Accuracy and Stability of Numerical Integration Methods in Penning Trap Simulations

Lars Bogner
(Dated: October 16, 2025)

Using integration algorithms for ordinary differential equations, we predict the time evolution of charged particles in the electromagnetic fields of a Penning trap. Comparing the computational cost and numerical accuracy of the Forward Euler, Velocity-Verlet, Boris and Runge-Kutta 4th order methods, we find that the Boris algorithm is the most efficient choice for this problem. It is phase-space preserving, energy conserving and reduces the computational cost by up to 700 % compared to general purpose approaches. To study ensemble properties of the trapped particles, we simulate the dynamics of 100 particles to find resonances and observe energy conservation of the different algorithms. Comparing to analytical solutions, at 32 000 time steps of $h = 1.56\,\mathrm{ns}$, the Boris algorithm achieves a relative numerical error of the trajectory of less than 2×10^{-3}.

I. INTRODUCTION

Penning traps are useful appliances to capture charged particles without contact to any matter surrounding the particles [1, 11]. This property of Penning traps is especially useful in the study of antimatter, as any contact to matter would lead to the direct annihilation of the antimatter. Therefore, Penning traps have become an essential tool in the antimatter research. Experiments like ALPHA-g [2] and AEgIS [3] at CERN have successfully deployed Penning traps for such applications.

To plan future antimatter experiments it is essential to be able to simulate Penning traps. Without the ability of the behavior of particles within such a trap, a precise planning of the Penning trap and antimatter experiment is not possible. Key criterion for a successful simulation in this context is accuracy of the numerical simulation in terms of precise trajectories and key statistical properties of the captured particles, e.g. the kinetic energy. A second metric that is very important to the usefulness of such numerical simulations is the computational performance of the simulation as this decides whether the simulation is even possible or not with finite computing resources. To find the optimal balance between the finite computing power available and an accurate result, different integration algorithms will be compared to solve the ordinary differential equations at play in the Penning Trap.

In section II the methods employed in the simulation will be expanded on. In detail the implementation of the source code will be discussed as well as the specific Penning trap used for the study, introduced. Thirdly the algorithms behind the numerical integration methods will be explained. Section III will show the results obtained from our implementation and discuss their practical implications. This section will be split into four parts expanding on the accuracy of the trajectories, the energy conservation in the system, the computational performance of the different algorithms as well as a discussion on the limit of many particles in the trap. Lastly in section IV our findings will be summarized and consequences of the results derived.

II. METHODS

Penning trap

Penning traps use a combination of a magnetic and electric field to confine the trajectory of a charged particle to a finite volume over long periods of time[1, p. 9-10]. Finite volume in this context refers to volumes in the order of mm^3 up to cm^3. Long time ranges imply a period that exceeds the period of motion of the particle by multiple orders. For the studied case we will use an ideal Penning trap, i.e. there are no inhomogenities in the fields. The magnetic field will be defined as

$$B(\vec{r}, t) = B_0 \Theta\left(d - |\vec{r}|\right) \cdot \vec{e}_z, \tag{1}$$

with the Heaviside function $\Theta(x)$, and we define two electric fields via the electric potentials

$$V_{\mathrm{stat.}}(\vec{r}, t) = \frac{V_0 \cdot (2z^2 - x^2 - y^2)\Theta\left(d - |\vec{r}|\right)}{2d^2} \tag{2}$$

and

$$V_{\mathrm{dyn.}}(\vec{r}, t) = (1 + f \cos \omega_V t) \cdot V_{\mathrm{stat.}}(\vec{r}, t). \tag{3}$$

Equations of motion

Using Newton's equations of motion, $\ddot{\vec{r}} = \vec{F}/m$, we can derive ordinary differential equations for the position of a particle which is subject to eqs. (1) and (2). As the magnetic field is parallel to the z-axis, the differential equation for the vertical component is straightforward to derive. Using the electric field resulting from $V_{\mathrm{stat.}}$ (see appendix A for details on all electric field components) we find

$$\ddot{z} + \omega_z^2 z \equiv \ddot{z} + \frac{2qV_0}{md^2} z = 0. \tag{4}$$

This is a simple harmonic oscillator equation which can be solved using standard methods. In the transversal plane, the equations of motion are coupled due to the

Lorentz force $\vec{F} = q\vec{v} \times \vec{B}$. Using the electric field resulting from $V_{\text{stat.}}$ (see appendix A for details on all electric field components) we find

$$\ddot{x} - \omega_0 \dot{y} - \frac{1}{2}\omega_z^2 x \equiv \ddot{x} - \frac{qB_0}{m}\dot{y} - \frac{qV_0}{md^2}x = 0, \tag{5}$$

$$\ddot{y} + \omega_0 \dot{x} - \frac{1}{2}\omega_z^2 y \equiv \ddot{y} + \frac{qB_0}{m}\dot{x} - \frac{qV_0}{md^2}y = 0. \tag{6}$$

