Lab 14

Numerical Derivatives

Lab Objective: *Understand and implement finite difference approximations of the derivative in single and multiple dimensions. Evaluate the accuracy of these approximations.*

Derivative Approximations in One Dimension

The derivative of a function f at a point x_0 is

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}. \tag{14.1}$$

In this lab, we will investigate one way a computer can calculate $f'(x_0)$.

Forward Difference Quotient

Suppose that in Equation (14.1), instead of taking a limit, we just pick a small value for h. Then we would expect $f'(x_0)$ to be close to the quantity

$$\frac{f(x_0 + h) - f(x_0)}{h}. \tag{14.2}$$

This quotient is called the *first order forward difference approximation* of the derivative. Because $f'(x_0)$ is the limit of such quotients, we expect that when h is small, this quotient is close to $f'(x_0)$. We can use Taylor's formula to find just how close.

By Taylor's formula,

$$f(x_0 + h) = f(x_0) + f'(x_0)h + R_2(h),$$

where $R_2(h) = \left(\int_0^1 (1-t)f''(x_0 + th)dt\right) h^2$. (This is called the *integral form* of the remainder for Taylor's Theorem; see Volume 1 Chapter 6). When we solve this equation for $f'(x_0)$, we get

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{R_2(h)}{h}. \tag{14.3}$$

Thus, the error in using the first order forward difference quotient to approximate $f'(x_0)$ is

$$\left|\frac{R_2(h)}{h}\right| \leq |h| \int_0^1 |1-t||f''(x_0+th)|dt.$$

If we assume f'' is continuous, then for any δ, set $M = \sup_{x\in(x_0-\delta,x_0+\delta)} f''(x)$. Then if $|h| < \delta$, we have

$$\left|\frac{R_2(h)}{h}\right| \leq |h| \int_0^1 M dt = M|h| \in O(h).$$

Therefore, the error in using (14.2) to approximate $f'(x_0)$ grows like h.

Centered Difference Quotient

In fact, we can approximate $f'(x_0)$ to the second order with another difference quotient, called the *centered difference quotient*. We begin by trying to find the *backward difference quotient*. Evaluate Taylor's formula at $x_0 - h$ to derive

$$f'(x_0) = \frac{f(x_0) - f(x_0-h)}{h} + \frac{R_2(-h)}{h}. \tag{14.4}$$

The first term on the right hand side of (14.4) is called the *backward difference quotient*. This quotient also approximates $f'(x_0)$ to first order, so it is not the quotient we are looking for. When we add (14.3) and (14.4) and solve for $f'(x_0)$ (by dividing by 2), we get

$$f'(x_0) = \frac{\frac{1}{2}f(x_0+h) - \frac{1}{2}f(x_0-h)}{h} + \frac{R_2(-h) - R_2(h)}{2h} \tag{14.5}$$

The *centered difference quotient* is the first term of the right hand side of (14.5). Let us investigate the remainder term to see how accurate this approximation is. Recall from the proof of Taylor's theorem that $R_k = \frac{f^{(k)}(x_0)}{k!}h^k + R_{k+1}$. Therefore,

$$\begin{aligned}
\frac{R_2(-h) - R_2(h)}{2h} &= \frac{1}{2h}\left(\frac{f''(x_0)}{2}h^2 + R_3(-h) - \frac{f''(x_0)}{2}h^2 - R_3(h)\right) \\
&= \frac{1}{2h}(R_3(-h) - R_3(h)) \\
&= \frac{1}{2h}\left(\left(\int_0^1 \frac{(1-t)^2}{2}f'''(x_0+th)dt\right)h^3 - \left(\int_0^1 \frac{(1-t)^2}{2}f'''(x_0-th)dt\right)h^3\right) \\
&= \left(\int_0^1 \frac{(1-t)^2}{4}(f'''(x_0+th) - f'''(x_0-th))\right)h^2 \\
&\in O(h^2)
\end{aligned}$$

once we restrict h to some δ-neighborhood of 0. So the error in using the centered difference quotient to approximate $f'(x_0)$ grows like h^2, which is smaller than h when $|h| < 1$.

