Project 3

Contents

- Practicalities
- Introduction
- Problems

Practicalities

- **Deadline**: Wednesday, October 23, 23:59.
- **Format:**
  - A scientific report, typeset in LaTeX, delivered as a pdf file on Canvas.
  - Use the report template we have provided here.
  - Code (with comments, of course) on a UiO GitHub repo (github.uio.no), with the URL to you repo written in the pdf document.
  - You *must* deliver via your group on Canvas (even if you are working alone).
- **A scientific report:** In contrast to projects 1 and 2, we now require you to write a proper scientific report. The details of this is discussed during the lectures, and we provide you with a `.tex` file template – see this page.
- **Collaboration:** We *strongly* encourage you to collaborate with others, in groups of up to three students. The group hands in a single pdf. Remember to list everyone's name in the pdf.
- **Reproducibility:** Your code should be available on a GitHub repo. You can refer to relevant parts of your code in your answers. Make sure to include a README file in the repo that briefly explains how the code is organized, and how it should be compiled and run in order for others to reproduce your results.
- **Figures:** Figures included in your LaTeX document should be made as vector graphics (e.g. `.pdf` files), rather than raster graphics (e.g. `.png` files). If you are making plots with `matplotlib.pyplot` in Python, this is as simple as calling `plt.savefig("figure.pdf")` rather than `plt.savefig("figure.png")`.

Introduction

In this project, you'll study the effects of a device used to store charged particles. This device is known as a **Penning trap** and utilizes a static configuration of electric and magnetic fields to confine charged particles, so that they can be used for various types of measurements and experiments. In particular, experiments at CERN such as ALPHA, AEgIS and BASE use Penning traps to control the antimatter in their experiments.

Reminder on electrodynamics and classical mechanics

Not everyone in this course study physics as their main subject. Therefore we'll summarize the main equations from electrodynamics that you need to work on this project. Good old classical physics is sufficient to study the basics of Penning trap, so we won't worry about quantum aspects.

- The **electric field** ($\mathbf{E}$) is related to the **electric potential** (V) through the equation

$$\mathbf{E} = -\nabla V, \tag{7}$$

where ∇ denotes the gradient operator.

- The electric field $\mathbf{E}(\mathbf{r})$ at a point $\mathbf{r}$ set up by point charges $\{q_1, \ldots, q_n\}$ distributed at points $\{\mathbf{r}_1, \ldots, \mathbf{r}_n\}$ is

$$\mathbf{E} = k_e \sum_{j=1}^{n} q_j \frac{\mathbf{r} - \mathbf{r}_j}{|\mathbf{r} - \mathbf{r}_j|^3} \tag{8}$$

  where k_e is Coulomb's constant.
- The force $\mathbf{F}$ on a particle with charge q from an electric field $\mathbf{E}$ and magnetic field $\mathbf{B}$, known as the **Lorentz force**, is given by

$$\mathbf{F} = q\mathbf{E} + q\mathbf{v} \times \mathbf{B}, \tag{9}$$

  where $\mathbf{v}$ is the velocity of the particle.
- Finally, the time evolution of a particle is governed by Newton's second law,

$$m\ddot{\mathbf{r}} = \sum_i \mathbf{F}_i, \tag{10}$$

  where m denotes the mass of the particle and $\ddot{\mathbf{r}} \equiv d^2\mathbf{r}/dt^2$.

Note

Moving particles will in reality also induce magnetic forces on each other, but these forces will be smaller than the Coulomb forces by a factor $\mathcal{O}(v^2/c^2)$. We ignore these magnetic interactions since we are working with particle velocities far below the speed of light in vacuum, c. This simplification is often referred to as the *electrostatic approximation*.

The Penning trap

A schematic of a Penning trap is shown in figure Fig. 3. Charged particles are confined using constant electric and magnetic fields. The electric field is set by three electrodes: the two end caps (a) and a ring (b) (only the ring cross-section is shown). The constant and homogenous magnetic field in the trap is set up by a cylinder magnet (c) (only cross-section shown). The red dot represents a (positively charged) trapped particle.

