Exercise 1.

Write a program that interpolates in the interval $[-5, 5]$ with a Newton's polynomial P the function

$$f(x) = \frac{1}{1+x^2}$$

evaluated at the nine points

$$\left\{-4.\overline{4},\ -3.\overline{3},\ -2.\overline{2},\ -1.\overline{1},\quad 0,\quad 1.\overline{1},\quad 2.\overline{2},\quad 3.\overline{3},\quad 4.\overline{4}\right\}$$

(note that they are equally spaced). Plot the interpolating polynomial and the function f using 1001 equally spaced points in the interval, as well as the nine interpolation points, and print an estimate of the maximum absolute error

$$\max_{x\in[-5,5]} |f(x) - P(x)|$$

For this, you can modify the code "polyfit.py" that comes with this homework and solves the exact same problem, but using a Lagrange polynomial. Note that you'll have to find the exact same result, because, given the same interpolation points, Newton's and Lagrange's polynomials are the same, just written in a different way.

Exercise 2.

Write a python function that, given the two extrema a and b of an interval, and a positive integer N, returns a 1D array containing all the zeros of the Chebishev polynomial T_N of degree N, rescaled from $[-1, 1]$ into the interval $[a, b]$. Then solve again the problem of exercise 1, but this time evaluate the function at the zeros of T_9 rescaled in $[-5, 5]$.

Exercise 3.

Solve the problem of exercise 1 (same function, same interval, same interpolation points) but using cubic natural splines. Just as in exercise 1, plot the interpolating spline, the function and the interpolation points and write an estimate of the maximum absolute error.

The efficient implementation of spline interpolation is a cottage industry in its own. In your code don't try to be fancy, or to "optimize" (premature optimization is the root of all evil). Aim at writing a clear and correct code. I suggest you to split the code in three python functions.

The first accepts as input the abscissas and ordinates of the N interpolation points, and returns a matrix having four columns and N-1 rows. The values on the $i-th$ row shall be the coefficients of the $i-th$ cubic polynomial S_i.

The second python function accepts these coefficients, the abscissas of the interpolation points and a single float number x and evaluates the spline interpolant at x. Care must be taken so that if x is smaller than the smaller abscissa, then S_1 is used (it's going to be an extrapolation to the left), and if x is larger than the largest abscissa then S_{N-1} is used (extrapolation to the right).

The third python function accepts the matrix of the coefficients, the abscissas of the interpolation points and a 1D array of floats, and returns the spline interpolant evaluated at the values of the 1D array. This third function is trivially a loop over the 1D array that calls the second function, and you'll find my version of it in "polyfit.py". You'll also find the declaration and doc string of the other two functions.

The most important part of the code is solving eq. 3.24 in the book. Because we haven't covered linear equations, we'll just use a canned routine. At the beginning of the code make sure you have the following line

```
from scipy.linalg import solve
```

Then prepare a the matrix `A` containing the deltas and the vector `K` of the known terms, and call:

```
c = solve(A, K)
```

The vector `c` will be the the numerical solution of eq. 3.24. With that you can then compute 3.23 and 3.22.

Exercise 4.

Repeat the computations of exercises 1, 2, 3 using $N = 3, \cdots, 40$ interpolation points. Use the same interval and function to be interpolated. With Newton's and splines interpolation use equally spaced points (if you recycle the code of "polyfit.py", you just need to change `Npts`: the assignment that defines the vector `a` will compute the correct abscissas). Plot on the same graph, as a function of N, the estimate of the maximum absolute error of each interpolant. Use a logarithmic scale on the vertical axis. Comment the results that you obtain.