AA 2021/2022 Computational Methods
Midterm exercises

Mario Galante*, Almaz Khabibrakhmanov†, Alexandre Tkatchenko

Theoretical Chemical Physics group, FSTM,
Campus Limpertsberg, University of Luxembourg

*mario.galante@uni.lu
†almaz.khabibrakhmanov@uni.lu

Elastic and inelastic collisions

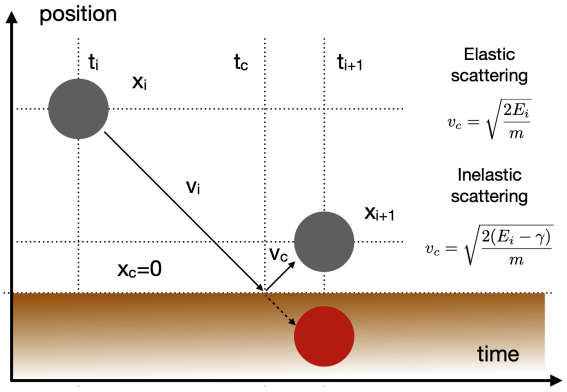

Figure 1: Schematic of a spherical object bouncing off the ground. Since the total energy is conserved away from the collision, we can extract the velocity after the impact from the energy before the collision (note that the potential energy at $x = 0$ is zero).

Algorithm 1 Vertical motion of a ball with elastic scattering off the ground

given the total number of steps, `Nt`, the time-step, `dt`, and the maximum time, `tmax`
Initialization: $x(t = 0) = x_0$, $v(t = 0) = v_0$
for `i` $= 1, \ldots,$`Nt` **do**
    $v_{half} = v_i + dt/2 * (-g)$ ▷ Half step
    $x_{i+1} = x_i + dt * v_{half}$ ▷ Update of the variables
    $v_{i+1} = v_i + dt * (-g)$
    if $x_{i+1} <= 10^{-6}$ **then** ▷ Check if we have a collision
        $t_c = -x_i / v_{half}$ ▷ Calculate the collision time
        $v_c = \sqrt{2E}$ ▷ Calculate the velocity after the elastic collision
        $x_{i+1} = v_c * (dt - t_c)$ ▷ Time evolution accounting for the collision
        $v_{i+1} = v_c - g * (dt - t_c)$
    end if
    if `i*dt==tmax` **then**
        exit loop
    end if
end for

All collisions between two objects can be separated into two categories: elastic, where the total energy is conserved, and inelastic, where some part of the total energy is dissipated. Here we consider the vertical motion of a tennis ball falling from rest at a given starting position and bouncing off the ground (see Figure 1 for a schematic representation). Here we neglect the influence of air resistance, meaning that the total energy,

$$E = \frac{1}{2}mv^2 + gx, \tag{1}$$

is conserved at all times except during a collision. As a consequence, the motion at times that do not involve a collision is given by the Newton equation of a particle in free fall,

$$\frac{d^2x}{dt^2} = -\frac{g}{m}, \quad v(0) = 0, \quad x(0) = x_0. \tag{2}$$

1. Write a Python function that employs the midpoint method to solve the dynamics of the position of the particle as described by Eq. 2. Use an if statement to stop the dynamics when the ball touches the ground, meaning when $|x| < 10^{-6}$, analogously to the exercise of lesson 5. Set the time step, dt, to 0.1, the maximum simulation time, Nt, to 20. Plot the trajectory for a mass of 1 kg and an initial height of $10m$ and verify that it produces a parabolic trend.

2. Now modify the same Python function to include *elastic* collisions with the ground in the dynamics. A collision will occur when our standard time evolution leads to a negative position. In that case, we need to calculate at which time $t_c \in [t_i, t_{i+1}]$ the collision takes place and set the velocity at t_c to be $\sqrt{2E/m}$ and directed upwards. We then continue our dynamics by evolving $x(t = t_c) = 0$ until the following time-step, t_{i+1}. The procedure just described is summarised in Algorithm 1.

   What kind of dynamics do you observe now? How many bounces does the ball do during the simulation? How does the maximum height reached for each bounce changes?

