Chapter 3 Numerical Differentiation

3.1 NUMERICAL DIFFERENTIATION PROBLEM STATEMENT

A **numerical grid** can be defined as an evenly spaced set of points over the domain of a function(i.e., the independent variable), over some interval. The **spacing** or **step size** of a numerical grid is the distance between adjacent points on the grid. For the purpose of this text, if x is a numerical grid, then x_j is the j^{th} point in the numerical grid, and h is the spacing between x_{j-1} and x_j. Fig. 3.1 shows an example of a numerical grid.

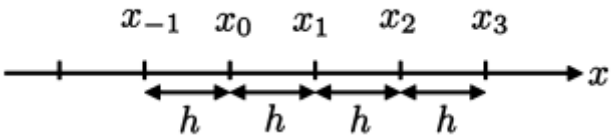


FIGURE 3.1

Numerical grid used to approximate functions.

There are several functions in Python that can be used to generate numerical grids. For numerical grids in one dimension, it is sufficient to use the `linspace` function, which you have already used for creating regularly spaced arrays.

In Python, a function $f(x)$ can be represented over an interval by computing its value on a grid. Although the function itself may be continuous, this **discrete** or **discretized** representation is useful for numerical calculations and corresponds to datasets that may be acquired in engineering and science practice. Specifically, the function value may only be known at discrete points. For example, a temperature sensor may deliver temperature versus time pairs at regular time intervals. Although temperature is a smooth function of time, the sensor only provides values at discrete time intervals; in this particular case, the underlying function would not even be known.

Whether f is an analytic function or a discrete representation of one, we would like to derive methods of approximating the derivative of f over a numerical grid and determine its accuracy

3.2 USING FINITE DIFFERENCE TO APPROXIMATE DERIVATIVES

The derivative $f(x)$ of a function $f(x)$ at the point $x = a$ is defined as.

$$f'(a) = \lim_{x \to a} \frac{f(x) - f(a)}{x - a}$$

The derivative at $x = a$ is the slope at this point. In "finite difference" approximations of this slope, one uses values of the function in a neighborhood of the point $x = a$ to achieve the goal. There are various finite difference formulas used in different applications, and three of these, where the derivative is calculated using the values at two points, are presented below.

The **forward difference** estimates the slope of the function at x_j using the line that connects

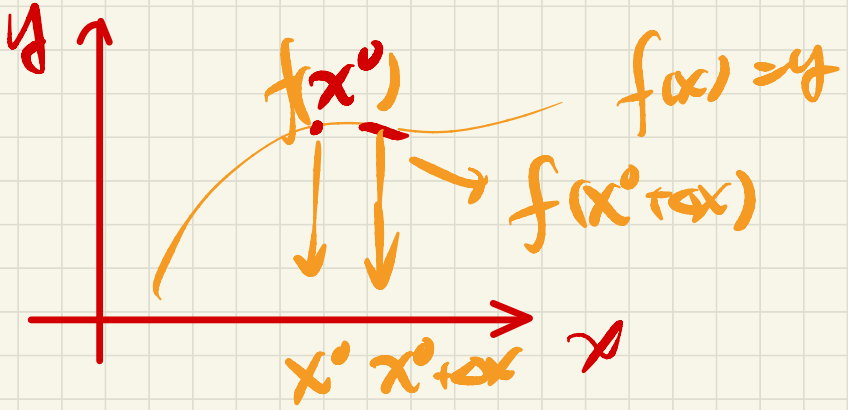
前项差分

4/30 Midterm

4/23 Mock

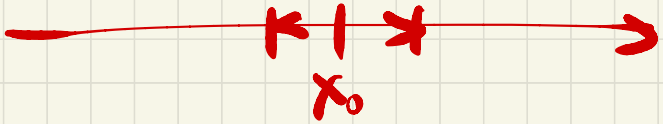
① open book

120



$$m = \frac{\Delta y}{\Delta x} = \frac{f(x^\circ+\Delta x) - f(x^\circ)}{\Delta x}$$

$$\Rightarrow \lim_{\Delta x \to 0} \frac{f(x^\circ+\Delta x) - f(x^\circ)}{\Delta x}$$


x_0

$\left(x_j, f(x_j)\right)$ and $\left(x_{j+1}, f(x_{j+1})\right)$:

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j}.$$

The **backward difference** estimates the slope of the function at x_j using the line that connects $\left(x_{j-1}, f(x_{j-1})\right)$ and $\left(x_j, f(x_j)\right)$:

$$f'(x_j) = \frac{f(x_j) - f(x_{j-1})}{x_j - x_{j-1}}.$$

The **central difference** estimates the slope of the function at x_j using the line that connects $\left(x_{j-1}, f(x_{j-1})\right)$ and $\left(x_{j+1}, f(x_{j+1})\right)$:

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_{j-1})}{x_{j+1} - x_{j-1}}.$$

Fig. 3.2 illustrates the three different formulas required to estimate the slope.

