CHAPTER 2

NUMERICAL DERIVATIVES

When we study physics, we often investigate the change of certain quantities. That is why basic calculus plays a major role in physics. Evaluating the derivative of a function $f(x)$ is an ubiquitous operation in physics. However, an analytical expression of the derivative is not always available. Then, a numerical method must be deployed to evaluate it. That is not only the reason we need numerical derivative. In some cases an analytical expression of the function itself is not available. For example, when a function is experimentally obtained by measurement, it is given as a set of numerical data, (x_i, f_i), $i = 1, \cdots, N$. Then, analysis of such data is always numerical. There are many algorithms for numerical derivative. In this chapter, we study some of well-known algorithms useful for studying physics.

Numerical methods are in general not exact and involve systematic errors. Understanding the source of errors and their magnitude is very important. It is possible to estimate the magnitude of expected errors based on mathematical analysis and the study of numerical error is an important part of numerical analysis. We will investigate the degrees of errors through theory and examples.

In this section, we assume that an analytical expression of function $f(x)$ is given and that we can numerically evaluate it at any point x. Furthermore, we assume that the numerical error in the evaluation of the function is negligiblly small. In other words, the main errors occur when the derivative is evaluated.

2.1 First order derivatives

Let us look at the mathematical definition of derivative:

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) = \lim_{h \searrow 0} \frac{f(x+h) - f(x)}{h} = \lim_{h \searrow 0} \frac{f(x) - f(x-h)}{h} \tag{2.1}$$
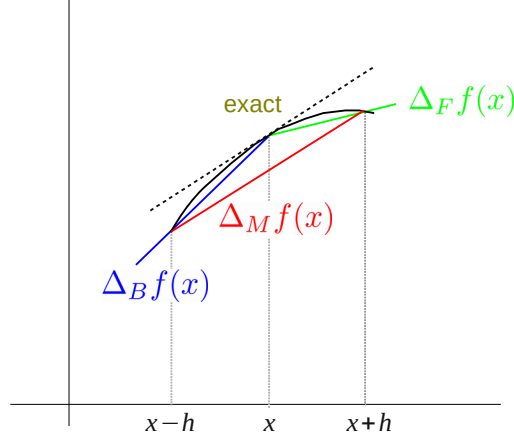
Figure 2.1: Illustration of various numerical derivatives. The exact derivative is the slope of the curve at x, which is shown as the dotted line. The forward finite difference method shown in green underestimates the slop whereas the backward finite difference method shwon in blue overestimates it. The mean finite different method shown in red looks very close to the exact derivative.

which involves limit operation which floating point calculation cannot not perform due to the quantization (See Chap. 1). If $h = 0$ is used, we have `0/0=NaN`. If h=`realmin` is used, $f(x+h) - f(x)$ is very inaccurate due to round-off error as discussed in Chapter 1. If $1 \gg h >$ `realmin` is used, the both numerator and denominator are finite but the ratio is not exactly the limit.

It is clear that direct numerical evaluation of Eq (1.1) is not possible. However, it is expected to be close to the limit if an *appropriately small* value of h is used. Based on this naive argument, we hope that the following *forward finite difference* method is close to the actual derivative with a certain small value of h.

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) \approx \Delta_\mathrm{F} f(x) \equiv \boxed{\frac{f(x+h) - f(x)}{h}}. \tag{2.2}$$

where Δ_F is the forward finite difference operator. Similarly, we define the *backward finite difference* method

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) \approx \Delta_\mathrm{B} f(x) \equiv \boxed{\frac{f(x) - f(x-h)}{h}}. \tag{2.3}$$

where Δ_B is the backward finite difference operator. Unlike the exact limit (1.1) the forward and backward finite difference methods do not agree each other due to the finite h. As illustrated in Fig. 1.1, one of them overestimates and the other underestimates.

Now, we investigate how accurate the forward and backward finite difference methods are. We estimate the order of error using Taylor expansion[1],

$$f(x + h) = f(x) + h\, f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{3!} f^{(3)}(x) + \mathcal{O}(h^4) \tag{2.4a}$$

$$f(x - h) = f(x) - h\, f'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{3!} f^{(3)}(x) + \mathcal{O}(h^4) \tag{2.4b}$$

where $\mathcal{O}(h^4)$ mean that the remaining terms with h^4 and higher order.