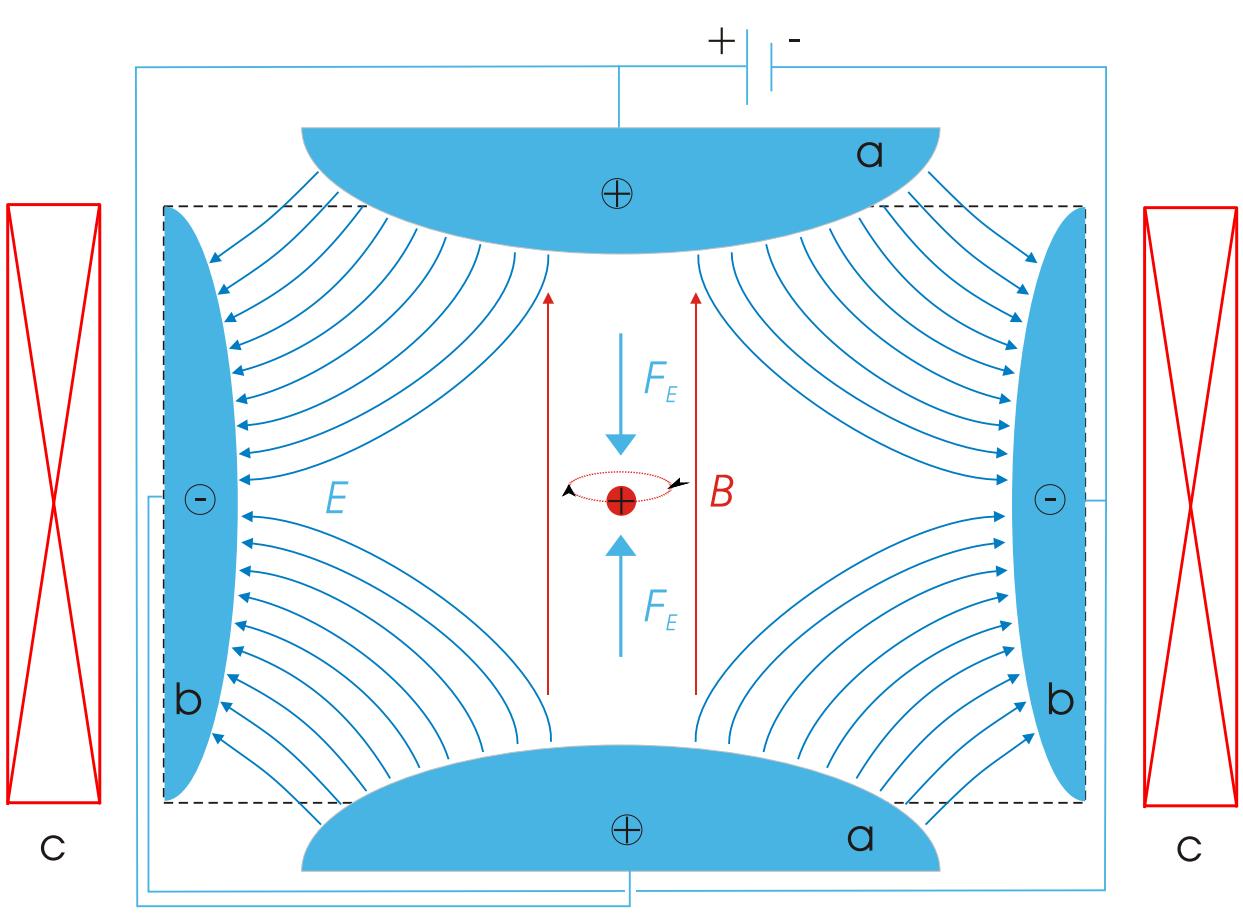

Fig. 3 Schematic of a Penning trap. See the text for further details. This illustration is by Arian Kriesch and taken from Wikimedia Commons.

We will consider an ideal Penning trap in which the electric field is defined by the electric potential

$$V(x,y,z) = \frac{V_0}{2d^2}(2z^2 - x^2 - y^2). \tag{11}$$

Here V_0 is the potential applied to the electrodes and d is called the *characteristic dimension*, which represents the length scale for the region between the electrodes. (Technically, d is defined as $d = \sqrt{z_0^2 + r_0^2/2}$, where z_0 is the distance from the center to one of the end caps and r_0 is the distance from the center to the ring.)

The electric field traps the particle in the z-direction, but the particle can still escape in the radial direction (in the xy-plane). Therefore, a homogeneous magnetic field is imposed in the z-direction of the apparatus, i.e

$$\mathbf{B} = B_0\hat{e}_z = (0, 0, B_0), \tag{12}$$

where B_0 is the field strength, with $B_0 > 0$. If this field is strong enough, the resulting force will stop particles from escaping outwards in the xy-plane by instead forcing the particles into orbital motions.