Using the sum of equations: (5) + i(6), we can use the general definition $f(t) \equiv x(t) + iy(t)$ to derive a single complex differential equation for the transversal motion

$$\ddot{f} + i\omega_0 \dot{f} - \frac{1}{2}\omega_z^2 f = 0. \tag{7}$$

After solving the differential equation, the real and imaginary parts of $f(t)$ will correspond to the x and y components of the trajectory, respectively. Equation (7) is a damped harmonic oscillator equation with the general solution

$$f(t) = A_+ e^{-i(\omega_+ t + \phi_+)} + A_- e^{-i(\omega_- t + \phi_-)}, \tag{8}$$

with the two characteristic frequencies

$$\omega_\pm = \frac{\omega_0 \pm \sqrt{\omega_0^2 - 2\omega_z^2}}{2}. \tag{9}$$

This solution is valid for $\omega_0^2 > 2\omega_z^2$. In the case of $\omega_0^2 \leq \omega_z^2$, the oscillation frequencies become complex, leading to exponential growth of the trajectory in the transversal plane. This condition is known as the stability criterion for a Penning trap [1, p. 67]. In terms of the physical parameters of the trap, the stability criterion can be expressed as

$$\frac{q}{m}B_0^2 > \frac{2V_0}{d^2}. \tag{10}$$

Given this condition, the solution for the transversal motion will be bound, i.e. $|f(t)| < \infty$ for all $t > 0$. Given the stability criterion, the two terms in eq. (8) can be identified as sinusoidal motions with amplitudes $A_\pm$. If the two terms are in phase the upper limit of $|f| \equiv R_+$ will be $R_+ = A_+ + A_-$, while the lower limit for the case of antiparallel phases will be the absolute of the difference in the two amplitudes, $R_- = |A_+ - A_-|$. A analytical solution for a special case of initial conditions is derived in appendix B.

Multiple particles

For the case of multiple particles, it is important to include the particle-particle interactions. For this analysis we will only consider the Coulomb interaction between the particles, neglecting magnetic interactions. The force on particle i due to all other particles j is then given by

$$\vec{F}_i = \frac{q_i}{4\pi\epsilon_0} \sum_{j \neq i} \frac{q_j(\vec{r}_j - \vec{r}_i)}{|\vec{r}_j - \vec{r}_i|^3}. \tag{11}$$

Numerical Integration Methods

Forward Euler method

The simplest numerical method to solve the ordinary differential equation system is the Forward Euler method. It is a first order method, meaning that the local truncation error per step is on the order of $\mathcal{O}(h^2)$, with h being the step size. The global error after N steps is therefore on the order of $\mathcal{O}(h)$ [4]. The method is explicit, meaning that the state of the system at the next time step can be calculated directly from the current state. Given a general ordinary differential equation of the form

$$\dot{y}(t) = f(t, y(t)), \tag{12}$$

the Forward Euler method updates the state of the system as follows:

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{13}$$

For our second order differential equations, we first rewrite them as a system of first order equations. The next state of the system is then calculated using the current position and velocity as [4]

$$\vec{r}_{n+1} = \vec{r}_n + h\vec{v}_n, \tag{14}$$

$$\vec{v}_{n+1} = \vec{v}_n + h\frac{\vec{F}(\vec{r}_n, \vec{v}_n, t_n)}{m}. \tag{15}$$

This method is computationally inexpensive, as we only need a single force evaluation per time step. However, it is known to be not very accurate.

Runge-Kutta 4 method

The Runge-Kutta 4 (RK4) method is a popular and more accurate method for solving ordinary differential equations. It is a fourth-order method, meaning that the local truncation error per step is on the order of $\mathcal{O}(h^5)$, and the global error after N steps is on the order of $\mathcal{O}(h^4)$ [4]. The RK4 method calculates the next state of the system using a weighted average of four different estimates of the slope (the derivative) at different points within the time step. Given a general ordinary differential equation of the form

$$\dot{y}(t) = f(t, y(t)), \tag{16}$$

the RK4 method updates the state of the system as follows [4, 5]:

$$k_1 = hf(t_n, y_n), \tag{17}$$

$$k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right), \tag{18}$$

$$k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right), \tag{19}$$

$$k_4 = hf(t_n + h, y_n + k_3), \tag{20}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4). \tag{21}$$

As this method is again a generalized solver for ODE of the first order, we again rewrite our second order differential equations as a system of first order equations.

Velocity-Verlet method

In contrast to the previous two methods, the Velocity-Verlet method is a symplectic integrator which is specifically designed for second order differential equations of the form

$$\ddot{\vec{r}}(t) = \frac{\vec{F}(\vec{r}(t), t)}{m}. \tag{22}$$

The method is time-reversible and conserves energy better over long time periods compared to non-symplectic methods like Forward Euler and RK4 [6]. The Velocity-Verlet method updates the position and velocity of the system as follows [6]:

$$\vec{r}_{n+1} = \vec{r}_n + h\vec{v}_n + \frac{h^2}{2}\vec{a}_n, \tag{23}$$

$$\vec{a}_{n+1} = \frac{\vec{F}(\vec{r}_{n+1}, t_{n+1})}{m}, \tag{24}$$

$$\vec{v}_{n+1} = \vec{v}_n + \frac{h}{2}(\vec{a}_n + \vec{a}_{n+1}), \tag{25}$$

It is important to note that the Velocity-Verlet integrator requires forces to be independent of velocity. In the case, that the algorithm is applicable, it is a very efficient method, as it only requires a single force evaluation per time step, while still being a second order method with a local truncation error per step on the order of $\mathcal{O}(h^3)$ and a global error after N steps on the order of $\mathcal{O}(h^2)$[6]. Because of the symplectic nature, the low number of force evaluations and the good energy conservation properties, the Velocity-Verlet method is widely used and the quasi-standard in molecular dynamics simulations[6].