Substituting the expansions (1.4) to Eqs. (1.2) and (1.3)), we find

$$\Delta_{\mathrm{F}} f(x) = f'(x) + \frac{h}{2} f''(x) + \mathcal{O}(h^2) \tag{2.5a}$$

$$\Delta_{\mathrm{B}} f(x) = f'(x) - \frac{h}{2} f''(x) + \mathcal{O}(h^2) \tag{2.5b}$$

where f' and f'' indicate the first and second order derivatives of f. The leading term in the error $\Delta_{\mathrm{F,B}} f(x) - f'(x)$ is $\pm \frac{h}{2} f''(x)$, which is the order of h. This means that if h is small enough, the numerical methods agree with the exact derivative. However, the error decreases only linearly with h. We need to remember that if h is too small, the round-off error kills the accuracy. Hence, h cannot be too small. Error at the order h is in general not acceptable and these methods are not accurate enough for practical applications.

There is a better method. Noting that the error in the forward and backward finite difference method is exactly the same in magnitude but has opposite sign, we can get rid of the error at the order of h. Taking the mean of the two approximations, we obtain

$$\Delta_{\mathrm{M}} f(x) \equiv \frac{\Delta_{\mathrm{F}} f(x) + \Delta_{\mathrm{B}} f(x)}{2} = \boxed{\frac{f(x+h) - f(x-h)}{2h}}. \tag{2.6}$$

where Δ_{M} is a *mean finite difference* operator.

Substituting the expansions (1.4) to (1.6), we find

$$\Delta_{\mathrm{M}} f(x) = f'(x) + \frac{h^2}{3!} f^{(3)}(x) + \mathcal{O}(h^4) \tag{2.7}$$

Now, the error is at the order of h^2 which is smaller than the previous error at h. The improvement is clearly visible in Fig. 1.1.

Mathematically speaking, the small error in the mean finite difference method is related to the mean value theorem[2] which states that there is at least one number c between a and b such that

$$f'(c) = \frac{f(b) - f(a)}{b - a} \tag{2.8}$$

Let $b = x + h$ and $a = x - h$, the right hand side of Eq (1.8) mathces to $\Delta_{\mathbf{M}} f(x)$. Therefore, the mean finite different method gives the slope of the curve at some point c between $x + h$ and $x - h$. When h is small, c is apprximately x. Then, we obtain the mean finite difference formula (1.6).

The mean finite element method (1.6) is good enough for most application. If even a higher degree of accuracy is needed, use the symmetric four-point method[3]

$$\Delta_{S4} f(x) \equiv \boxed{\frac{f(x+2h) - 8f(x+h) + 8f(x-h) - f(x-2h)}{12h}}. \tag{2.9}$$

Its error is at the order of h^4.

Exercise 2.1 Using the Taylor expansion, verify that the error of symmetric four-point method is order of h^4.

How do we find an appropriate value of h? Considering the definition of derivative (1.1), one might expect that smaller h provides more accurate result. However, when h is too small, the finite different methods suffer from the round-off error. Therefore, we must choose the value of h carefully, not too big and not too small. Noting that $x + h = x(1 + x/h)$, it is pointless to use $h < \epsilon_{\mathrm{m}} x$ where ϵ_{m} is the machine epsilon. (See Section 1.6). Example 1.1 illustrates that the finite difference method fails when h is too small.
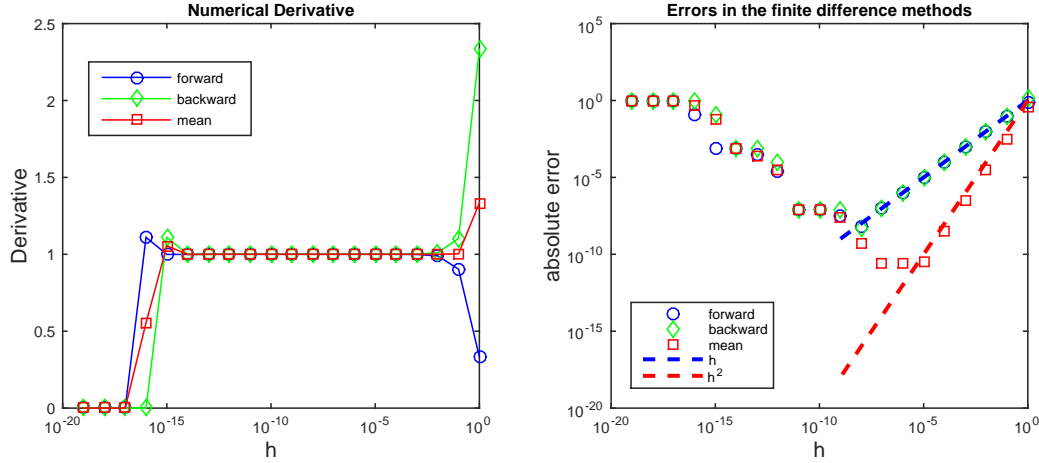
Figure 2.2: Output of Example 1.1. The left panel shows numerical derivatives for wide ranges of h. As h decreases from $h = 1$ to $h = 0.01$, the derivative converges to 1 (at least in our eyes). As h further decreases, the values of all methods remain the same until $h \approx \epsilon_{\mathrm{m}}$. Below it, the derivative abruptly goes to 0. The numerical method fails due to round-off error. To see more details, the right panel plots the error. As h decreases, the error of the forward and backward finite difference methods decreases in the same way as h until $h \approx 10^{-8}$ but the error increases when h is further reduced. The mean finite difference method shows smaller error than the two other methods and the error decreases as h^2 up to $h \sim 10^{-5}$. The best result is given by the mean finite difference method with $h \approx 10^{-5}$.

