Python exercises for Chapter 3. Session 2

Instructions for uploading the exercises

1. File names:
   - **Names of python scripts** are given according to the numbering of the list of exercises. Like `exercise_1.py`, `exercise_2.py`, etc.
   - **Names of output files** where the outputs are written to follow a similar naming format:
     - `exercise_1.txt`, if using the functions `print`, and `open` and `close`,
     - `exercise_1.npz`, if using the function `numpy.savez`, etc.
   - The **name of the zip file** must be `Surname1Surname2Name`, without white spaces, and excluding non-ASCII characters, such as tildes and *ñ*. For instance,

     Lucía Martín Cañas must write `MartinCanasLucia.zip`

     Include only the `exercise_*.py` files in your zip.
2. Ckeck that:
   - **Each script runs without errors**. To do this, in Spyder, or in any other IDE, restart the kernel (to clean variables) and run the script in the command window.
   - **The solution, and only the solution, is printed to the required output file.** Do not print intermmediate results in the final version of the script.

Exercises

3. Write a script for computing the cubic Natural Spline interpolating polynomial. To do this, create the function `spline_natural` with
   - Input: a point for evaluation, x, and the nodes and values of the interpolation problem.
   - Output: the value of the cubic Natural Spline interpolating polynomial.

   Use them for the following data:

   $$\text{Nodes: } 2,3,4,5,6, \quad \text{Values: } 3,\ 6,\ 2,\ 1,\ 3$$

   and evaluate the spline in a mesh of the interval $(2,6)$ with 100 equi-distant points. If your evaluation is `v = spline_natural(mesh, nodes, values)` save `v` through `numpy.savez('exercise_3', v)`.

   To check your result, make a plot containing the interpolating polynomial and the values at the nodes and compare it to Figure .

   Hint: Since the distance between adjacents nodes is the same ($=1$), the algorithm in Section 3.1 of the Handbook may be simplified. Start defining the matrix $\mathbf{H}$ in (3.15), and solving the system (3.14). Then, use formula (3.13), and compute the interpolator, $\tilde{f}$, in each subinterval $[x_i, x_{i+1}]$, using (3.12).

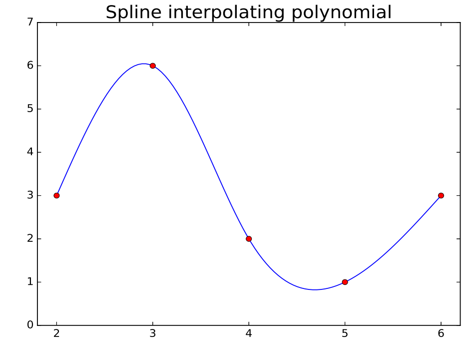

Figure : Exercise 3

4. Write a script for computing the trigonometric polynomial interpolator. To do this, create the function `fourier_interpolation` with

   - Input: a point for evaluation, x, and the nodes and values of the interpolation problem.
   - Output: the value of the interpolating polynomial.

   Use them for the following data:

   - Nodes: 100 equispaced nodes in the interval $[0, 2\pi)$. Do it like this:

   ```
   nodes = np.linspace(0, 2*np.pi, 101)
   nodes = nodes[0:-1] # remove the last one (2pi must not be among the nodes)
   ```

   - Values: the corresponding values of $f(x) = \sin(5\pi x)\cos(x)$ at these nodes.

   Evaluate the Fourier polynomial in a mesh of the interval $[0, 2\pi]$ with 1000 equi-distant points. If your evaluation is `v = fourier_interpolation(mesh, nodes, values)` save `v` through `numpy.savez('exercise_4', v)`.

   To check your result, make a plot containing the interpolating polynomial and the values at the nodes and compare it to Figure .

   Hint: Implement formula (3.19) of the Handbook. Observe that the imaginary number $i = \sqrt{-1}$ is denoted on Python as `1j`. Use the function `numpy.real` to neglect the (tiny) complex part of the values of your function `fourier_interpolation`.

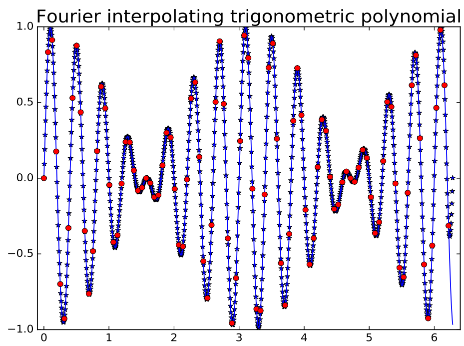

Figure : Exercise 4