Boris algorithm

The Boris algorithm is a widely used method for integrating the equations of motion of charged particles in electromagnetic fields [7, 8]. It is particularly well-suited for problems where the Lorentz force plays a significant role, as it is designed to handle the velocity-dependent nature of the magnetic force. The Boris algorithm is a symplectic integrator[7][1], which means it conserves the phase space volume and exhibits good long-term energy conservation properties. The algorithm updates the position and velocity of a charged particle in a magnetic field as follows: We define two auxiliary velocities $\vec{v}^-$ and $\vec{v}^+$, which represent the velocity before and after the magnetic field rotation, respectively. The algorithm proceeds in the following steps [9]:

$$\vec{v}^- = \vec{v}_n + \frac{q\vec{E}h}{2m} \tag{26}$$

$$\vec{v}' = \vec{v}^- + \vec{v}^- \times \vec{t} \tag{27}$$

$$\vec{v}^+ = \vec{v}^- + \vec{v}' \times \vec{s} \tag{28}$$

with

$$\vec{t} = \frac{q\vec{B}h}{2m} \tag{29}$$

$$\vec{s} = \frac{2\vec{t}}{1 + |\vec{t}|^2}. \tag{30}$$

The final update of the velocity and position is then given by

$$\vec{v}_{n+1} = \vec{v}^+ + \frac{q\vec{E}h}{2m} \tag{31}$$

$$\vec{r}_{n+1} = \vec{r}_n + h\vec{v}_{n+1} \tag{32}$$

Code Structure

The basic framework for the numerical analysis is based on a `PenningTrap` class, which contains all the particles present in the trap, as well as a parametrization of the electric and magnetic fields. The particles are represented by a `Particle` class, which contains the physical properties of the particle, as well as its current position and velocity. With this information, the `PenningTrap` class can calculate the forces acting on each particle, including the external fields and the particle-particle interactions. The particle-particle interactions can be toggled on and off, allowing for a simulation of both scenarios. The external fields can be modified by supplying a field-method of the form `external_field(const arma::vec& r, double t, const PenningTrap& trap)`. The reference to the `PenningTrap` allows for the parameters of the field to be stored in the trap object. The implementation of the `Particle` and `PenningTrap` classes can be found in `/src/project3/include/classes.hpp` of the project repository, as well as in the corresponding source file `/src/project3/src/classes.cpp`.

Numerical Methods Implementation

All numerical methods are implemented as classes inheriting from a base class `Solver`. The base class contains a reference to the `PenningTrap` object, as well as the time step size. The recording of particle properties over time, like position and velocity is part of the general `Solver` class. Each de-

[1] The simplecticity of Boris algorithm is controversial with disagreement between papers [8]. All publications agree on the fact, that Boris algorithm is phase-space preserving at the very least.

rived class implements the `step()` method, which updates the state of the system by one time step using the respective numerical method. The implementation of the `Solver` class and its derived classes can be found in `/src/project3/include/solvers.hpp` of the project repository, as well as in the corresponding source file `/src/project3/src/solvers.cpp`. The following solvers are implemented:`ForwardEulerSolver`, `RK4Solver`, `VelocityVerletSolver` and `AnalyticalSolver`, which implements the analytical solution for a special case of initial conditions (see appendix B).

Unless otherwise specified, the simulations are run with the following parameters:

- Trap parameters: $V_0 = 25.0\,\text{mV}$, $B_0 = 1.00\,\text{T}$, $d = 500\,\mu\text{m}$
- Particle parameters: $q = e$, $m = 40.078\,\text{u}$
- Simulation parameters: $T = n_{\text{steps}} \cdot h = 50\,\mu\text{s}$.

Tools

All simulations are written in C++ and compiled using `g++ (GCC) 15.2.1 20250808 (Red Hat 15.2.1-1)`. For optimization, the `-O3` flag is used. The code makes use of the Armadillo library[10, 11] for linear algebra operations and the argparse library[12] for command line argument parsing. To visualize the results and analyze the data, Python 3.13.7 with the libraries `numpy` [13], `matplotlib` [14] and `pandas` [15] is used. To create easy to use command line interfaces for the Python scripts, the `typer` library[16] is used.

As a large number of simulations need to be run for different parameters, a bash script is used to automate the process. The script can be found in `/src/project3/generate_results.sh` of the project repository. It uses the `pueue` task management tool[17] to run multiple simulations in parallel, making use of all available CPU cores. To automate the plot generation, the script `generate_plots.sh` is used, which can be found in the same directory.

To aid in the development process, the large language model GitHub Copilot[18] was used to generate code snippets and provide suggestions for code completion. Other large language models were not used in the writing of this report.

III. RESULTS AND DISCUSSION

Numerical Accuracy of Trajectories

As there is no closed form solution to the trajectories in a Penning trap containing multiple particles, which interact through Coulomb forces, the case of a single particle is used to verify the numerical accuracy of the implemented algorithms. As a metric for the accuracy, the relative error

$$\epsilon = \frac{|\vec{r}_{\text{exact}} - \vec{r}_{\text{numerical}}|}{|\vec{r}_{\text{exact}}|} \tag{33}$$

is used, where $\vec{r}_{\text{exact}}$ as derived in section II, and $\vec{r}_{\text{numerical}}$ is the numerical solution obtained from the implemented algorithms. The relative error is calculated at each time step, and the maximum value over the entire simulation is reported as the error for that particular simulation.