3. Modify a third time the function by including energy dissipation for each collision. The only modification needed is the subtraction of the dissipated energy, $\gamma = 15$, to the total energy when calculating v_c, i. .e. step 9 of Algorithm 1. Run again the dynamics. How many bounces do you observe now, and what happens to the maximum height reached after each bounce?

Heat equation

General theory

The general formulation of the problem for 1D heat equation is the following:

$$u_t = Du_{xx},\ 0 \le x \le a,\ 0 \le t \le t_{max} \tag{3}$$

$$u(0,x) = f(x), \quad \begin{cases} u(t,0) = \phi(t) \\ u(t,a) = \psi(t) \end{cases} \tag{4}$$

In this case, the function $u(t,x)$ describes the temperature distribution along the string of length a at different time. Note also that the heat equation contains only the first derivative in time. Therefore, we need to specify only one initial condition.

The explicit Euler method to solve the 1D heat equation uses the forward difference in time and the central difference in space to approximate the derivatives:

$$u_t = \frac{u_j^{k+1} - u_j^k}{\Delta t},\ u_{xx} = \frac{u_{j+1}^k - 2u_j^k + u_{j-1}^k}{\Delta x^2} \tag{5}$$

The lower index j refers to the space axis, and the upper index k refers to the time axis.

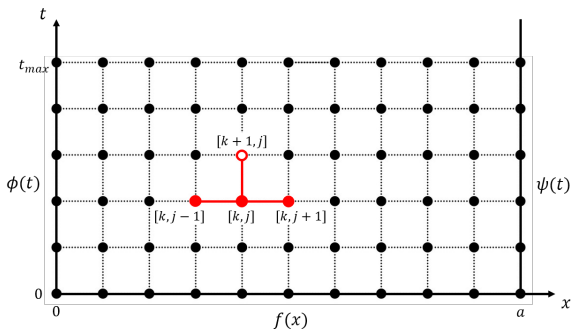

Figure 2: The numerical grid with the stencil of the computational scheme shown in red.

Solving then Eq.(1) for u_j^{k+1} at the timestep $k+1$, we obtain:

$$u_j^{k+1} = (1-2q)u_j^k + q(u_{j+1}^k + u_{j-1}^k)\ , \tag{6}$$

where $q = \frac{D\Delta t}{\Delta x^2}$ is the Courant number for the heat equation. This is the key parameter determining the stability of the numerical scheme. The explicit Euler method is stable, if:

$$q = \frac{D\Delta t}{\Delta x^2} \le \frac{1}{2} \tag{7}$$

Exercises

1. Initialize the variables which define the parameters of the problem: $D = 0.2$, $a = 1$, $t_{max} = 3$, $\Delta t = 0.02$, $\Delta x = 0.1$.

2. Set up the variables, `x` and `t`, which store the grid points in space and time, respectively. Remember that $0 \leq x \leq a$ and $0 \leq t \leq t_{max}$. The steps of the grid are defined by Δx and Δt. Calculate the numbers of points N_x and N_t required to obtain these grid steps. Then use `np.linspace` to create `x` and `t`.

3. Write the function `InitCond` implementing the following initial condition:

$$f(x) = 4x(a - x),\ 0 \leq x \leq a \tag{8}$$

The function should take a scalar x as an argument and return a scalar $f(x)$. Physical meaning of this function is the initial distribution of the temperature along the string. Plot this function and save the figure. You have to include this figure in your report. Therefore, set the proper axis limits and add the axis labels.

4. Write two functions, `LeftEnd` and `RightEnd`, implementing the following boundary conditions on the left and right ends of the string, respectively:

$$\phi(t) = 0, \qquad \psi(t) = 0 \tag{9}$$

Physically this means that we keep zero temperature at both ends of the string.

5. Write the function `HeatSolver` which implements the explicit Euler method from the Eq. (6). Below is the pseudocode that should help you.

Algorithm 2 1D heat equation solver

1: Given the arrays `x` and `t`, the parameters `a` and `q`, and the functions `InitCond`, `LeftEnd` and `RightEnd`
2: Initialize the 2D `numpy` array `u` which will store the solution. What should be the sizes of this array?
3: Assign the values to u_j^0 for all $j = 1, \ldots, N_x$ using the `InitCond`.
4: Assign the values to u_0^k and $u_{N_x}^k$ for all $k = 1, \ldots, N_t$ using the `LeftEnd` and `RightEnd`.
5: Write a double `for` loop to calculate all the unknown values of u_j^k using the Eq. (6).
6: Return the calculated 2D array `u`.