3.2.1 Using finite difference to approximate derivatives with Taylor series

To derive an approximation for the derivative of f, we return to the Taylor series. For an arbitrary function $f(x)$, the Taylor series of f around $a = x_j$ is

$$f(x) = \frac{f(x_j)(x-x_j)^0}{0!} + \frac{f'(x_j)(x-x_j)^1}{1!} + \frac{f''(x_j)(x-x_j)^2}{2!} + \frac{f'''(x_j)(x-x_j)^3}{3!} + \cdots.$$

If x is on a grid of points with spacing h, we can compute the Taylor series at $x = x_{j+1}$ to obtain

$$f(x_{j+1}) = \frac{f(x_j)(x_{j+1}-x_j)^0}{0!} + \frac{f'(x_j)(x_{j+1}-x_j)^1}{1!} + \frac{f''(x_j)(x_{j+1}-x_j)^2}{2!} + \frac{f'''(x_j)(x_{j+1}-x_j)^3}{3!} + \cdots.$$

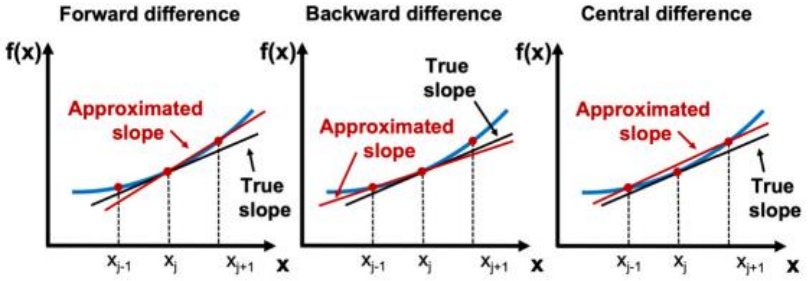


FIGURE 3.2

Finite difference approximation of the derivative.

Substituting $h = x_{j+1} - x_j$ and solving for $f'(x_j)$ gives the equation

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_j)}{h} + \left(-\frac{f''(x_j)h}{2!} - \frac{f'''(x_j)h^2}{3!} - \cdots\right).$$

Taylor series

$$f(x) = f(x) + f'(x)\cdot\Delta x + \frac{f''(x)}{2!}\Delta x^2 + \cdots + \frac{f^n(x)}{n!}\Delta x^n$$

Linear approach → high order term

Δx 趋向0 → 高次项可忽略

$$f(x) = f(x) + f'(x)\Delta x + O(\Delta x^2)$$

1. Δx 趋向0, $O(\Delta x^2)$ 趋向0 (二次收敛)

complexity

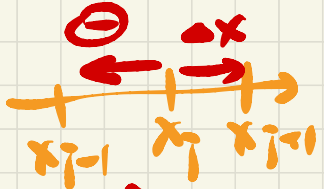
- $f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} + O(\Delta x)$

  前项插分、线性收敛

- $f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{\Delta x} + O(\Delta x)$

  后项插分 线性收敛

- 中央插分



$\Delta x > 0$

$$f(x_{i+1}) = f(x_i) + f'(x_i)\Delta x + \frac{1}{2}f''(x_i)\Delta x^2 + \frac{1}{3!}f'''(x_i)\Delta x^3$$

$$f(x_{i-1}) = f(x_i) + f'(x_i)\Delta x + \frac{1}{2}f''(x_i)\Delta x^2 + \frac{1}{3!}\cdots$$

$$f(x_{i+1}) - f(x_{i-1}) = 2f'(x_i)\Delta x + \frac{2}{3!}f'''(x)\Delta x^3 + \cdots$$

$$\Rightarrow f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} + O(\Delta x^2)$$

= 二次收敛

The terms that are in parentheses, $-\frac{f''(x_j)h}{2!}-\frac{f'''(x_j)h^2}{3!}-\cdots$, are **called higher order** terms of h. The higher order terms can be rewritten as

$$-\frac{f''(x_j)h}{2!}-\frac{f'''(x_j)h^2}{3!}-\cdots=h\big(\alpha+\epsilon(h)\big),$$

where α is some constant, and $\epsilon(h)$ is a function of h that goes to zero as h goes to zero. You can verify using algebra that this is true. We use the abbreviation "$O(h)$" for $h\big(\alpha+\epsilon(h)\big)$, and in general, we use the abbreviation "$O(h^p)$" to denote $h^p\big(\alpha+\epsilon(h)\big)$.