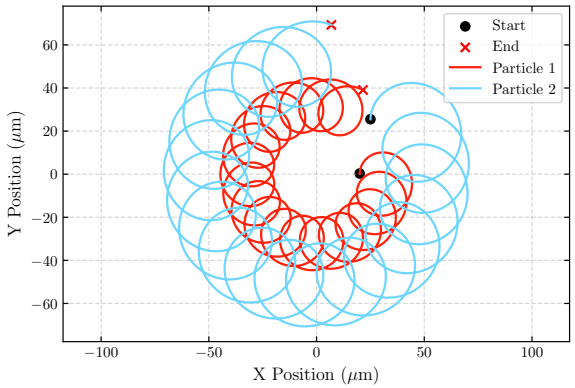

Figure 1. Trajectory of two particles in a Penning trap, simulated using the RK4 method. The particles are initialized as: $\vec{r}_1 = (20\,\mu\text{m}, 0\,\text{um}, 20\,\mu\text{m})$, $\vec{v}_1 = (0\,\mu\text{m}/\mu\text{s}, 25\,\mu\text{m}/\mu\text{s}, 0\,\mu\text{m}/\mu\text{s})$, $\vec{r}_2 = (25\,\mu\text{m}, 25\,\mu\text{m}, 0\,\mu\text{m})$, $\vec{v}_2 = (0\,\mu\text{m}/\mu\text{s}, 40\,\mu\text{m}/\mu\text{s}, 5\,\mu\text{m}/\mu\text{s})$.

Apart from the quantitative measure of the accuracy, the qualitative behavior of the trajectories is also examined. The numerical solutions are plotted and compared to the expected behavior. Especially properties like energy drift of the solution and the general shape of the trajectory can give insight into the performance of the algorithms.

The Runge-Kutta 4 method shows a very good qualitative behavior, with no apparent inaccuracies in the trajectory, as shown in fig. 1. Comparing against the same trajectories calculated using the Velocity-Verlet method, we can immediately see shortcomings of the latter. Trajectories for the Velocity-Verlet algorithm, the Boris algorithm and Euler algorithm are included in figs. 5 to 7 in appendix C. The trajectory shows a significant drift over time, which is not present in the RK4 solution. The Verlet solution rapidly increases its radius, which is not physical in a system with conserved energy. While one would expect the symplectic Verlet method to perform better in this regard, the velocity-dependant Lorentz force is not suited for computations using the Velocity-Verlet algorithm, as discussed in section II. A similar behavior of non-physical qualitative behavior is observed for the Euler method, which also shows a significant energy drift. The Boris algorithm, which is specifically designed to handle electromagnetic forces, shows a qualitatively correct behavior, with no apparent energy drift and a stable orbit over time.

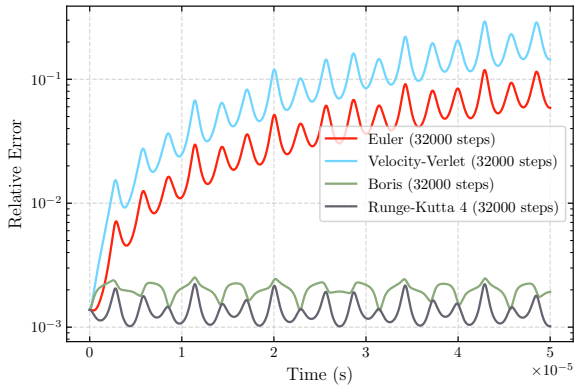

Figure 2. Relative error ϵ for a single particle in a Penning trap simulated for 32 000 time steps. The particle is initialized as: $\vec{r} = (20\,\mu\text{m}, 0\,\text{um}, 20\,\mu\text{m})$, $\vec{v} = (0\,\mu\text{m}/\mu\text{s}, 25\,\mu\text{m}/\mu\text{s}, 0\,\mu\text{m}/\mu\text{s})$.

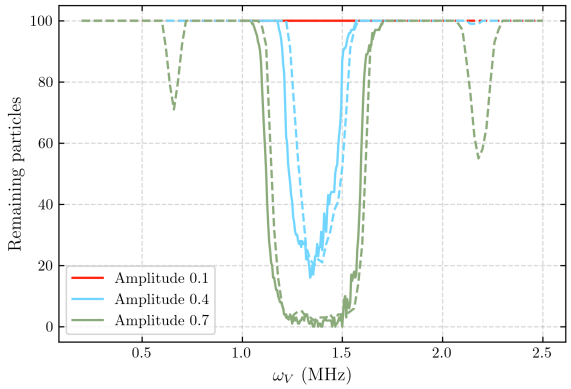

Figure 3. Fraction of particles remaining in the trap after 50 µs of simulation time using Boris algorithm, as a function of the frequency of the dynamic trapping potential $V_{\text{dyn.}}$ with different oscillatory amplitudes. The first scan (dashed) is performed without Coulomb interactions, while the second scan (solid) is performed with Coulomb interactions enabled.

As well as the trajectory in the transversal plane, the osciallatory behavior in the z direction and the phase space usage in all three dimensions is examined. As there is no influence of the magnetic field in the z direction, the oscillations should be perfectly harmonic. Furthermore should the Velocity-Verlet method exhibit correct behavior, as there is no velocity-dependant force in this direction. The very good performance of RK4, Boris and Velocity-Verlet in this regard is confirmed, while the Euler method shows energy drift and a non-harmonic behavior with increasing amplitude over time.

The numerical analysis of the trajectories shows similar behavior for the two algorithms designed to handle electromagnetic forces, RK4 and Boris. The evolution of the relative error ϵ as defined in eq. (33) over time is shown in fig. 2. Both algorithms show a steady oscillation in the relative error around 1×10^{-3} to 2×10^{-3}, with RK4 showing a slightly lower error overall. The Velocity-Verlet and Euler methods show a significantly higher error, with the Euler method reaching a relative error of up to 0.1 at certain times. The Velocity-Verlet method shows a relative error of up to 0.2.

