Sympathetic Cooling Optimization for Precision Spectroscopy of Chiral Molecular Ions

Doron Behar

Sympathetic Cooling Optimization for Precision Spectroscopy of Chiral Molecular Ions

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Physics

Doron Behar

Submitted to the Senate
of the Technion — Israel Institute of Technology
Av 5785 Haifa July 2025

This research was carried out under the supervision of Prof. Yuval Shagam and Yotam Soreq, in the Phyiscs Department.

Some results in this thesis have been published as articles by the author and research collaborators in conferences and journals during the course of the author's research. The most up-to-date versions of these publications are:

Doron Behar. Sympathetic cooling optimization for chiral molecular ions precision spectroscopy. Poster Presented in CCMI 2024 conference, September 2024.

The research underlying this thesis was conducted honestly, in accordance with the ethical standards customary in academia. This applies, in particular, to the collection, processing and presentation of data, the description of and comparison with previous research work, etc. Also, the report on research activities and findings in this work is thorough and honest in accordance with the aforementioned standards.

The generous financial help of the Technion is gratefully acknowledged.

Contents

List of Figures

Abstract

The standard model predicts the weak force causes variations between the spectra of left and right-handed chiral molecules. Our group's goal is to measure this shift, also termed parity violation (PV) which has never been measured in molecules. This PV shift in IR vibrational excitations is predicted to be as high as a few Hz at best, thus measuring it requires cold chiral molecules. Molecular ions were chosen because they can be easily trapped and cooled for long time periods.

Cooling chiral molecules directly is challenging, so we co-trap and directly cool simpler ions (Yb^+), that cool the molecule via collisions – a process known as sympathetic cooling. The proposed PV detection scheme relies on photo-dissociation of the molecule, so performing the measurement repeatedly requires reloading new cold molecules every time. The goal of this research is to optimize the cooling rate with respect to the coherence time, to reduce the dead time in each repetition.

We discuss the results of trapping & cooling processes from many body simulations that include Coulomb interactions. One propensity rule we find, is that higher secular frequencies have proven to increase the cooling rate – something that can be done experimentally as part of the proposed experimental setup.

Chapter 1

Introduction

1.1 Motivation

The word chirality is derived from the Greek word for 'hand'. We define an object as chiral if it is distinguishable from its mirror image, with the classic example of our hands. Alternatively, the two mirror images of a chiral object, usually referred to as the left and right hands of the object, cannot be superimposed. This also means they cannot be related by rotational and/or translational transformations alone. The first documented observation of chirality in molecules was of L. Pasteur in 1848, where he examined salt crystals and found they appear in two mirrored structures [VC21], called *enantiomers*.

Chirality is a fundamental property of many molecules in nature. Despite the structural symmetry allowed by chemistry, nature consistently uses only one enantiomer in living organisms in many biologically relevant molecules. Some researchers suggest that Parity Violation (PV), which is part of the Standard Model of Particle Physics can explain this phenomenon [Qua02]. We aim to measure the energy difference between opposite enantiomers' vibrations, denoted Δ_{PV}.

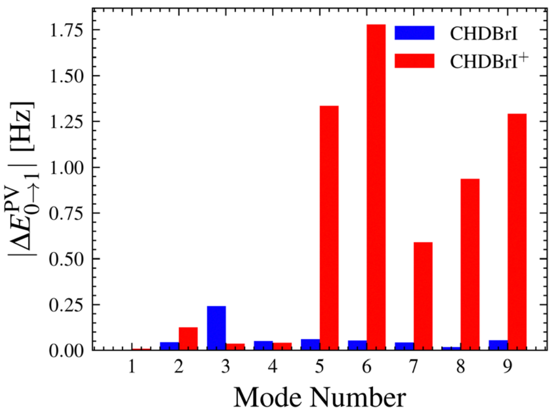
1.1.1 Why Chiral Molecular Ions?

Calculations predict that for many chiral molecules, $\Delta_{\text{PV}} \ll 1$ Hz [QS01], while the relevant vibrational mode frequencies lie in the mid- to far-IR regime - $16 - 18$ orders of magnitude larger. Measuring such a tiny energy difference is challenging. However, using chiral molecular ions offers two advantages: (1) long trapping durations allow for extended coherence times, enhancing experimental precision, and (2) the parity-violating energy shift Δ_{PV} is often larger in ions than in their neutral counterparts. Additionally, cooling the molecular ions can reduce thermal Doppler broadening and further extend coherence times.

Calculations predict a maximal $\Delta_{\text{PV}} \approx 1.8$ Hz [ESL+23] for CHDBrI^+, which is significantly larger than the estimated $\Delta_{\text{PV}} \approx 0.25$ Hz in the neutral species [QS01], as illustrated in Figure 1.1.

1	CBrI scissors
2	Cl stretch
3	CBr stretch
4	CHD rock
5	CD wag
6	CH wag
7	CHD scissors
8	CD stretch
9	CH stretch

(a) Names & indices of the vibrational modes.



(b) Δ_{PV} per vibrational mode.

Figure 1.1: Overview of the absolute values of Δ_{PV} between the two enantiomers of CHDBrI and CHDBrI$^+$, for all 9 fundamental vibrational transitions [ESL+23], where $\nu_6 \approx 33.3$ THz.

1.1.2 How to Cool a Molecular Ion?

Direct laser cooling of molecules is challenging due to their complex internal structure, including many rotational and vibrational modes that prevent the formation of closed cycling transitions. While a few specially designed molecules have been successfully laser cooled in recent years [SBD10; VHA+22], more complex species such as CHDBrI$^+$ - our target molecule, will be too challenging for laser cool.

The solution we went with is to use sympathetic cooling - a technique where two different ion species are co-trapped, and only one species is directly laser cooled [WBGS08]. The Laser-Cooled (LC) ions absorb kinetic energy from the Sympathetically Cooled (SC) ions through Coulomb collisions, thereby cooling their external degrees of freedom indirectly. For a comparative sympathetic cooling method, see Hudson's review of sympathetic cooling of molecular ions using laser cooled *neutral* atoms [Hud16].

For efficient sympathetic cooling there are two criteria that the laser cooled ions must meet: (1) They need to have a cycling transition that can be used for laser cooling, and (2) their mass needs to be close to the mass of the sympathetically cooled ions [BW02a]. We chose Yb$^+$, with a mass of $m_{LC} = 174$amu, where CHDBrI$^+$'s mass is $m_{SC} = 222$amu. Other ions we considered that are known to have been laser cooled, are Ca$^+$ & Be$^+$, but Yb$^+$'s mass is the closest to CHDBrI$^+$'s mass.

1.1.3 Why Cool the Molecular Ions Fast?

Δ_{PV} is planned to be measured using Ramsey spectroscopy due to numerous advantages [LEB+23; ESL+23; Ere23]. However with molecular ions, we cannot use fluorescence for state detection due to too low ion number, so the plan is to use instead photo-dissociation, as depicted in figure 1.2. The important implication of using photo-dissociation state detection after every Ramsey sequence, is that we need to reload new

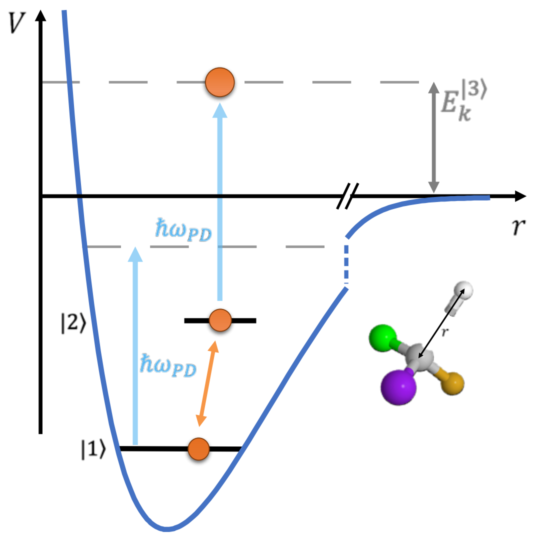


Figure 1.2: Illustration of a molecule's dissociation, done with a photon of energy $\hbar\omega_{PD}$. If the photon's energy was tuned right, the molecule will be dissociated, depending on the energy level it was in.

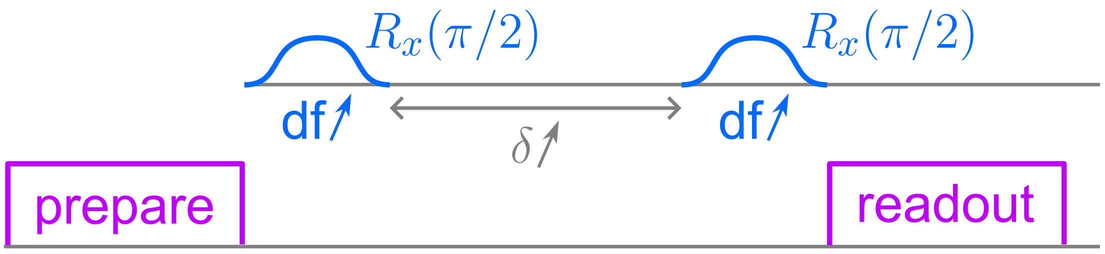


Figure 1.3: Schematic depiction of a Ramsey sequence along with preparation and readout. Preparation includes molecule synthesis and cooling.

molecules before every Ramsey sequence. The preparation, which includes reloading the molecules and cooling them, and the Ramsey sequence are depicted schematically in figure 1.3.

The uncertainty in a frequency measurement of a Ramsey sequence scales as $1/\left(\tau\sqrt{N}\right)$, where τ is the coherence time, and N is the amount of measurements [PSHL25]. Increasing τ (by cooling) increases the amount of Ramsey oscillations after which we focus the measurement efforts. Cooling fast however, means that the preparation time is shorter, allowing us to also increase N, and decrease the uncertainty even further.

1.1.4 Ion Trapping

The basic challenge of trapping charged particles is to overcome the limitation of the Laplace equation:

$$\nabla^2\phi = 0 \tag{1.1}$$

Meaning that we cannot get a point in space in which the potential is at a minima in all spatial dimensions. We can artificially overcome this limitation by imposing a force resembling a saddle-point, and oscillate the sign of it. If the oscillation is fast enough, a mass in the center will not have enough time to escape the origin in the direction of the maxima, before it will become a minima. [Gho95]

The saddle-point oscillation frequency is usually in the RF regime, and is denoted $\omega_{\text{rf}} \equiv 2\pi f_{\text{rf}}$ [Gho95]. On time scales much longer then the RF period, the ion's motion under such an oscillating potential, is of lower frequency denoted $\omega_{\text{sec}} = 2\pi f_{\text{sec}}$.

Typical ion traps induce secular frequencies of ~ 100 kHz, with RF frequencies of at least a few MHz. However, the ion trap used in our lab, is uniquely designed [WSoT24] to apply a *range* of secular frequencies of usually less then 10 kHz, with typical RF frequencies of 100 kHz or 50 kHz. This design allows to trap more ions and enhance the fluorescence signal, and to manipulate the ions' positions and kick them to our detector, as explained in the next section. This flexibility in frequencies invites researching the effect of different secular frequencies over the cooling efficiency.

1.1.5 The Experiment's Velocity Map Imaging

Another motivating point for cooling the ions in general, is our detection scheme based upon a Velocity Map Imaging (VMI) system optimized [WSoT24] for our research. The basic idea depicted in figure 1.4 is to shutdown the ion trap's electric fields after the photo-dissociation is done, and then focus the fragments to a detector of charged particles with spatial resolution. The key feature is that different kinetic energies are mapped to different points in the detector, allowing you to infer the kinetic energy distribution of the particles you had before releasing them. This mapping has limited resolution and was optimized as much as possible [WSoT24].

1.2 Simulations Overview

Since the main mechanism of sympathetic cooling is Coulomb interactions, we chose to simulate trapping and cooling with LAMMPS[TAB+22]. It is a time propagation software package, capable of utilizing the GPU for simulating numerous classical models relevant in many contexts. In our case, we selected a simple model of charged point particles with only long-range Coulomb interactions.

Given N particles and C available GPU cores, the amount of Coulomb interactions LAMMPS is capable of computing in a single step is $N(N-1)/(2C)$, since each coulomb interaction can be computed by a single GPU core. With 1000 ions, and our GPU, we found that the bottleneck of such simulations is writing the results to disk, and not computing the Coulomb interactions. More technical details on how we save results to disk throughout a simulation are available in chapter B.

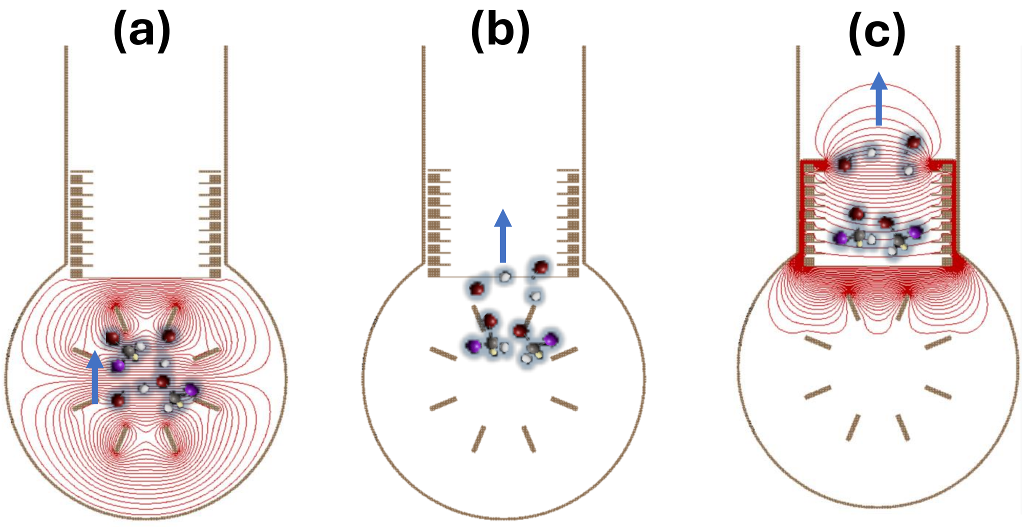


Figure 1.4: Our trapping and detection scheme using a VMI. In stage (a) the ions are kicked from the trap to the VMI; In stage (b) they pass freely to the electrostatic lens; In stage (c) they pass through the electrostatic lens focusing them onto a charge sensor located at the end of the apparatus.

Chapter 2

Theory

This chapter will iterate the theoretical background necessary for understanding how we trap (section 2.1) and cool (section 2.2) our ions, and how we analyzed the simulation results. With respect to trapping, I want to highlight two points, which are (1) inverse solution of the Mathieu equation (section 2.1.2) and (2) non-trivial dependence of mass to secular frequencies (section 2.1.4) which is particularly relevant for thermodynamically stable initial conditions.

2.1 Trapping

Realized ion traps are implemented using electrodes designed in a specific geometry, attached to oscillating electric potentials of specific forms. In literature, one often finds analytical expressions for electric potentials given a certain geometry and voltages [Ake12]. These expressions are highly tied to the parameters of the geometry, and cannot be easily generalized for our trap design. Our trap can be analyzed pretty easily in SIMION[Dah00], but it is not capable of including also ion-ion Coulomb interactions in simulations [SRKK12].

LAMMPS[TAB+22], is not designed for solving the Laplace equation (1.1) given a design of electrodes and voltages. It is great however for computing ion-ion interactions, within electric fields given as analytical expressions. Therefore we chose to focus our efforts on producing analytical expressions with abstract parameters, that trap our ions in desired secular frequencies. In contrast to the common expression found in the literature for an ion trap's electric field, we chose to use for the following expression for the i^{th} spatial dimension:

$$E_i = (a_i + 2q_i \cos(\omega_{\text{rf}} t))x_i \tag{2.1}$$

The dimensions of the *abstract* parameters a_i & q_i are naturally Volt/distance^2. The 1D equation of motion you get for a single ion under this field is:

$$(\tilde{a}_i + 2\tilde{q}_i \cos(2\tau))x_i = \partial_\tau^2 x_i \tag{2.2}$$

Where $2\tau \equiv \omega_{\text{rf}} t$, and:

$$a_i/\tilde{a}_i = q_i/\tilde{q}_i = m\omega_{\text{rf}}^2/e \tag{2.3}$$

with e the charge of the ion. This is called the Mathieu equation [Mat36], and I'll present here how I solved it, in a way that is more tailored for semi-numerical calculations, without involving physical parameters like the trap's geometry and voltages.

2.1.1 Solving the Mathieu Equation in 1D

The solution used in this work is obtained by substituting the following expression to equation 2.2.

$$x(\tau) = e^{i\beta\tau} \sum_{n=-\infty}^{\infty} \gamma_n e^{in\tau} + e^{-i\beta\tau} \sum_{n=-\infty}^{\infty} \gamma_n e^{-in\tau} \tag{2.4}$$

This expression is based upon the parametrization found in literature [Gho95; Dan20], but with n substituted by $n/2$. This parametrization is cleaner and hence more natural for programming. Putting it in the equation yields the infinite set of equations:

$$\gamma_{n+1} + \gamma_{n-1} = \frac{\tilde{a}_i - (\beta_i + n)^2}{\tilde{q}_i} \gamma_n, \quad \forall i \in \{x, y, z\},\ \forall n \in \mathcal{Z} \tag{2.5}$$

Where $n \in (-\infty, \infty)$ and $\beta = 2\omega_{\text{sec}}/\omega_{\text{rf}}$. The main point important to understand, is that we need to find a non-trivial solution, meaning a set γ_n coefficients that not all are 0. The best way to view the equations of all coefficients is via a matrix:

$$\begin{bmatrix} \ddots & \ddots & \ddots & & & & \vdots & \vdots & \\ \ddots & -D_{-N} & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots \\ \ddots & 1 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & \cdots \\ & 0 & \cdots & -D_{-1} & 1 & 0 & \cdots & 0 & \\ & \vdots & \cdots & 1 & -D_0 & 1 & \cdots & \vdots & \\ & 0 & \cdots & 0 & 1 & -D_1 & \cdots & 0 & \\ \cdots & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 1 & \ddots \\ \cdots & 0 & 0 & \cdots & 0 & 1 & -D_N & \ddots & \ddots \\ & \vdots & \vdots & & & \ddots & \ddots & \ddots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ \gamma_{-N} \\ \vdots \\ \gamma_{-1} \\ \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_N \\ \vdots \end{bmatrix} \overset{!}{=} \begin{bmatrix} \vdots \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \end{bmatrix} \tag{2.6}$$

Where similarly to 2.5:

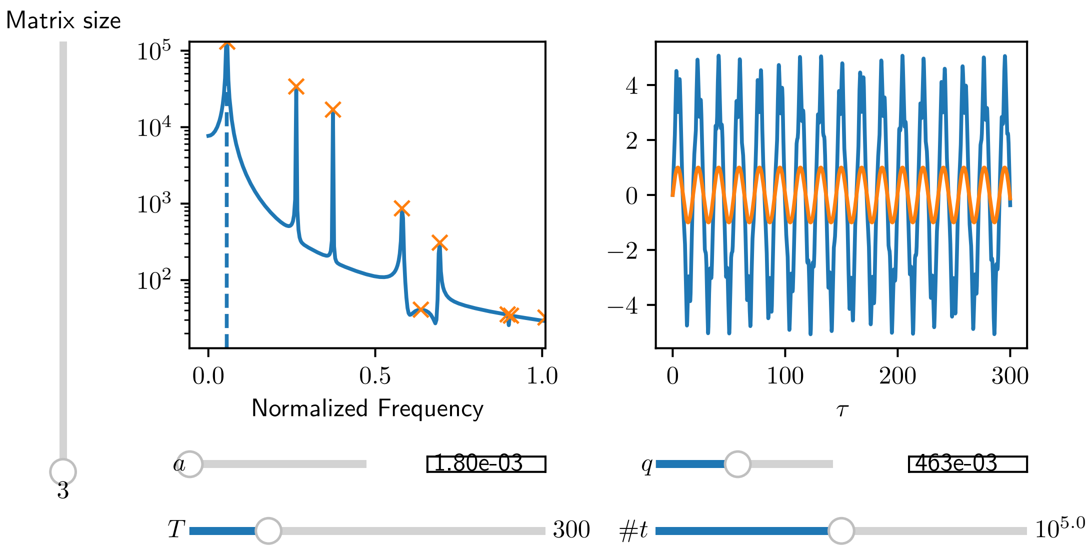


Figure 2.1: Matplotlib interactive plot presenting the 1 dimensional solution of the Mathieu equation, with a spectrum analysis demonstrating the correct matrix based solution of β in the dashed line. The orange sinusoidal line in the right axes is a simple cosine line with the right phase and the calculated β as a frequency. The T and $\#t$ sliders control the total time duration of the numerical ODE solution and the amount of time points, respectively.

$$D_n = \frac{a - (\beta + n)^2}{q}$$

One can easily see that D_n increases $\sim n^2$, which promises the target β solution might converge if a finite matrix will be used. To require a non-trivial solution, we simply require the matrix to have a 0 determinant, thus supplying us an equation for β that can be solved easily numerically, for matrix size $2N + 1$.

Inserting the β into the matrix, and inspecting the kernel of it, will give us the γ_n coefficients of the solution, and we'd be able to easily verify they indeed decrease with $|n|$, and that γ_0 is the most dominant.

The verification of this solution was implemented in a Matplotlib [Hun07] based interactive plot, presented in figures 2.2 & 2.1. A common approximation for a solution is:

$$\beta_i \approx \sqrt{\tilde{a}_i + \tilde{q}_i^2/2} \tag{2.7}$$

But it wasn't used in these plots at all. In the next section however, this approximation *was* helpful.

