Chapter 1

Numerical derivatives

The derivative of a function is a slope of a tangent line to the plot of the function at a given input value (see for example fig. 1.1a). There is often a need to calculate the derivative of a function, such as when using the Newton-Raphson root finding algorithm from section 8.7. The function could be quite complex and spending effort to derive the analytical expression for the derivative yourself may not be ideal. Alternatively, the function implementation might not even be available to us. For these cases, we resort to a numerical estimate of the function derivative.

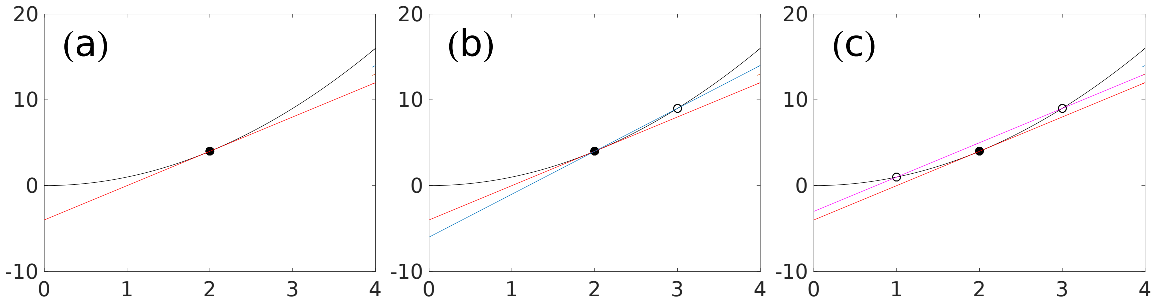

Figure 1.1: (a) The plot of the $f(x) = x^2$ function and its tangent line at the point $(x = 1, y = 1)$. The comparison between the tangent line calculated analytically, as it shown in (a), and by the difference methods: the forward difference (b) and the central difference methods (c). The step h is equal to 1.

1.1 Estimate of the derivative via the forward difference

We might look at the mathematical definition of the derivative

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{1.1}$$

and implement the numerical estimate ($f'_c(x)$) of the derivative via the *forward difference*

$$f'_c(x) = \frac{f(x+h) - f(x)}{h} \tag{1.2}$$

which essentially approximates the derivative with the finite step h (see fig. 1.1b). The MATLAB implementation is shown below

Listing 1.1: `forward_derivative.m` (available at `http://physics.wm.edu/programming_with_MATLAB_book/ch_derivatives/code/forward_derivative.m`)

```
function dfdx = forward_derivative( f, x, h )
% Returns derivative of the function f at position x
% f is the handle to a function
% h is step, keep it small
dfdx = (f(x+h) - f(x))/h;
end
```

Let's check this implementation with $f(x) = x^2$. The derivative of this is $f'(x) = 2x$. So we expect to see $f'(1) = 2$. At first, we calculate the derivative at `x=1` with `h=1e-5`

```
>> f = @(x) x.^2';
>> forward_derivative(f, 1, 1e-5)
ans =
    2.0000
```

It is very tempting to make `h` as small as possible to mimic the limit in the mathematical definition. We decrease `h`

```
>> forward_derivative(f, 1, 1e-11)
ans =
    2.0000
```

So far so good, the result is still correct. But if we decrease `h` further, we get

```
>> forward_derivative(f, 1, 1e-14)
ans =
    1.9984
```

which is imprecise. Somewhat surprisingly, if we make `h` even smaller, we get

```
>> forward_derivative(f, 1, 1e-15)
ans =
    2.2204
```

This deviates from the correct result even further. This is due to round-off errors (see section 1.5) and they will get worse as `h` gets smaller.

1.2 Algorithmic error estimate for numerical derivative

Unfortunately, there is more to worry about than just round-off errors. Let's assume for a second that the round-off errors are not a concern and a computer can do the calculation precisely. We would like to know what kind of error we are facing by using eq. (1.2). Recall that according to the Taylor series

$$f(x+h) = f(x) + \frac{f'(x)}{1!}h + \frac{f''(x)}{2!}h^2 + \cdots \tag{1.3}$$

So we can see that the computed approximation of derivative (f'_c) calculated via the forward difference approximates the true derivative f' as

$$f'_c(x) = \frac{f(x+h) - f(x)}{h} = f'(x) + \frac{f''(x)}{2}h + \cdots$$

From the above equation, we can see to the first order of h

Algorithm error for the forward difference

$$\varepsilon_{fd} \approx \frac{f''(x)}{2}h \tag{1.4}$$

This is quite bad since the error is proportional to h.

Example

Let's consider the function

$$\begin{aligned} f(x) &= a + bx^2 \\ f(x+h) &= a + b(x+h)^2 = a + bx^2 + 2bxh + bh^2 \\ f'_c(x) &= \frac{f(x+h) - f(x)}{h} \approx 2bx + bh \end{aligned}$$

For small $x < b/2$, the algorithm error dominates the derivative approximation.

For a given function and a point of interest, there is an optimal value of h, where both the round-off and the algorithm errors are small and about the same.

1.3 Estimate of the derivative via the central difference

Let's now estimate the derivative via the average of the forward and backward difference.

$$f'_c(x) = \frac{1}{2}\left(\frac{f(x+h) - f(x)}{h} + \frac{f(x) - f(x-h)}{h}\right) \tag{1.5}$$

We can see that the *backward difference* (the second term above) is identical to the forward formula once we plug $-h$ to eq. (1.2) and flip the overall sign. With trivial simplification we get the *central difference* expression

The central difference estimate of the derivative

$$f'_c(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{1.6}$$

We probably would not expect any improvement, since we are combining two methods which both have algorithmic errors proportional to h. However, the errors come with different signs, and thus cancel each other. We need to follow the Taylor series up to the term proportional to the f''' to calculate the algorithmic error.

Algorithm error of the central difference derivative estimate

$$\varepsilon_{cd} \approx \frac{f'''(x)}{6} h^2 \tag{1.7}$$

The error is quadratic to h, which is a significant improvement (compare cases (b) and (c) in fig. 1.1).

Example

Using the same function as in previous example

$$\begin{aligned} f(x) &= a + bx^2 \\ f(x+h) &= a + b(x+h)^2 = a + bx^2 + 2bxh + bh^2 \\ f(x-h) &= a + b(x-h)^2 = a + bx^2 - 2bxh + bh^2 \\ f'_c(x) &= \frac{f(x+h) - f(x-h)}{2h} = 2bx \end{aligned}$$

This is the exact answer, which is not very surprising since all derivatives with order higher than 3 are zero. The algorithmic error in this case is zero.

We get a much better derivative estimate for the same computational price: we still have to evaluate our function only twice to get the derivative. Thus, the central difference should be used whenever possible, unless we need to reduce computational load by reusing some values of the function calculated at prior steps[1].

[1] In some calculations, a single evaluation of a function can take days or even months.

1.4 Self-Study

Problem 1: Plot the $\log_{10}$ of the absolute error (when compared to the true value) of the $\sin(x)$ derivative at $x = \pi/4$ calculated with forward and central difference methods vs. the $\log_{10}$ of the step size h value. See `loglog` help for plotting with the logarithmic axes. The values of h should cover the range $10^{-16} \cdots 10^{-1}$ (read about MATLAB's `logspace` function designed for such cases).

Do the errors scale as predicted by eq. (1.4) and eq. (1.7)?

Why does the error decrease with h and then start to increase?

Note: the location of the minimum of the absolute error indicates the optimal value of h for this particular case.