Many-Body Simulations

In addition to the case of perfect trajectories without Coulomb interactions, the behavior of a cloud of up to 100 particles is examined. This use case is of particular interest, as it is closer to real-world applications of Penning traps, such as in mass spectrometry. The limit of increasing particle number is also of interest, as the computational complexity increases significantly with the number of particles, due to the expected $\mathcal{O}(N^2)$ scaling of the Coulomb interaction calculations. Additionally, the collective behavior of particles, i.e. the statistical properties of the ensemble, and the accuracy thereof is studied.

All particles are initialized with random positions and velocities, sampled from a normal distribution. The standard deviation of the position distribution is set to 50 µm in all three dimensions, while the standard deviation of the velocity distribution is set to 50 µm/µs. To compare the influence of Coulomb interactions on the simulation performance and statistical properties, simulations with and without interactions are performed.

Resonant Behavior

A further behavior that can be studied in the many-body simulations is the resonant behavior of the particle cloud when a dynamic trapping potential $V_{\text{dyn.}}$ is used. By varying the frequency of the dynamic potential ω_V, we can study how the stability of the trap is influenced by resonances. The stability is measured by the fraction of particles remaining in the trap after a fixed simulation time of 50 µs. A particle is considered to be inside the trap, if its distance from the center of the trap is less than d, where d is the characteristic dimension of the trap, as defined in section II. In a first scan of the frequency range from 0.2 MHz to 2.5 MHz, three resonances are found, at 0.7 MHz, 1.4 MHz and 2.1 MHz. The first scan is performed without Coulomb interactions between the particles, as to speed up the simulation. In a second scan, the resonance in the range of 1.0 MHz to 1.8 MHz is studied in more detail, with a finer frequency resolution and with Coulomb interactions enabled. The results of both scans are shown in fig. 3.

Table I. Relative run times of the different solvers in µm per particle and time step. The times are averaged over 4 runs with different time steps. Coulomb interactions are enabled and $V_{\text{stat.}}$ is used as the trapping potential. Output during the simulation is enabled.

Num. Particles Solver	10	50	100
Euler	1.45(30)	5.90(41)	$1.244(90) \times 10^1$
Velocity Verlet	1.46(17)	6.22(52)	$1.271(61) \times 10^1$
Runge-Kutta 4	3.71(65)	$1.57(13) \times 10^1$	$3.58(57) \times 10^1$
Boris	1.43(22)	5.95(51)	$1.263(60) \times 10^1$

Table II. Total run times of the different solvers in s for a simulation of 100 particles and 40 000 time steps. The dynamic potential $V_{\text{dyn.}}$ is used, and Coulomb interactions are both enabled and disabled. The output during the simulation is disabled to get a more accurate measure of the run time.

	Interactions Disabled	Interactions Enabled
Solver		
Velocity Verlet	1.23(26)	$3.14(16) \times 10^1$
Runge-Kutta 4	8.8(11)	$1.267(48) \times 10^2$
Boris	1.07(24)	$3.11(16) \times 10^1$

Performance and Efficiency

For different step sizes, i.e. a different number of total time steps, the total run time of a simulation with different numbers of particles is measured. For measuring the run time, the `ctime` module with the `std::clock()` function is used. All runs were performed on the same machine, with an AMD Ryzen 7 4750U CPU and 16 GB of RAM, running Fedora 42. For parallelization of the runs the scheduler `pueue` was used with 14 simultaneous jobs. To reduce the influence of other processes on the machine, the runs were performed at night when the machine was otherwise idle. Furthermore, the runs were interweaved in a way, that no solving algorithm could take dvantage from time dependant properties of the machine, such as thermal throttling. The output during the simulation was enabled, as the output is reused for the analysis of the statistical properties of the particle ensemble. The I/O operations were performed outside of the timed section of the code, to get a more accurate measure of the run time of the algorithms themselves. The times were averaged over 4 runs with different time steps, and the relative time per particle and time step was calculated. The results are shown in table I.

The results show a clear linear scaling of the relative time per particle and time step with increasing number of particles, as expected from the $\mathcal{O}(N^2)$ scaling of the Coulomb interaction calculations. The Euler, Velocity-Verlet and Boris methods show a very similar performance, with the RK4 method being slower by a factor of approximately 2.8. The similar performance of the Euler, Velocity-Verlet and Boris methods is expected, as all three methods require a single force calculation per time step. The RK4 method requires four force calculations per time step, which explains the significantly higher run time. From the scaling behavior for simulations with Coulomb interactions, we can derive that the major contributing factor to the run time is the calculation of the interactions.

For a more precise comparison on how the four different algorithms compare, with respect to run time, a simulation with 100 particles and 40 000 time steps is performed. There is no output recording during the simulation, as the simulations for the resonant behavior are reused. The lack of output recording allows for a more accurate measure of the run time of the algorithms themselves. The results are shown in table II. The Boris algorithm will serve as a baseline for the comparison, as it shows the lowest execution time of the three algorithms tested. The Velocity-Verlet algorithm shows a similar performance, with an increase in CPU time of 0.75(733) % with interactions and 14.90(3495) % without interactions. The RK4 method shows a significantly higher run time, with an increase of 307(26) % with interactions and 721(210) % without interactions. The stark difference in performance between the RK4 method and the other two methods is expected, as the RK4 method requires four force calculations per time step, while the other two methods only require a single force calculation per time step. The slight increase in run time of the Velocity-Verlet method compared to the Boris method is unexpected, but within the margin of error of the measurements. The error of each run time is estimated via the standard deviation of 348 runs without interactions and 483 runs with interactions. The error on the relative speed decrease is estimated via Gaussian error propagation assuming uncorrelated errors.