Interactive plot 2.1 can be also useful for inspecting the roughness of the motion as a function of a and q. As expected, a high q and a lower a produce more 'RF-noisy' motion near the peaks of the oscillatory, secular motion. Also, satisfyingly, there's barely any difference in the β accuracy as N is increased, even when starting from

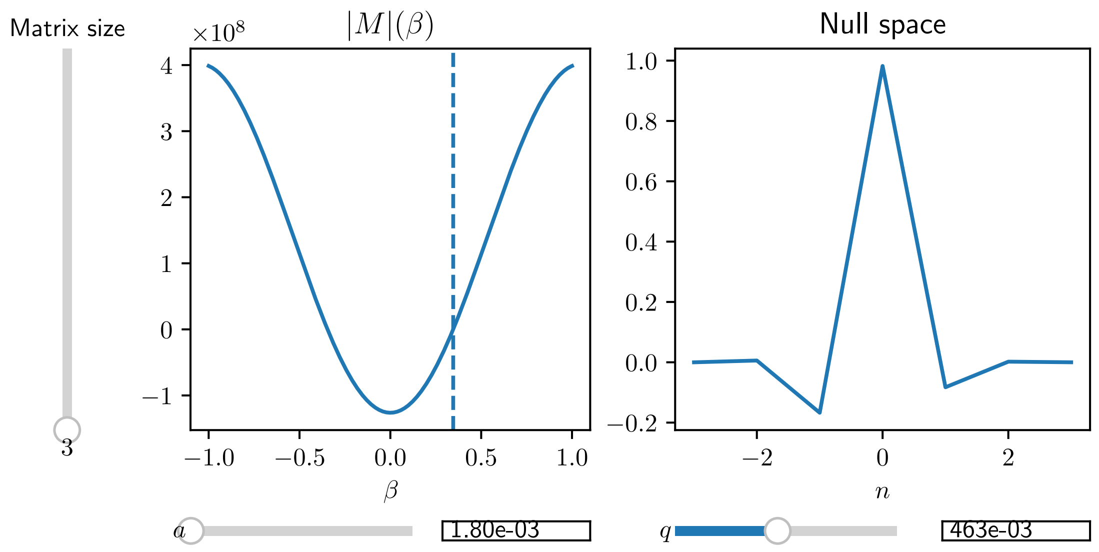


Figure 2.2: Matplotlib interactive plot demonstrating the fast decay of the Mathieu equation's solution's γ_n (normalized) coefficients, as $|n$ increases. The dashed β line represents the β which zeros the matrix determinant.

$N = 3$. Hence, this parameter was practically hard-coded in all the simulations etc. to $N = 7$.

2.1.2 Solving Mathieu Equation Inversely

Now that we know how to solve the Mathieu equation given (a_i, q_i), and get a secular frequency β_i, we wish to be able to find a set of 3 (a_i, q_i) pairs per spatial dimension, that will produce a requested set of secular frequencies. The (a_i, q_i) must satisfy the Laplace equation, meaning:

$$\sum_{i\in x,y,z} a_i = \sum_{i\in x,y,z} q_i = 0 \tag{2.8}$$

So we have now 5 equations, and 6 unknowns. A single solution to this system of equations can be obtained by adding a 6th equation, based on our ion trap geometry: Using SIMION [Dah00] simulations of our trap geometry, we found that the RF potential along the z-axis is approximately 2 orders of magnitude weaker than in the radial directions. Therefore, we demanded also:

$$q_z = 0 \tag{2.9}$$

The approximation in equation 2.7 for all 3 axes, along with equations 2.8 and 2.9, can be used to compute analytically an *approximated* expression of three $\{\tilde{a}_i, \tilde{q}_i\}$ pairs, in terms of all three β_i:

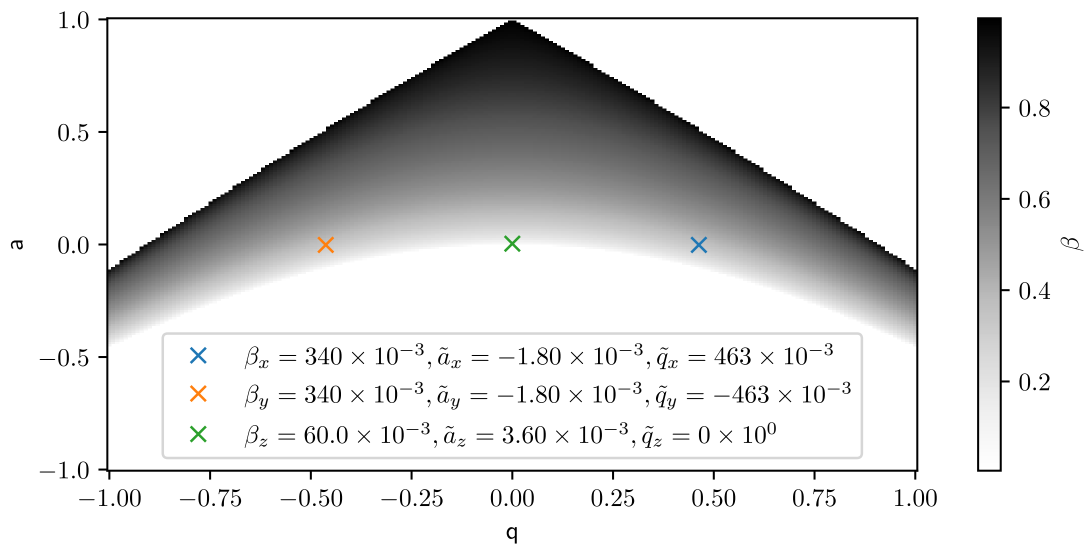


Figure 2.3: Mathieu Stability diagram, with shades of gray marking the magnitude of β. The $\times$ signs are specific β values, that were used in many simulations of this work - βs corresponding to secular frequencies $(8.5, 8.5, 1.5)$kHz, with $f_{\text{rf}} = 50$kHz.

$$\begin{pmatrix} \tilde{a}_x & \tilde{a}_y & \tilde{a}_z \\ \tilde{q}_x & \tilde{q}_y & \tilde{q}_z \end{pmatrix} \approx \begin{pmatrix} \frac{1}{2}\left(+\beta_x^2 - \beta_y^2 - \beta_z^2\right) & \frac{1}{2}\left(-\beta_x^2 + \beta_y^2 - \beta_z^2\right) & +\beta_z^2 \\ \sqrt{\beta_x^2 + \beta_y^2 + \beta_z^2} & -\sqrt{\beta_x^2 + \beta_y^2 + \beta_z^2} & 0 \end{pmatrix} \tag{2.10}$$

These expressions were used in the software as an initial guess for the numerical optimizer.

The best way I found to illustrate the exact solution to this system of equations, is via a Mathieu stability diagram. Most Mathieu stability diagrams found in literature, assume that geometry dictates $a_x = a_y$, and like us, that $q_z = 0$[BW02a; SKH+16; ZOR+07a]. Thanks to equation 2.8, this suggests that one can plot a single Mathieu stability diagram for a 3 dimensional trap using a single Mathieu stability diagram for (a_r, q_r) where $a_r \equiv a_x = a_y = -a_z/2$ and $q_r = q_x = -q_y$ as axes. These stability diagrams [Ake12; JPYM92; Gho95] need to take care of correctly mirroring $q_x = -q_y$ to satisfy stability in both x & y spatial dimensions (where the z axis is trivially stable). Figure 2.3 presents a different approach to show the stability of a 3 dimensional trap, without even imposing cylindrical symmetry which shouldn't be physically impossible.

Now that we know how to implement an ion trap with any secular frequencies we want, we need to discuss how to initiate the simulations in a thermodynamically stable state. Distributing the velocities is trivial with a Maxwell-Boltzmann distribution, but distributing the positions is harder. The next two sections describe implications relevant to the initial positions.

2.1.3 Initial Conditions & Ion-Ion Coulomb Energy

The questions we wish to answer in this section are: How significant is the ion-ion Coulomb energy (ICE) in the Boltzmann ensemble? Should it be considered when initiating the positions and velocities of the ions? If so, how? To assess the ICE in the temperature scale, we divided it by N - the amount of particles, and denoted it as E_c/N. This is essentially an assessment of the mean-field ICE energy.

Under the secular approximation, and when ignoring the ICE, the Maxwell-Boltzmann partition function dictates the probability of finding an ion in a position $\mathbf{x}$ to be:

$$\prod_{i=1}^{3} \sqrt{\frac{k_i}{2\pi K_{\mathrm{B}} T}} \exp\left(\frac{-k_i x_i^2}{2K_{\mathrm{B}}T}\right) \tag{2.11}$$

Where k_i is the spring-like coefficients of the harmonic trap. Since the real temperature is affected by the ICE, in the next usages if this expression we will not use the symbol T, but rather we'll use τ.

This distribution couples naturally density and temperature. More precisely: E_c/N is a function N, and the distribution parameters of expression 2.11 - $\{k_i\}$ and τ. The relation of E_c/N to $\{k_i\}$, τ and N is not analytical of course, yet we still expect a smooth behavior if we'd average over enough randomness. In figure 2.4 we can see that (expectedly), dense and cold ion clouds produce a higher ICE, that even exceeds τ.[1]

To produce this color-mesh, we distributed the ions in space multiple times per (τ, N) point, until the relative standard deviation of all E_c/N results in that point was lower then 0.12 (arbitrary number), with a minimum of 5 distributions (also termed 'iterations'). Statistical results of figure 2.4 is depicted in figures A.2 and A.3.

Naturally, the value of the highest ICE found escalates with the trap's tightness, as can be seen in figure 2.5.

Summary

The ICE is not negligible for dense & cold distributions. Our ability to initiate a distribution of ions in a thermodynamically stable set of positions is limited by the effects of strong ICE, and by the tightness of our trap. To avoid too strong ICE effects, we thereby avoid distributing ions in $T < 5\mathrm{K}$.[2]

More importantly, we expect time dependent temperature measurements to be offset from the initial temperature T_i on the order of magnitude of E_c/N. Since we also wish to start cooling when the system is thermodynamically stable, the ICE offset requires us

[1]Figure A.1 shows the same results normalized to τ.

[2]During the research period I tried to construct an algorithm that would get a real temperature $T \neq \tau$, N and a set of secular frequencies, and compute a τ with which the system will be more thermodynamically stable initially. When this algorithm was used for a single ion species it was slightly effective. However when 2 ion species were used, (with different masses and different $\{k_i\}$ - see section 2.1.2), accommodating the algorithm for that case has proved to be too complex, and too slow.

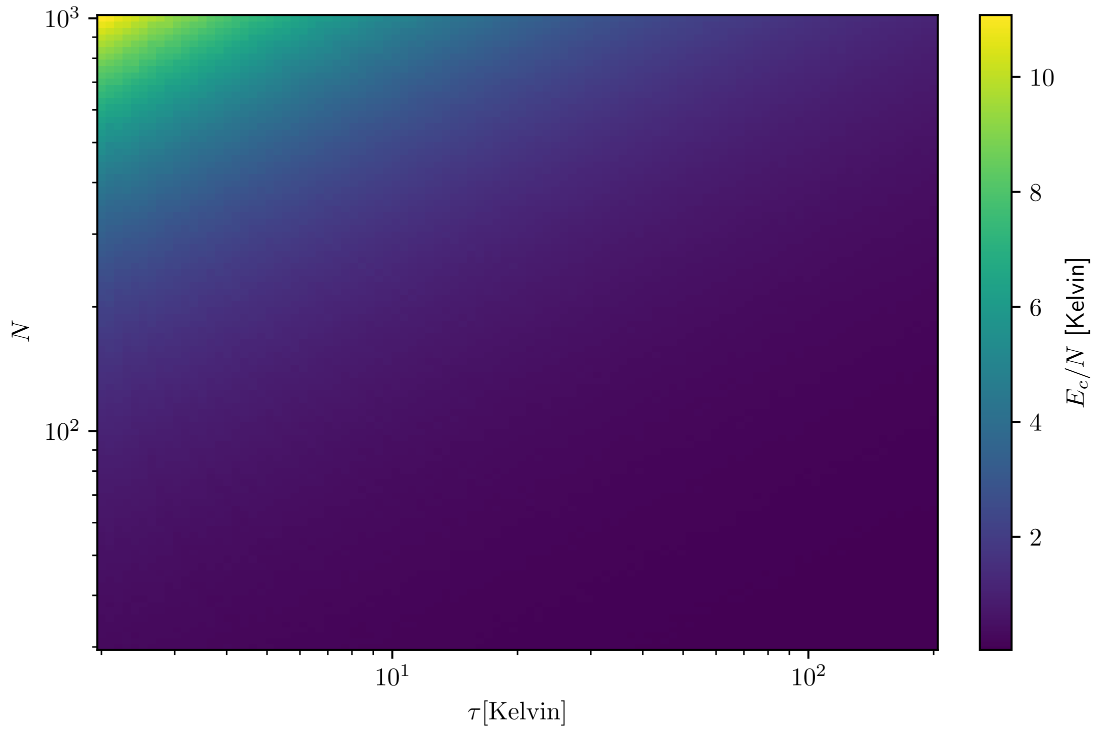


Figure 2.4: Dependence of E_c/N – the ion-ion Coulomb energy per particle, upon N and τ for a trap of Yb$^+$ (in varying amounts), with secular frequencies of $(8.5, 8.5, 1.5)$kHz, in the range of $\tau \in [2, 200]$K.

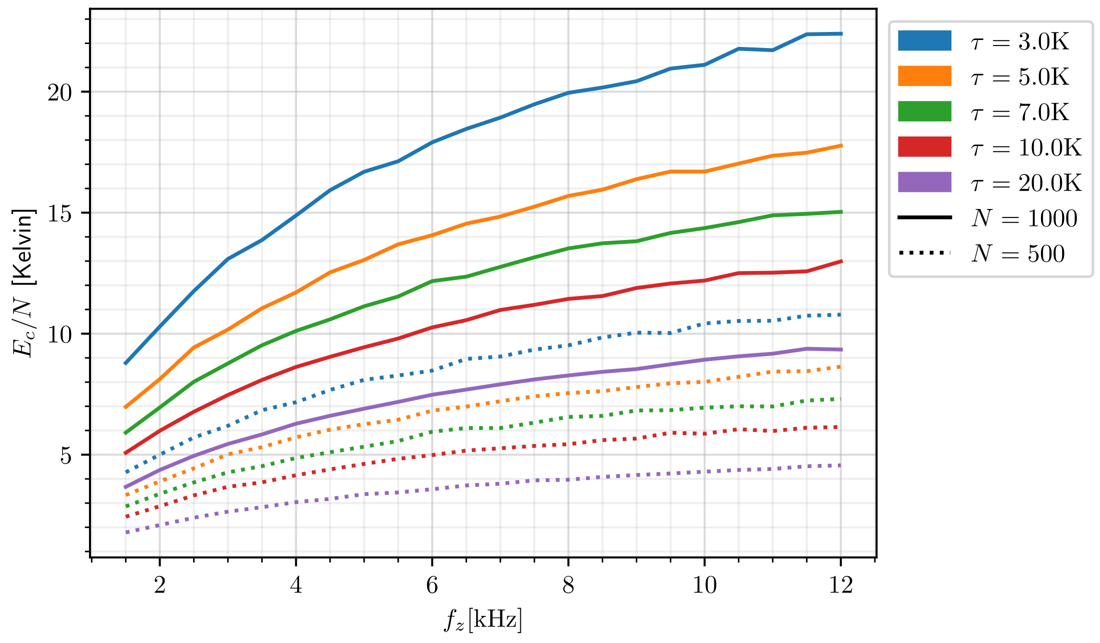


Figure 2.5: Dependence of E_c/N – the ion-ion Coulomb energy per particle, upon the axial secular frequency f_z, calculated from randomly sampled spatial distributions for a few specific (τ, N) pairs, with radial secular frequencies $f_x = f_y = 8.5$kHz.

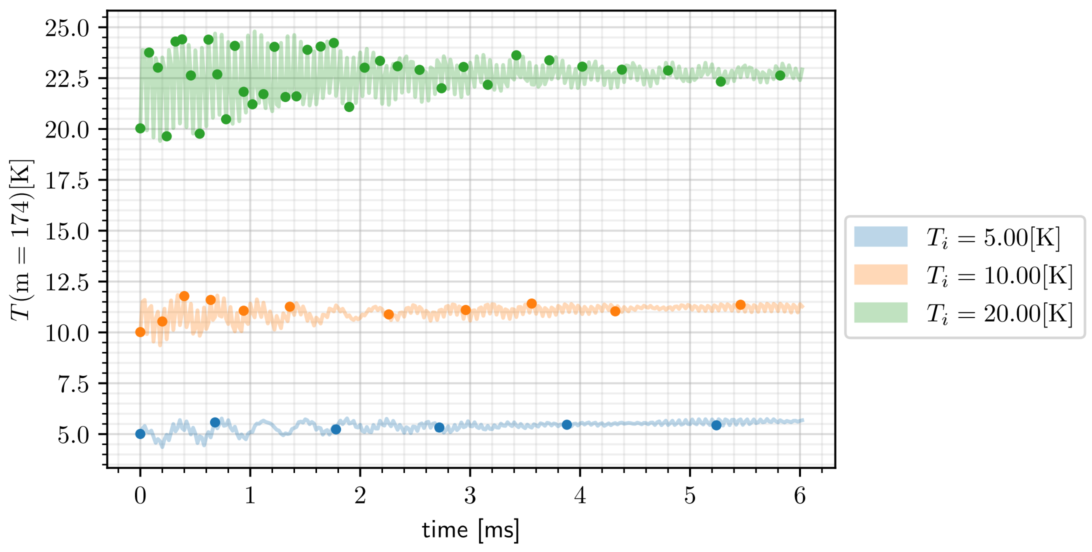


Figure 2.6: Simulations showing the required time for Yb^+ to reach equilibrium for several distributions in 3 different initial temperatures. Temperatures were calculated by averaging over the variances of the v_x, v_y, and v_z velocity distributions (see section 2.5 for additional temperature calculation methods). RF heating is observed for $T_{\text{initial}} = 5$ K.

to wait for a few ms, before beginning cooling, as can also be seen from the oscillations in figure 2.6.

2.1.4 A Non-Trivial Secular Frequency Mass Dependence

In the literature, analysis of two different masses in an ion trap commonly employs the pseudo-potential approximation [WAMS12a; BW02b; KKM+00], also called the pondermotive potential [Hom13], which yields an RF-averaged, perfectly harmonic potential. This approximation is widely used and sufficiently accurate for many applications, particularly in quantum computing implementations where the number of trapped ions is small. In such a potential, given a:

$$k = m_{cooler}\omega^2_{\text{sec(cooler)}} \tag{2.12}$$

Applied to any mass, in a certain spatial dimension, one obtains a secular frequency for m_{target} of:

$$\omega_{\text{sec(target)}} = \omega_{\text{sec(cooler)}}\sqrt{\frac{m_{cooler}}{m_{target}}} \tag{2.13}$$

For our simulations, we solve the Mathieu equations directly without the pseudo-potential approximation to determine the secular frequencies experienced by the target mass. This approach is particularly relevant when initiating simulations in thermodynamic equilibrium according to equation 2.11.

`mm`	$\sqrt{m_r}$	**Physical**	**Description**
✓	×	✓	A single electric field exhibiting 3 dimensional Mathieu equations - The only physically possible kind of trap, with non-trivial secular frequencies as presented in figure 2.7.
×	✓	×	A single $k\mathbf{x}$-like force is acting upon all ions - implementing the naive frequencies you'd expect from a perfect Harmonic oscillator.
✓	✓	×	Two different Mathieu forces are applied to each mass, such that the naive frequencies related by $\sqrt{m_r} \equiv \sqrt{m_{cooler}/m_{target}}$ are experienced by the two ion species.
×	×	×	Two different $k\mathbf{x}$-like forces are applied to each mass, such that the non-naive frequencies are experienced by the two ion species.

Table 2.1: Truth table for `micromotion` (`mm`) and `naive_freqs` ($\sqrt{m_r}$) combinations and their physical validity.

According to equations 2.1 and 2.3:

$$(\tilde{a}_i, \tilde{q}_i) \propto 1/m_{cooler}. \tag{2.14}$$

Hence the real secular frequency of m_{target} can be obtained by calculating β_{target} given the pair:

$$(\tilde{a}_i, \tilde{q}_i) \cdot \frac{m_{cooler}}{m_{target}} \tag{2.15}$$

The dependence of β_{target} as a function of the mass ratio is depicted in figure 2.7. This insight invites defining 2 decoupled, boolean parameters for the simulation, named `micromotion` & `naive_freqs`. Table 2.1 is a truth table explaining what trapping forces are applied in each combination of these parameters, and a separate definition of these parameters is laid out below.

`naive_freqs` ($\sqrt{m_r}$)

Controls whether the secular frequencies experienced by the target CHDBrI$^+$ ion are defined using the 'naive' $\sqrt{m_r}$ relation. Turning this option on naturally implements a trap that is not physically implementable.

`micromotion` (`mm`)

Controls whether the secular frequencies experienced by both ion species are implemented using a Mathieu like force like in equation 2.1, or using a perfect harmonic trap force like $k\mathbf{x}$. A perfectly harmonic trap is of course not implementable.

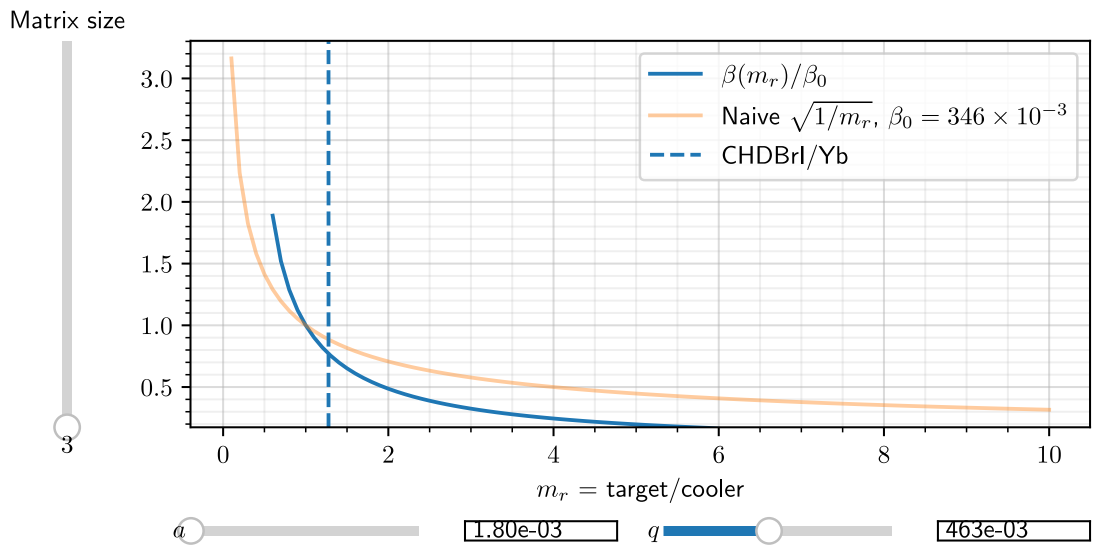


Figure 2.7: A Matplotlib interactive plot showing the non-trivial dependence of the secular frequencies of a 'target' mass (represented in blue), given a Mathieu trap with (a, q) that define a certain secular frequency for a 'cooler' mass, termed β_0. The (a, q) values used here, produce $\beta_0 = 2{\cdot}8.5/50$ - like the x & y axis Mathieu forces of figure 2.8.