Substituting $O(h)$ into the previous equation gives

$$f'(x_j)=\frac{f(x_{j+1})-f(x_j)}{h}+O(h).$$

This gives the **forward difference** formula for approximating derivatives as

$$f'(x_j)\approx\frac{f(x_{j+1})-f(x_j)}{h},$$

and we denote this formula as $O(h)$.

Here, $O(h)$ describes the **accuracy** of the forward difference formula for approximating derivatives. For an approximation that is $O(h^p)$, we denote p as the **order** of the accuracy of the approximation. With few exceptions, higher-order accuracy is better than lower order. To illustrate this point, assume $q<p$, Then as the spacing, h > 0, goes to zero, h^p goes to zero faster than h^q. Therefore as h goes to zero, an approximation of a value that is $O(h^p)$ moves closer to the true value faster than one that is $O(h^q)$.

By computing the Taylor series around $a=x_j$ at $x=x_{j-1}$ and again solving for $f'(x_j)$, we obtain the **backward difference** formula

$$f'(x_j)\approx\frac{f(x_j)-f(x_{j-1})}{h},$$

which is also $O(h)$. Verify this result on your own.

Intuitively, the forward and backward difference formulas for the derivative at x_j are just the slopes between the point at x_j and the points x_{j+1} and x_{j-1}, respectively.

We can construct an improved approximation of the derivative by a clever manipulation of Taylor series terms taken at different points. To illustrate, we can compute the Taylor series around $a=x_j$ at both x_{j+1} and x_{j-1}. Written out, these equations are

$$f(x_{j+1})=f(x_j)+f'(x_j)h+\frac{1}{2}f''(x_j)h^2+\frac{1}{6}f'''(x_j)h^3+\cdots$$

and

$$f(x_{j-1})=f(x_j)-f'(x_j)h+\frac{1}{2}f''(x_j)h^2-\frac{1}{6}f'''(x_j)h^3+\cdots.$$

Subtracting the formulas above gives

$$f(x_{j+1})-f(x_{j-1})=2f'(x_j)h+\frac{2}{3}f'''(x_j)h^3+\cdots,$$

which, when solved for $f'(x_j)$, gives the **central difference** formula

$$f'(x_j) \approx \frac{f(x_{j+1}) - f(x_{j-1})}{2h}$$

Because of how we subtracted the two equations, the h terms canceled out; therefore, the central difference formula is $O(h^2)$, even though it requires the same amount of computational effort as the forward and backward difference formulas! Thus the central difference formula gets an extra order of accuracy for free. In general, formulas that utilize symmetric points around x_j, for example, x_{j-1} and x_{j+1}, have better accuracy than asymmetric ones, such as the forward and background difference formulas.

Fig. 3.3 shows the forward difference (line joining $(x_j,\ y_j)$ and $(x_{j+1},\ y_{j+1})$) backward difference (line joining $(x_j,\ y_j)$ and $(x_{j-1},\ y_{j-1})$), and central difference (line joining $(x_{j-1},\ y_{j-1})$ and $(x_{j+1},\ y_{j+1})$) approximation of the derivative of a function f. As can be seen, the difference in the value of the slope can be significantly different based on the size of the step h and the nature of the function.

Problem3.1 Take the Taylor series of f around $a = x_j$ and compute the series at $x = x_{j-2}, x_{j-1}, x_{j+1}, x_{j+2}$. Show that the resulting equations can be combined to form an approximation for $f'(x_j)$ which is $O(h^4)$.

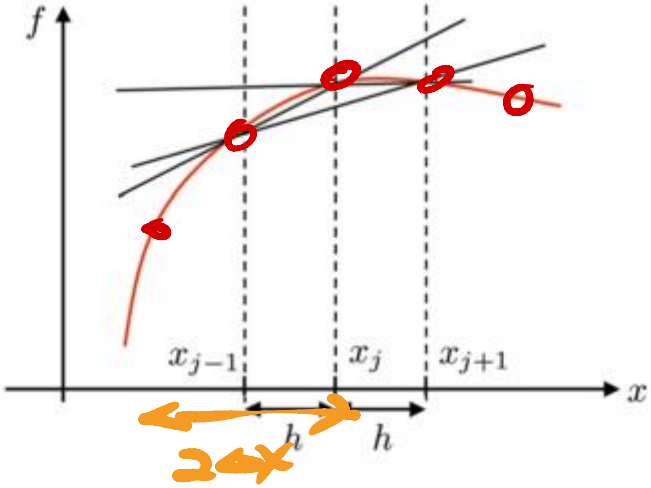


FIGURE 3.3

Illustration of the forward difference, the backward difference, and the central difference. Note the difference in slopes depending on the method used.