The high performance of the Boris method in comparison to RK4 is also an important factor to consider when assessing the numerical accuracy of the trajectories. While both algorithms performed similarly in terms of trajectory accuracy at the same step size, the decreased computation time for Boris algorithm allows for a significantly smaller step size at the same computation time. This in turn allows for a higher numerical accuracy of the trajectories, as the error generally decreases with decreasing step size.

Symplectic Properties

While a full analysis of the symplectic properties of the different algorithms is outside the scope of this project, a brief qualitative analysis is performed. The symplectic nature of an algorithm is especially important in long-term simulations, as non-symplectic algorithms may produce incorrect ensemble properties over long time scales, even if the short-term accuracy is high. The Velocity-Verlet algorithm is symplectic by design for Hamiltonian

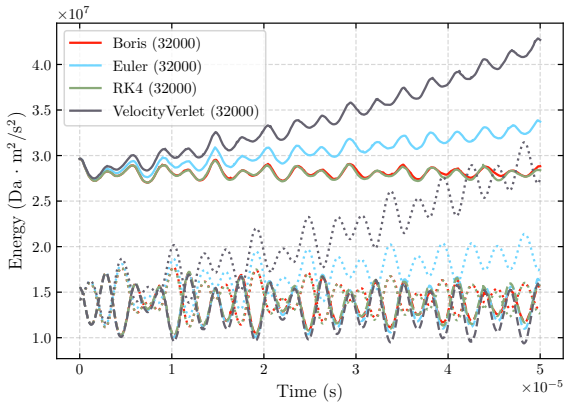

Figure 4. Total energy of a system of 100 particles in a Penning trap over 32 000 time steps, simulated using different algorithms. The kinetic component of the total energy is shown as a dotted line, while the potential component is shown as a dashed line. The solid line shows the total energy of the system.

systems, while the Boris algorithm is also known to exhibit symplectic properties in electromagnetic systems. The RK4 method and the Euler method are not symplectic.

A qualitative analysis of the symplectic properties is performed by examining the systems total energy

$$E = \sum_{i=1}^{N} \left(\frac{m_i v_i^2}{2} + q_i V(\vec{r}_i) + \frac{1}{2} \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{q_i q_j}{4\pi\epsilon_0 |\vec{r}_i - \vec{r}_j|} \right) \quad (34)$$

over the simulation duration. As the system is isolated, the total energy should be conserved. A non-conservation of the total energy indicates a non-symplectic behavior of the algorithm.

Figure 4 shows the total energy development of the system for the different algorithms. A more detailed analysis on the dependance on step size is shown in appendix C with figs. 8 to 11. The RK4 and Boris methods show a very stable total energy over the entire simulation duration, with slight oscillations around a mean value. The Velocity-Verlet method shows a significant drift to increasing total energy over time, indicating a non-symplectic behavior in this system. The Euler method shows a similar behavior, with a slightly less pronounced increase in total energy over time.

The non-symplectic behavior of the Velocity-Verlet method in this system is expected due to the velocity-dependant Lorentz force, as discussed in section II. The non-symplectic behavior of the Euler method is also expected, as it is not a symplectic algorithm. The good energy conservation of the RK4 method is noteworthy, as it is not a symplectic algorithm either. However, a conservation of total energy in a single simulation for a specific time span does not imply symplectic properties of the algorithm. In general RK4 is not considered a symplectic algorithm, and may show non-symplectic behavior in other systems or over longer time spans.

Boris method also shows a very good conservation of total energy, which is expected due to its design for electromagnetic systems and symplectic properties therein.

IV. CONCLUSION

While general purpose integrators like the Runge-Kutta methods can be applied to a wide range of problems, specialized algorithms like the Velocity-Verlet and Boris methods are tailored for specific types of systems, such as those encountered in molecular dynamics and charged particle dynamics, respectively. While Velocity-Verlet offers very good energy conservation properties for systems with position-dependent forces, it is not applicable for problems involving charged particles in magnetic fields, as the Lorentz force depends on velocity. The Boris algorithm, on the other hand, is specifically designed to handle the velocity-dependent nature of the magnetic force and is widely used in plasma physics and charged particle simulations. It delivers a similar numerical accuracy as the Runge-Kutta methods, while being phase-space preserving and requiring a fraction of the computational cost.

All algorithms including Forward Euler integration improve their accuracy with decreasing time step size, as the local truncation error per step is proportional to a power of the time step size h. However, due to limitations in computational resources, purpose built algorithms like Boris algorithm are needed to efficiently simulate complex systems over long time periods with sufficient accuracy. Thus, Boris algorithm is the preferred choice for simulating Penning traps.

If there is no knowledge about the system to be simulated, we have shown near perfect results for the RK4 method at a slightly increased computational cost. Furthermore, RK4 was energy conserving in our simulations. If phase-space conservation is not of uttermost importance, using Runge-Kutta methods is a good choice for general purpose simulations.