LAMMPS Verification of Mathieu Understanding

Now that we have gained confidence in our Mathieu solving capabilities, Let's demonstrate that indeed the reversely solved Mathieu equations presented in section 2.1 indeed produces (a_i, q_i) pairs with the requested secular frequencies. Figure 2.8 demonstrates that decoupling the `micromotion` and `naive_freqs` parameters was done successfully.

The color-varied behavior in figure 2.8, demonstrates the non-negligible difference between the naive, $\sqrt{m_r}$-based calculation and the correct 'Mathieu frequencies' that are experienced in a real ion trap. The fact the two $\times$ & $\circ$ markers appear on the same spots, proves that indeed the simulation code has successfully created the correct forces no matter whether `micromotion` was enabled or not. The only physically implementable dots are the blue $\times$ points.

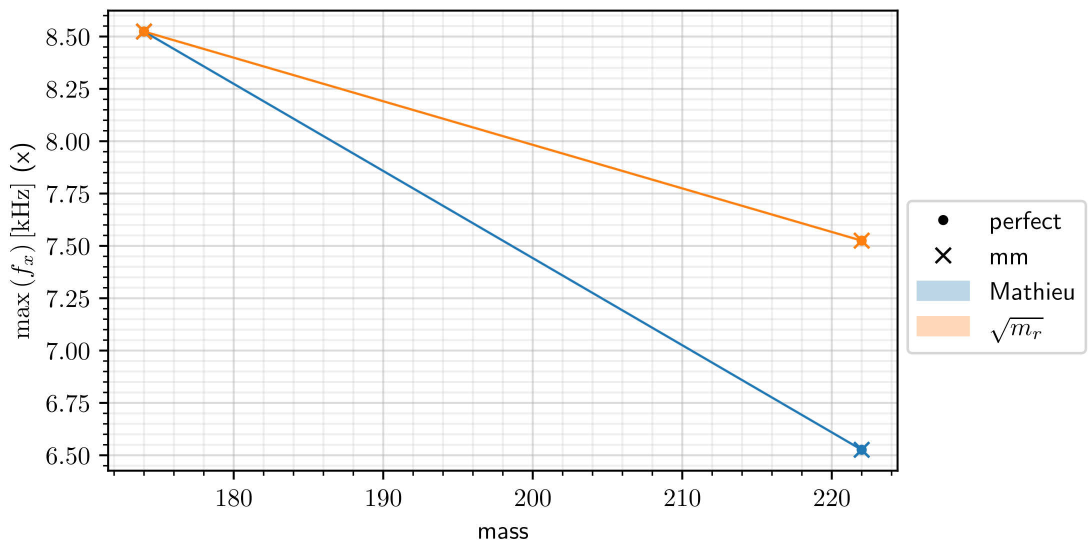


Figure 2.8: The collective x-axis movement's frequencies of a 2-species-mixed ion cloud, positioned in a spatial offset from the origin at $t = 0$. Different markers represents whether `micromotion` was enabled, and the `naive_freqs` parameter is represented in color. The frequencies were obtained via fitting [VGO$^+$20] the cloud's center to a cosine. Identical results can be obtained via using `numpy.fft.fft`[HMvdW$^+$20].

2.2 Laser Cooling

The theory of laser cooling is well established for 2-level systems. The semi-classical model implemented in the simulations presented in this work is explained well in literature [Coh92; Ste24a] and can be expressed as follows:

$$\mathbf{F}(\mathbf{v}) = \frac{h\pi\gamma\hat{k}}{\lambda} \cdot \frac{\omega^2/2}{1/4 + \omega^2/2 + (d - \hat{k}\cdot\mathbf{v}/\lambda/\gamma)^2} \tag{2.16}$$

Where:

- h is Planck's constant.
- $\hat{k}$ is the laser's spatial direction.
- λ is the cooling transition.
- γ, is the cooling transition's lifetime in Hz dimensions. The more commonly used rad/s scaled lifetime is usually denoted as $\Gamma = 2\pi\gamma$.
- d is the laser's detuning normalized to Γ. Sometimes the not normalized detuning in the dimensions of rad/sec is denoted as δ and with it we can denote $d \equiv \delta/\Gamma$.
- $\mathbf{v}$ is the velocity of the cooled atom.
- ω is the laser's *dimensionless* Rabi frequency. Given a Rabi frequency Ω in units of rad/s, $\omega \equiv \Omega/\Gamma$.

Apparently, applying such a force on a trapped particle induces an offset that is particularly relevant in ion traps like ours - ion traps of kHz secular frequencies. This offset is calculated in the next section, and was verified in simulations too in figure 2.9. We also describe how we overcame this offset as much as we could, in the section afterwards.

2.2.1 Ion Cloud Offset Induced by Laser Cooling Force

In principal, there is a Taylor expansion for expression 2.16, for $\mathbf{v}/\lambda/\gamma \ll 1$. Let's observe this expansion for a slightly simplified, 1 spatial dimensional form of the force - with $\mathbf{v} = (v, 0, 0)$ and $\hat{k} = (1, 0, 0)$:

$$\frac{h\pi\gamma}{\lambda} \cdot \frac{\omega^2/2}{1/4 + \omega^2/2 + d^2} \left(1 + \frac{2d}{1/4 + \omega^2/2 + d^2} \cdot v/\lambda/\gamma\right) \tag{2.17}$$

To understand the effect of the 0th order force component, let's observe a simplistic system of a 1 dimensional harmonic oscillator with a damping force of the form above:

$$\ddot{x} = -\omega_{\text{sec}}^2 x + F_0/m - 2\xi\omega_{\text{sec}}\dot{x} \tag{2.18}$$

Where ω_{sec} is the secular frequency well defined by our trap as explained in section 2.1 and ξ is derived from the 1st order force component in expression 2.17. The well known solution is:

$$x(t) = \frac{F_0}{m\omega_{\text{sec}}^2} + Ae^{-\xi\omega_{\text{sec}}t} \sin(\sqrt{1-\xi^2}\ \omega_{\text{sec}}t + \phi) \tag{2.19}$$

Where A and ϕ are determined by initial conditions. The steady state's offset from the origin $F_0/m\omega_{\text{sec}}^2$ in the case of a laser cooling force is

$$x_o = \frac{h\pi\gamma}{\lambda} \cdot \frac{\omega^2/2}{1/4 + \omega^2/2 + d^2} \cdot \frac{1}{m\omega_{\text{sec}}^2} \tag{2.20}$$

If we'll take:

- Secular frequency ω_{sec} in the order of magnitude of $\sim 2\pi \cdot 1.5$kHz.
- laser intensity of $\omega \equiv \Omega/\Gamma = 1$.[3]
- A common [Coh92] laser detuning of $d \equiv \delta/\Gamma = 1/2$.
- Yb$^+$ and the cooling transition at $c/\lambda = 811.2891$THz with the line width $\Gamma/2\pi \equiv \gamma = 19.6$MHz.[Ran20]

We'd get $x_o \approx 2.1$mm, where the trap's harmonic region is of about 25mm from the origin. This offset might not be very significant for only laser cooling a single ion like Yb$^+$, but sympathetic cooling will suffer from this offset, as the target molecule

[3]See also relation to actual intensity in subsubsection 2.2.3

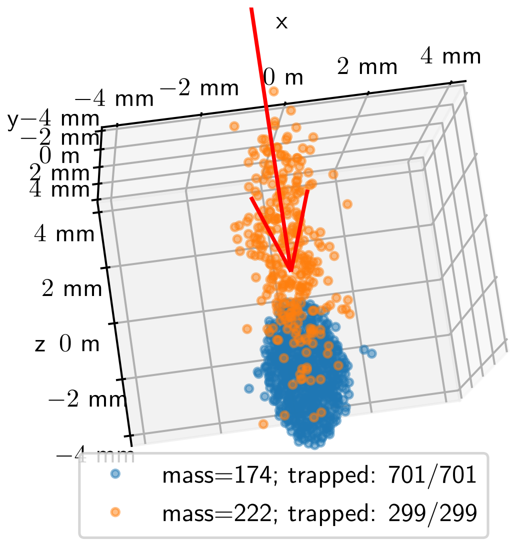


Figure 2.9: The offset produced by a single laser coming in the z axis, where secular frequencies are $f_x = f_y = 8.5$ kHz and $f_z = 1.5$ kHz. The offset measured fits exactly the analytical prediction of $x_0 \approx 2.1$mm.

won't experience an identical offsetting force, and would not collide frequently enough with the cooling ion and therefore will not lose kinetic energy, as seen in figure 2.9. Naturally increasing the secular frequencies of the trap greatly reduces this offset, but our trap's design doesn't allow much higher secular frequencies.

2.2.2 Overcoming the Offset

As suggested in a more classical laser cooling model presented by Dan Steck [Ste24b], we can use 2 counter propagating laser beams, that will apply 2 forces which sum up to $\mathbf{F}(\mathbf{v}) - \mathbf{F}(-\mathbf{v})$, forming an odd function in $\mathbf{v}$, with no 0th order component. Creating 2 such counter-propagating beams in experiment is simply done with a mirror. However, the vacuum chamber's windows induce a small attenuation on the intensity, as illustrated in figure 2.10.

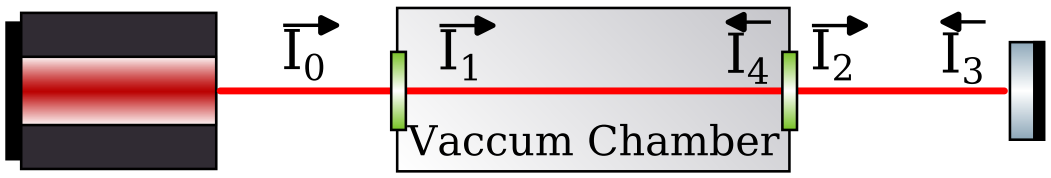


Figure 2.10: The effect of the Vacuum Chamber's windows' attenuation upon the laser cooling intensities.

Since $\omega \equiv \Omega/\Gamma$ and $I \propto \Omega^2$, if we'd define[4] $\alpha \equiv I_4/I_1$, the actual force we are imposed to apply upon our ions is:

[4]The experimentalist's most intuitive measurement is to measure I_2/I_0, but it should equal I_4/I_1 assuming all windows are made of identical glass, and the intensity lost in the mirror is 0.

$$\mathbf{F}(\mathbf{v},\omega) - \mathbf{F}(-\mathbf{v},\sqrt{\alpha}\omega) \tag{2.21}$$

Causing a slight imperfect oddness (with respect to $\mathbf{v}$) of the force, depending on the windows attenuation parameter α.

2.2.3 Summary of Laser Cooling Parameters

This is a good place to pause and list all the simulation parameters we collected from the theory of laser cooling:

- The laser cooled ion. Choosing an ion and a cooling transition is effectively choosing λ, γ and the mass m.[5] Current experimental efforts as well as all simulations presented focus on Yb^+, but in principal this parameter is not hard-coded, and
- The normalized laser's intensity $\omega \equiv \Omega/\Gamma$.
- The normalized laser's detuning $d \equiv \delta/\Gamma$.
- The spatial direction(s) of the laser(s). This can be parameterized by a list of (θ, ϕ) pairs - a pair per laser, defining each laser's $\hat{k}$ using a spherical coordinates transformation, as depicted in figure 3.1.
- The Vacuum chamber's windows' attenuation. Our experiment's attenuation is $\alpha = 0.7$, and most simulations used this value.[6]

Lastly, the relation of the Rabi frequency Ω to the actual laser intensity [Foo05], is worth mentioning[7]:

$$I = \frac{2\pi hc\Gamma}{3\lambda^3} \cdot \left(\frac{\Omega}{\Gamma}\right)^2 \tag{2.22}$$

With Yb^+'s parameters [Ran20], we conveniently get:

$$I \approx \frac{1\text{mW}}{\text{cm}^2} \cdot \left(\frac{\Omega}{\Gamma}\right)^2 \tag{2.23}$$

Now that we know how to fully characterise laser cooling and ion trapping, we will take revise some models we found in literature that try to predict cooling rates and energy exchange rates in multi-species ensembles.

[5] Cooling transitions and line-widths were taken from [IUH+93; WvdBG+08; HRK+15; Ran20; HFC+22]

[6] Besides figure 2.9 which displays it's effect.

[7] The software is capable of displaying Ω dependence in the scale of the intensity I, but this behavior wasn't activated in the results presented.

2.3 State of the art Sympathetic Cooling Models

Among the literature we surveyed, the only theoretical model we found that resembles the most to our simulations, is the one developed by Baba & Waki [BW02a]. Other works deal with:

- Laser cooling of single-species crystals [TP00]
- Plasma-like behavior [SL03; FN22] where temperatures evolve through bulk approximations
- A single ion per species (2 altogether), under a pseudo-potential approximation [WAMS12b]

In contrast, Baba & Waki's model directly addresses the exchange of energy between a laser-cooled species and a second, sympathetically cooled species, both in the gas-phase regime of a linear radio-frequency quadrupole (RFQ) trap, without using a pseudo-potential.

Their work combines detailed molecular dynamics (MD) simulations with a simplified analytical model based on binary energy exchange. In their simulations, 35 laser-cooled (lc) $^{24}\mathrm{Mg}^{+}$ ions form a cold, dense cloud in the trap. When 5 sympathetically cooled (sc) ions of various masses are introduced with higher initial kinetic energies, their behavior depends critically on their mass: $m_{\mathrm{sc}} \gtrsim (0.54 \pm 0.04) m_{\mathrm{lc}}$ are efficiently cooled, while lighter ions are instead heated. This mass threshold arises from the structure of micromotion in the linear RFQ trap, which lacks radio-frequency confinement in the axial z direction.

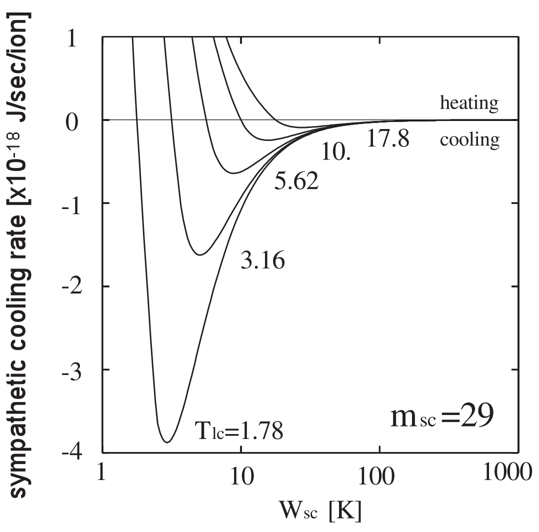
To interpret the MD results, the authors developed a 'heat-exchange' model, where the sympathetic cooling rate is described as the product of two components:

1. The *energy transfer per collision*, derived using elastic, two-body kinematics for ions moving along Mathieu trajectories in the RFQ field. This term incorporates the mass ratio, the trap parameters, and the kinetic energy.
2. The Coulomb *collision rate*, modeled using a differential cross-section of the Rutherford scattering.

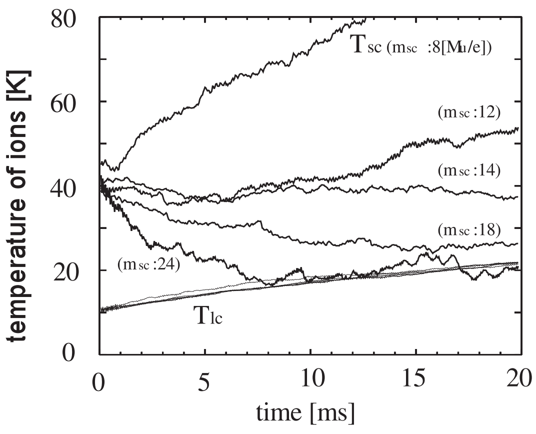
The resulting model yields a sympathetic cooling rate of the form

$$\frac{dW}{dt} \propto \frac{W_{\mathrm{sc}} - W_{\mathrm{lc}}}{(W_{\mathrm{sc}} W_{\mathrm{lc}})^{3/2}}$$

where $W_i \propto T_i$ and $i \in \{\mathrm{lc}, \mathrm{sc}\}$. This equation correctly captures the behavior observed in their simulations. As shown in figure 2.11, the cooling becomes ineffective both at very low and very high temperatures, and is strongly suppressed for $m_{\mathrm{sc}} < 0.54\, m_{\mathrm{lc}}$.



(a) Cooling rate as a function of both T_{sc} and T_{lc}. $W_{\mathrm{sc}} \propto T_{\mathrm{sc}}$. The mass ratio used in this calculation - $m_{\mathrm{sc}}/m_{\mathrm{lc}} = 29/24 \approx 1.32$ is somewhat similar to our mass ratio - $222/174 \approx 1.28$.



(b) MD simulation results for various m_{sc} values. Only ions heavier than roughly $0.54\,m_{\mathrm{lc}}$ are cooled effectively. Lighter ions are heated.

Figure 2.11: Key results from Baba & Waki's model, for $m_{\mathrm{lc}} = 24\,\mathrm{amu}$ (mass of $^{24}\mathrm{Mg}^{+}$). Their simulation and analytical work establish a critical mass ratio for effective sympathetic cooling in linear RFQ ion traps.

Their model incorporates many key physical ingredients: Coulomb interactions, energy partitioning, and trap-specific parameters. However, like other temperature-based differential equation models, it cannot reproduce more complex dynamical phenomena we observed in our simulations - phenomena that occur in a mixed gas-crystal phases. Moreover, we will see in the following chapter a *hysteresis* or *Mpemba* phenomenon in the temperature evolution, due to non-Boltzmann distributions of kinetic energies. This explicitly disagrees with the results of figure 2.11a.

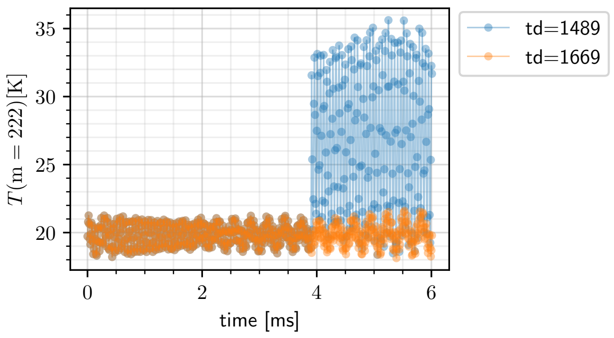
2.4 Simulations Parameters & Convergence

To fully define a simulation, one has to define a few more parameters, laid out in this section.

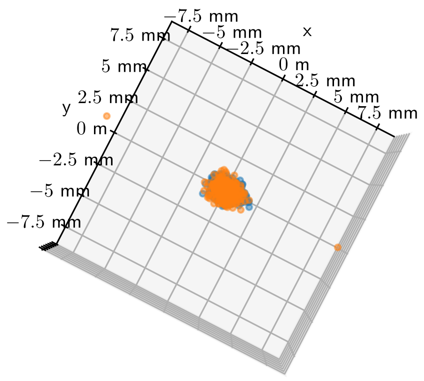
2.4.1 Time Periods

We define two subsequent time periods for each simulation: *stabilization* and *cooling*. During the stabilization period, ions evolve under only the trapping forces without laser cooling, allowing thermodynamic instabilities from the initial conditions to dissipate. After stabilization, the cooling laser is turned on for the cooling period.

For researching cooling parameters, and for most initial temperatures, a stabilization time of $5 - 7$ ms is sufficient. Figure 2.6 demonstrates this: the width of $T(t)$



(a) Temperature of CHDBrI$^+$ over a 6ms period of stabilization starting at a temperature of $T_i = 20$K, simulated with 2 different RF divisor `td` values. With `td = 1489` the temperature calculated is oscillating very broadly, and using a higher `td` fixes this issue.



(b) The coulomb force made 2 CHDBrI$^+$ ions fly roughly 8 mm from the center.

Figure 2.12: *Coulomb explosion*, demonstrated both in 3D simulation and in $T(t)$ measurement.

decreases adequately after 6 ms of stabilization for several initial temperatures.

Another usage for varying the stabilization time, is for measuring the eigenfrequencies of the trap, without cooling (0 cooling time), and all cooling parameters irrelevant.

2.4.2 Time Dividing Fineness

Another technical parameter is the fineness of the time division. As further justified in section 2.5.3, we want to always divide the simulation's RF cycles to an integer number of time steps. Since the RF frequency is also the highest frequency of the system, the natural thing to do is to divide each RF cycle to an integer number of steps, and this slightly arbitrary parameter is called the RF divisor.

One can perform many simulations with a certain RF divisor, and only in a specific simulation, in a specific advanced time, notice a certain ion has experienced a atypically strong Coulomb force. We term this phenomenon *Coulomb Explosion*, as it causes the ion to fly far away from the origin, which in turn make the temperature measurements noisy and essentially incorrect. An example of a coulomb explosion is depicted in figure 2.12.

With too low RF divisors, the temperatures measurements will be noisy and incorrect from the start. After a few months of optimizing this parameter, a value for the RF divisor was stabilized around ≈ 1500, depending on the trap's RF frequency. When increasing the RF divisor doesn't change substantially the results, we say the simulations *converge*. To further tighten the demand of the RF division's independence from potential numerical errors, a prime number was chosen [Wal10].

2.4.3 Initial Conditions

To gather conclusions from the simulations as accurately as possible, we wanted to initiate the simulation in a state that is thermodynamically stable as much as possible, to avoid two undesired phenomena: (1) 'Sloshing', which is the collective movement of the ion cloud in high amplitudes around the origin, and (2) 'breathing', which is a collective and periodic expansion and contraction of the ion cloud.

The initial conditions were obtained using a Random Number Generator (RNG). Each RNG seed integer represents a reproducible set of initial conditions, given the algorithm distributing positions and velocities stays the same. To decouple the effects of initial conditions upon the results, the same simulations but with different seeds were ran, and in most results displayed this dimension was averaged over.

