the times t_j and corresponding Y_j. *Binary* format means that the files contain the exact bit sequences of the IEEE format floats that represent λ, etc. You put data in binary files because the binary representation is more efficient (makes smaller files) and does not require you to format floats as strings and then convert the strings back to floats (which is not exact and can be slow). This can make a big difference for large datasets. You (probably) cannot open a binary file in a text editor or view it in a web browser window.

[3] *Fake data* is a technical term for artificial data generated by a computer model rather than real data that consists of actual measurements of something. Data analysis often starts seeing how the analysis algorithm works on fake data where you know the *ground truth* (the exact form of the model and parameters) before trying it on real data.

There are two sets of fake data. The first set is `DataInfo.py`, which points to binary files `ModelParameters` (containing λ and a) and `TimeSeries` (containing the numbers t_j and Y_j). Download these along with `DataAnalysis.py`. Then run `DataAnalysis.py`. You should get printout of the 20 data pairs t_j and Y_j and the six (two sets of three) model parameters λ_k and a_k. Then download and run `FakeData.py`. This is the module that created the fake data you just saw. It also prints the data and model values, which should match the ones you printed before exactly.

The second set of fake data files are called `DataInfo2.py`, `ModelParameters2` and `TimeSeries2`. Download these and edit `DataAnalysis.py` to import `DataInfo2.py` (line 15). You will see that these files are bigger, with more model parameters and more (fake) observations. Edit out the lines that print the data (lines 30 to 33 and 42 to 45). Use this data for the tasks below.

This exercise asks you to compute derivatives of u with respect to λ with the amplitude parameters a fixed. Therefore, we write $u(\lambda)$ for $u(\lambda, a)$. The values of a never change. Use calculus (the chain rule, etc.) to find formulas for the gradient and hessian of u:

$$g = \nabla u(\lambda) \quad , \quad H = D^2 u(\lambda) \; .$$

To be clear, g is a vector with n components and H is a symmetric $n \times n$ matrix. Add code to `DataAnalysis.py` that evaluates u, g and H using these formulas. You can copy the code that evaluates $f(t)$ from `FakeData.py`.

Create a few (more than one but not a lot more than one) random directions v, which can be a vector with n independent standard normal components. For a step size s, take $\Delta\lambda = sv$. Compare the actual change in u to predicted changes using first or first and second derivatives

$$\begin{aligned}
\Delta u &= u(\lambda + \Delta\lambda) - u(\lambda) \\
\Delta u_1 &= g^T \Delta\lambda \\
\Delta u_2 &= g^T \Delta\lambda + \frac{1}{2}\Delta\lambda^T H \Delta\lambda \; .
\end{aligned}$$

Make a table of the absolute and relative errors as a function of s with fixed v to give computational evidence that the absolute error is order s^2 or s^3 and the relative error is order s or s^2, depending on whether you use second derivatives. For example, you can show that an error is order s^2, you divide the error by s^2, take a decreasing sequence of s values, and look for a limit. The smallest values of s in your sequence should be so small that roundoff spoils the results. Maybe decreasing s by a factor of ten each time will be good, or maybe a sequence that decreases by a smaller factor. Note, absolute error is

$$\Delta u_1 - \Delta u$$

(or Δu_2). The error may be negative. The relative error is

$$\frac{\Delta u_1 - \Delta u}{\Delta u} \ .$$

Use the integrated `numpy` routines for evaluating $g^T \Delta\lambda$ and $g^T H g$ (a combination of `@` and `np.dot`), not scalar loops.