Problems

The analytical part

For now, assume that we're dealing with a *single* particle with charge q and mass m.

Problem 1

Starting from Newton's second law, show that the differential equations governing the time evolution of the particle's position (the *equations of motion*) can be written as

$$\ddot{x} - \omega_0\dot{y} - \frac{1}{2}\omega_z^2 x = 0, \tag{13}$$

$$\ddot{y} + \omega_0\dot{x} - \frac{1}{2}\omega_z^2 y = 0, \tag{14}$$

$$\ddot{z} + \omega_z^2 z = 0, \tag{15}$$

where we define

$$\omega_0 \equiv \frac{qB_0}{m}, \quad \omega_z^2 \equiv \frac{2qV_0}{md^2}.$$

Write down the general solution for eq. (15). While the above equations of motion are of course equally valid for $q < 0$ and $q > 0$, you can assume $q > 0$ for this project.

Problem 2

Eqs. (13) and (14) introduces a challenge because the two equations are coupled.

Introduce a complex function $f(t) = x(t) + iy(t)$ and show that eqs. (13) and (14) can be rewritten to a single differential equation for f as

$$\ddot{f} + i\omega_0\dot{f} - \frac{1}{2}\omega_z^2 f = 0. \tag{16}$$

Problem 3

The general solution to eq. (16) is

$$f(t) = A_+ e^{-i(\omega_+ t + \phi_+)} + A_- e^{-i(\omega_- t + \phi_-)}, \tag{17}$$

where ϕ_+ and ϕ_- are constant phases, the amplitudes A_+ and A_- are positive, and

$$\omega_{\pm} = \frac{\omega_0 \pm \sqrt{\omega_0^2 - 2\omega_z^2}}{2}.$$

The physical coordinates are then found as $x(t) = \mathrm{Re} f(t)$ and $y(t) = \mathrm{Im} f(t)$.

What is the necessary constraint on ω_0 and ω_z to obtain a bounded solution for the movement in the xy-plane (i.e. a solution where $|f(t)| < \infty$ as $t \to \infty$). Also express this as a constraint that relates the Penning trap parameters (B_0, V_0, d) to the particle properties (m, q).

Problem 4

Show that the upper and lower bounds on the particle's distance from the origin in the xy-plane are given by $R_+ = A_+ + A_-$ and $R_- = |A_+ - A_-|$, respectively.

Specific analytical solution

For later tests of the code you develop, you will need a specific analytical solution to use for comparison. For this we will consider the case of a single charged particle with charge q and mass m in the Penning trap, with the following initial conditions:

$$\begin{aligned} x(0) &= x_0, \qquad \dot{x}(0) = 0, \\ y(0) &= 0, \qquad \dot{y}(0) = v_0, \\ z(0) &= z_0, \qquad \dot{z}(0) = 0. \end{aligned}$$

From eq. (15) and the initial values for $z(0)$ and $\dot{z}(0)$ we get that the specific solution for $z(t)$ in this case is

$$z(t) = z_0 \cos(w_z t).$$

For the movement in the xy-plane, the specific solution for $f(t)$ is given by eq. (17) with

$$\begin{aligned} A_+ = \frac{v_0 + \omega_- x_0}{\omega_- - \omega_+}, \qquad A_- = -\frac{v_0 + \omega_+ x_0}{\omega_- - \omega_+}, \\ \phi_+ = 0, \qquad \phi_- = 0. \end{aligned}$$

With multiple particles

We remind you that so far we have only been looking at the special case of a single particle. In a trap with more particles, the equations of motion for each particle are coupled to those of the other particles. Specifically, consider the case where we fill our Penning trap with a set of n particles with charges $\{q_1, \ldots, q_n\}$ and masses $\{m_1, \ldots, m_n\}$. Each particle will then experience both the force from the external electric and magnetic fields and the Coulomb force from all the other particles. In this case our set of equations of motion would become

$$\ddot{x}_i - \omega_{0,i}\dot{y}_i - \frac{1}{2}\omega_{z,i}^2 x_i - k_e \frac{q_i}{m_i} \sum_{j \neq i} q_j \frac{x_i - x_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} = 0, \tag{18}$$

$$\ddot{y}_i + \omega_{0,i}\dot{x}_i - \frac{1}{2}\omega_{z,i}^2 y_i - k_e \frac{q_i}{m_i} \sum_{j \neq i} q_j \frac{y_i - y_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} = 0, \tag{19}$$

$$\ddot{z}_i + \omega_{z,i}^2 z_i - k_e \frac{q_i}{m_i} \sum_{j \neq i} q_j \frac{z_i - z_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} = 0. \tag{20}$$

Here i and j are particle indices. We will not attempt an analytical treatment of this problem, but rather move on to a numerical simulation.

