Python exercises for Chapter 4. Session 1

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

1. Write a script for computing the (first order) backward, forward, and centered finite differences approximation to the derivative of a function, using the following formulas at the borders:

$$f'(x_0) \approx \frac{1}{2h}\big(-3f(x_0)+4f(x_1)-f(x_2)\big), \quad f'(x_n) \approx \frac{1}{2h}\big(3f(x_n)-4f(x_{n-1})+f(x_{n-2})\big).$$

   Use it for the function $f(x) = \sin(x)\cos(2\pi x)$ in a mesh of the interval $(0, 2\pi)$ with 100 points, and compute also the approximation to the derivative using the function `numpy.gradient`. Then, calculate the relative errors in the infinity norm between the approximations (including the numpy function) and the exact derivative (computed by hand).

   To check your results,

   - Make a plot in the interval $(0, 2\pi)$ containing all the approximations, and the exact derivative evaluated in the mesh. See Figure 1, left.
   - Same question, but in the interval $(1.65, 1.85)$. See Figure 1, right.
   - Compare your results to the following table of errors

Backward	Forward	Centered	Numpy gradient
0.203710316929	0.201181355516	0.0282294001191	0.0632500853384

Write the relative errors to a numpy array, like `err = np.array([b, f, c, g])`, in the same order than in the Table, and save it through `numpy.savez('exercise_1', err)`.

Hint: For computing the errors, use the function `norm` from `numpy.linalg`.

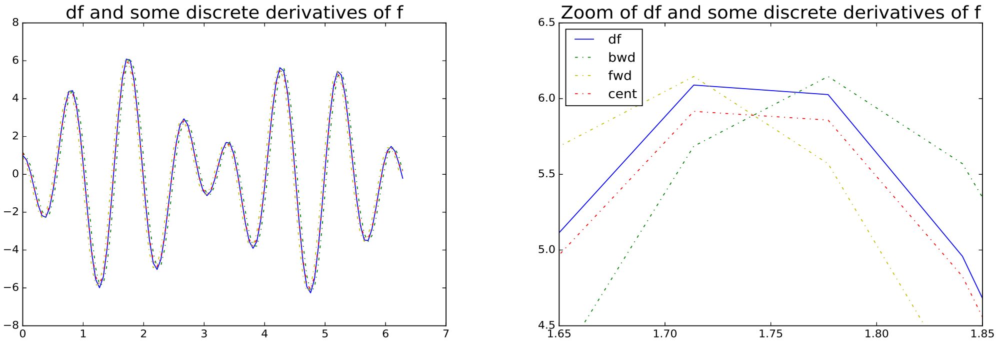

Figure : Exercise 1

2. Write a script for computing the explicit Euler scheme for approximating the solution of the differential equation $f'(t) = F(t, f(t))$, see Example 4.2, and formula (4.4). To do this, define the function `euler_explicit` with

   - Input: Function F, the initial condition, f_0, the final time, T, and the time step τ.
   - Output: The approximated solution, f, and the time mesh, $\{t_i\}$.

   Use it for the following data:

   $$F(t, x) = 2x, \quad f_0 = 1, \quad T = 1, \quad \tau = 0.01,$$

   for which the exact solution is $exact(t) = e^{2t}$.

   If your evaluation is `f, t = euler_explicit(F, f0, T, tau)` save `f` through `numpy.savez('exercise_2', f)`.

   To check your results, plot the approximated solution and the exact solution and compare to Figure 2. You can also compute the relative error in the infinity norm (with function `norm` from `numpy.linalg`), which should be 0.0195437656373.

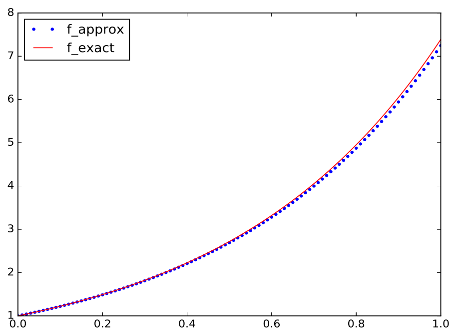

Figure : Exercise 2

3. The solution of a second order equation $f''(t) = F(f(t), f'(t))$ may be found by introducing the new unknowns $f_1 = f$, and $f_2 = f'$, which satisfy

$$f_1'(t) = f_2(t),$$
$$f_2'(t) = F(t, f_1(t)).$$

Write a script for computing the explicit Euler scheme for approximating the solution of this system of differential equations, that is, the scheme

$$f_1(t_{i+1}) = f_1(t_i) + \tau f_2(t_i),$$
$$f_2(t_{i+1}) = f_2(t_i) + \tau F(t_i, f_1(t_i)).$$

To do this, define the function `euler_explicit_system` with

- Input: Function F, the initial conditions (numbers), f_{10}, f_{20}, the final time, T, and the time step τ.
- Output: The approximated solution, f_1, f_2, and the time mesh, $\{t_i\}$ (arrays).

Use it for the following data:

$$F(x_1, x_2) = -x_1, \quad f_{10} = 0, \quad f_{20} = 1, \quad T = 6\pi, \quad \tau = 0.01,$$

for which the exact solution is $exact(t) = \sin(t)$.

If your evaluation is `f1, f2, t = euler_explicit_system(F, f10, f20, T, tau)` save the approximated solution $f \equiv$`f1` through `numpy.savez('exercise_3', f1)`.

To check your results, plot the approximated solution and the exact solution and compare to Figure 3. You can also compute the relative error in the infinity norm (with function `norm` from `numpy.linalg`), which should be 0.0904341417498.

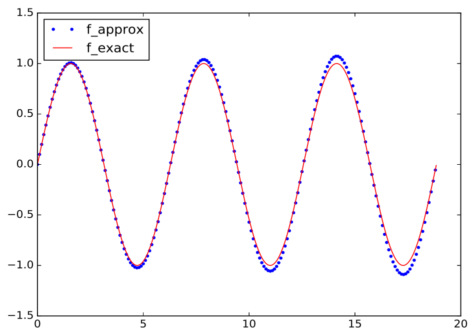

Figure : Exercise 3

4. Write a script for computing the approximations to the integral

$$\int_0^1 xe^{-2\pi x} \cos(4\pi x) dx$$

given by the composite formulas of the middle point, the trapezoidal, and the Simpson rules, see formulas (4.6), (4.7), and (4.8).

Knowing that the exact solution is $I = -0.00309340171237$, compute the absolute errors between the exact solution and the approximations obtained for the different methods for meshes of the interval $(0, 1)$ with the following step sizes, h:

$$5.e-1, \quad 1.e-1, \quad 5.e-2, \quad 1.e-2, \quad 5.e-3, \quad 1.e-3.$$

Save them in the variables (arrays) `err_mp`, `err_t`, and `err_s`, and then use `numpy.savez('exercise_4', err_mp=err_mp, err_t=err_t, err_s=err_s )`.

For instance, in the terminal, `err_mp` must give

```
array([ 2.62602795e-02, 4.62338588e-04, 1.07898248e-04, 4.21214528e-06,
1.05221701e-06, 4.20781893e-08]).
```