Exercise 1.

Work out finite-differences formulas for the first derivative of a generic function f by evaluating the derivative $P'(x)$ of Newton's polynomial interpolation of f through the interpolation points specified in the following table

Location of interpolating points
$x,\ x+h$
$x-h,\ x,\ x+h$
$x-h,\ x,\ x+h\ , x+2h$

Example:

$$P_1(y) = f[x] + f[x\ x+h]\,(y-x)$$

Using the notation $f_i = f(x+ih)$ we write Newton's divided differences as

$$f[x] = f_0, \qquad f[x\ x+h] = \frac{f_1 - f_0}{h}$$

The derivative is $P_1'(y) = f[x\ x+h]$. Thus, evaluating P_1' at $y = x$ we have

$$f'(x) \approx P_1'(x) = \frac{f(x+h) - f(x)}{h}$$

Exercise 2.

Write three python functions that evaluate the first derivative of a generic f using the three finite differences expressions computed in the previous exercise. Also write a python function for the fourth-order approximation

$$f'(x) \approx \frac{f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)}{12h}$$

(if you wish, you could derive this expression with the technique of exercise 1).

Use those functions to evaluate the derivative of e^x at $x = 0$ using the values of $h = 10^{-i/5}$, $i = 1, 2, \ldots, 75$. Plot the absolute error as a function of h on a log–log graph.

Exercise 3.

Find a finite-difference formula for the third derivative of a generic function f by interpolating f at $x-h$, x, $x+h$, $x+2h$ with a third degree Newton's polynomial and evaluating the third derivative of the polynomial at x. (Just derive three times the last polynomial that you have computed for exercise 1). Once you have the finite differences formula, Taylor expand it, and give an estimate of the order of the error. Then evaluate the same formula at $z = x + 1/2 h$, Taylor expand it, and give an estimate of the order of the error. What do you notice?

Note: By "Taylor expand it" I mean systematically perform substitutions like $f(x-h) = f(x) - f'(x)h + \cdots$, or $f(z + 3/2 h) = f(z) + 3/2 f'(z)h + \cdots$, and extending the expansion up to the order that is necessary to get a meaningful result.

Exercise 4.

Write a python function that integrates an arbitrary function f over the interval $[a, b]$ by using repeatedly the method of trapezoid over the N subintervals $[x_i, x_{i+1}]$, where

$$x_i = a + i\frac{b-a}{N}, \qquad i = 0, \cdots, N.$$

Then write a similar python function that, instead, uses the midpoint method over each subinterval. The midpoint method is the following approximation:

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \approx (x_{i+1} - x_i)\, f\left(\frac{x_{i+1} - x_i}{2}\right)$$

Finally use these two functions for the test case

$$\int_0^{\pi} \sin(x)\,dx = 2$$

Use several different values of N and plot (use log-log axes) the absolute error vs the subinterval size $h = \pi/N$. If you have not made mistakes you'll find that, for both methods,

$$AbsoluteError(h) \propto h^2$$

Now answer the following two questions:

1) The midpoint method is a 1-point integration method, and the trapezoid method is a two-points integration method. How comes that they have an error that scales with the same power of h?

2) In class we showed that the trapezoid method has an error $\propto h^3$: how comes that now it is $\propto h^2$?

Exercise 5.

Write python functions that integrate that integrate an arbitrary function f over the interval (a, b) by using repeatedly the method of Gauss with 2, 3 and 4 points over the N subintervals (x_i, x_{i+1}), where

$$x_i = a + i\frac{b-a}{N}, \qquad i = 0, \cdots, N.$$

See Table 5.1 in the book for the coefficients. Then use the same test function and interval as in the previous exercise and plot (use log-log axes) the absolute error as a function of the size $h = (b-a)/N$ of the subinterval for various values of h.

Suggestions:

1) Remember that the roots and the coefficients for Gauss integration are given for the interval $(-1, 1)$. To integrate in the interval (x_i, x_{i+1}) you need to perform a change of variables. Just in case your calculus is rusty, I mean this:

$$\int_{x_i}^{x_{i+1}} f(x)\,dx = \int_{-1}^{1} f(x(t))\,\frac{dx}{dt}(t)\,dt$$

(pick the simplest expression for $x(t)$ that does the job, don't try to be fancy).

2) Don't attempt to write a function that does all at once. Break down the problem into sub-problems. Write three python functions that compute the Gauss 2-points, 3-points, 4-points integral on the interval $(-1, 1)$. Then write one (or three, if you feel more comfortable) wrapper function that loops over the subintervals, performs the change of variables and calls the Gauss–integration python functions.

3) Don't wait to finish writing the whole thing to start testing! Test along the way. In particular test that your Gauss integration functions over $(-1, 1)$ integrate polynomials exactly up to the degree stated by the theory. The following table might be useful.

(%i20) integrate(1+2*x, x, -1, 1);
(%o20) 2
(%i21) integrate(1+2*x+3*x^2+4*x^3, x, -1, 1);
(%o21) 4
(%i23) integrate(1+2*x+3*x^2+4*x^3+4*x^4, x, -1, 1);
(%o23) 28/5
(%i24) integrate(1+2*x+3*x^2+4*x^3+4*x^4+5*x^5, x, -1, 1);
(%o24) 28/5
(%i25) integrate(1+2*x+3*x^2+4*x^3+4*x^4+5*x^5+6*x^6, x, -1, 1);
(%o25) 256/35
(%i26) integrate(1+2*x+3*x^2+4*x^3+4*x^4+5*x^5+6*x^6+7*x^7, x, -1, 1);
(%o26) 256/35

```
(%i27) integrate(1+2*x+3*x^2+4*x^3+4*x^4+5*x^5+6*x^6+7*x^7+8*x^8,
x, -1, 1);
(%o27) 2864/315
```