Note

While the above equations are the equations our simulation will solve, our code does not have to contain these equations as single, long mathematical formulas. First of all, the equations above are *second-order* differential equations, while we will use Forward Euler and Runge-Kutta as methods for solving *first-order* differential equations. So when approaching the numerical part of this project, the first step is to formulate the problem in terms of coupled, first-order differential equations:

$$\dot{\mathbf{r}} = \mathbf{v}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{F}}{m}$$

This suggests that all the physics-specific details in the second-order differential equations above will effectively be contained in the function(s) you use to compute the total force $\mathbf{F}$ acting on a particle at position $\mathbf{r}$ with velocity $\mathbf{v}$ at time t.

The code snippets we provide at the end will outline an approach where the contributions from different forces are specificed in separate functions in the code. This will allow us to write a quite *general* Penning trap simulator, for which the specific Penning trap configuration described by the equations above is just one special case.

Code development and numerical investigations

In this part, you'll develop a code to simulate a set of n particles with charges $\{q_1, \ldots, q_n\}$ and masses $\{m_1, \ldots, m_n\}$. You are **required** to write an **object-oriented** code in this project. Don't worry, though – we'll help you with suggestions for *how* you might structure your code. Also, writing object-oriented code does not mean that *everything* in the code should be solved using classes. Below we only require you to write two classes, `Particle` and `PenningTrap`. The other aspects of your code you are free to design in the way you think is best.

Numbers and units

For this project we'll work with the following set of base units:

- Length: micrometre ($\mu\mathrm{m}$)
- Time: microseconds ($\mu\mathrm{s}$)
- Mass: atomic mass unit (u)
- Charge: the elementary charge (e)

In these units, the Coulomb constant takes the value

- $k_e = 1.38935333 \times 10^5 \ \frac{\mathrm{u}\ (\mu\mathrm{m})^3}{(\mu\mathrm{s})^2\ \mathrm{e}^2}$

and the derived SI units for magnetic field strength (Tesla, T) and electric potential (Volt, V) become

- $\mathrm{T} = 9.64852558 \times 10^1 \ \frac{\mathrm{u}}{(\mu\mathrm{s})\ \mathrm{e}}$
- $\mathrm{V} = 9.64852558 \times 10^7 \ \frac{\mathrm{u}\ (\mu\mathrm{m})^2}{(\mu\mathrm{s})^2\ \mathrm{e}}$.

Our default Penning trap configuration will have

- $B_0 = 1.00\,\mathrm{T} \simeq 9.65 \times 10^1 \ \frac{\mathrm{u}}{(\mu\mathrm{s})\ \mathrm{e}}$
- $V_0 = 25.0\,\mathrm{mV} \simeq 2.41 \times 10^6 \ \frac{\mathrm{u}\ (\mu\mathrm{m})^2}{(\mu\mathrm{s})^2\ \mathrm{e}}$
- $d = 500\,\mu\mathrm{m}$.

We note that V_0 and d only appear as the ratio V_0/d^2, which now takes the value

- $V_0/d^2 = 9.65 \ \frac{\mathrm{u}}{(\mu\mathrm{s})^2\ \mathrm{e}}$.

(Note that we have here used the precise $V_0 = 25.0\,\mathrm{mV}$.)

We'll use singly-charged Calcium ions (Ca^+) as our charged particles.

Problem 5

Create a class named `Particle` that should at least store the following properties of a single particle:

- Charge (q)
- Mass (m)
- Position ($\mathbf{r}$)
- Velocity ($\mathbf{v}$)

You are of course free to extend the class with more attributes (member variables) and/or methods (member functions) if you think it is useful.