■ EXAMPLE 2.1 Round-off errors in finite difference methods

Let's numerically evaluate the derivative of $f(x) = \dfrac{1}{3}x^3$ at $x = 1$ and compare the results with the exact value. The analytic form of derivative is $f'(x) = x^2$ and thus the exact value is $f'(1) = 1$. Program 1.1 computes the derivative using the forward, backward and mean finite difference methods for $h = 1, 0.1, 0.01, \cdots, 10^{-19}$. The error is measured by $\delta = |f'(1) - \Delta f(1)|$. The results are plotted in Fig. 1.2. The error of forward and backward methods is almost the same and decreases in the same way as h. However, the improvement stop at $h \sim 10^{-8}$ and the error increases for smaller h due to the round-off error. The best answer is obtained with $h = 10^{-8}$. The mean difference method shows much better result. The error decreases with h^2. The best value is obtained at $h = 10^{-5}$ and the result is better than the best value obtained by the forward or backward method. It is quite clear that the mean difference method is much better than the two others.

In practice, we don't know the actual error and therefore we cannot use a plot like the right panel of Fig. 1.2 to find an appropriate value for h. However, the left panel of Fig. 1.2 shows that the numerical derivative approaches a certain value and the output does not change much when h is smaller than a certain value until it hits the limit of round-off error. Suppose that we calculate the derivative using h and $h' = h/2$. We expect that the output is more accurate with h' than h. Using Eq. (1.7), the change of the mean finite

difference is given by

$$|\Delta_{\mathrm{M}}f(x) - \Delta'_{\mathrm{M}}f(x)| = \frac{|f^{(3)}|}{3!}\frac{3}{4}h^2 = \frac{3}{4} \times [\text{error in } \Delta_{\mathrm{M}}f(x)] \tag{2.10}$$

which suggests that the error is estimated by

$$\text{Error in } \Delta_M f(x) \approx |\Delta_{\mathrm{M}}f(x) - \Delta'_{\mathrm{M}}f(x)| \tag{2.11}$$

. Note that the exact value of $f'(x)$ is not needed to find the error. Now we have an algorithm to find numerical derivative with a desired accuracy using this error estimate.

Algorithm 2.1 Numerical derivative with tolerance

1. Set a value of *tolerance* (allowed error).

2. Set a reference step size h_1 and evaluate a reference derivative $g_1 = \Delta_{\mathrm{M}}f(x)$ using h_1.

3. Set a new step size $h_2 = h_1/2$ and evaluate a new derivative $g_2 = \Delta_{\mathrm{M}}f(x)$ using h_2.

4. Evaluate error $\delta = |g_2 - g_1|$.

5. If $\delta <$ tolerance, g_2 is the desired result. Stop the loop.

6. If not, let $g_1 = g_2$ and $h_1 = h_2$ (previous g_2 and h_2 are now the reference).

7. Go back to Step 3.

Here the absolute error is used. We can also use a relative error $\left|\dfrac{g_2 - g_1}{g_1}\right|$ instead. Then, the tolerance specifies a desired relative error.

EXAMPLE 2.2   Automatic adjustment of the step size h.