2.5 Simulation Results Analysis

In this section we will discuss how we analyzed the simulation results. Naturally, the main quantity that interests us is the temperature, as a measure for the average kinetic energy. In our case since we use a VMI, it is also a way to estimate the velocities' distribution's width. A related property of our ion clouds is their size, that can be computed per spatial dimension simply via a `std`. As for the temperatures, there are a apparently many more methods to extract them, laid out below.

2.5.1 Temperature's Random Variables

Under the approximated treatment of the Ion trap's potential as a perfect harmonic potential, and with the long range Coulomb potential neglected, one can treat the positions analogously, and hence almost identically to the velocities. This is done by multiplying each particle's position in the i^{th} spatial dimension, by $2\pi f_i$, where f_i is the secular frequency of the Ion trap in the i^{th} axis.[8]

2.5.2 Temperature's Probabilistic Methods

Given N particles, which were sampled during the computer simulation in a certain time, 2 arrays of shapes `(N,3)` are obtained for velocities and positions. The positions' array can be scaled by the proper secular frequencies as described above, and the question of how to compute a temperature given such an array has multiple legitimate answers. To layout all available answers we'll denote the random variable $u_i \in \{v_i, 2\pi f_i x_i\}$ for axis i. The i^{th} slice of length `N` from the above `(N,3)` shaped array is a distribution of the u_i random variable.

[8] Although not implied by an existence of an additional index besides i, there is a different, non-trivial set of f_i secular frequencies per particle species, as described in section 2.1.4.

One simple answer which is most intuitive in the context of VMI based measurements is to simply compute $m\mathrm{Var}(u_i)/K_\mathrm{B}$, as derived from the Maxwell-Boltzmann distribution expression:

$$f(\mathbf{u})\mathrm{d}^3\mathbf{u} = \left[\frac{m}{2\pi k_\mathrm{B}T}\right]^{3/2} \exp\left(-\frac{m\mathbf{u}^2}{2K_\mathrm{B}T}\right)\mathrm{d}^3\mathbf{u} \tag{2.24}$$

Another approach, is to integrate over a solid angle of $\mathbf{u}$ in this probabilistic model, and write a speeds-like distribution function:

$$f(|\mathbf{u}|) = \left[\frac{m}{2\pi K_\mathrm{B}T}\right]^{3/2} 4\pi|\mathbf{u}|^2 \exp\left(-\frac{m|\mathbf{u}|^2}{2k_\mathrm{B}T}\right) \tag{2.25}$$

Which has a variance equal to $K_\mathrm{B}T/m \cdot (3 - 8/\pi)$, and a squared mean equal to $K_\mathrm{B}T/m \cdot 8/\pi$, thus providing 2 more methods to calculate a temperature given a `(N,3)` shaped array of either positions or velocities.

2.5.3 Temperature Under the Secular Approximation

In addition to the velocities and/or positions choice of random variable(s), and in addition to the above probabilistic model approaches, one has to decide how to imitate the secular approximation computationally. The physical justification for doing it, is the fact that in a crystal phase (which we aim achieving), the movement of the ions is strongly coupled to the RF field which we have direct control over. Essentially the model states that the collective movement due to the RF field is predictable and so well defined, that it is not a random variable.

Experimentally speaking, we are capable of measuring positions and velocities always at the beginning/end of each RF cycle, and hence always use an integer number of RF cycles. This is the simplest and probably the only way of implementing a secular approximation experimentally, and it should work best for ions that have formed a crystal. In the following, we'll describe what other techniques are available to a computer simulation that might help achieve time dependent, secularly approximated temperature assessments.[9]

Besides sampling the velocities and positions in the beginning of each RF cycle, many simulations based research found in literature average over an RF period each particle's per-axis velocity, and then construct a temperature [ZOR$^+$07b; MD21]. You might hope that the same can be done for $2\pi f_i x_i$, but apparently, even for ion clouds initiated in a thermal gas phase equilibrium in temperatures as low as 5K, each particle reaches many spatial areas of the trap, and it's RF averaged position produce (incorrect) temperatures in the μK regime! This proves that this approach is not accurate also

[9] There's also a technical motivation to imitate some kind of secular approximation when measuring temperatures, and that is reducing the amount of disk writes required when simulating $\sim 3,000$ RF cycles divided to ~ 1500 time steps. Reducing the amount of disk writes not only reduces disk usage of simulation result files, but also speeds up the total time required for simulating.

Dimension	Optional values				
`T_coord`	$\vec{v}$	$\overrightarrow{\omega x}$			
`T_rf_type`	$\langle\circlearrowright\rangle$	$\min\left(\circlearrowright\right)$	$\max\left(\circlearrowright\right)$		
`T_method`	$\mathrm{Ave}\left(\lvert u\rvert\right)^2\left(\pi/8\right)$	$\mathrm{Var}\left(\lvert u\rvert\right)/\left(3-8/\pi\right)$	$\mathrm{Var}\left(u_x\right)$	$\mathrm{Var}\left(u_y\right)$	$\mathrm{Var}\left(u_z\right)$

Table 2.2: Methods to extract temperatures, obtained from a generalized maxwell-Boltzmann model. $\circlearrowright$ marks an RF cycle, and $\langle\circlearrowright\rangle$ marks an average over RF cycle. The functions Ave and Var operate upon a 1 dimensional array of length N - the number of per specie's particles.

for the velocities, at least for gas like phases, as it suggests that ions go through a big part of the phase-space. Never the less, I decided to do save the RF averaged velocities (but not the positions) to the disk during simulations, and the software displaying the simulation results provides the option to use this data.

Another interesting RF cycles related option, is to sample the velocities and positions in the middle of each RF cycle - where the kinetic energy induces by the RF field should be at it's peak. The difference from the energies in the beginning of the RF cycles may help extract the RF motion energy. This data was also saved for both positions and velocities, for consistency.

2.5.4 Temperatures Options Summary

The above temperatures related options are modeled in software as 3 dimensions with the names and symbolic optional values as defined in table 2.2. We can select specific, multiple temperature values and average over the different methods, coordinates and RF handling methods. A common example is to take the positions and velocities sampled in the beginning of each RF cycle ($\min\left(\circlearrowright\right)$), and calculate the temperature according to all `T_methods`. Whatever specific `T_coords` / `T_rf_types` / `T_methods` were chosen, the code analyzing and displaying the simulation results averages over these dimensions for each time index separately.[10]

2.5.5 Further Summarizing Analysis

Up until now we discussed time dependent properties of our ion clouds - the per spatial axis size and the various temperatures. We can also summarize these results to obtain various measures of how fast did we cool our ions, and how much smaller did the cloud get due to the laser cooling. These kind of results are termed both in the code and here as summarized results.

[10]Technically, the temperatures obtained from the RF averaged ($\langle\circlearrowright\rangle$) positions ($\overrightarrow{\omega x}$) are considered `NaN` and hence ignored.

Temperatures and Cloud Size

The simplest summarized result type is the temperature/size before/after cooling. The time periods before/after the cooling are termed *stabilizing part*/*cooling part* respectively. If we'll denote the stabilizing & cooling times t_s and t_c, the summarized temperatures/sizes are defined (slightly arbitrarily) as the average temperature/size in the time periods $(3/4, 1)t_s$ and $t_s + (3/4, 1)t_c$. The standard deviation of the temperature/size in each of those *part*s defines the uncertainty of the summarized measurement.

Cooling Frequency

Another way to summarize the temperature measurements is as follows: Given a $T(t)$ defined using any of the parameters of section 2.5.4, we take $\text{Log}(T(t)/\text{kelvin})$, and apply a continuous piecewise linear fit [JV19], with 2 segments. This gives 2 cooling frequencies in the units of kHz that depend on a dimension referred to as `regime` - corresponding to each of the segments.

We chose to use 2 segments due to the observation that in many simulations most of the ion cloud is cooled quickly and efficiently whereas afterwards the rest of the cooling happens in a mixed thermodynamic phase of an ion crystal (massive and highly charged) with a gas of ions surrounding it. An example of such a fit is available in figure 3.8b.

Collective Cloud Movement Frequencies

Lastly, only for verifying the frequencies expected from the Mathieu equations of the ion trap, as explained in section 2.1, we define the following analysis referred to as the *cloud's frequencies*: Given the time dependent positions of all ions, we average over the ions to get the cloud's center position (still time and spatial axis dependent). With it, we can take the Fourier transform and get the dominant frequency via `scipy.signal.find_peaks`[VGO+20], or simply via `numpy.argmax`[HMvdW+20]. Such a Fourier transform is depicted in figure 2.13. The uncertainty of this computation is defined by the Fourier transform resolution limit, dictated by the total time range available.

If the collective movement is very harmonic (due to e.g collective offset at $t = 0$), a fit for a $\cos(2\pi f x_i + \phi)$ can be attempted, and the fit's frequency is the dominant frequency of the movement. The uncertainty of this fit, is obtained from the covariance matrix returned by `scipy.optimize.curve_fit`. An example of such a fit is available in figure 2.14.

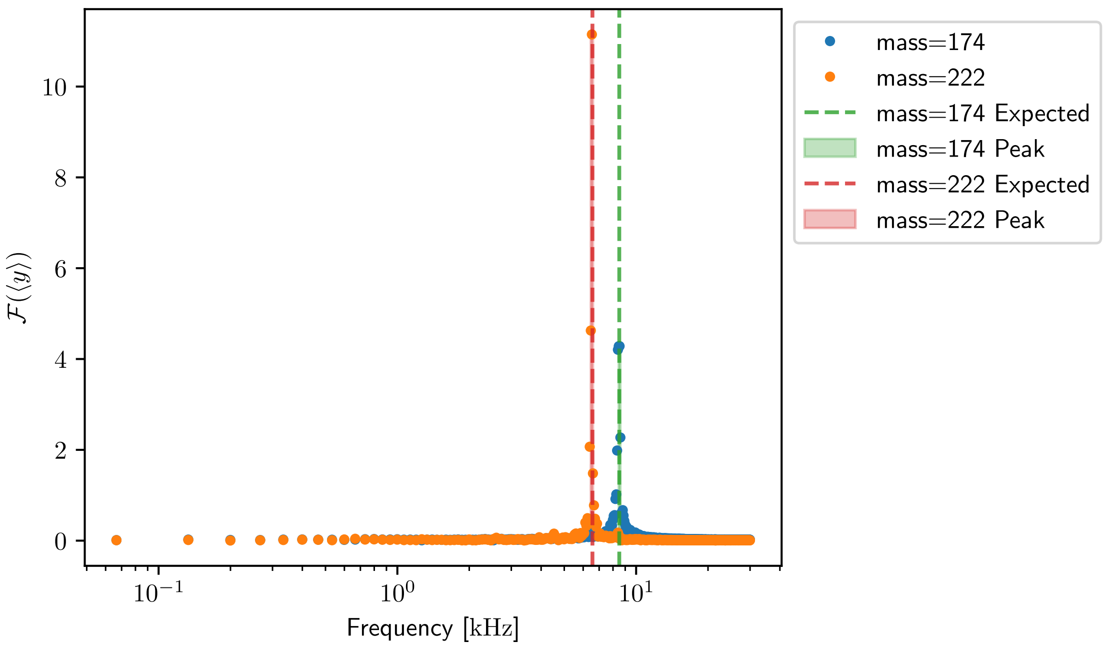


Figure 2.13: Fourier transform of the movement of the ion cloud's center, for 2 masses co-trapped in a physical Mathieu trap of frequencies $(f_x, f_y, f_z) = (8.5, 8.5, 1.5)$kHz for mass 174amu. The ion cloud was centered at $t = 0$ in an x axis offset of 4mm. The peaks of the expected frequencies per mass are marked in the dashed lines, and fit satisfyingly to the Fourier transform's peaks.

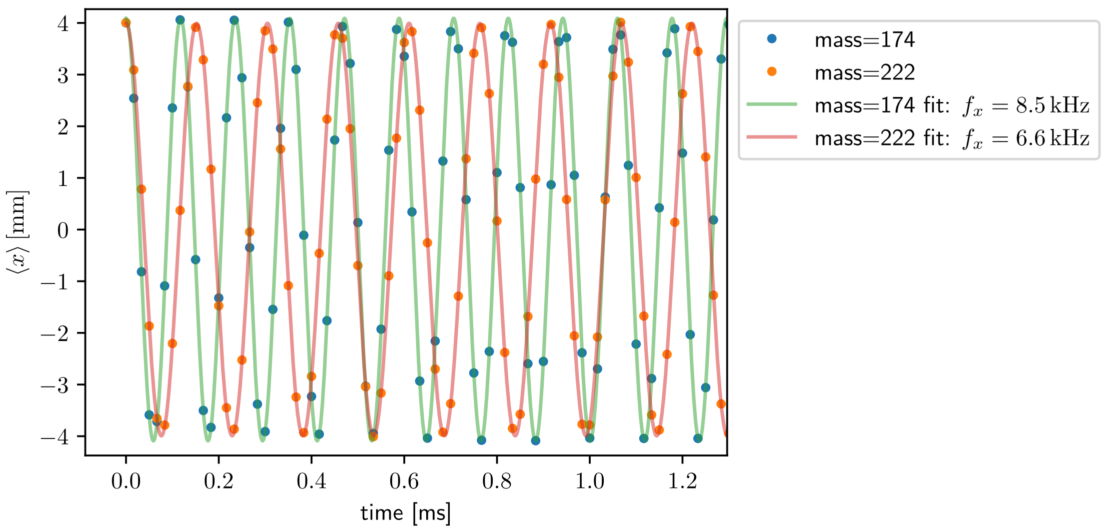


Figure 2.14: Cosine fit for one the movement of the ion cloud's center, for 2 masses co-trapped in a physical Mathieu trap of frequencies $(f_x, f_y, f_z) = (8.5, 8.5, 1.5)$kHz for mass 174amu. The ion cloud was centered at $t = 0$ in an x axis offset of 4mm. Only a few cycles are presented in order for the fit's convergence to be clear.

Chapter 3

Cooling Simulation Results

Before trapping both Yb^+ and $CHDBrI^+$, we wanted to make sure we are able to cool efficiently only Yb^+. We believe that cooling fast the Yb^+ ions, must help cool fast the $CHDBrI^+$ ions. This led us to investigate first how to cool Yb^+ alone fast, and the main results of this research are in section 3.1. Next we added $CHDBrI^+$ and varied the intensities and the initial temperatures, to see how sensitive is the sympathetic cooling to these parameters.

3.1 Laser Cooling Intrusion Angle(s) & Trap Geometry

When our experimentalists first tried to cool the Yb^+ ions, they had less than a total of 1mW of power. This constraint motivated several questions that our simulations address: Is using 1 laser enough? If so, which intrusion angle should be used? If we split the available power among multiple laser beams, which angles are optimal? Is recycling laser power (by reflecting the beam back through the trap with mirrors) worth the technical effort of beam alignment?

In the case of 1 laser beam, the naive choice might be to use the z axis, which is the symmetry axis of the trap, and in which there is also no micromotion. It turns out that using the z axis is actually the worst choice of them all, and in fact you *do* want the laser beam to mix the principal axes of the trap as much as possible. The next sections describe how we parameterized this, and why we think mixing the principal axes helps.

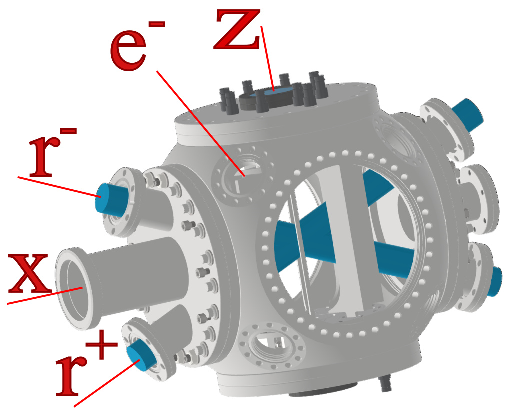
3.1.1 Laser Angles Parametrization

Our vacuum chamber has only a finite set of windows we can use, so we gave each of them a name, as depicted in figure 3.1a. Each intrusion window is mapped to a (θ, ϕ) pair, as listed in table 3.1, according to the standard spherical solid angle parametrization depicted in figure 3.1b.

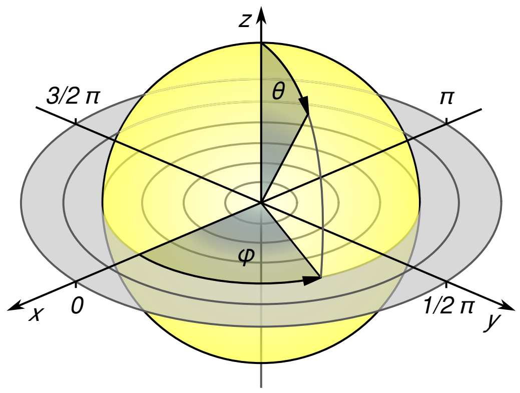
No matter how many lasers we use, we give **all of them** a single intensity $I \propto (\Omega/\Gamma)^2$. This means that comparing 3 lasers of a total intensity of I, to a single laser

	z	r^-	r^+	e^+	e^-	e^*
θ	$0°$	$67.5°$	$112.5°$	$45°$	$45°$	$-45°$
ϕ	$0°$	$0°$	$0°$	$45°$	$-45°$	$45°$

Table 3.1: Our available intrusion angles mapping from names to (θ, ϕ) pairs.



(a) Our vacuum chamber CAD design [WSoT24], annotated with the names we gave of the available windows through which our laser can intrude.



(b) Our (standard) solid angles parametrization

Figure 3.1: Our laser intrusion angles parametrization.

of the same intensity, would require using $\Omega \propto \sqrt{I/3}$ in the 3 lasers simulations. In figure 3.2, to simulate a total intensity of $\Omega/\Gamma = 1$ split to 3 laser beams, we used an intensity of $\Omega/\Gamma = 0.55 < 0.58 \approx \sqrt{1/3}$ – taking possible experimental splitting inefficiency into account.

In the results presented below, markers represent the intensities of **each one** of the lasers, and colors represent laser angles *group*s. A group includes one or more lasers. They were performed with a laser detuning of $\delta/\Gamma = -2.6$, over 1000 Yb^+ ions and in a trap with frequencies $(f_x, f_y, f_z, f_{\text{rf}}) = (8.5, 8.5, 1.5, 100)$ kHz.

3.1.2 Simulation Results

In figure 3.2, we see a wide range of cooling rates, produced by several configurations of a total laser intensity $\Omega/\Gamma = 1$. The cooling rates are summarized in table 3.2. The propensity rule we derive from this comparison is that mixing the principal axes of the trap is what helps cool faster.

This is not only a matter of breaking the cylindrical symmetry of the trap, as e.g the x laser does that too, and it cools slower then r^-, which is almost parallel to it. We also think there is no meaning to the ϕ direction of all laser angles, and the most important parameter is how much the laser is mixing z principal axis movement with the x, y plane.

Using 3 lasers which are all mixing the principal axes seems to work best. However,

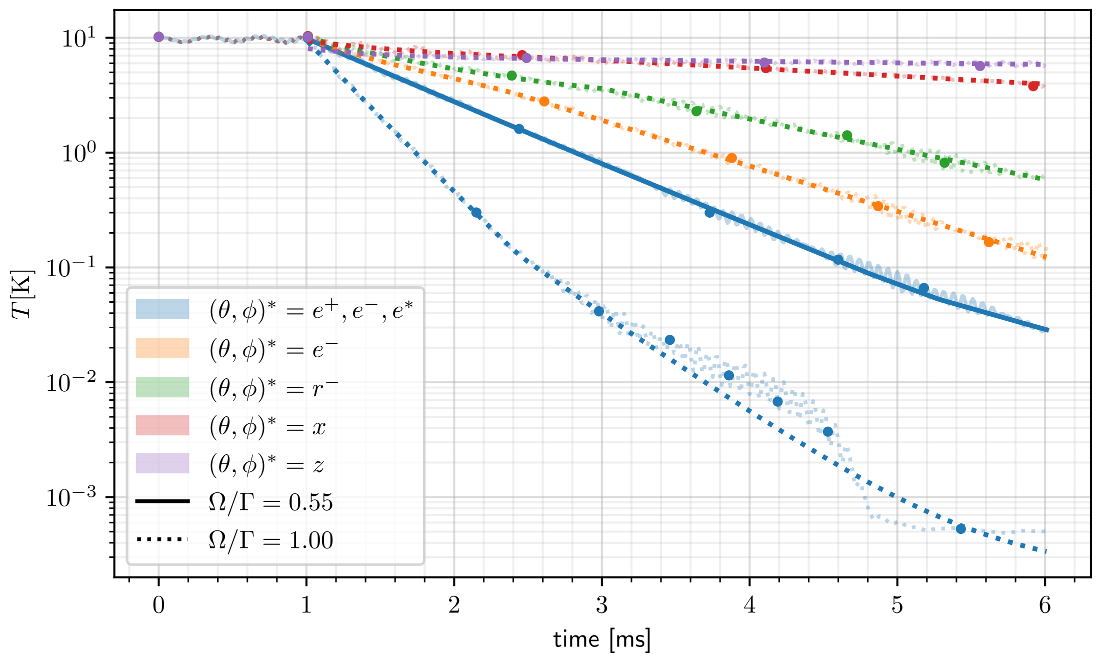


Figure 3.2: Temperatures as a function of time, for several different laser angles groups. As explained in section 3.1.1, the blue-solid line represents a single laser beam of $\Omega/\Gamma = 1$, split upfront to 3 laser beams. On the other hand, *recycling* the laser power is simulated in the blue-dashed line – where *all* beams' intensities are $\Omega/\Gamma = 1$.

Power	e^+, e^-, e^*	e^-	r^-	x	z
Recycled	2.3(1) kHz	981(28) Hz	806(8) Hz	349(8) Hz	353(22) Hz
Split	1.3(0.0) kHz				

Table 3.2: Cooling rates computed with a linear fits to $\text{Log}(T(t))$ lines presented in figure 3.2. Whether the laser power is recycled or split upfront is naturally relevant only to a case with multiple lasers.

if obliged to use only one laser, using e^- gives slightly better results in comparison to r^-. The latter however, is parallel to the optical table, and hence is slightly easier to setup technically in the lab. This is why we chose r^- for the next scan, in which we also added CHDBrI$^+$ ions and modulated the intensity (Ω), in multiple initial temperatures.