6. Call your function `HeatSolver` to solve the heat equation and save the results to the variable `u_num`.

7. Paste the function `Animator` from the lesson 7 into your code. Use it to visualize your results. Call this function adding to your code the line: `Animator(x,t,u_num,a,tmax)`. Wait for the animation to be created and saved (this may take some time). Analyze the result. Save the '.gif' animation to submit it together with your report and the code.

8. The 2D array `u_num` contains the values of the temperature at all space points and time moments. Use `u_num` to create the array `Tmid` which store the time evolution of the temperature from $t = 0$ to $t = t_{max}$ at the fixed point $x = 0.5$ (i. e. you just 'slice' your 2D array along one dimension). Plot `Tmid` vs. `t` and save the figure. You have to include this figure in your report. Does this plot agrees with what you saw in animation at point $x = 0.5$?

9. Calculate the Courant number for the numerical scheme that you used. Is the scheme stable?

Damped driven harmonic oscillator

The equation of motion for a damped harmonic oscillator with a mass of 1 kg can be written as

$$\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx = 0, \quad x(0) = x_0, \quad \frac{dx}{dt}(0) = 0 \tag{10}$$

where b is the friction constant and k is the spring constant. The general solution is given by

$$x(t) = A\ \mathrm{e}^{-bt/2+i\omega t} + B\ \mathrm{e}^{-bt/2+i\omega t}, \quad \omega = \sqrt{\frac{k}{m} - \frac{b^2}{4m^2}}, \tag{11}$$

with A and B determined given the initial condition.

1. Write a Python function that employs the midpoint method to solve Eq. 10. It should take as arguments only the friction constant, b, and the starting point, x_0, and should return the time dependence of the position. Set the spring constant, k, to 4 N/m, the time step, dt, to 0.1 s and the total number of time steps, Nt, to 250.

2. Plot in the same graph the trajectories for $b = 0.8,\ 2,\ 4,\ 8,\ 20$. How many qualitative different behaviours can you see? What is the condition that separates the different behaviours? (*Hint:* the answer can be seen from Eq. 11)

3. Eq. 10 is valid in absence of external driving. When an external force is introduced, the equation of motion becomes

$$\frac{d^2x}{dt^2} + b\frac{dx}{dt} + kx = F_0\cos(\omega_f t), \quad x(0) = x_0, \quad \frac{dx}{dt}(0) = 0, \tag{12}$$

   where F_0 and ω_f are input parameters. Modify the Python function you defined in part 1 to include the presence of an external driving of amplitude F_0 and frequency ω_f. *Note:* the external driving now has an explicit dependence on the time

4. Fix F_0 to 4 and set ω_f to 1,2,4. What features of the dynamics change with ω_f? What phenomenon are we observing?

Newton equation for a projectile under the influence of air resistance

1. The Newton equation for a spherical projectile in free fall, in absence of friction, is given by

$$\frac{d^2x(t)}{dt^2} = -g, \quad x(0) = x_0, \tag{13}$$

where $g = 9.81$ m/s^2 and the initial velocity is assumed to be 0. Define a Python function that employs the midpoint method to solve the differential equation of Eq. (1), where the initial position of the projectile should be given as argument. Set the maximum number of steps to 200, the time-step to 0.1 and the maximum simulation time to 20. The function should return an array containing the evolution of the position x with time.

2. When air friction is included the equation becomes

$$\frac{d^2x(t)}{dt^2} = -g - \frac{c\ D^2}{m}\frac{dx(t)}{dt}\left|\frac{dx(t)}{dt}\right|, \tag{14}$$

where D is the diameter of the projectile in meters, m is its mass in kg and $c = 0.25$ Ns/m^2. Define a second Python function that employs the midpoint method to solve the differential equation of Eq. (2), where the mass, diameter and initial position of the projectile should be given as arguments.

3. Test your implementation of the points 1 and 2 by setting $D = 1$ m and $m = 0.1$ kg. Try different values of the starting point, x_0: 5, 10 and 15. What is the difference between the time the sphere takes to touch the ground with and without friction? How does the starting point influence such difference?