We numerically evaluate the derivative of $f(x) = \dfrac{1}{3}x^3$ at $x = 1$ again. This time, we do not specify the step length h. Instead we specify a tolerance and the program will automatically find an appropriate h for the given tolerance. Program 1.2 implements Algorithm 1.1. When the tolerance is 0.001, we obtain the following output. It appears that $h = 0.03125$ is good enough for this problem.

```
Enter tolerance =0.001
    h         derivative              error
 5.000e-01 1.0833333333e+00 2.5000000000e-01
 2.500e-01 1.0208333333e+00 6.2500000000e-02
 1.250e-01 1.0052083333e+00 1.5625000000e-02
 6.250e-02 1.0013020833e+00 3.9062500000e-03
 3.125e-02 1.0003255208e+00 9.7656250000e-04
 Tolerance is OK.
```

Exercise 2.2 Evaluate of the derivative of $\sin(x)$ at $x = \pi/4$ using the mean finite difference method. The first three digits of the answer should be correct.

2.2 Second order derivatives

We can evaluated the second order derivative using the mean value method twice. First, we pretend that the first order derivative is given. Using the mean value method with step size $h/2$, we obtain

$$f''(x) \approx \Delta_M f'(x) = \frac{f'(x + \frac{h}{2}) - f'(x - \frac{h}{2})}{h} \tag{2.12}$$

Then, we replace the first order derivatives with approximated ones using the mean value method again.

$$f'(x + \frac{h}{2}) \to \Delta_M f(x + \frac{h}{2}) = \frac{f(x + h) - f(x)}{h} \tag{2.13a}$$

$$f'(x - \frac{h}{2}) \to \Delta_M f(x - \frac{h}{2}) = \frac{f(x) - f(x - h)}{h}. \tag{2.13b}$$

The result is

$$f''(x) \approx \Delta_M^{(2)} f(x) \equiv \boxed{\frac{f(x + h) + f(x - h) - 2f(x)}{h^2}}. \tag{2.14}$$

where $\Delta_M^{(2)}$ is the second order mean finite difference operator. Substituting Eqs. (1.4a) and (1.4b) into (1.14), we find that the error of this approximation is order of h^2.

Exercise 2.3 The second order derivative of $\sin(x)$ is $-\sin(x)$. Evaluate the second order derivative of $\sin(x)$ at $x = 2n\pi/N$ where $n = 0, 1, \cdots, N$. Use $N = 100$ and plot the result. Verify that it is $-\sin(x)$ at least within the accuracy of the numerical method.

If a higher accuracy is needed, use the symmetric five-point method

$$f''(x) \approx \Delta_{S5}^{(2)} f(x) \equiv \boxed{\frac{-f(x + 2h) + 16f(x + h) - 30f(x) + 16f(x - h) - f(x - 2h)}{12h^2}} \tag{2.15}$$

whose error is the order of h^4.[4]

Exercise 2.4 Using the Taylor expansion, verify that the error of symmetric five-point method is order of h^4.

Problems

2.1 Evaluate the first order derivative of $\sin(x)$ for interval $(0, 2\pi)$ with tolerance 0.0001. What value of h is needed to satisfy the tolerance? Compare the numerical derivative with the exact one $\cos(x)$ and check if the tolerance is indeed satisfied.

2.2 Evaluate the second order derivative of $f(x) = \dfrac{1}{12}x^4$ at $x = 1$ using $h = 1, 0.1, 0.01, \cdots, 10^{-10}$. Plot the error as a function of h. Plot also the theoretical error h^2 for the three-point method (1.14). [Optional: Try also the five-point method (1.15) and check how the error increases. Unexpectedly, the larger h gives better answer. The symmetric five-point formula takes into account up to the fourth order derivative. The fifth and higher order derivative of x^4 vanishes and thus the five-point formula is theoretically exact for x^4. Nevertheless, you encounter numerical errors due to the loss of significance. Therefore, the larger h is better for this particular function. Try the same calculation with $f(x) = \dfrac{1}{20}x^5$. You will see the usual error profile.]