h	1e-1	1e-3	1e-5	1e-7	1e-9	1e-11
Error	5e-3	5e-7	6e-11	6e-11	7e-9	1e-5

Table 14.1: This table shows that it is best not to choose h too small when you approximate derivatives with difference quotients. Here, "Error" equals the absolute value of $f'(1) - f_{app}(1)$ where $f(x) = e^x$ and f_{app} is the centered difference approximation to f'.

Accuracy of Approximations

Let us discuss what step size h we should plug into the difference quotients to get the best approximation to $f'(x_0)$. Since f' is defined as a limit as $h \to 0$, you may think that it is best to choose h as small as possible, but this is not the case. In fact, dividing by very small numbers causes errors in floating point arithmetic. This means that as we decrease $|h|$, the error between $f'(x_0)$ and the difference quotient will first decrease, but then increase when $|h|$ gets too small because of floating point arithmetic.

Here is an example with the function $f(x) = e^x$. A quick way to write f as a function in Python is with the `lambda` keyword.

```
>>> import numpy as np
>>> from matplotlib import pyplot as plt
>>> f = lambda x: np.exp(x)
```

In general, the line `f = lambda <params> : <expression>` is equivalent to defining a function `f` that accepts the parameters `params` and returns `expression`.

Next we fix a step size `h` and define an approximation to the derivative of `f` using the *centered difference quotient*.

```
>>> h = 1e-1
>>> Df_app = lambda x: .5*(f(x+h)-f(x-h))/h
```

Finally, we check the accuracy of this approximation at $x_0 = 1$ by computing the difference between `Df_app(1)` and the actual derivative evaluated at 1.

```
# Since f(x) = e^x, the derivative of f(x) is f(x)
>>> np.abs( f(1)-Df_app(1) )
0.0045327354883726301
```

We note that our functions `f` and `Df_app` behave as expected when they are passed a NumPy array.

```
>>> h = np.array([1e-1, 1e-3, 1e-5, 1e-7, 1e-9, 1e-11])
>>> np.abs( f(1)-Df_app(1) )
array([  4.53273549e-03,   4.53046679e-07,   5.85869131e-11,
         5.85873572e-11,   6.60275079e-09,   1.04294937e-05])
```

These results are summarized in Table 14.1.

Thus, the optimal value of h is one that is small, but not too small. A good choice is `h = 1e-5`.

Problem 1. Write a function that accepts as input a callable function object `f`, an array of points `pts`, and a keyword argument `h` that defaults to `1e-5`. Return an array of the *centered difference quotients* of `f` at each point in `pts` with the specified value of `h`.

Problem 2. Write a function that accepts as input a callable function object `f`, the derivative `df` of the function `f`, an array of points `pts`, and a keyword argument `h` that defaults to `1e-5`. Return an array of the errors for the *centered difference quotients* at each point in `pts` with the specified value of `h`.

Problem 3. Use the *centered difference quotient* to approximate the derivative of $f(x) = (\sin(x) + 1)^x$ at $x = \frac{\pi}{3}, \frac{\pi}{4}$, and $\frac{\pi}{6}$. Calculate the error of the approximations.

You may wonder if the forward or backward difference quotients are ever used, since the centered difference quotient is a more accurate approximation of the derivative. In fact, there are some functions that in practice do not behave well under centered difference quotients. In these cases, one must use the forward or backward difference quotient.

Finally, we remark that forward, backward, and centered difference quotients can be used to approximate higher-order derivatives of f. However, taking derivatives is an *unstable* operation. This means that taking a derivative can amplify the arithmetic error in your computation. For this reason, difference quotients are not generally used to approximate derivatives higher than second order.