Ans:

② $f(x_{j-1}) = f(x_j) - \Delta x f'(x_j) + \frac{1}{2!} f''(x_j)(-\Delta x)^2 + \frac{1}{3!}(-\Delta x)^3 f'''(x_j) + \frac{1}{4!} f^{(4)}(x_j)(-\Delta x)^4$

③ $f(x_{j+1}) = f(x_j) + \Delta x f'(x_j) + \frac{1}{2!} f''(x_j)(\Delta x)^2 + \frac{1}{3!}(\Delta x)^3 f'''(x_j) + \frac{1}{4!} f^{(4)}(x_j)(\Delta x)^4$

④ $f(x_{j+2}) = f(x_j) + 2\Delta x f'(x_j) + \frac{1}{2!} f''(x_j)(2\Delta x)^2 + \frac{1}{3!}(2\Delta x)^3 f'''(x_j) + \frac{1}{4!} f^{(4)}(x_j)(2\Delta x)^4$

① − ② + ③ − ④ : $f(x_{j-2}) - 8f(x_{j-1}) + 8f(x_{j+1}) - f(x_{j+2}) = 12\Delta x + \frac{24}{120} f^{(5)}(x_j)(\Delta x)^5$

TIP! Python has a command that can be used to compute finite differences directly: for a vector f, the command `d=np.diff(f)` produces an array d in which the entries are the differences of the adjacent elements in the initial array f. In other words, $d(i) = f(i+1) - f(i)$.

WARNING! When using the command `np.diff`, the size of the output is one less than the size of the input since it needs two arguments to produce a difference.

$$f'(x) = \frac{f(x_{i-2}) - 8f(x_{i-1}) + 8f(x_{i+1}) - f(x_{i+2})}{12\Delta x} + O(\Delta x^4)$$

为.

Problem3.2 Consider the function $f(x) = cos(x)$. We know that the derivative of $cos(x)$ is $-sin(x)$. Although in practice we may not know the underlying function we are finding the derivative for, we use the simple example to illustrate the aforementioned numerical differentiation methods and their accuracy. The following code computes the derivatives numerically Python

Ans:

As the above figure shows, there is a small offset between the two curves, which results from the numerical error in the evaluation of the numerical derivatives. The maximal error between the two numerical results is of the order 0.05 and expected to decrease with the size of the step.

As illustrated in the previous example, the finite difference scheme contains a numerical error due to the approximation of the derivative. This difference decreases with the size of the discretization step, which is illustrated in the following example.

Problem3.3 The following code computes the numerical derivative of f (x) = cos(x) using the forward-difference formula for decreasing step size, h. It then plots the maximum error between the approximated derivative and the true derivative versus h as shown in the generated figure.

Ans:

3.3 APPROXIMATING OF HIGHER ORDER DERIVATIVES

It also possible to use the Taylor series to approximate higher-order derivatives (e.g.,$f''(x_j)$,$f'''(x_j)$, etc.). For example, taking the Taylor series around $a = x_j$ and then computing it at $x = x_{j-1}$ and x_{j+1} gives

$$f(x_{j-1}) = f(x_j) - hf'(x_j) + \frac{h^2 f''(x_j)}{2} - \frac{h^3 f'''(x_j)}{6} + \cdots$$

and

$$f(x_{j+1}) = f(x_j) + hf'(x_j) + \frac{h^2 f''(x_j)}{2} + \frac{h^3 f'''(x_j)}{6} + \cdots.$$

If we add these two equations together, we get

$$f(x_{j-1}) + f(x_{j+1}) = 2f(x_j) + h^2 f''(x_j) + \frac{h^4 f''''(x_j)}{24} + \cdots.$$

and, with some rearrangement, this gives the approximation

$$f''(x_j) \approx \frac{f(x_{j+1}) - 2f(x_j) + f(x_{j-1})}{h^2}$$

which is $O(h^2)$.

3.4 Numerical differentiation with noise

As stated earlier, sometimes *f* is given as a vector where *f* is the corresponding function value for independent data values in another vector *x*, which is gridded. Sometimes data can be contaminated with noise, meaning its value is off by a small amount from what it would be if it were computed from a pure mathematical function. This can often occur in engineering due to inaccuracies in measurement devices or the data itself can be slightly modified by perturbations outside the system

of interest. For example, you may be trying to listen to your friend talk in a crowded room. The signal *f* might be the intensity and tonal values in your friend's speech; however, because the room is crowded, noise from other conversations is heard along with your friend's speech, and he becomes difficult to understand.