Appendix A: Electric Field Equations

Here we derive the electric field equations used in section II. The electric field following from $V_{\text{stat.}}$ defined in eq. (2) is given by

$$E_{\text{stat.},x} = -\frac{\partial V_{\text{stat.}}}{\partial x} = \frac{V_0}{d^2} x, \quad (A1)$$

$$E_{\text{stat.},y} = -\frac{\partial V_{\text{stat.}}}{\partial y} = \frac{V_0}{d^2} y, \quad (A2)$$

$$E_{\text{stat.},z} = -\frac{\partial V_{\text{stat.}}}{\partial z} = -\frac{2V_0}{d^2} z, \quad (A3)$$

for all $|\vec{r}| < d$ and zero otherwise. The electric field following from $V_{\text{dyn.}}$ is simply

$$E_{\text{dyn.}} = (1 + f\cos\omega_V t) \cdot E_{\text{stat.}}. \tag{A4}$$

Appendix B: Special Case Analytical Solution

In the special case of a single particle in the potential $V_{\text{stat.}}$, with the inital conditions $\vec{r}(t_0) = (x_0, 0, z_0)$ and $\vec{v}(t_0) = (0, v_0, 0)$, we can derive an analytical solution for the trajectory. The solution for the z component is straightforward

$$z(t) = z_0 \cos(\omega_z t). \tag{B1}$$

From the equation system

$$f(0) = A_+ e^{-i\phi_+} + A_- e^{-i\phi_-} \equiv x_0, \tag{B2}$$
$$\dot{f}(0) = -i\omega_+ A_+ e^{-i\phi_+} - i\omega_- A_- e^{-i\phi_-} \equiv iv_0, \tag{B3}$$

we derive $\phi_\pm = 0$ as $f(0) \in \mathbb{R}$ and therefore $A_+ + A_- = x_0$ and $\omega_+ A_+ + \omega_- A_- = -v_0$. Solving this system of equations for $A_\pm$ we find

$$A_\pm = \pm\frac{v_0 + \omega_\mp x_0}{\omega_- - \omega_+}. \tag{B4}$$

Appendix C: Further Results

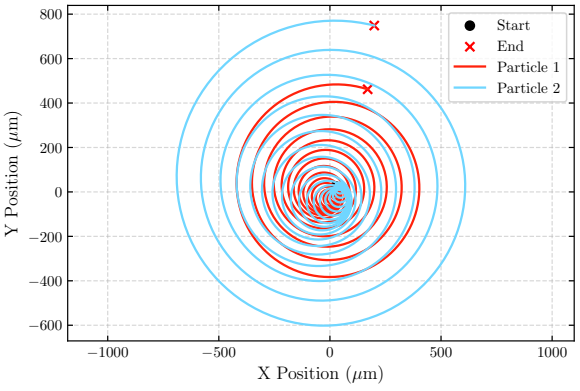

Figure 5. Trajectory of two particles in a Penning trap simulated using the Velocity-Verlet solver (4000 steps). The particles are initialized as $\vec{r}_1 = (20\,\mu\text{m}, 0\,\mu\text{m}, 20\,\mu\text{m})$, $\vec{v}_1 = (0\,\mu\text{m/}\mu\text{s}, 25\,\mu\text{m/}\mu\text{s}, 0\,\mu\text{m/}\mu\text{s})$, $\vec{r}_2 = (25\,\mu\text{m}, 25\,\mu\text{m}, 0\,\mu\text{m})$, $\vec{v}_2 = (0\,\mu\text{m/}\mu\text{s}, 40\,\mu\text{m/}\mu\text{s}, 5\,\mu\text{m/}\mu\text{s})$.

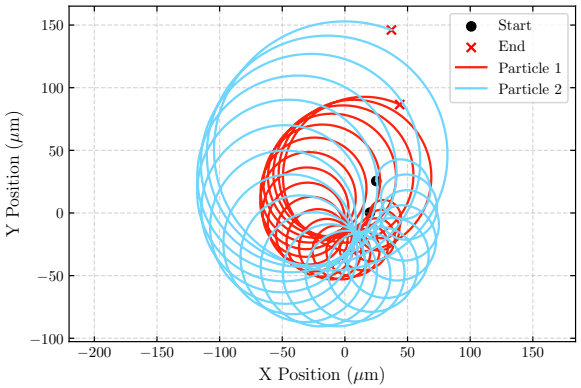

Figure 6. Trajectory of two particles in a Penning trap simulated using the Euler solver (4000 steps). Same initial conditions as in Fig. 5.

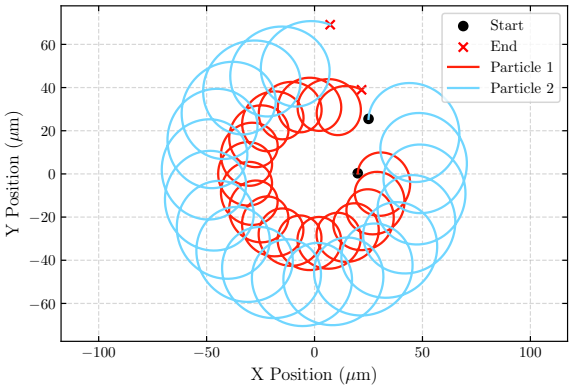

Figure 7. Trajectory of two particles in a Penning trap simulated using the Boris solver (4000 steps). Same initial conditions as in Fig. 5.