MATLAB Source Codes

Program 2.1

```
%******************************************************************************
%*     Example  2.1                                                          *
%*     filename: ch02pr01.m                                                  *
%*     program listing number: 2.1                                          *
%*                                                                           *
%*  This program evaluates the derivative of a given function func(x)        *
%*  at x=1 using the three finite difference methods.                        *
%*  Errors in forward, backward and mean value methods are plotted.          *
%*                                                                           *
%*     Programed by Ryoichi Kawai for Computational Physics Course.          *
%*     Last modification:  12/25/2013.                                       *
%******************************************************************************
clear all;

% define the function
func = @(x) x^3/3;

x=1;  % the point at which derivative is evaluated.
imax=20; % number of different displacements

% title and headers
display('Absolute Errors')
fprintf('%4s %16s %18s %20s\n','h','forward','backward','mean value')

for i=1:imax

    % Small displacement
    h(i)=10^(-i+1);

    % Evaluation of numerical derivative
    d_f(i)=(func(x+h(i))-func(x))/h(i); % Forward diffrence
    d_b(i)=(func(x)-func(x-h(i)))/h(i); % Backward diffrence
```

```
    d_m(i)=(func(x+h(i))-func(x-h(i)))/(2*h(i)); % Mean value

    % Errors
    err_f(i)=abs(1-d_f(i));
    err_b(i)=abs(1-d_b(i));
    err_m(i)=abs(1-d_m(i));

    % Display the errors
    fprintf('%6.1e %18.10e %18.10e %18.10e\n',...
            h(i),err_f(i),err_b(i),err_m(i));
end

hh = h.*h; % h^2 (the error of the mean value formula is order of h^2)

% Plot data
subplot(1,2,1)  % left panel
q=semilogx(h(1:imax),d_b(1:imax),'-o',...
    h(1:imax),d_f(1:imax),'-d',...
    h(1:imax),d_m(1:imax),'-s');
title('Numerical Derivative');
xlabel('h','fontsize',14);
ylabel('Derivative','fontsize',14);
set(q(1),'Color','blue');
set(q(2),'Color','green');
set(q(3),'Color','red');
legend(q,{'forward','backward','mean'});
legend(q,'Location','NorthWest');

subplot(1,2,2) % right panel
p=loglog(h(1:imax),err_b(1:imax),'o',...
    h(1:imax),err_f(1:imax),'d',...
    h(1:imax),err_m(1:imax),'s',...
    h(1:imax/2),h(1:imax/2),'--',h(1:imax/2),hh(1:imax/2),'--');
title('Errors in the finite difference methods');
xlabel('h','fontsize',14);
ylabel('absolute error','fontsize',14);
set(p(1),'Color','blue');
set(p(2),'Color','green');
set(p(3),'Color','red');
set(p(4),'Color','blue','LineWidth',2);
set(p(5),'Color','red','LineWidth',2);
legend(p,{'forward','backward','mean','h','h^2'});
legend(p,'Location','SouthWest');
```

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━▲▲▲━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━

Program 2.2

```
%*****************************************************************************
%*  Example  2.2                                                            *
%*  filename: ch02pr02.m                                                    *
%*  program listing number: 2.2                                            *
%*                                                                          *
%*  This program evaluates the derivative of a given function func(x)       *
%*  at x=1 using the mean finite difference method with the accuracy        *
%*  specified by tolerance.                                                 *
%*                                                                          *
%*     Programed by Ryoichi Kawai for Computational Physics Course.         *
%*     Last modification:  01/01/2017.                                      *
%*****************************************************************************
clear all;
```

```
% define the function
func = @(x) x^3/3;

% Read tolerance from keyboard.
tol=input('Enter tolerance =');  %

x=1;  % point where derivative is evaluated
h=1; % initial interval
diff_old=(func(x+h)-func(x-h))/(2*h); % initial numerical derivative

% any value bigger than tol is OK here.
delta = tol+1;

fprintf('%5s %18s %14s\n','h','derivative','error')

% Repeat until error is smaller than tolerance.
while delta>tol
    h=h/2;
    diff_new=(func(x+h)-func(x-h))/(2*h);
    delta=abs(diff_new-diff_old);
    fprintf('%10.3e %16.10e %16.10e\n',h,diff_new,delta)
    diff_old=diff_new;
end
fprintf('Tolerance is OK.\n')
```