3.2 Scanning Ω & Initial Temperatures with CHDBrI$^+$

In this section we will see several phenomena that are hard to model, and hence demonstrate the significance of our results. We will start by focusing on the Yb$^+$ temperatures, as a function the laser intensity and the initial temperature. For reference, in this section all of the simulations had 299 CHDBrI$^+$ ions, 701 Yb$^+$ ions, and used the parameters presented in table 3.3.

δ/Γ	Laser's $(\theta, \phi)^*$			t_d	$f_{\rm rf}$[kHz]	t_c[ms]	t_s[ms]	f_x[kHz]	f_y[kHz]	f_z[kHz]
-2.6	r^-	67.5	0	1583	50	30.0	6.0	8.5	8.5	1.5

Table 3.3: Parameters commonly used for multi-dimensional scan of Intensities (Ω/Γ) and initial temperatures (T_i)

3.2.1 Yb$^+$ Temperatures in the Presence of CHDBrI$^+$

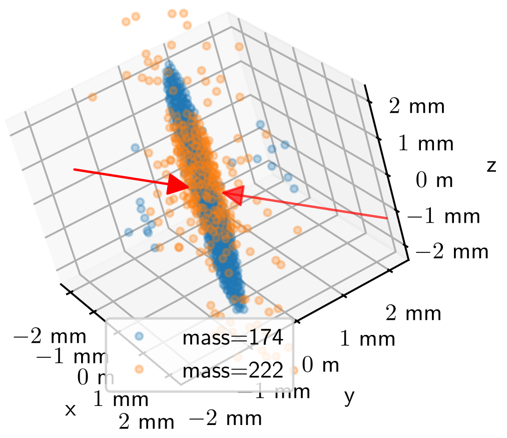
In figure 3.3b we can see the temperatures of only the Yb$^+$ ions, with colors for different intensities. Intriguingly, we see they stabilize to a certain value, after an intensity dependent amount of time. When we looked at the animations for each of those lines, we saw a varying number of rogue Yb$^+$ outside of the crystal, that are precluded from joining it, due to CHDBrI$^+$ that have already adsorbed. In figure 3.3a we see an example of this phenomenon, with roughly 10 rogue Yb$^+$ ions.

The energy distributions of the Yb$^+$ ions when rogue Yb$^+$ are present is depicted in figure 3.4. In particular, the orange $\Omega/\Gamma = 1$ histogram exhibits a small maximum around 40K that demonstrates that rogue Yb$^+$ shift the distribution away from being Boltzmann.

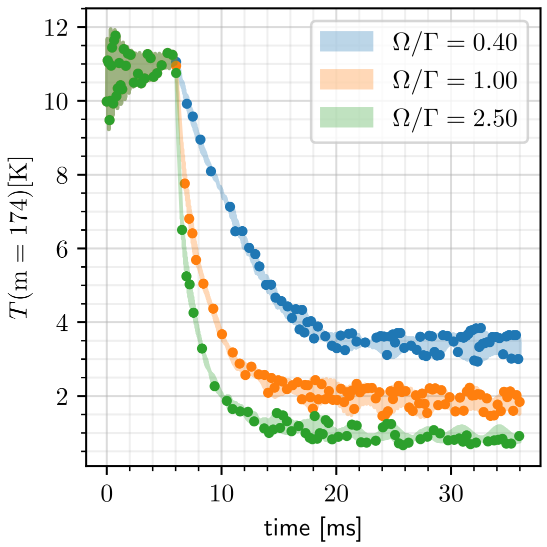
In the examples above both species started at 10 K, but if we start at 5 K, we *are* capable of assembling a fuller Yb$^+$ crystal without rogue ions. This is apparent in the animation snapshot in figure 3.3c, where we also used an intensity of $\Omega/\Gamma = 2.5$. The lack of rogue Yb$^+$ ions is also apparent in the temperatures extracted to figure 3.3d.

To avoid rogue Yb$^+$, one can suggest to cool all Yb$^+$ ions slower (with e.g lower laser intensity), so that the CHDBrI$^+$ ions will be adsorbed more gradually. This indeed might increase the total number of crystallized ions after a long time. However we focus in this research on improving sympathetic cooling on time scales of roughly 50 ms, so we decided to stick with the attempt to cool the Yb$^+$ ions as fast as we can.

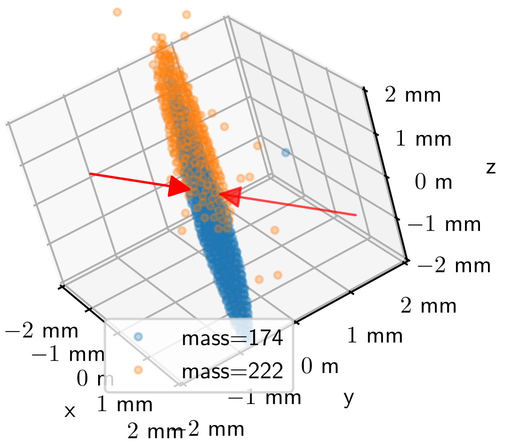
After observing these results we realized we might not be able to completely crystallize all Yb$^+$ ions with only a single laser. However, this may not be a devastating



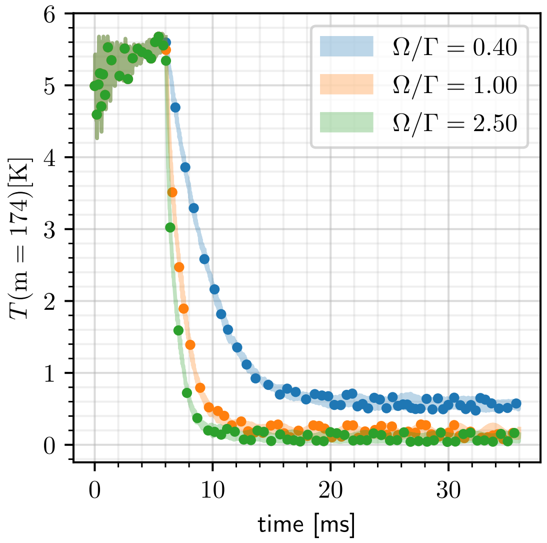
(a) Positions of all ions at the end of a simulation that started at $T_i = 10$ K with a laser power of $\Omega/\Gamma = 1$ – orange temperatures in figure (b).



(b) Yb$^+$'s temperatures (mass 174 amu), after they started at $T_i = 10$ K.



(c) Positions of all ions at the end of a simulation that started at $T_i = 5$ K with a laser power of $\Omega/\Gamma = 2.5$ – green temperatures in figure (d).



(d) Yb$^+$'s temperatures (mass 174 amu), after they started at $T_i = 5$ K.

Figure 3.3: Simulation results showing the effect of rogue Yb$^+$ ions on the temperatures of the Yb$^+$ ion cloud. Results were obtained using 299 CHDBrI$^+$ and 701 Yb$^+$ ions in a trap with secular frequencies $(8.5, 8.5, 1.5)$ kHz. The right plots show Yb$^+$ temperatures, averaged over 4 different initial conditions, for 3 different laser intensities represented by line colors. The left figures show animation snapshots from the end of 2 selected simulations, with red arrows showing the laser cooling intrusion angle. In the top figures, the initial temperature is $T_i = 10$ K, whereas in the bottom figures it is $T_i = 5$ K. Rogue Yb$^+$ ions are most apparent in figure (a).

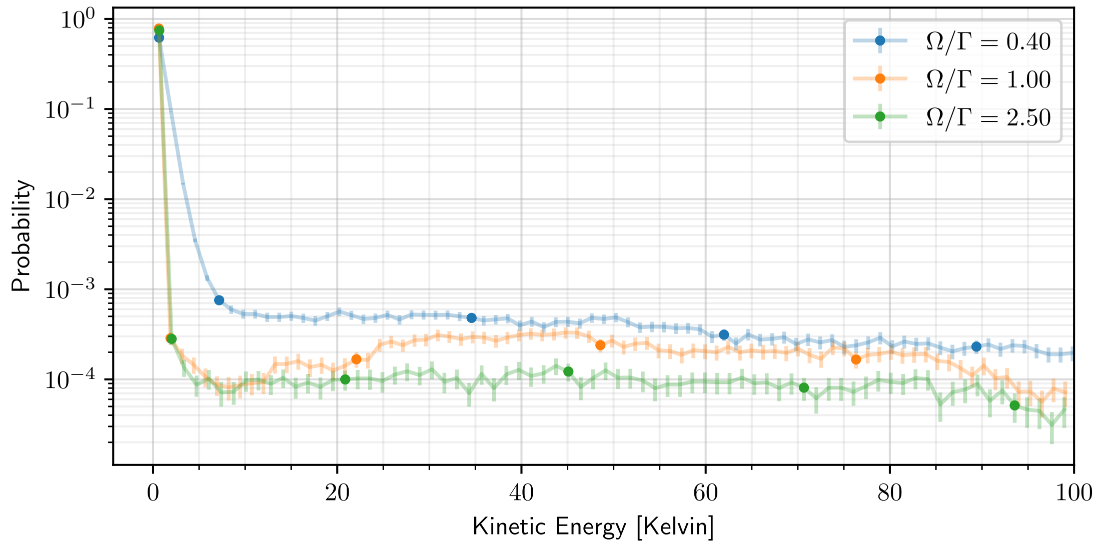


Figure 3.4: Histograms of Yb^+'s kinetic energies of the last 3ms of the cooling period shown in figure 3.3b. The orange $\Omega/\Gamma = 1$ line shows the energy distribution depicted in figure 3.3a. Energies above 100 K are not shown due to large statistical uncertainties (relative errors $\sqrt{N}/N > 0.5$).

issue for sympathetically cooling the $CHDBrI^+$ ions, as it might still be possible to cool it efficiently. The sympathetic cooling results, also with only r^- laser are shown in the next section.

3.2.2 Sympathetic Cooling with Varying Intensities

In figure 3.5 we can see the $CHDBrI^+$ ions' temperatures for initial temperatures $\{5, 10\}$K, and with a different color per laser intensity. It seems that in this range of intensities (targeting only Yb^+ ions of course), one can conclude that a stronger laser helps cooling. This is easily justified by expression 2.16 of the laser cooling force introduced in section 2.2, and also by the phenomenon of power broadening [Ste24a].

A more intriguing phenomenon we noticed in the $\Omega/\Gamma \geq 1$ lines of figure 3.5 is the inconsistent cooling rates (slopes). The descent is steeper when starting from lower initial temperatures, and the cooling rate appears to depend on the initial temperature rather than the instantaneous temperature. Figure 3.6 demonstrates this hysteresis effect by comparing the $\Omega/\Gamma = 1$ simulations from both initial temperatures.

We see in figure 3.5 that when the $CHDBrI^+$'s temperature starts at $T_i = 10$ K, the rate is moderated when it reaches $T_i = 5$ K, whereas the rate is higher when it starts at $T_i = 5$ K initially! We identify this phenomenon as *hysteresis* - the $CHDBrI^+$'s cooling rate seem to depend on the initial temperature on long time scales, and that limits their cooling rate.

Our attempts to fit our results to the models presented in section 2.3 failed due to this hysteresis. We managed to identify the root cause for the conservation of energy, when we looked at the animations, as described in the next section.

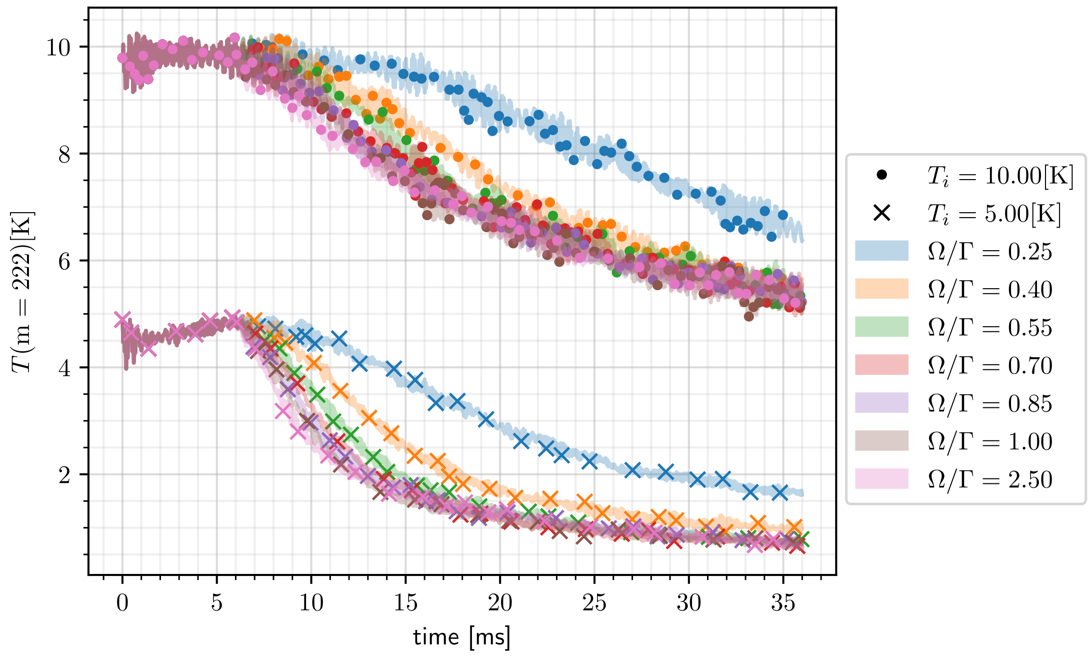


Figure 3.5: The $CHDBrI^+$ ions' temperatures, starting in initial temperatures $\{5\text{ K}, 10\text{ K}\}$, and in various laser intensities.

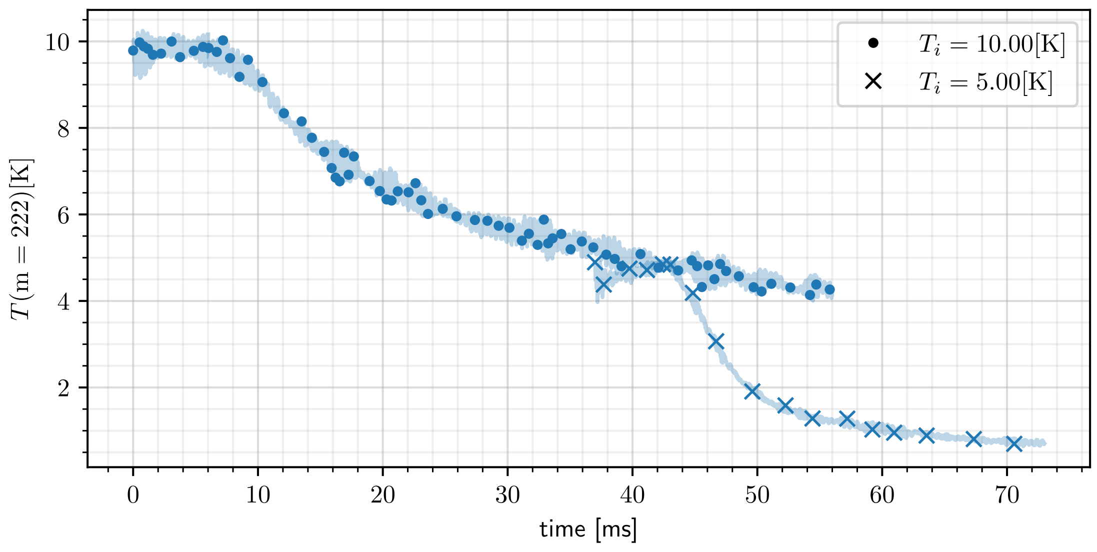
3.3 Angular Momentum Conservation

As depicted in figure 3.7, we noticed that many gas-phase $CHDBrI^+$ ions are rotating around the crystal. Their rotation axis is the z axis, which is also the symmetry axis of our cylindrically symmetric trap, meaning $f_x = f_y > f_z$.

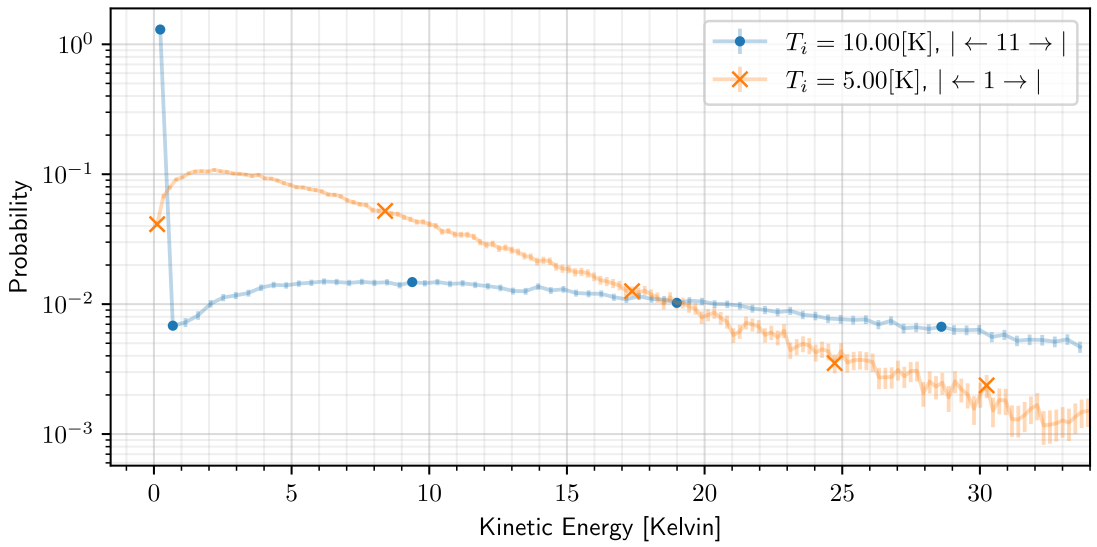
This collective movement around the z axis limits the sympathetic cooling rate because the rotating ions can rotate for long periods of time without losing kinetic energy. This occurs because the Coulomb crystal is smaller in comparison to the sympathetically cooled ion cloud, and can only exert radially directed forces on the ion cloud. This is the main missing piece in Baba & Waki's model described in section 2.3 - the angular momentum is not treated explicitly in their model, and this behavior is not detected.

Baba & Waki's sympathetic cooling rate depicted in figure 2.11a, does state that for very low T_{lc} the sympathetic cooling rate is suppressed. However we identify it to be strongly dependent on the *way* the sympathetic cooled ions' velocities are distributed (with or without a collective rotation around the z axis), and not only on T_{lc} and T_{sc}. Most importantly, what limits the cooling rate is the changes the velocities' distribution goes through, and that depends on the initial conditions for time scales of at least 10 ms - hence we identify this as hysteresis.

Since the effectiveness of sympathetic cooling naturally depends on the ion-ion collision strengths, we tried to increase the secular frequencies, to strengthen the collisions. We scanned multiple sets of trap frequencies, all with a cylindrically symmetry, and



(a) CHDBrI$^+$ temperature evolution for simulations starting at $T_i = 5$ K and $T_i = 10$ K, both with $\Omega/\Gamma = 1$. The $T_i = 5$ K curve has been shifted forward in time by 37.0 ms to align with the point where the $T_i = 10$ K simulation reaches 5 K. The steeper slope of the blue curve demonstrates that ions starting at 5 K cool faster than ions that have cooled down to 5 K from 10 K - a hysteresis effect where cooling rate depends on thermal history.



(b) Kinetic energy distributions of CHDBrI$^+$ ions comparing two scenarios at different times. Orange: ions that started at $T_i = 10$ K sampled at $t =$ 27-30 ms (marked $|\leftarrow 11 \rightarrow|$). Blue: ions that started at $T_i = 5$ K sampled at $t =$ 3-6 ms (marked $|\leftarrow 1 \rightarrow|$). Despite both having mean temperatures of approximately 5 K, the distributions have different shapes, explaining the different cooling rates observed in figure 3.6a, specifically at $t \approx 42$ms. Energies above 34 K are not shown due to large statistical uncertainties (relative errors $\sqrt{N}/N > 0.4$).

Figure 3.6: Hysteresis in sympathetic cooling of CHDBrI$^+$: cooling rate depends on initial temperature. Simulations show that CHDBrI$^+$ ions cool faster when starting at $T_i = 5$ K compared to ions that cooled from $T_i = 10$ K to the same temperature. The energy distribution comparison in figure 3.6b reveals that this effect arises from non-Boltzmann distributions that persist from initial conditions.

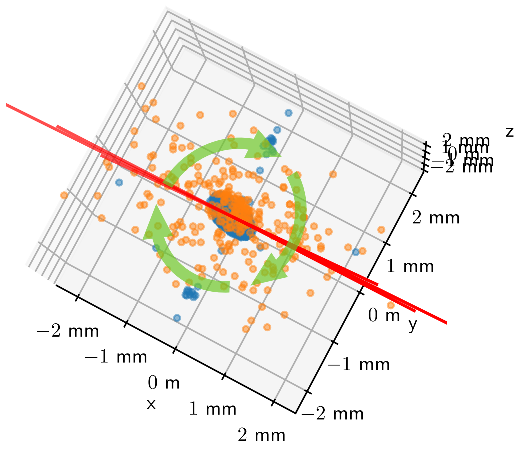


Figure 3.7: A snapshot of an animation in which one can clearly see the ions are rotating around the z axis - which is also the symmetry axis of the cylindrically shaped trap.

t_c[ms]	t_s[ms]	Ω/Γ	δ/Γ	Laser's $(\theta, \phi)^*$		
				r^-	67.5	0
30.0ms	6.0ms	1	-2.6	r^+	112.5	0
				e^-	45	-45

Table 3.4: Parameters common for the multi-dimensional scan of trap secular frequencies and initial temperatures (T_i)

the results are described in the next section.

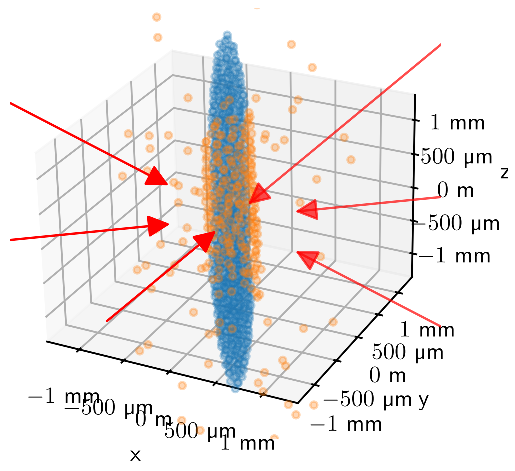
3.4 Varying Secular Frequencies

In this scan of many different trap frequencies, we wanted to eliminate the effects of rogue Yb$^+$ ions we saw on section 3.2. We did that by ensuring they are all cooled almost instantly, using multiple high intensity lasers. We also chose secular frequencies with similar ratios between $f_x = f_y$ and f_z - to reduce the risk to mix effects related to the trap's eccentricity. Lastly, these simulations had 299 CHDBrI$^+$ ions, 701 Yb$^+$ ions, and the parameters presented in table 3.4.