To illustrate this point, we numerically compute the derivative of a simple cosine wave corrupted by a small sin wave. Consider the following two functions:

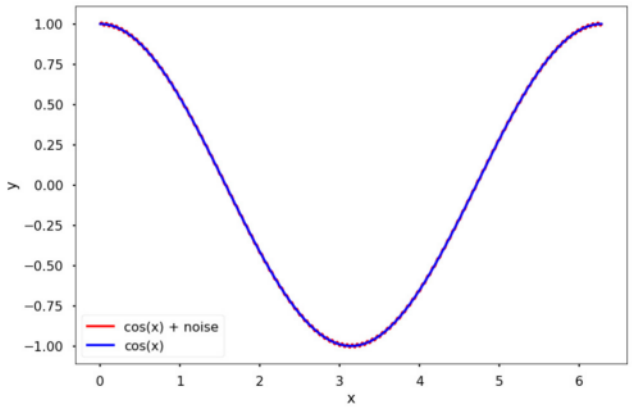
$$f(x) = \cos(x)$$

and

$$f_{\epsilon,\omega}(x) = \cos(x) + \epsilon \sin(\omega x)$$

Where $0 < \epsilon \ll 1$ is a very small number, and ω is a large number. When ϵ is small, it is clear that $f \cong f_{\epsilon,\omega}$. To illustrate this point, we plot $f_{\epsilon,\omega}(x)$ for $\epsilon = 0.01$ and ω = 100, and we can see it is very close to $f(x)$, as shown in the following figure.

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        plt.style.use("seaborn-poster")
        %matplotlib inline
In [2]: x = np.arange(0, 2*np.pi, 0.01)
        # compute function
        omega = 100
        epsilon = 0.01
        y = np.cos(x)
        y_noise = y + epsilon*np.sin(omega*x)
        # Plot solution
        plt.figure(figsize = (12, 8))
        plt.plot(x, y_noise, "r-", label = "cos(x) + noise")
        plt.plot(x, y, "b-", label = "cos(x)")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.legend()
        plt.show()
```



The derivatives of our two test functions are

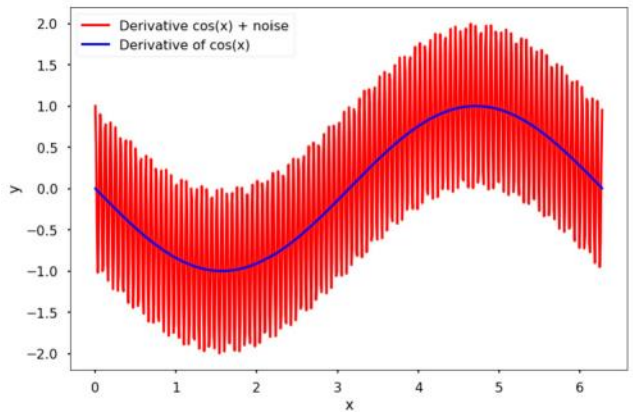
$$f'(x) = -\sin(x)$$

and

$$f'_{\epsilon,\omega}(x) = -\sin(x) + \epsilon\omega\cos(\omega x).$$

Since $\epsilon\omega$ may not be small when ω is large, the contribution of the noise to the derivative may not be small. As a result, the derivative (analytic and numerical) may not be usable. For instance, the following figure shows $f'(x)$ and $f'_{\epsilon,\omega}(x)$ for $\epsilon = 0.01$ and $\omega = 100$.

```
In [3]: x = np.arange(0, 2*np.pi, 0.01)
        # compute function
        y = -np.sin(x)
        y_noise = y + epsilon*omega*np.cos(omega*x)
        # Plot solution
        plt.figure(figsize = (12, 8))
        plt.plot(x, y_noise, "r-", label = "Derivative cos(x) + noise")
        plt.plot(x, y, "b-", label = "Derivative of cos(x)")
        plt.xlabel("x")
        plt.ylabel("y")
        plt.legend()
        plt.show()
```



3.5 Summary

1. Because explicit derivation of functions is sometimes cumbersome for engineering applications, numerical approaches are preferable.
2. Numerical approximation of derivatives can be done using a grid on which the derivative is approximated by finite differences.
3. Finite differences approximate the derivative by ratios of differences of the function values over small intervals.
4. Finite difference schemes have different approximation orders depending on the method used.
5. When the data is noisy, there are issues using finite differences for approximating derivatives.