▲▲▲

Python Source Codes

Program 2.1

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
%***************************************************************************
%*  Example  2.1                                                          *
%*  filename: ch02pr01.py                                                 *
%*  program listing number: 2.1                                          *
%*                                                                        *
%*  This program evaluates the derivative of a given function func(x)     *
%*  at x=1 using the three finite difference methods.                    *
%*  Errors in forward, backward and mean value methods are plotted.      *
%*                                                                        *
%*     Programed by Ryoichi Kawai for Computational Physics Course.      *
%*     Last modification:  01/01/2017.                                   *
%***************************************************************************
"""
import numpy as np
import matplotlib.pyplot as plt

def func(x):
    return x**3/3

def main():
```

```
    x=1.0
    imax=20
    h=np.zeros(imax)
    d_f=np.zeros(imax)
    d_b=np.zeros(imax)
    d_m=np.zeros(imax)
    err_f=np.zeros(imax)
    err_b=np.zeros(imax)
    err_m=np.zeros(imax)
    i=0
    print("{0:^62}".format('Absolute Errors'))
    print("{0:^6} {1:^18} {2:^18} {3:^20}"
          .format('h','forward','backward','mean value'))
    while(i<imax):
    # Small displacement
        h[i]=10**(-i)

    # Evaluation of numerical derivative
        d_f[i]=(func(x+h[i])-func(x))/h[i] # Forward diffrence
        d_b[i]=(func(x)-func(x-h[i]))/h[i] # Backward diffrence
        d_m[i]=(func(x+h[i])-func(x-h[i]))/(2*h[i]) # Mean value

    # Errors
        err_f[i]=abs(1.-d_f[i])
        err_b[i]=abs(1.-d_b[i])
        err_m[i]=abs(1.-d_m[i])
        print("{0:6.1e} {1:18.10e} {2:18.10e} {3:18.10e}"
              .format(h[i],err_f[i],err_b[i],err_m[i]))
        i=i+1

    # Plot data
    plt.ioff()
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    plt.semilogx(h,d_f, '--ob', label='forward')
    plt.semilogx(h,d_b, '--dg', label='backword')
    plt.semilogx(h,d_m, '--sr', label='mean')
    plt.legend(loc=2)
    plt.xlabel('h')
    plt.ylabel('Derivative')

    plt.subplot(1,2,2)
    plt.loglog(h,err_f, '--ob', label='forward')
    plt.loglog(h,err_b, '--dg', label='backword')
    plt.loglog(h,err_m, '--sr', label='mean')
    plt.legend(loc=3)
    plt.xlabel('h')
    plt.ylabel('Absolute error')
    plt.show()

if __name__ == "__main__":
    main()
```

▲▲▲

```
    #!/usr/bin/env python3
    # -*- coding: utf-8 -*-
    """
    %**************************************************************************
    %*   Example  2.2                                                        *
    %*   filename: ch02pr02.py                                               *
    %*   program listing number: 2.2                                         *
```

```
%*                                                                        *
%*   This program evaluates the derivative of a given function func(x)     *
%*   at x=1 using the mean finite difference method with the accuracy      *
%*   specified by tolerance.                                               *
%*                                                                        *
%*      Programed by Ryoichi Kawai for Computational Physics Course.       *
%*      Last modification:  01/01/2017.                                    *
%*************************************************************************
"""

import numpy as np
import matplotlib.pyplot as plt

def func(x):  # define a function
    return x**3/3

def main():
    tol=input("Enter tolerance =")  # Read a tolerabce from the console
    tol=np.float(tol)
    x=1.0
    h=1.0    # initial interval
    diff_old=(func(x+h)-func(x-h))/(2.0*h)  #  derivative first try
    delta=np.finfo(float).max  # any value bigger than tol is OK.

    print("{0:^10} {1:^16} {2:^16}"
          .format('h','derivative','error'))
    while (delta>tol):
        h=h/2.0
        diff_new=(func(x+h)-func(x-h))/(2.0*h)  # improved derivative
        delta=np.abs(diff_new-diff_old)
        print("{0:10.3e} {1:16.10e} {2:16.10e}"
              .format(h,diff_new,delta))
        diff_old=diff_new

    print("Tolerance is OK.")

if __name__ == "__main__":
    main()
```

Bibliography

[1] Mary L. Boas. *Mathematical Methods in the Physical Sciences.* Wiley, 3rd edition, 2006.

[2] James Stewart. *Essential Calculus.* Cengage Learning, 2nd edition, 2012. Section 3.2.

[3] Daniel Zwillinger. *CRC Stanbdard Mathematical Tables and Formula.* CRC Press, 35th edition, 2012.

[4] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Dover Publications, 1970. Table 25.2.