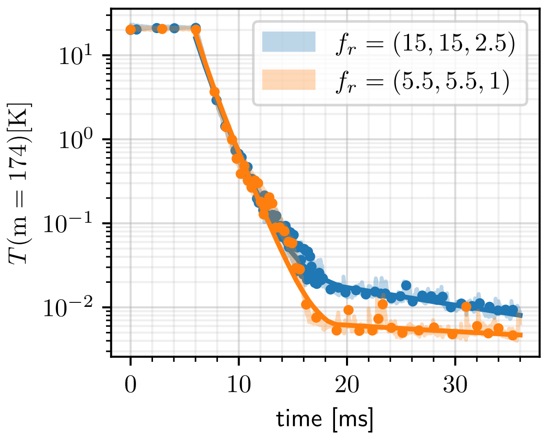
In all of these simulations we had 4 different initial conditions per trap and initial temperature. We found consistently rogue Yb$^+$ ions only for $f_r = (8.5, 8.5, 1.5)$ and $f_r = (17, 17, 3)$. Their amounts for for these traps, with $T_i = 10$ K was (respectively) 7.25 ± 2.3 and 5.5 ± 1. The amount of rogue Yb$^+$ for $T_i = 20$ K, was roughly twice as much.

Never the less, the lower amounts of rogue Yb$^+$ ions is observable in their temperatures, as depicted in figure 3.8. The absence of rogue Yb$^+$ ions in the traps given there, allows us to observe fast and efficient laser cooling in the presence of a sympathetically cooled ions, with rates of almost 1kHz[1].

[1]Calculated with PieceWise Linear Fits [JV19].



(a) Positions at the end of a simulation, with no rogue Yb^+ ions outside the crystal, and arrows depicting the multiple laser beams used in these simulation.



(b) Temperatures of Yb^+ ions when no rogue Yb^+ were observed, and hence are all fully crystallized. The initial temperature is $T_i = 20$ K and we reach the sub-kelvin regime in less then 10 ms, with the presence of CHDBrI^+. The exact rates for the 1st piecewise linear fit segments, are 933.7(3.9)Hz & 986.1(5.4)Hz for $f_r = (15, 15, 2.5)$ & $f_r = (5.5, 5.5, 1)$ respectively. The lower steady-state temperatures observed for $f_r = (5.5, 5.5, 1)$ are probably due to the weaker trap inducing less collisions.

Figure 3.8: Temperatures of Yb^+, along with a positions snapshot of $f_r = (15, 15, 2.5)$ (in kHz).

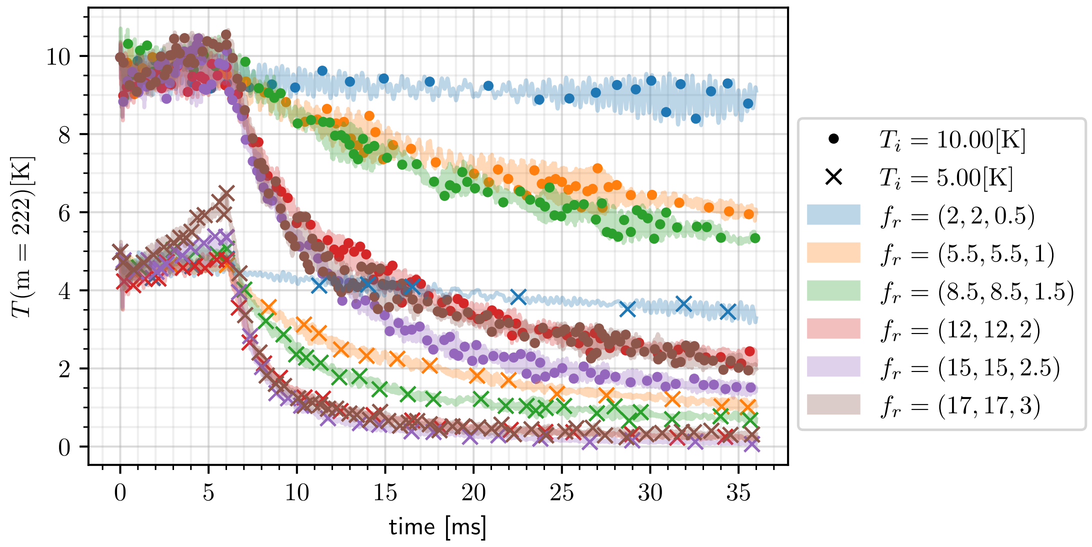


Figure 3.9: Comparison of CHDBrI$^+$ temperatures, with $T_i \in \{5, 10\}$K, and many different traps varied by color. As always, ion trap's frequencies are in kHz. RF Heating is also observed for the strong traps, especially when we start at $T_i = 5$K, and is explained well in literature [vMHG$^+$22; OMS96].

The next thing we show, is how much better is the sympathetic cooling with the stronger traps. In figure 3.9 we see a similar $T(t)$ plot with various colors and markers per traps and initial temperatures, respectively. Clearly using a stronger trap helps for most of the simulated scenarios, especially with high initial temperatures. This rule of thumb is intuitive since higher secular frequencies induce stronger collisions between CHDBrI$^+$ ions and the ion crystal.

However, with $T_i = 10$ K, and comparing $f_r = (17, 17, 3)$ v.s $f_r = (15, 15, 2.5)$, we observe a violation of this rule of thumb. Our best explanation for this violation is the 5.5 ± 1 rogue Yb$^+$ ions we counted earlier for $f_r = (17, 17, 3)$, which seem to undermine the sympathetic cooling.

When comparing $f_r = (5.5, 5.5, 1)$ v.s $f_r = (8.5, 8.5, 1.5)$, we don't see such a violation, even though for $f_r = (8.5, 8.5, 1.5)$ we see even more rogue ions - 7.25 ± 2.3. We think that the higher secular frequencies, inducing stronger CHDBrI$^+$ collisions compete with the disturbance of the rogue Yb$^+$ ions.

These results are summarized in figure 3.10 in the following way: We normalize the final temperatures (T_f) of the CHDBrI$^+$ ions to the temperatures before cooling.[2] This gives us a dimensionless scalar out of every simulation, which is plotted as a function of the trap frequencies, with colors per T_i. The bare difference we see between the points of $f_r = (15, 15, 2.5)$ v.s $f_r = (17, 17, 3)$, is well explained also by the rogue Yb$^+$ ions discussed earlier, observed consistently in all simulations with $T_i \geq 10$ K.

[2]denoted T_s, not T_i - see section 2.5.5.

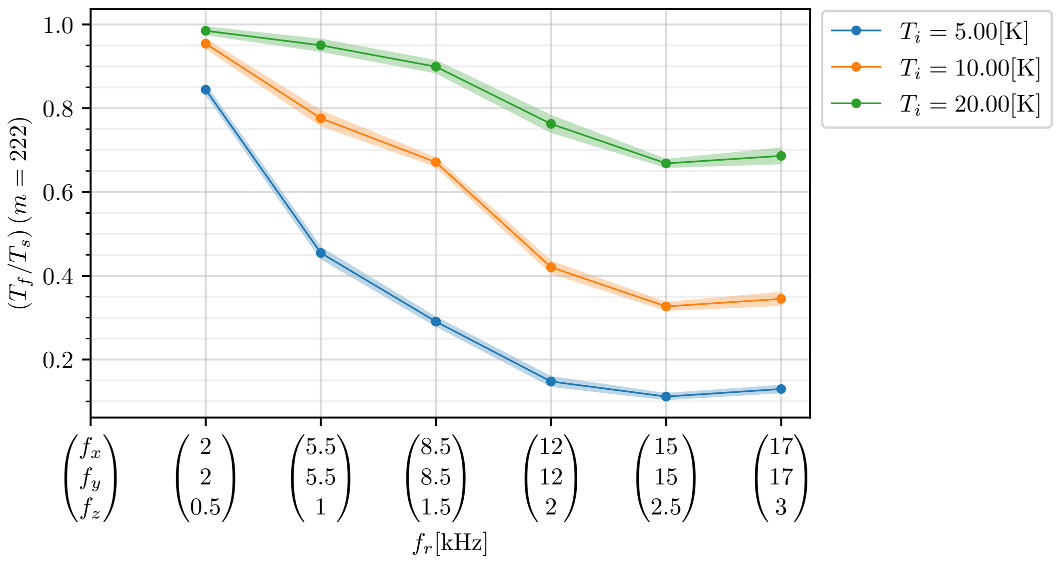


Figure 3.10: Simulation results of the dependence of the sympathetic cooling rate of the secular trap frequency and the initial temperatures. The plot shows higher cooling rates for higher trapping frequencies. Both T_f & T_s were calculated by averaging over the temperatures in the last quarter of the cooling and stabilization times, respectively (for more details see section 2.5.5). The errors of each temperature was calculated via standard deviation of the temperatures vectors. A total of $6 \cdot 3 \cdot 4 = 72$ measurements are presented here, for 6 traps, 3 initial temperatures, and 4 different initial conditions.

Chapter 4

Conclusion & Outlook

The software developed for this thesis has enabled us to explore phenomena that were hard to anticipate, and hard to analytically model. These include:

- Choosing wisely laser cooling intrusion angles is significant.
- Rogue laser-cooled ions lead to a kinetic-energy distributions that don't follow Boltzmann statistics.
- Angular momentum conservation of the sympathetically cooled ions occurring outside the atomic crystal, limits their cooling rate. This is a finite size effect of the ion crystal.
- The sympathetically cooled ions' conserved angular momentum, depends indirectly on their initial temperature. We identify this phenomena as hysteresis or an Mpemba effect, because the cooling rate depends on the energy distribution, which could be a remainder of the initial conditions.
- Increasing the trap's secular frequencies, seem to help overcome the angular momentum conservation limit, by inducing more frequent collisions with the cooled crystallized laser cooled ions.

Additionally, Baba & Waki's analytical model [BW02a], provide useful insight into the temperature dynamics of sympathetic cooling. However, since their model focuses primarily on temperature evolution, they do not account for conserved quantities such as angular momentum, and therefore cannot describe the Mpemba effect or hysteresis phenomena we observed.

Outlook

- Using a non-isotropic trap can reduce angular momentum conservation, and can be simulated without further modifications to the software. These results could inspire experimentalists to implement non-isotropic traps which might enhance sympathetic cooling rates.

- Modeling the angular momentum conservation in cylindrically symmetric traps might be feasible, using a $T = 0$ K ion crystal, surrounded by a dilute cloud other ions. A reliable model for the ion crystal size is available in the literature [HA91] and can be generalized to non-spherical traps.
- Single-species laser cooling can be further optimized using our software. Given that the cooling force 2.16 depends on velocity, it would be interesting to explore time-dependent modulations of detuning or intensity to further enhance cooling rates.

Appendix A

More Coulomb Energies Dependence Figures

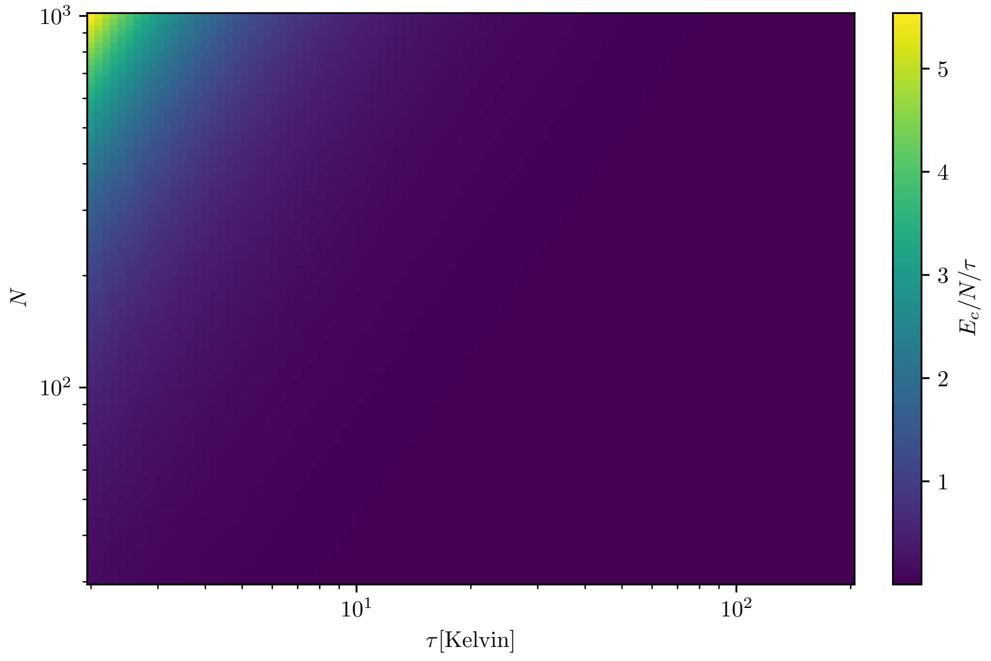


Figure A.1: Same as figure 2.4, but with a normalization to τ. Added here to show how E_c/N can exceed τ for some values of (τ, N).

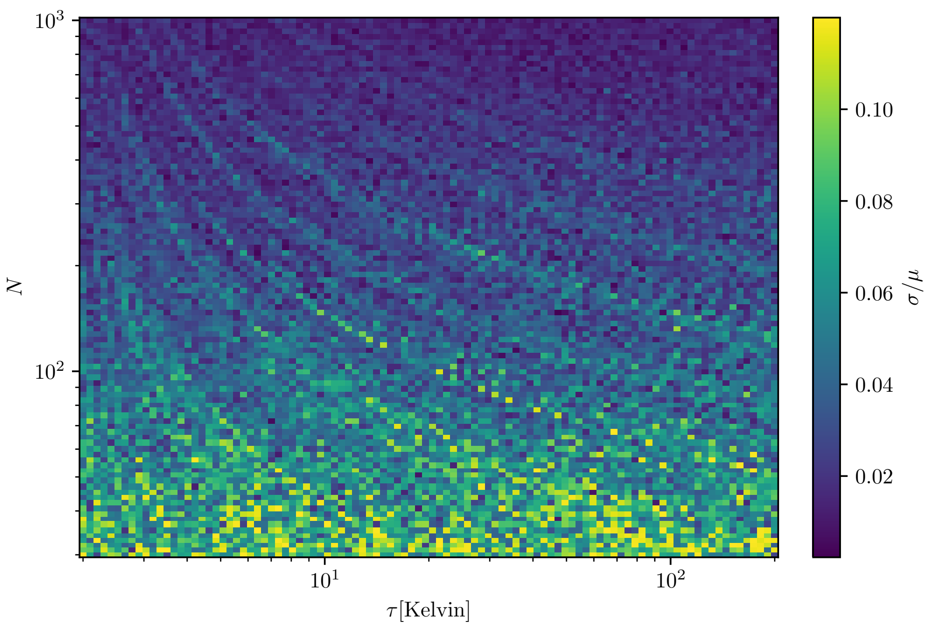


Figure A.2: The relative standard deviation received when generating figure 2.4. Naturally, in low N (low-densities), the randomness is larger.

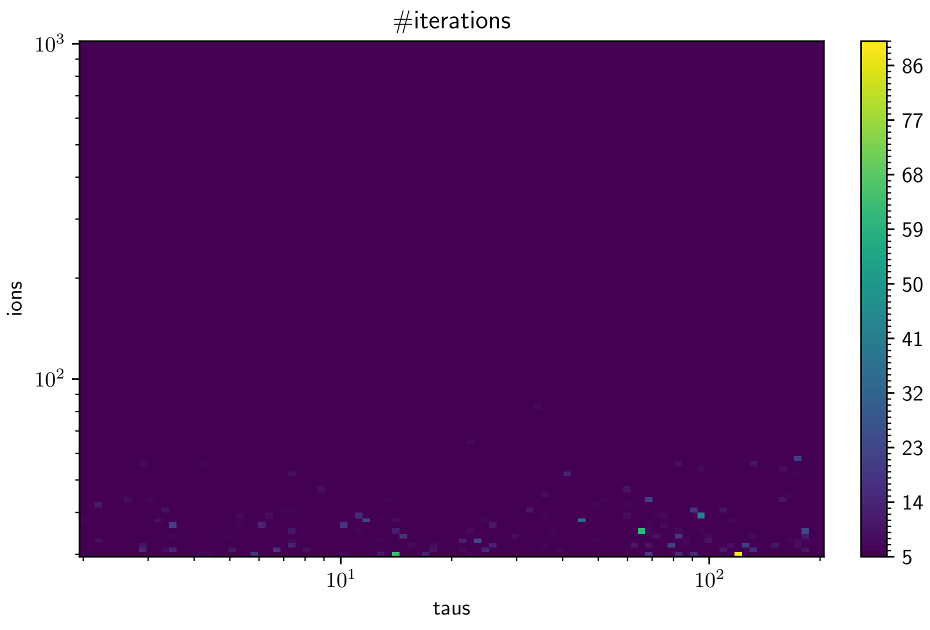


Figure A.3: The number of iterations that had to be performed to generate each point in figures 2.4. Almost always 5 random number generations is enough to reach a relative standard deviation smaller then 0.12, however sometimes a few more iterations are needed, naturally for small N (low densities).

Appendix B

Simulation Software Manual & Technical Details

The repository of the Simulations' software code is available online at: `https://gitlab.com/doronbehar/lab-ion-trap-simulations`. It consists of several Python executable scripts, each with a different role. The main important scripts are `sim.py` and `plot.py`. Running `./sim.py` creates a set of `.h5` suffixed files (in HDF5[The] format), and running `./plot.py` with the `.h5` files as arguments opens an interactive window for viewing the results, as depicted in figure B.1.

The following sections will simply describe each file in the repository, hopefully in an order that will make it easy to understand the complete architecture.

B.1 `flake.nix`: Setting up a Development Environment

Before we'll begin describing the actual Python scripts, it is important to setup a proper development environment for running the scripts. The best software I recommend for managing the development environment on a per-project basis, is the package manager *Nix*[DT]. Nix is a functional package manager [Dol06], meaning it is capable of producing (packages and) development environments from a pure set of inputs. The intention is to increase the likelihood that the resulted development environment will be reproducible [Del24] on any computer running Nix.

The main file that defines the development environment is `flake.nix`. If you have `nix` installed you should be able to run `nix develop`[1] in a command line shell and enter a Bash shell that would be capable of running any script in the repository just like I did - with the same versions of Python dependencies, and same LAMMPS version and build variant.

Don't decry these development environment instructions. Many crucial details and

[1]Don't forget to enable the (currently, as of writing) `experimental-features = nix-command flakes`. See the relevant Nix' Manual page for more details.

features in the Python scripts depend on specially built variants of the Python dependencies. Worth noting, are:

- 2 important Matplotlib patches. Without 1 of the patches, `./plot.py` won't work at all.
- An unreleased version of a Python dependency called `pint`, including another patch for it not accepted upstream.
- The `lammps` package built with very specific CMake flags that make the simulations run faster.
- A TeXLive distribution with specific dependencies that are necessary to create all the LaTeX generated texts in the plots.

Nix can be installed on Darwin, and any GNU/Linux distribution, including Windows' Subsystem for Linux (WSL). I personally used NixOS on (WSL) on the office's computer, with this OS configuration, which also includes a few NVIDIA hardware settings which might be relevant if you want to actually enjoy GPU support by LAMMPS. The fact the operating system's settings are declared and are reproducible just as the development environment, is another outstanding feature of Nix.

To reduce the hassle of running `nix develop` every time you enter the repository's directory, I recommend using `direnv`, which is also supported on the same platforms as Nix. The following sections assume you have a working development environment set up.

B.2 `sim.py`

`sim.py` is the script that actually calls LAMMPS' shared object[2], and runs the simulation, while also saving the results into a set of `.h5` files. The full set of available command line options is printed when you run `./sim.py --help`, and is explained in general in this section.

B.2.1 Parameters

The total amount of simulation parameters is approximately 22. Most of them are of type float, some are booleans, and some are integers. Although some of the parameters are constants for most simulations, simply iterating the multidimensional space of the rest of the parameters is computationally incomprehensible. The sensible thing to do instead is to study the effects of a few parameters one by one, while other parameters are constant.

[2]Also referred to as `DLL` in MS windows terminology.

`./sim.py` is basically built with a command line option per parameter, enabling you to choose which parameters to scan. Any command line option not specified will prompt you to choose the values to iterate over interactively, or a default value will be chosen.

Scanning-Enabled Parameters

Most of the float typed parameters' possible values are defined in `PARAMETERS_INFO` defined in `physical_constants.py`, along with complementary metadata per parameter. For example, the set of initial temperatures is marked as `T_init`, and possible values are $[5, 10, 20, 40, 80]$K, defined using `numpy.geomspace`. Similar ranges are defined for other parameters.

When you run `./sim.py` and you want to choose specific values of a certain float like parameter, you use the indices of the array's values defined in `PARAMETERS_INFO`. For example to scan the initial temperatures $[10, 40]$K, you would use `--T_init 1 3`. If you want to specify a certain range of values, you can use Bash shell syntax to choose e.g $[5, 10, 20]$K with `--T_init {0..2}`, which simply is expanded by Bash to `--T_init 0 1 2`. If the option `--T_init` is not specified at all, you will be prompted to pick the temperatures you wish to simulate over.

To view all available parameters and their values, `./sim.py --list` can be used, which lists all available parameters to the terminal in 2 tables. Tables B.10 and B.11 serve as a copy of these tables. Eventually, to run scans with a sensible amount of varying parameters (2-3 dimensions), you'd need to write `./sim.py` commands with lots of options, every time. This could be a nightmare without using Shell command line history, accessible with the keyboard's ↑ & ↓ keys. For a more versatile and efficient command line history browsing, I'd recommend using fzf with this Bash configuration.

Lastly, the `--seed` parameter allows one to choose a specific random number generation (positive integer) seed. Multiple values are treated as multiple initial conditions to perform - hence increasing the dimensionality of the scan. A negative integer argument like e.g `--seed=-5`, means 'take 5 random seed integers'.