Some advice:

- The constructor should assign values to the member variables.
- For the position and velocity, use `arma::vec` objects.
- Make all member variables public. This is just to ease the interplay between the `Particle` class and the `PenningTrap` class. (Alternatively, a nice approach in this case would be to make `PenningTrap` a friend class of `Particle`.)

Problem 6

Create a class named `PenningTrap` that should at least contain member variables for the following:

- The magnetic field strength (B_0)
- The applied potential (V_0)
- The characteristic dimension (d)
- A `std::vector<Particle>` to contain all the `Particle` objects in the Penning trap.

Further, the class should contain some member functions for evaluating

- The external electric field
- The external magnetic field
- The force due to the interaction among the particles

Remember to run small tests during your code development. In particular, since the analytic solution we have found are for the case of a single particle, you can consider carrying out the single-particle tests in Problem 8 before implementing any of the code related to particle interactions.

Problem 7

Now we need functionality for actually solving the equations of motion, i.e. evolving our system in time. Our main method will be the fourth-order Runge-Kutta (RK4) method, but for comparison (and debugging) we'll also implement the simple forward Euler method.

So, write functions that can evolve your `PenningTrap` system in time using

- the forward Euler method
- the RK4 method

You can implement these methods either

- via regular functions, that take a reference to a `PenningTrap` object as input;
- via some type of "solver class" or "integrator class", that contains/interacts with the `PenningTrap` object; or

- via member functions implemented directly in the `PenningTrap` class.

The code snippets below outline the third approach, but feel free to choose the approach you prefer.

We will use RK4 in the problems below, unless specified otherwise. However, having the simple forward Euler method implemented is still very useful – for instance it can help you check if a problem in your results is due a problem in your RK4 code or some other part of the code.

> **Note**
>
> In the Runge-Kutta algorithm you have to compute a series of k's. More specifically, for every particle in the Penning trap you need four 3-vectors $\mathbf{k}_{\mathbf{r},1}$, $\mathbf{k}_{\mathbf{r},2}$, $\mathbf{k}_{\mathbf{r},3}$, $\mathbf{k}_{\mathbf{r},4}$ to update the position, and four 3-vectors $\mathbf{k}_{\mathbf{v},1}$, $\mathbf{k}_{\mathbf{v},2}$, $\mathbf{k}_{\mathbf{v},3}$, $\mathbf{k}_{\mathbf{v},4}$ to update the velocity. Recall that each k is associated with a specific whole/half timestep. Since the accelleration for one particle depends on the positions of all other particles (due to the Coulomb interactions), this means that the positions and velocities for *all* particles in the trap must be updated to the correct whole/half timestep before the next set of k's can be computed.
>
> In other words, your implementation of the Runge-Kutta algorithm for one timestep should probably go something like this:
>
> - Make a temporary copy of all the particles in the Penning trap, since we'll need the original positions and velocities to perform the final RK4 update step.
> - For each particle, compute $\mathbf{k}_{\mathbf{r},1}$ and $\mathbf{k}_{\mathbf{v},1}$.
> - For each particle, update the position and velocity using the corresponding $\mathbf{k}_{\mathbf{r},1}$ and $\mathbf{k}_{\mathbf{v},1}$.
> - For each particle, compute $\mathbf{k}_{\mathbf{r},2}$ and $\mathbf{k}_{\mathbf{v},2}$.
> - For each particle, update the position and velocity using the corresponding $\mathbf{k}_{\mathbf{r},2}$ and $\mathbf{k}_{\mathbf{v},2}$
> - …
> - For each particle, compute $\mathbf{k}_{\mathbf{r},4}$ and $\mathbf{k}_{\mathbf{v},4}$
> - Final step: For each particle, perform the proper RK4 update of position and velocity using the original particle position and velocity, together with all the $\mathbf{k}_{\mathbf{r},i}$ and $\mathbf{k}_{\mathbf{v},i}$ computed above.

