Computational Methods

Python exercises

Matteo Barborini[*,a], Valentin Vassilev Galindo[*,b], Prof. Alexandre Tkatchenko[*]

Theoretical Chemical Physics group

Faculté des Sciences, des Technologies et de Médecine - Département Physique et sciences des matériaux

[*]Campus Limpertsberg, Université du Luxembourg - 162 A, avenue de la Faïencerie, L-1511 Luxembourg

[a]matteo.barborini@uni.lu, [b]valentin.vassilev@uni.lu

Rules of the game

Dear students, because of the Covid-19 emergency many of you are arriving during the course and for this reason we have chosen to change the rules of the game. During this course you will receive one (or more) exercise every two weeks, and the exercises will cover in total 6 arguments. The full list of the exercises will be update constantly.
All the exercise will be delivered to me or Valentin on the 16th of December 2020 (the last lesson of the course), in which you will also receive the topics for the final project. The project will be delivered in January, and discussed with us soon afterwards.
The total grade of the course will be given by

$$\text{Final Grade} = 0.3 \cdot \text{Exercise Grade} + 0.7 \cdot \text{Project Grade}$$

Please consider that, me and Valentin expect you to constantly upgrade us on your progresses and to constantly ask us for help. During the laboratories we will answer all your questions and show you how to code the operations that you think are required to reach the result.
Following this intro you can find the list of exercise that have been given at the beginning of the course and in the last lesson.

1. Interpolation, differentiation and integration

1.1. Exercise on derivation.

1. Using as reference the function for the forward derivative `derForward` defined in `der.py`, write two functions `derBackwards` and `derCentral` that compute the derivatives of a given function respectively with the backwards and central approaches.

2. Compute the three derivatives for different values of the step h, (call the functions inside a loop), and store the values of the three derivative and the values of h into individual lists. (For the loops follow the examples inside `example08.py` and to add elements to the lists check the file `example07.py`)

3. Plot the values of the three derivatives as a function of h all together. What do you see? How do they converge as $h \to 0$?.

1.2. Exercise on integration.

1. Using as a reference the function `intMidPoint` defined in `int.py` to compute the integral with the *midpoint* method, write two functions for the integration with the *Trapezoid Rule* and *Simpson's rule*.

2. Compute the three integrals for different values of the step h (as done for the derivatives). Store the values of the integrals and the values of h into individula lists. (For the loops follow the examples inside `example08.py`) and to add elements to the lists check the file `example07.py`).

3. Plot the values of the three integrals as a function of h all together. What do you see? How do the three integration methods converge as $h \to 0$?.

1.3. Exercise on interpolation with Lagrange polynomials.

1. Build a function that, given a set of n datapoints $[(x_1, y_1), (x_2, y_2), \dots (x_n, y_n)]$ (two separate lists of equal length for the x and y values), constructs the Lagrange polynomial of order $n-1$ and returns the value of the polynomial at a given input value of x.
2. Choose a continues function (for example $sin(x)$, $cos(x)$, $exp(-x)$) in an given interval of your choice $[a, b]$: create n datapoints from the function and construct the associated Lagrange polynomial.
3. Build a plot in which you compare the original function you have chosen, with the approximated polynomials obtained by 2,3,4 and 5 datapoints respectively (Use example `plotFunction.py`).

2. Ordinary differential equations

2.1. Exercise on Euler, Euler-Cromer and Leapfrog.

1. Consider the harmonic oscillator. Build a function that computes the energy given the value of x, v and m and k. Build a function the computes $\phi(x) = -\omega^2 x$ given as imput x and $\omega = \sqrt{\frac{k}{m}}$.
2. Build two functions that given as inputs the initial position and velocity x_n, v_n and the values of $\phi(x_n)$ and Δt, compute the final values of x_{n+1}, v_{n+1} with the Euler and the Euler-Cromer algorithms.

   Build a third function that takes as input the position and the velocity $x_n, v_{n-1/2}$ and returns the values of $x_{n+1}, v_{n+1/2}$ with the Leapfrog method. Beware that the leapfrog method requires a first step to compute the velocity at the intermediate step $v_{1/2}$ and position at x_1.
3. Use these three functions to evolve your system from time $t_0 = 0$ to the time $t = 100$, with the initial conditions $x_0 = 2.0$ and $v_0 = 1.0$ and with the parameters $m = 1.0$ and $k = 3.0$ (Use $\Delta t = 0.01$ at the beginning). Plot the position $x(t)$ and the velocity $v(t)$ for the three algorithms as functions of time t and compare the results with Figure 2 of the notes in Lesson 3. What happens in the case of the leapfrog algorithm? What happens to the evolution of the relative error of the energy defined as $\Delta E = \frac{E(t)-E_0}{E_0}$ (E_0 is the energy computed with the initial conditions)?
4. Consider the three algorithms and use them to evolve the system until the time $t = 2.5$ with different time steps $\Delta t = [0.001, 0.003, 0.005, 0.007, 0.01, 0.02]$. Compute the energy of the last step (at $t = 2.5$) and plot the values of the relative error ΔE as a function of Δt for the three algorithms (see Figure 3b in Lesson 3). How does the error behave?

3. Partial differential equations

3.1. Fourier's equation of heat.

1. Consider Fourier's equation of Heat on a final interval L. Take as a reference Lesson 6, and the python scripts `fourier.py`, `solvers.py`, `main_euler.py`, `main_crnic.py`. Modify the initial condition so that

   $$u(x, t_0) = 50.0 \qquad x \in (0, L)$$

   with boundary conditions, constant during the evolution, set to $u(0) = 100.0$ and $u(L) = 0.0$. Describe the evolution of this system with the Euler method and with the Crank-Nicolson algorithm, and compare in a plot the evolutions of the solution in three points of the space, $x = L/4, L/2, 3L/4$ for $t \to \infty$.
2. Compare the full evolution of $u(x, t)$ for the two methods; that is, calculate the root mean square deviation accumulated along the time evolution:

   $$\sigma = \sqrt{\frac{1}{N_x N_t} \sum_{j=1}^{N_t} \sum_{i=1}^{N_x} (u_{eu}(x_i, t_j) - u_{cn}(x_i, t_j))^2}$$

   for different values of space Δx and time Δt ($u_{eu}(x_i, t_j)$ refers to the Euler solution and $u_{cn}(x_i, t_j)$ is the Crank-Nicolson one). Be carefull about the stability of the Euler algorithm. Plot the σ values for the various values of Δx and Δt (Use a 2D plot with different curves with σ vs Δx using Δt as a parameter).

3.2. Time-independent Schrödinger equation.

1. Consider the shooting method used to solve the time-independent Schrödinger equation using as a reference the `spectrum_shooting.py` script. Modify the integration algorithm that is used (Euler-Cromer) to the 4th order Runge-Kutta, and compare the results (with the same integration parameters) obtained with the two methods: compare the first 3 **even** eigenfunctions in one plot, and write the corresponding eigenvalues.