Single value Parameters

The following parameters always get a single value, even if not specified explicitly on the command line:

- `--naive-freqs` (boolean): Put forces that generate $\sqrt{m_1/m_2}$ secular frequencies (default: False)
- `--micromotion` (boolean): Enable RF like micromotion (default: True)
- `--cooler {Be,Ca,Yb,Ra}`: Which cooler ion to use (default: [Yb])
- `--time-stabilizing`: How much time before cooling (default: [5.0ms])

- `--time-cooling`: How much time after cooling (default: [15ms])
- `--rf-divisor`: How many timesteps to put in a single RF cycle (default: [1597]), see also section 2.4.
- `--windows-attenuation`: By what fraction the intensity decreases after passing 2 windows (default: [0.7])
- `--approximate-laser-force` (boolean): Simulate laser cooling with the approximated $\alpha \cdot v$-like expression
- `--offset x[mm] y[mm] z[mm]`: Put the ion cloud in an offset from the center (default: [0.0, 0.0, 0.0])

`./sim.py` doesn't allow scanning over values of these parameters. However if you really want to you can use GNU's `parallel`[Tan11] program to iterate such values artificially. Below are a few example usages:

```
parallel --tmux \
  ./sim.py \
    --x_secular_freq 16 --y_secular_freq 16  --z_secular_freq 2 \
    --rf_freq 2 \
    --omega 7 --delta 13 --laserAngles 22
    --cloud 10 --total 9 \
    --T_init 1 \
    --seed {90..93} \
    --time-stabilizing 15 --time-cooling 0.1 \
    --rf-divisor ::: 1301 1381 1427 1523 1583
```

This command tests the effect of the RF divisors on the stability and convergence (see 2.4). Note that `--seed {90..93}` is expanded by the shell and eventually interpreted as `--seed 90 91 92 93`, which means: Run each simulation with 4 different initial conditions defined by the seeds $90 - 93$. Using a constant set of seeds, and not `-seed=-4`, makes sure that every process initiated by GNU `parallel` will use the same 4 initial conditions, and not 4 randomly picked initial condition seeds (picked by each `./sim.py` process).

```
parallel --tmux \
  ./sim.py \
    --x_secular_freq 16 --y_secular_freq 16  --z_secular_freq 2 \
    --rf_freq 2 \
    --omega 7 --delta 13 --laserAngles 22
    --cloud 10 --total 9 \
    --T_init 1 \
    --seed {90..93} \
    --time-stabilizing 15 --time-cooling 0.1 \
  ::: --{no-,}naive-freqs ::: --{no-,}micromotion
```

In the above, the 2×2 boolean matrix of `naive_freq` and `micromotion` is measured. A similar command was used to generate figure 2.8. The `{no-,}OPTION` syntax is expanded by the shell to `--no-OPTION --OPTION`, and GNU `parallel` runs 1 `./sim.py` process with `--no-OPTION` and another with `--OPTION`, and does the same for the other list of arguments appearing after the 2nd `:::`.

Other command line options

Besides the obvious `--help` option, a few more non physical options of `./sim.py` are of worth noting:

- `--date`: When creating `.h5` files, part of the files' name include a date string. When you want to easily allocate the files later, this option can be used.
- `--coulomb`: When simulating, measure the coulomb energies into a dedicated HDF5 group. This data can be plotted in a separate figure window with `./plot.py --show-coulomb`.
- `--high-resolution`: When simulating, extract the velocities and the positions every simulation step, and not only in the beginning of every RF cycle. NOTE this makes the simulation run much slower due to slow excessive disk writing.
- `--list`: Don't actually simulate, only list the available parameters. Essentially prints into the terminal tables B.10 and B.11.
- `--cores`: How many CPU cores to use when distributing jobs, defaults to the number of CPU cores available on your machine. See section B.2.4 for more details.

B.2.2 Managing Simulation Parameters with `xarray`

Since we have so many simulation parameters, and we are interested in manipulating their results in various ways, it can be incomprehensibly hard to do it with traditional multi-dimensional Numpy arrays. Almost all of the repository uses `xarray`[HH17] to manage arrays of 2 dimensions and more. With `xarray` you can be absolutely sure you are performing operations on the right dimensions, no matter how they are ordered internally.

Additionally, `xarray`-like objects can be saved to files of various formats, like HDF5[The], *Zarr*[3] and more. Specifically in this project, the HDF5 format was picked, because:

[3] https://zarr.readthedocs.io/en/stable/

- (Like Zarr), it has a concept of Groups, allowing you to save the xarray objects into 1 group, and other data in another group.
- In comparison to Zarr, HDF5 files are single files, and not directories, making them less prone to corruption due to cloud storage sync conflicts, and manual recursive copying issues.
- Can be faster then Zarr when reading data that needs to be read fast enough for animations like in figure B.1.

There are also a few worth noting advantages of Zarr over HDF5:

- The parallel writing support, which could have been an advantage for section B.2.4.
- Somewhat related to the above advantage, and also explained in section B.2.3, appending simulation data like positions and velocities can be done a bit more efficiently with Zarr, as this operation doesn't require extending the file and make sure all the pointers in the file point to the correct place.

These Zarr advantages were realized in a late part of the research period, and were not assessed thoroughly.[4]

The format in which simulations results were saved is described below:

- Each scan of parameters is saved to a single HDF5 file.
- The parameters scanned over are defined via an `xarray.Dataset` saved into an HDF5 group named `ranges`.
- Each dimension in the dataset is named like the parameters in tables B.10 and B.11, as described in section B.2.1.
- Each scanning point, has a hash, calculated[5] from the dictionary of input parameters values.
- The hash is part of the saved `ranges` dataset, and it serves as a link to actual simulation measurements of that set of input parameters.
- Per hash `h`, the HDF5 group `measurements/{h}` holds all the simulation's raw results, in a hierarchical format described in the next section.

[4]Also due to rough edges with the Zarr 2.18 $\rightarrow$ 3.0 version update, that happened around that time.
[5]Using DeepHash.

B.2.3 `measurements/` HDF5 groups format

The raw simulations' results are not suitable to be modelled as an xarray object. They are calculated much more dynamically, and include much more data in comparison to the `xarray.Dataset` saved in the `ranges` HDF5 group, and hence invite a more dynamic format that allows appending and truncating data without loading it all into memory. The next subsections describe the HDF5 'paths' to the HDF5 datasets.[6]

`times`

The first direct HDF5 dataset in the measurement group: A simple, usually[7] linearly spaced time stamps, that always starts with 0, and is appended floating point numbers as the simulation's time proceeds.

`mass=AMU` HDF5 Datasets

Since we are interested in measuring every ion species separately, every HDF5 group described in the next subsections, contains at least 1 HDF5 dataset in a path that ends with `mass=AMU`. If the simulation's `cloud`$\times$`total` $\in (0,1)$, the paths `mass=X` & `mass=222` will be included - for the masses of the cooler and target $CHDBrI^+$. It may also be that `cloud`$\times$`total` $\in \{0,1\}$, and then there will be only a single `mass=AMU` there.

The `cloud` parameter explained in section B.2.1, and the `--cooler` parameter in section B.2.1. Table B.11 lists all possible amounts of coolers v.s targets, including those in which `cloud`$\times$`total` $\in \{0,1\}$.

`positions` prefixed Groups

There are 3 types of positions arrays, each with shape `(M, N, 3)`, where `N` is the number of ions of the species, and `M` is the number of time samples. They are saved under the following groups:

- `positions_rf_min`: The positions measured when the RF oscillation is at it's minimum.
- `positions_rf_max`: The positions measured when the RF oscillation is at it's maximum - at the middle time step of the oscillation.
- `positions`: When `--high-resolution` is not used, the arrays in this group are identical to those saved to `positions_rf_min`. If it *is* used, the positions in *every* time step are saved there.

[6]Note the slightly confusing terminology: An `xarray.Dataset` is not an HDF5 dataset - these are 2 completely different Python objects.

[7]Why not always? See section B.3

`velocities` prefixed Groups

Very similarly to the `positions` prefixed groups, the analogous `velocities` groups are saved as well. An additional group named `velocities_rf_averaged` is also saved, and it includes the per particle, per axis velocities averaged over an RF cycle, as described in section 2.5.3.

The `temperatures` Group of Groups

Per every `velocities` and `positions` related group of HDF5 Datasets, multiple 1 dimensional HDF5 datasets of length `M` are saved into the `temperatures` group. Given a `T_coord` & `T_rf_type`, there are 5 `T_methods` one can use to get a temperature, as described in section 2.5.2 and table 2.2. The full template of the paths to these per mass HDF5 datasets, under the `measurements/{h}` group, is:

```
temperatures/{T_coord}_rf_{T_rf_type}_{T_method}/mass={AMU}
```

Note that no `temperatures/potisions_rf_averaged*` datasets are saved, as explained in section 2.5.4.

B.2.4 Scan Parallelizing Algorithm (`parallel_hdf5_splitting.py`)

Running a single simulation of $20 - 30$ms can take ≈ 30min. Hence scanning can obviously take days, depending on the dimensionality of the scan. A great way to speed up such scans is via distributing scanning points to different CPU cores. The algorithm described below, implemented in `parallel_hdf5_splitting.py`[8], is in charge of deciding how to distribute the scanning points to CPU cores.

The splitting algorithm is best explained using examples. In table B.1, C is the argument given to `./sim.py --cores`, that defaults to the number of CPU cores on the machine (usually 8 or 16). C^* is the resulted number of CPU cores the scan is distributed to, decided by the algorithm ($C^* \leq C$).

The definition of C^* is simple to explain in general: If we'll denote T to be the total number of scanning points, C^* is the highest divisor of T such that $C^* <= C$. The progress bars `./sim.py` creates, are aware of the sub processes' progress, and updates them accordingly.

When `./sim.py` runs, it creates multiple files in the following scheme:

```
scan@{basic_parameters_hints}@vars=({v1,v2,v3})@{date(core#)}.h5
```

Where:

- `basic_parameters_hints` is a comma separated list of `param=value` strings of specific parameters worth mentioning early upfront in the file name - parameters that should have substantial influence on the results.

[8]Was implemented with https://claude.ai.

Original Shape	**C**	**C***	**Each Core's Shape**
$(7,3)$	7	%	$(1,3)$
$(7,3)$	8	7	$(1,3)$
$(5,4)$	5	%	$(1,4)$
$(3,4)$	6	%	$(1,2)$
$(12,7)$	7	%	$(12,1)$
$(12,8)$	8	%	$(12,1)$
$(12,6)$	8	%	$(4,2)$
$(4,5,6)$	10	%	$(2,5,1)^*$
$(2,3,4)$	8	%	$(1,3,1)$
$(1,1,1,1,7,8,2)$	7	%	$(1,1,1,1,1,8,2)$
$(1,1,1,1,9,4,5,1,1,1)$	9	%	$(1,1,1,1,1,4,5,1,1,1)$
$(1,1,1,1,9,4,5,1,1,1)$	15	15	$(1,1,1,1,3,4,1,1,1,1)$
$(100,100)$	25	%	$(20,20)^*$
$(11,13)$	11	%	$(1,13)$
$(11,13)$	10	1	$(11,13)$

Table B.1: Test cases for `./sim.py` splitting function. $C^* = \%$ means $C^* = C$, as in most of the cases. A shape superscripted with * is marking a shape which is 1 solution of the algorithm among a few more valid solutions possible.

- `{v1,v2,v3}` is comma separated list of float like parameters that were scanned, from section B.2.1.
- `date` is the date specified via `--date` (defaults to the OS' date).
- `#` is the index of the CPU core upon which these simulations were scanned.

While a scan is running, that file is added a `.now-scanned` suffix, to help you avoid touching it while it runs, and the parent `./sim.py` script should remove this prefix when the simulation is finished. Unfortunately, there is a hard to debug but minor issue with this parallelisation, that might cause this renaming to fail, but this can be easily solved later with `sim-continue.py` as described in the next section.

B.3 `sim-continue.py`

Imagine you run a long scan that takes a day or two, and something crashes in the middle of the night. Letting go of all the results obtained so far would be too expensive, and if there's a bug that caused this crash, you'd have to run the same command and wait an insensible long time for it to happen again. This scenario invites writing a script, named `sim-continue.py` that will be able to recover from such crashes, and perhaps be able to do a bit more.

The basic recovery usage of it is:

```
parallel ./sim-continue.py ::: scan*.h5.now-scanned
```

Where the `.now-scanned` suffixed files are scan files that were not finished (or when finished but not renamed, as described in section B.2.4). In principal `./sim-continue.py` accepts only a single `.h5` file, so GNU parallel [Tan11] can very useful with it.

The next subsections, explain how to use `./sim-continue.py` with successfully finished scans - scan files ending with `.h5`, and a few details regarding the `.h5` file naming schema.

B.3.1 Simulating More Time then Originally Prescribed

Sometimes, you look at the results of a particular scan point with `./plot.py` (see section B.6), and you want to extract a few simulations and run them for a few more ms. This is possible with `./sim-continue.py`'s `--time-cooling` option, which replaces the original cooling time with the one you specify. Likewise, for the sake of UI symmetry, the option `--time-stabilizing` also exists and replaces the original stabilization time.

B.3.2 Simulating with a Finer Time Division

Another common scenario where `./sim-continue.py` is very useful, is running part of a simulation from a certain point in time, but with a higher RF divisor. You can observe example results where this is needed in section 2.4. The relevant self-explanatory command line options are `--rf-divisor` and `--from`. The argument to `--from` is a time point in ms, rounded downwards to the nearest time point found.

B.3.3 Removing Abruptly Some Ions

One very rarely used, yet still maintained feature of `./sim-continue.py` is the `--remove TYPE FRACTION` command line option. Basically it enables you to remove a certain `FRACTION` of the ions of group `TYPE`, at the time point specified by `--from`. Results where this option was used where not presented in the thesis, but it was still used during the research period to debug several physical phenomena, now well understood.

B.3.4 File Names Details

When `./sim-continue.py` is given an unfinished `.h5.now-scanned` file path, it copies it to a file with the extension `.h5.now-appended`, and tries to fill it. Then when it finishes, the `.h5.now-appended` file is renamed to `.h5`.

Somewhat similarly, when `./sim-continue.py` is given a (hopefully valid) `.h5` suffixed file, a `.h5.now-appended` file is still created and filled to, and renamed upon completion to `.h5`. However, for a valid `.h5` file, and when one of the `--time-*` or `--rf-divisor` options are used, the `basic_parameters_hints` part of the file name[9] will change, and thus ensure you won't override the original `.h5` file.

[9] Explained in section B.2.4

B.4 `sim-reconcile.py`

Another likely scenario one may encounter when researching, is best explained in the following example: Assume you ran the scans marked in table B.2, using e.g a 2 dimensional scan of parameters $(A, B) \in \{a_1, a_2\} \times \{b_1, b_2\}$, and then a 1 dimensional scan of $B \in \{b_2, b_3\}$ with $A = a_3$. You may suspect there's some coupling between parameters A and B, and you want to get a full picture of the dependence. Therefore you wish to simulate the missing configurations, marked with $\times$ in table B.2.

	b_1	b_2	b_3
a_1	✓	✓	×
a_2	✓	✓	×
a_3	×	✓	✓

Table B.2: Example of partially overlapping scans requiring reconciliation. ✓ indicates completed simulations; × indicates missing ones to be filled.

Since there is no guarantee the collection of missing scanning points iterated is a full grid, the file name template of scans generated by `./sim-reconcile.py` is:

```
scan@reconciling@vars=(...)@{date}.h5
```

Where the only dynamic part of it is `{date}`, and the `.now-scanned` suffix is added during the scan (and removed when it finishes successfully). In principle, it should be possible to parallelize this scan too, but this feature is not yet implemented.

B.4.1 Merging Threshold

When `./sim-reconcile.py`, and other scripts described further on, read multiple `.h5` files, they merge all scans' `ranges` xarray datasets, using the default `join="outer"`[10] argument, which means: Unionize all simulations' input parameters, and fill the missing values of data variables with `NaN`. In our case, our only data variable is called `hash`.

Now what if 2 scans have cooling times of 15.05ms and 15.0ms? The naive behavior of xarray would be to consider these two values different. Unfortunately, this can cause the merged `xarray.Dataset` to be very sparse, when in fact we'd probably like to consider these 2 points the same. This is why merging scans' `ranges` parameters is done using a non-trivial merging algorithm[11] called 'smoothly merging datasets', described briefly in the next paragraph.

Per dimension, compute a 'cluster' of distances between the values normalized to their mean.[12] Now if there are 2 values in the cluster in a distance d apart, they are considered the same. Then, the only thing left is to group the values considered the

[10]See xarray.merge documentation.
[11]Developed with `https://openai.com/`.
[12]Implemented with `linkage` and `fcluster` functions from `scipy.cluster.hierarchy`[VGO+20].

same by the clusters' into, and take the only valid hash we find there among NaN hashes. If multiple non-NaN hashes are found in each such closely located parts of the cluster, an error message is raised. The d parameter defaults to 0.01, and can be modified in the command line with `--merge-threshold`.

B.5 `time-plot.py`

The most basic plotting functionality is plotting multiple simulation results as a function of time, on the same axes. The per spatial axis' standard deviation, and the temperatures, are the only time dependent variables calculated by the code, as introduced in section 2.5 These are selected via the `-y` command line option, and it defaults to temperatures.

Several worth noting features of `time-plot.py` (common also to `plot.py`) are explained in the next subsections.

B.5.1 Handling Multi-Dimensional Scans

Differentiating between the lines is done by the script via the following line properties, termed *displayers*:

- Colors: (easily differentiable:[13])
- Line Styles (—, ····, - - , -·- etc.)
- Markers ($\times$, $\circ$, $\cdot$ etc.)

The `ranges` `xarray.Dataset` saved in each `.h5` file given as argument, is read and merged before the dimensions and displayers are iterated. All non-trivial dimensions of the merged `xarray.Dataset` can be represented by any of the above displayers. Besides the standard simulation parameters, the masses are also considered as a potential dimension to iterate over. If a mixed species cloud was simulated, a `mass` dimension is added, and it can be displayed by one of the displayers above. An algorithm[14] to choose a displayer per a dimension, is described below.

The displayers itemized above are ordered according to their clarity when displayed on axes, with the clearest displayer being the colors. The non-trivial dimensions of the scans, including the mass dimension, are also ordered by their size, starting with the largest. Then the algorithm matches each dimension to each displayer, until all displayers are consumed. When more non-trivial dimensions exist, and no more displayers are available, the dimensions left are marked 'to be averaged over'.

An exceptional dimension in the above iteration is the initial conditions' `seed`, which the algorithm by default marks it to be averaged over. The command line

[13]Provided by Matplotlib via `rcParams["axes.prop_cycle"]`.
[14]Also written with https://openai.com/.

options `--color`, `--linestyle`, `--marker`, `--average` can be used to modify the default behavior of the algorithm, which might be a bit arbitrary if dimensions of the same length are in the scan.

`./time-plot.py` prints to the terminal a table of the matching made between displayers and dimensions, along with their lengths. Tables B.3, B.4 and B.5 demonstrate how this matching is printed, for several different displayers command line arguments.

	T_i[K]	Ω/Γ	Seed	t_c[ms]
coordinate length:	5	9	3	2
displayed by:	linestyle	color	average	marker

Table B.3: The default match made by the plotting scripts between displayers and `xarray.Dataset` dimensions, for a scan with only 1 mass.

	T_i[K]	Ω/Γ	Seed	t_c[ms]
coordinate length:	5	9	3	2
displayed by:	linestyle	color	average	marker

Table B.4: The dimensions-displayers match for the same `.h5` files as in B.3, but when `--color omega` was given.

	T_i[K]	Ω/Γ	Seed	t_c[ms]
coordinate length:	5	9	3	2
displayed by:	color	linestyle	marker	average

Table B.5: Like B.4, but with `--color T_init --marker seed`.

The algorithm is flexible enough to use a different displayer for a dimension if you picked it with the command line, and never discard a displayer unless you chose enough dimensions to `--average` over. Averaging over multiple dimensions is done simply via multiple arguments to `--average`.

B.5.2 Handling Different Traps & Offsets (`collapse_ds.py`)

A common scenario one may encounter is the desire to choose a different displayer per trap. Take as an example the secular frequencies in figure Since the trap's secular frequencies are modeled as 3 parameters, before even assigning displayers to dimensions, `xarray`'s merging will create a large and sparse `xarray.Dataset`, with every possible combination of all secular frequencies used.

The natural thing to do of course is to consider instead a single dimension per trap. This is termed in our context as *collapsing*, and can be done using `--collapse-trap`. If you have a scan where (f_x, f_y) were changed together, but f_z was constant, `--collapse-trap xy` can be used (`xyz` is the default argument).

Very similar to the trap's frequencies, a similar `--collapse-offset` has the same possible arguments and behavior, acting upon the offsets. This functionality is implemented in `collapse_ds.py` and can be tested by executing it.

B.5.3 Showing `pwlf` Fit Results

The piecewise fit results, described in section 2.5.5, are added to the plot if the command line options `--show-fits` and/or `--show-regimes` are used. `--show-fits` simply plots the lines of the fits in addition to the raw measurements, and `--show-regimes` also plots an opaque background marking the cooling regimes detected by `pwlf`[JV19]. Since only 1 pair of regimes can be sensibly plotted on one axes, the fits' break points are averaged over all dimensions of the scans.

B.5.4 Miscellaneous Options

- Since reading multiple datasets involves merging, the same merging behavior described in section B.4.1 is applied here, and `--merge-threshold` is available as well.
- Saving the plot in any format supported by Matplotlib is possible using `--save-fig`, with the format specified by the given file's extension. If '`-`' is used as a file path, the contents of a PDF file containing the plot are printed to `stdout`, useful for piping it straight to the clipboard.
- `--wfrac` Takes the intended LaTeX `\linewidth` fraction used when including the saved figure.
- `--hprop` Takes the height/width proportion to use in the generated figure.
- `--always-show`: If saving, open the figure before finishing.
- `--legend`: When saving, put legend in the specified location. Use 'none' to disable legend altogether.
- `--legend-columns`: Specify the amount of legend columns to use. Relevant only when saving, and when using a legend location outside and 'upper,lower'. The default value, is picked by the maximal input datasets' dimensions.
- `--log` makes the y axis scale logarithmically.
- `--list` makes the script not open any figure, and instead prints information related to the simulation and displaying parameters. Various formats are supported, and can be specified as optional arguments to `--list`.
- `--where` accepts 2 arguments, a dimension, and an index of a value of a parameter. The list of values and their indices can be printed with `--list all`. Choosing multiple values `v1, v2, ...` of a dimension `d`, can be done with `--where d v1 --where d v2 ...`.