Problem 8

For the simulations in this problem we'll consider only one or two particles. Use the following initial conditions for all these simulations:

- Particle 1:
  - $(x_0, y_0, z_0) = (20, 0, 20)\,\mu\mathrm{m}$
  - $(v_{x,0}, v_{y,0}, v_{z,0}) = (0, 25, 0)\,\mu\mathrm{m}/\mu\mathrm{s}$
- Particle 2:
  - $(x_0, y_0, z_0) = (25, 25, 0)\,\mu\mathrm{m}$
  - $(v_{x,0}, v_{y,0}, v_{z,0}) = (0, 40, 5)\,\mu\mathrm{m}/\mu\mathrm{s}$

It's now time to test and explore your simulation:

- Simulate the movement of a single particle in your Penning trap for a total time of $50\mu\mathrm{s}$. Make a plot of the motion in the z direction as a function of time. Does the result look as expected, given the value of ω_z?
- Simulate two particles in your Penning trap and make a plot of their motion in the (x, y)-plane *with* and *without* particle interactions.
- Similarly, for the case of two particles, make plots of the trajectories in the (x, v_x) and (z, v_z) planes (i.e. *phase space* plots) for the cases *with* and *without* interactions. How do the trajectories change when you include interactions? Are the results reasonable from a physical point of view?

- Make a 3D plot of the trajectory of two particles in the (x, y, z) space *with* and *without* interactions.

> **Note**
>
> An example of 3D plotting with matplotlib can be found here.

- Now consider the case of a single particle, which means we can easily compare to the analytical solution. Use the initial values for Particle 1 given above, and let the total simulation time be $50\mu s$. Run the simulation four times, using $n_1 = 4000$, $n_2 = 8000$, $n_3 = 16000$ and $n_4 = 32000$ timesteps. (Corresponding stepsizes: $h_k = 50/n_k\ \mu s$). For each of the four simulations, make a graph showing the size of the relative error in $\mathbf{r}_i$ at each time step t_i. Present the four graphs in a single plot. Do the same using the forward Euler method.
- Using the simulation results for the four different stepsize values h_k, estimate the error convergence rate r_{err} for forward Euler and RK4 with the expression

$$r_{\text{err}} = \frac{1}{3}\sum_{k=2}^{4}\frac{\log(\Delta_{\max,k}/\Delta_{\max,k-1})}{\log(h_k/h_{k-1})}.$$

where

$$\Delta_{\max,k} = \max_i |\mathbf{r}_{i,\text{exact}} - \mathbf{r}_i|$$

is the maximum error (taken over all timesteps i) of the simulation with stepsize h_k. Here $\mathbf{r}_{i,\text{exact}}$ is the analytical solution and $\mathbf{r}_i$ our numerical approximation.

> **Note**
>
> When trying to interpret the plots from the simulations above, you may find it useful to also look at other plots, e.g. plots of the (t, x) and (t, y) planes, or plots of the Coulomb force between the two particles as function of time, or something else. (But we do not expect you to include these plots in the report.)

> **Note**
>
> Plots with particle trajectories can be complicated to read. You may find it useful to use e.g. `plt.scatter` to put some markers in your plot that show the start point and end point of each trajectory.

> **Note**
>
> To make sure that e.g. circular orbits actually look circular in your plots, the pyplot command `plt.axis('equal')` might come in handy.

> **Note**
>
> If your simulation results look wrong, it can be useful to start the debugging by only simulating a few timesteps rather than running the full $50\mu s$ simulation.

Problem 9

Now that we have explored the basics of our simulation setup it's time to use it to explore some Penning trap physics.

In a system with complicated periodic motions we should expect the possibility for seeing resonance phenomena. In a Penning trap, this can be investigated by subjecting the system to a time-dependent electromagnetic field and study *the loss of trapped particles as function of the*

applied frequency.

There are multiple ways of subjecting the system to a time-dependent field. Here we will do it by including a time-dependent perturbation to the applied potential, i.e. make the replacement

$$V_0 \rightarrow V_0 \left(1 + f \cos(\omega_V t)\right), \tag{21}$$

where f is a constant amplitude and ω_V is the angular frequency of the time-dependent potential term.

For this task you should implement (at least) the following extensions to your code:

- A check that sets the external E- and B-fields to zero in the region outside the trap. We'll use the characteristic distance d as a simple measure for the trap size, so what we need is a check that sets the external fields $\mathbf{E}(\mathbf{r})$ and $\mathbf{B}(\mathbf{r})$ to zero when $|\mathbf{r}| > d$.
- An extension of the code for the external E-field such that it can have a time dependence. (For the different steps in RK4, make sure you evaluate the now time-dependent force at the correct time steps.)
- A small function (either as part of the `PenningTrap` class or outside it) that can count how many of the particles are still inside the trap region, i.e. the number of particles with $|\mathbf{r}| < d$.
- An option to switch the Coulomb interactions on/off.
- Code for filling the `PenningTrap` with particles with randomly generated initial positions and velocities. Initial positions and velocities for each particle can be sampled from normal distributions, suitably scaled relative to the length scale of our Penning trap. Here's a snippet illustrating this for a single particle, using the `vec::randn()` function of Armadillo to sample a position and velocity (`my_trap` is an instance of `PenningTrap`):

  ```
  vec r = vec(3).randn() * 0.1 * my_trap.d;  // random initial position
  vec v = vec(3).randn() * 0.1 * my_trap.d;  // random initial velocity
  ```