Problem 4. The radar stations A and B, separated by the distance a = 500 m, track the plane C by recording the angles α and β at one-second intervals (See figure 14.1). Three successive readings are give in table 14.2. Use centered difference quotients to calculate the speed v of the plane at t = 10 s. The coordinates of the plane can be shown to be

$$x = a\frac{\tan(\beta)}{\tan(\beta) - \tan(\alpha)} \tag{14.6}$$

$$y = a\frac{\tan(\beta)\tan(\alpha)}{\tan(\beta) - \tan(\alpha)} \tag{14.7}$$

(Kiusalaas, Jaan. Numerical Methods in Engineering with Python 3)

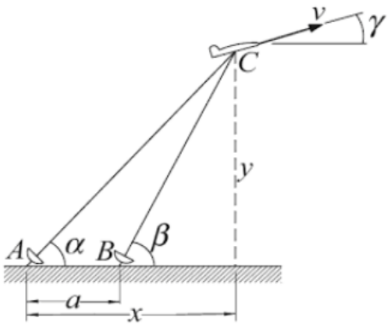

Figure 14.1: Radar stations in relation to plane

t (s)	9	10	11
α	54.80°	54.06°	53.34°
β	65.59°	64.59°	63.62°

Figure 14.2: Angles α and β at one-second intervals

Derivative Approximations in Multiple Dimensions

Finite difference methods can also be used to calculate derivatives in higher dimensions. Recall that the Jacobian of a function $f : \mathbb{R}^n \to \mathbb{R}^m$ at a point $x_0 \in \mathbb{R}^n$ is the $m \times n$ matrix $J = (J_{ij})$ defined component-wise by

$$J_{ij} = \frac{\partial f_i}{\partial x_j}(x_0).$$

For example, the Jacobian for a function $f : \mathbb{R}^3 \to \mathbb{R}^2$ is defined by

$$J = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \frac{\partial f}{\partial x_3} \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \end{pmatrix}.$$

The Jacobian is useful in many applications. For example, the Jacobian can be used to find zeros of functions in multiple variables.

The forward difference quotient for approximating a partial derivative is

$$\frac{\partial f}{\partial x_j}(x_0) \approx \frac{f(x_0 + he_j) - f(x_0)}{h},$$

where e_j is the j^{th} standard basis vector. Similarly, the centered difference approximation is

$$\frac{\partial f}{\partial x_j}(x_0) \approx \frac{\frac{1}{2}f(x_0 + he_j) - \frac{1}{2}f(x_0 - he_j)}{h}.$$

Problem 5. Write a function that accepts

1. a function handle `f`,
2. an integer `n` that is the dimension of the domain of `f`,
3. an integer `m` that is the dimension of the range of `f`,
4. an `1 x n`-dimensional NumPy array `pt` representing a point in $\mathbb{R}^n$, and
5. a keyword argument `h` that defaults to `1e-5`.

Return the approximate Jacobian matrix of `f` at `pt` using the centered difference quotient.

Problem 6.

Let $f : \mathbb{R}^2 \to \mathbb{R}^2$ be defined by

$$f(x, y) = \begin{bmatrix} e^x \sin(y) + y^3 \\ 3y - \cos(x) \end{bmatrix}$$

Find the error between your Jacobian function and the analytically computed derivative on the square $[-1, 1] \times [-1, 1]$ using ten thousand grid points (100 per side). You may apply your Jacobian function to the points one at a time using a double `for` loop. Once you get the error matrix for a given point, calculate the Frobenius norm of this matrix (`la.norm` defaults to the Frobenius norm). This norm will be your total error for that point. What is the maximum error of your Jacobian function over all points in the square?

Hint: The following code defines the function $f(x, y) = \begin{bmatrix} x^2 \\ x + y \end{bmatrix}$.

```
# f accepts a length-2 NumPy array
>>> f = lambda x: np.array([x[0]**2, x[0]+x[1]])
```