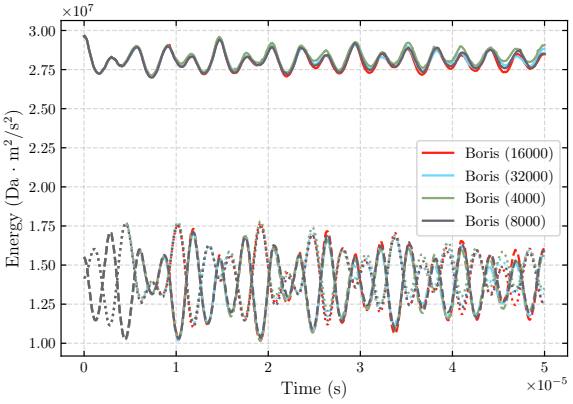

Figure 8. Total energy of 100 particles in a Penning trap simulated using the Boris solver with inter-particle interactions enabled for different time step sizes.

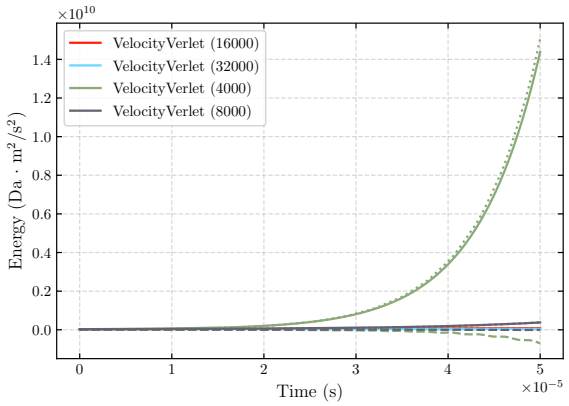

Figure 9. Total energy of 100 particles in a Penning trap simulated using the Velocity-Verlet solver with inter-particle interactions enabled for different time step sizes.

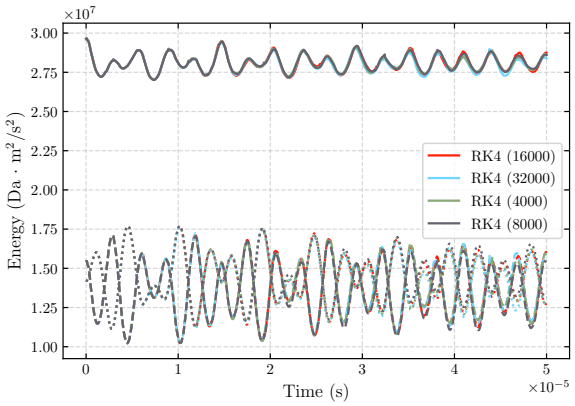

Figure 11. Total energy of 100 particles in a Penning trap simulated using the RK4 solver with inter-particle interactions enabled for different time step sizes.

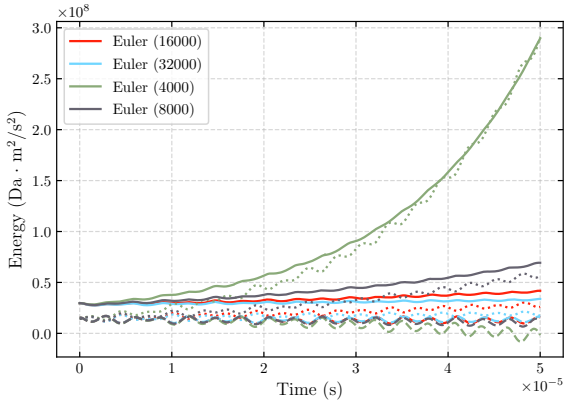

Figure 10. Total energy of 100 particles in a Penning trap simulated using the Euler solver with inter-particle interactions enabled for different time step sizes.

[1] Manuel Vogel. *Particle Confinement in Penning Traps: An Introduction*, volume 126 of *Springer Series on Atomic, Optical, and Plasma Physics*. Springer International Publishing.

[2] W. A. Bertsche. Prospects for comparison of matter and antimatter gravitation with ALPHA-g. 376(2116):20170265. Publisher: Royal Society.

[3] Paola Scampoli and James Storey. The AEgIS experiment at CERN for the measurement of antihydrogen gravity acceleration. 29(17):1430017. Publisher: World Scientific Publishing Co.

[4] Ulrik Skre Fjordholm. Numerical methods for ODEs.

[5] Erik Cheever. Fourth order runge-kutta.

[6] Tomas Kubar, Marcus Elstner, and Mariana Kozlowska. Lecture: Molecular dynamic simulations summer 2025.

[7] Stephen D. Webb. Symplectic integration of magnetic systems. 270:570–576.

[8] (PDF) why is boris algorithm so good?

[9] Ian Hoppock. iwhoppock/boris-algorithm. original-date: 2018-11-15T15:21:28Z.

[10] Conrad Sanderson and Ryan Curtin. Armadillo: An efficient framework for numerical linear algebra. In *2025 17th International Conference on Computer and Automation Engineering (ICCAE)*, pages 303–307.

[11] Conrad Sanderson and Ryan Curtin. Practical sparse matrices in c++ with hybrid storage and template-based expression optimisation. 24(3):70.

[12] Pranav. p-ranav/argparse. original-date: 2019-03-30T22:28:48Z.

[13] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. 585(7825):357–362. Publisher: Springer Science and Business Media LLC.
[14] J. D. Hunter. Matplotlib: A 2d graphics environment. 9(3):90–95. Publisher: IEEE COMPUTER SOC.
[15] The pandas development team. pandas-dev/pandas: Pandas.
[16] Sebastián Ramírez. Typer.
[17] Arne Christian Beer. Nukesor/pueue. original-date: 2015-09-04T16:24:23Z.
[18] GitHub copilot · your AI pair programmer.