  Note: To set the seed for Armadillo's random number generator, you can use `arma_rng::set_seed(value)` or `arma_rng::set_seed_random()`.

We want to use our simulation to search for resonance frequencies of the system. Starting from a system filled with *100 randomly initialized Ca^+ particles*, do the following:

- For each of the amplitudes $f = 0.1, 0.4, 0.7$, produce a graph that shows *the fraction of particles that are still trapped after* $500\,\mu\text{s}$ as a function of the applied angular frequency ω_V. Plot the three graphs in the same figure. You should explore frequencies in the range $\omega_V \in (0.2, 2.5)\,\text{MHz}$. For this broad exploration of frequencies you can *switch off* the Coulomb interactions between the particles in the trap, as your code should then run much faster. Make sure you use sufficiently small steps along the ω_V axis – steps of $0.02\,\text{MHz}$ can be a starting point. Comment on your results. Some suggested things to discusss:
  - A qualitative explanation of what's going on: Given the periodic / quasi-periodic nature of the Penning trap system, why is it that some frequencies can be particularly effective for pushing particles out of the trap? (Here we do *not* expect a detailed explanation of the physics behind why the resonances appear exactly where they do – the resonance structure of a many-particle Penning trap is a complicated topic beyond the scope of this project.)
  - How do the resonances change when the amplitude for the time-varying potential is increased?

- Now we want to check if the Coulomb interactions have some impact on the structure of these resonances. To do this you should "zoom in" on one of the resonances you've uncovered by performing fine-grained frequency scans around that resonance.
  - Perform two such scans, one where you include Coulomb interactions in your simulation and one without. (If you have time, you can run multiple scans for each case and use the average results, to lower the statistical uncertainty coming from the fact that we only simulate 100 randomly initialized particles.)

- Produce a plot that shows the fraction of trapped particles versus frequency for the two cases. Comment on noteworthy differences.

> **Note**
>
> Running a simulation with 100 particles is computationally quite demanding, and runtimes can vary greatly depending on how the code is written, the particular machine the code is run on, etc. Enabling compiler optimisation can have quite a big impact on the runtime. We recommend adding either `-O2` or `-O3` to your compilation command.
>
> As a reference, for my (Anders) code and laptop, running a 100-particle simulation for 40,000 timesteps using RK4 and *no* Coulomb interactions takes ~10 seconds. This reduces to ~2 seconds when compiling with `-O2`.

> **Note**
>
> Some of you may experience runtimes that are so long that you are unable to perform the series of 100-particle simulations for this problem. If you are unable to reduce the runtime by optimising your code, switching on compiler optimisation, etc, you can reduce the problem size by reducing the number of particles until you get workable runtimes. If you have to do this, please comment on it in your report.

Good luck!

Code snippets

Here is a suggested starting point for member functions of the `PenningTrap` class:

```
  // Constructor
  PenningTrap(double B0_in, double V0_in, double d_in);

  // Add a particle to the trap
  void add_particle(Particle p_in);

  // External electric field at point r=(x,y,z)
  arma::vec external_E_field(arma::vec r);

  // External magnetic field at point r=(x,y,z)
  arma::vec external_B_field(arma::vec r);

  // Force on particle_i from particle_j
  arma::vec force_particle(int i, int j);

  // The total force on particle_i from the external fields
  arma::vec total_force_external(int i);

  // The total force on particle_i from the other particles
  arma::vec total_force_particles(int i);

  // The total force on particle_i from both external fields and other
particles
  arma::vec total_force(int i);

  // Evolve the system one time step (dt) using Runge-Kutta 4th order
  void evolve_RK4(double dt);

  // Evolve the system one time step (dt) using Forward Euler
  void evolve_forward_Euler(double dt);
```

Note that for Problem 9 you probably want to modify the declarations of some of these functions, as well as add some new ones.