- `--mass-conf` allows you to reduce the results dataset by performing simple operations upon the masses. One option is to use an argument of the form `m=AMU` that selects the given mass. A 2nd option is performing a mathematical operation between the light v.s heavy mass. The special argument `a-h-l` is useful for `-y E_c` only, for showing the Coulomb interaction energy between the species. Using this option reduces the amount of displayers required to display all dimensions on the axes.
- `--smooth` smooths the time dependent data before analyzing it, using Scipy's `make_smoothing_spline`[15] and given a λ parameter.

Additional optional arguments like `--T_coord`, `--T_rf_type` and `--T_method` are described in table 2.2.

B.6 `plot.py`

`plot.py` is the main script with which you can view simulations' results of any kind, interactively, as shown in figure B.1. The basic command line usage is simply passing it `.h5` arguments. Many other optional command line arguments share the same behavior with other scripts, as described briefly below:

- `--collapse-trap` and `--collapse-offset` work as described in section B.5.2, though for here it is more relevant for the axes described in section B.6.1.
- `--merge-threshold`, `--where` & `--list` work exactly as described in section B.5.4.
- `--T_coord`, `--T_rf_type` & `--T_method` also work very similarly to `./time-plot.py`, and here they are relevant for axes described in sections B.6.1 and B.6.4.

B.6.1 Top Right: Summarized Results' Dependence on Simulation Parameters

The main purpose of `./plot.py` is to display *summarized* simulation results, on a single axes, as described in section 2.5.5. For a 1 dimensional scan of parameter p, the summarized results are plotted as a function of p, where results relevant of different masses in each simulation, are differentiated by color. This plot is termed here and in software as *x-vs-y*.

Like with `./time-plot.py` (see section B.5.1), each varying dimension of the scan is attached a *displayer*, with the horizontal axis considered a displayer as well, termed simply `x`. The `x` displayer is picked first - for the most varying dimension, and can be picked manually with the command line argument `-x`, just like `--color`, `--linestyle` and `--marker`.

[15]See https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.make_smoothing_spline.html.

Deciding which summarized results are displayed, can be done either in the GUI via the radio-buttons found at the right-bottom, or via the command line arguments described in tables B.6 and B.7.

-y choice	**Explanation**	**Dependent Arguments**
T	The temperature of the clouds [Kelvin]	`--part`
size	Cloud size [mm]	`--axis` `--part`
coolFreq	The cooling frequency [kHz], as calculated by cjekel's PWLF (PieceWise Linear Fit) algorithm [JV19]	`--regime`
cloudFreq	The cloud's collective movement most dominant frequency [kHz]	`--axis` `--freqMethod` `--freqOperator`
E_c	Coulomb energies[16] [eV]	`--part`

Table B.6: Available choices for the `-y` argument and their meanings.

Argument	**Explanation**
`--part`	The simulation part in comparison to cooling time point to use in the calculation (options: `stabilizing`, `cooling`, `s-f`, `f/s`; default: `stabilizing`)
`--regime`	The index of the cooling regime that was detected by cjekel's PWLF (PieceWise Linear Fit) algorithm [JV19] (options: `0`, `1`; default: `0`)
`--axis`	The axis for which to focus the calculation upon (options: `x`, `y`, `z`; default: `x`)
`--freqMethod`	How to compute the dominant movement frequency of the cloud (options: `fft`, `fit`; default: `fft`)
`--freqOperator`	How to compute the dominant movement frequency of the cloud (options: `none`, `diff`, `ratio`; default: `none`)

Table B.7: Arguments that other `-y` choices may depend upon.

Last notes:

- The *x-vs-y* figure can be saved to a file via the `--save-fig` command line option.
- If the `x` displayer is set by chance to the simulation parameter `omega`, you can use the boolean flag `--intensity` to make the x axis of this figure scale like laser intensity, as in equation 2.22.

B.6.2 Bottom Left: Scan's Dimensions Sliders

As described in section B.6.1, multiple dimensions of the input `.h5` files are attached to *displayer*s. Like with `./time-plot.py`, the `seed` parameter is averaged over by default, along with potentially other dimensions. The 3D animation (section B.6.3), and the time-dependent plot (section B.6.4), always display a single simulation, which can be picked by the sliders found in the bottom-left part of the figure.

Controlling the values of each slider can be done by clicking the mouse near a tick, and also via the keyboard shortcuts in table B.8. The command line option `--interactive-init` can be used to start the GUI in a specific state of the sliders, by choosing interactively a specific simulation in the command line - using *Beaupy*[17] and iterating the non-trivial dimensions.

displayer	+	−
x	ctrl + →	ctrl + ←
color	ctrl + ↑	ctrl + ↓
linestyle	ctrl + j	ctrl + k
marker	ctrl + n	ctrl + m
average	ctrl + u	ctrl + i

Table B.8: Keyboard shortcuts for iterating dimensions values, per their attachment to a *displayer*. ± is increasing/decreasing the dimension's value.

Lastly, ctrl + h can be used to save the currently focused simulation to a single file, which can be manipulated and inspected later for various purposes.

B.6.3 Top Left: 3D Animation

The central part of the GUI is the 3D animation showing the particles' movement in time, of the simulation picked by the sliders. Matplotlib allows to easily changing the view angle[18], and zooming in and out[19] with the mouse. A few relevant command line options for the animation are available:

- `--no-animate`: Disables animation - useful when you want to observe each frame.
- `--save-video`: Saves a video animation of the particles moving, including the rest of the GUI axes and other elements. To simulate a specific simulation, `--interactive-init` can be used.
- `--range`: Specifies a limit L in mm for the 3D space, used as $\pm L$ from the origin for all spatial dimensions.
- `--positions-label`: Choose a different positions label scheme. Defaults to `full`, which means show what masses are trapped, and how many of them are trapped at any given time. Other options are self explanatory: `only-mass`, `none`.

[17] https://petereon.github.io/beaupy/
[18] See https://matplotlib.org/stable/api/toolkits/mplot3d/view_angles.html
[19] See https://matplotlib.org/stable/users/explain/figure/interactive.html#interactive-navigation.

- `--separate-positions-figure`: Put the positions' axes in a separate figure. Useful for saving images of the 3D particles (projected onto 2D), in selected points in time

B.6.4 Middle Right: Time Dependent Results

This plot is very similar to plots generated by `./time-plot.py`, only that it shows only on a single simulation - the simulation picked by the sliders. Since only a single simulation's results are displayed, the piecewise fits and cooling regimes are always presented, unless the simulation is marked as ignored, in which case there are no fits available to begin with.

The command line option `--log` can be used to start the GUI with this plot's vertical axis displayed in a logarithmic scale. When the GUI is open, ctrl+l can be used to toggle on/off the logarithmic scaling.

B.7 `histogram-plot.py`

To plot the histograms in figure 3.6b, this script can be used. It acts similarly to `time-plot.py` and `plot.py` with regards to handling of multi-dimensional scans, but it includes one special command line option explained below.

To generate histograms with enough statistics and fine bins spacings, one has to take kinetic energies of the particles over many time steps. The script is hard-coded to divide the stabilization time period to 2 segments, and the cooling time period to 10 more segments. These segments are termed *time windows* by the script.

By default, the last cooling time window is used, but multiple time windows can be specified by the `--time-windows` option, using indices starting at 0, ending at 11. When more then 1 time window is specified, this dimension is added to the plot, and colors / markers / linestyles can be used to vary the time windows, just like in section B.5.1.

B.8 `h5doctor.py`

Throughout the research period the format in which simulation results are saved has changed a lot. To verify our scan files are valid and can be plotted, and to also apply a sort of quality assurance, this script was written.

All modifying actions this script may perform, depend on the input `.h5` files successfully going through a check phase. The `./h5doctor.py check` command can be used to perform only a check, which also verifies many HDF5 attributes related details are intact. Other actions possible of `h5doctor.py` are described in the next sections.

B.8.1 Modifying Parameters

A particular kind of change the simulations' scans can go through, is changes to the `ranges` `xarray.Dataset`, described below:

- Parameters renamed.
- A floating point parameter was rescaled by a scalar.
- A new parameter added (happened often).

When a new parameter is added, it is likely that old scans are still valid, but their `ranges` simply don't include that parameter but had it set to an implicit value. All of these scenarios (and a few more), are handled by `./h5doctor.py`.

Since the `ranges` `xarray.Dataset` also includes a `hash` `xarray.DataArray`, calculated based on the simulation's parameters, `./h5doctor.py` is also carefully changes these hashes' values, and renames the pointers in the `measurements/` HDF5 groups (see section B.2.3).

B.8.2 Coulomb Energies

When `./sim.py` is used with `--coulomb`, it saves a few time-dependent Coulomb energies HDF5 datasets, that can be plotted with `./plot.py --show-coulomb`. Usually these energies are not very interesting, but sometimes, it can be interesting to look at the Coulomb energies after the simulation finished. Without requiring simulating everything again with `./sim.py --coulomb`, `./h5doctor.py coulomb --save` can fill in those HDF5 datasets based on the positions.

Worth noting: `./sim.py --coulomb` is using a LAMMPS[TAB+22] function to calculate these energies, whereas `./h5doctor.py coulomb` uses *Freud*[RDH+20] to iterate all the pairs and sum the $1/r$ distances. Hence, by default this subcommand verifies there is a small numerically negligible difference between these two calculation methods and thus verifies the usage of both of these functions.

B.8.3 Ignoring Measurements

As described in section 2.4, sometimes only after a long scan has finished, you find out a certain specific simulation has had a numerical error due to a too coarse RF divisor, which change the summarized results so substantially, that they should be ignored. Ignoring is done via a dedicated HDF5 attribute that can be set or removed by `./h5doctor.py ignore`.

Handling multiple `.h5` files is done seamlessly, and usually you don't have to know which `.h5` file contains the simulation you wish to ignore. Rather, `./h5doctor.py ignore select` uses *Beaupy*[20] to help you find easily the simulation you are interested in. The other `./h5doctor.py ignore` subcommands are:

[20] https://petereon.github.io/beaupy/

- `search`: Search and report any ignored simulations
- `clean`: Clean all ignored simulations

B.9 Helpers, not Dealing Directly with Simulations

All other Python files in the repository collect functions related to a certain topic in simulating or plotting. Most of them are also executables, allowing to test the functions implemented in them, in various ways. Table B.9 briefly describes them.

Name	**Purpose**	**Execution**
`analyze.py`	Analyzing results, and structuring the results also into a `xarray.Dataset`.	Shows the Python representation of the computed `xarray.Datasets`, given `.h5` files.
`boltzmann.py`	Implement and tests various temperature computations, and temperatures-dependent distributions.	Given trap secular frequencies and other parameters, can print and plot statistics of distributions.
`coulomb.py`	Implement Coulomb energies calculation, using Freud'sA[RDH+20] `locality.AABBQuery - allowing` to iterate all nearest-neighbors efficiently.	Can calculate & plot data like presented in section 2.1.3 and in appendix A.
`laser_cool.py`	Implement laser cooling related functions.	Opens a Matplotlib plot solving the equation of motion of a single particle under a harmonic force and the laser cooling force. Including sliders for various parameters that affect the results, similarly to figure 2.1.
`mathieu.py`	Implement a `Trap` class solving the Mathieu equations given, as described in section 2.1.2.	Can open figures 2.2, 2.1, 2.3 & 2.7.Is also capable of exploring the dependence of $\{(a_i, q_i)\}$ solutions given β_i in other ways.
`optimal_cooling.py`	None for simulating and plotting.	Calculates a theory that was not fully verified in this work, and wasn't important enough, and hence was left out.
`physical_constants.py`	Put all physical and technical parameters in the same place.	Does nothing.
`pwlfWrapper.py`	Wraps the PWLF algorithm to ease handling of the fit parameters' uncertainties [Leb10].	Does nothing.
`bolg.py`	Implements an algorithm (written by Claude.ai), that performs an opposite of file path globbing (hence the name). Used for picking figures' titles.	Tests the algorithm using simply file paths arguments.

Table B.9: All helper Python files in the repository.

B.10 Reference Tables & Figures

Table B.10: The list of other float typed parameters with their indices and values, created using `./sim.py --list latex-most`

	f_x[kHz]	f_y[kHz]	f_z[kHz]	$f_{\rm rf}$[kHz]	Ω/Γ	δ/Γ	Laser's $(\theta, \phi)^*$		T_i[K]
							0	0	
0	0.50	0.50	0.50	20.00	0.10	-15.00	67.5	0	5.00
							112.5	0	
							0	0	
1	1.00	1.00	1.00	30.00	0.25	-13.97	67.5	0	10.00
							45	45	
							0	0	
2	1.50	1.50	1.50	40.00	0.40	-12.93	67.5	0	20.00
							45	-45	
							0	0	
3	2.00	2.00	2.00	50.00	0.55	-11.90	67.5	0	40.00
							-45	45	
							0	0	
4	2.50	2.50	2.50	60.00	0.70	-10.87	112.5	0	80.00
							45	45	
							0	0	
5	3.00	3.00	3.00	70.00	0.85	-9.83	112.5	0	
							45	-45	
							0	0	
6	3.50	3.50	3.50	100.00	1.00	-8.80	112.5	0	
							-45	45	
							0	0	
7	4.00	4.00	4.00	200.00	2.50	-7.77	45	45	
							45	-45	
							0	0	
8	4.50	4.50	4.50	400.00	4.00	-6.73	45	45	
							-45	45	
							0	0	
9	5.00	5.00	5.00	800.00	5.50	-5.70	45	-45	
							-45	45	
							67.5	0	
10	5.50	5.50	5.50		7.00	-4.67	112.5	0	
							45	45	

Table B.10: The list of other float typed parameters with their indices and values, created using `./sim.py --list latex-most`

	f_x[kHz]	f_y[kHz]	f_z[kHz]	f_{rf}[kHz]	Ω/Γ	δ/Γ	Laser's $(\theta, \phi)^*$		T_i[K]
11	6.00	6.00	6.00		8.50	-3.63	67.5 112.5 45	0 0 -45	
12	6.50	6.50	6.50		10.00	-2.60	67.5 112.5 -45	0 0 45	
13	7.00	7.00	7.00			-1.57	67.5 45 45	0 45 -45	
14	7.50	7.50	7.50			-0.53	67.5 45 -45	0 45 45	
15	8.00	8.00	8.00			0.50	67.5 45 -45	0 -45 45	
16	8.50	8.50	8.50				112.5 45 45	0 45 -45	
17	9.00	9.00	9.00				112.5 45 -45	0 45 45	
18	9.50	9.50	9.50				112.5 45 -45	0 -45 45	
19	10.00	10.00	10.00				45 45 -45	45 -45 45	
20	10.50	10.50	10.50				90	0	
21	11.00	11.00	11.00				0	0	
22	11.50	11.50	11.50				67.5	0	
23	12.00	12.00	12.00				112.5 67.5	0 0	
24	12.50	12.50	12.50				0 67.5	0 0	
25	13.00	13.00	13.00				0 45	0 -45	

Table B.10: The list of other float typed parameters with their indices and values, created using `./sim.py --list latex-most`

	f_x[kHz]	f_y[kHz]	f_z[kHz]	f_{rf}[kHz]	Ω/Γ	δ/Γ	Laser's $(\theta, \phi)^*$		T_i[K]
26	13.50	13.50	13.50				67.5	0	
							45	-45	
27	14.00	14.00	14.00				45	-45	
28	14.50	14.50	14.50						
29	15.00	15.00	15.00						
30	15.50	15.50	15.50						
31	16.00	16.00	16.00						
32	16.50	16.50	16.50						
33	17.00	17.00	17.00						
34	17.50	17.50	17.50						
35	18.00	18.00	18.00						
36	18.50	18.50	18.50						
37	19.00	19.00	19.00						
38	19.50	19.50	19.50						
39	20.00	20.00	20.00						

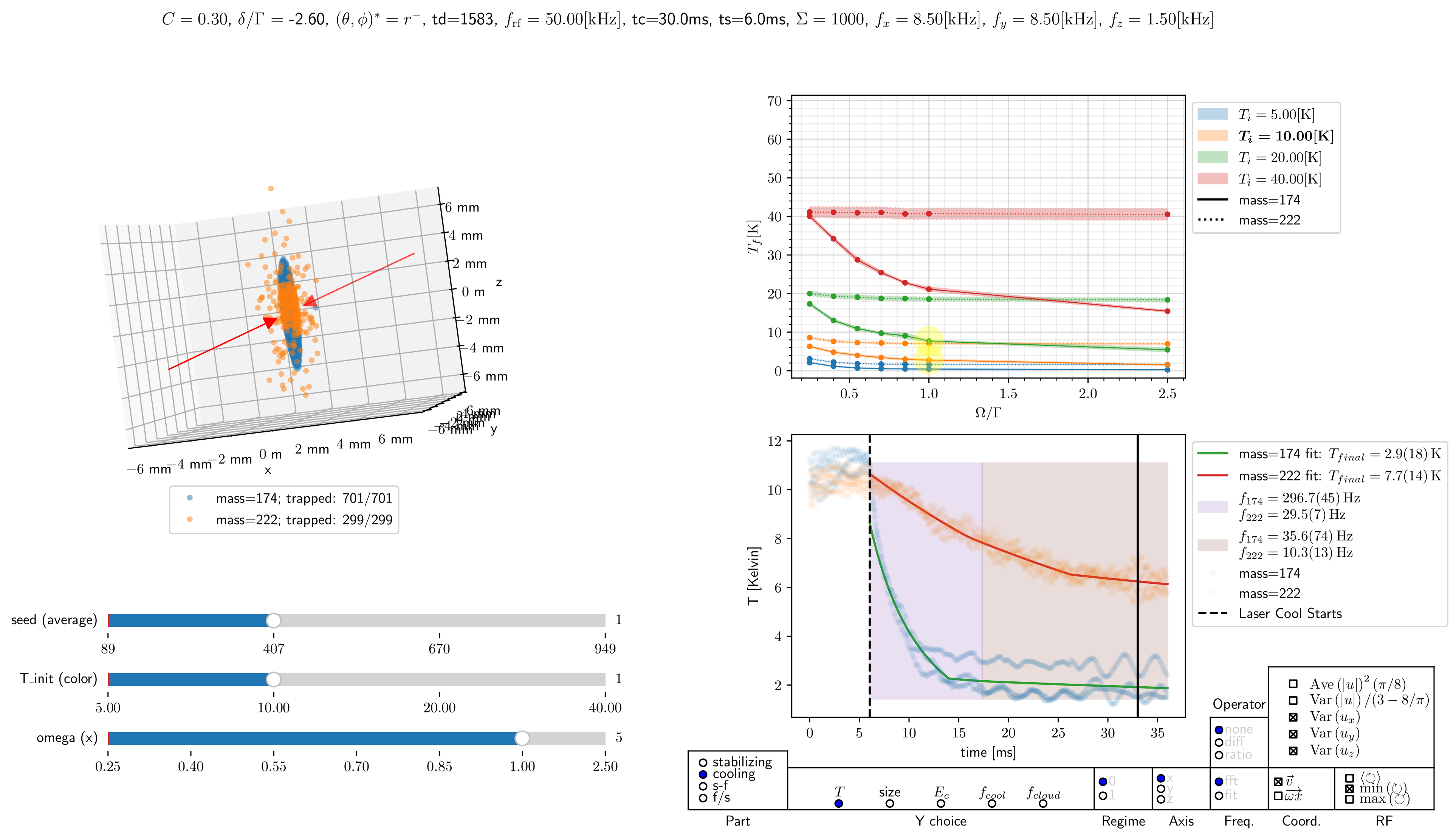


Figure B.1: A typical window opened by `./plot.py`. All features are described in section B.6.

total ↓	cloud →	0	1	2	3	4	5	6	7	8	9	10	11
0		0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	1/1
1		0/50	2/50	3/50	4/50	5/50	6/50	8/50	11/50	14/50	19/50	25/50	50/50
2		0/72	3/72	4/72	6/72	7/72	10/72	12/72	16/72	21/72	27/72	36/72	72/72
3		0/105	5/105	6/105	8/105	11/105	14/105	18/105	24/105	31/105	40/105	52/105	105/105
4		0/153	7/153	9/153	12/153	16/153	21/153	27/153	35/153	45/153	59/153	76/153	153/153
5		0/223	11/223	14/223	18/223	24/223	31/223	40/223	51/223	66/223	86/223	111/223	223/223
6		0/325	16/325	20/325	27/325	35/325	45/325	58/325	75/325	97/325	125/325	162/325	325/325
7		0/472	23/472	30/472	39/472	50/472	65/472	84/472	109/472	141/472	182/472	236/472	472/472
8		0/687	34/687	44/687	57/687	74/687	95/687	123/687	159/687	205/687	265/687	343/687	687/687
9		0/1000	50/1000	64/1000	83/1000	107/1000	139/1000	179/1000	232/1000	299/1000	387/1000	500/1000	1000/1000

Table B.11: The list of clouds' amounts. Calculated via rounding down cloud concentrations, and total amounts. Created using `./sim.py --list latex-amounts`.

Appendix C

Calculating Intra-molecular Vibration Redistribution (IVR)

Before we decided to use $CHDBrI^+$ for our Δ_{PV} measurement, we wanted to assess the natural life time of the vibrational levels. Apparently when a molecule is in an excited vibrational state, it's internal energy can be redistributed into other degrees of freedom of the molecule - hence the term Intra Vibrational Redistribution (IVR)[BR97]. This is a contributing factor to the life times of the vibrational levels, in addition to black body radiation.

The chance for IVR to happen depends on the structure of the molecule in ways out of scope for the assessment presented here. We did however developed a simple, brute-force model that would give as a hint on how likely this is to happen, while also focusing solely on vibrational levels.

To explain our model, we can start with an example: Say you have a molecule with the following vibrational eigen modes:

modes	1	2	3	4
energies [Arb.u]	1.5	2.5	5	6.5

Now, if the molecule's 3rd vibrational mode is excited, meaning $\nu_3 = 1$, the same internal energy can be redistributed into a double excitation of the 2nd mode, meaning $1\nu_3 = 2\nu_2$. Likewise, $1\nu_4$ can be redistributed into $2\nu_2 + 1\nu_1$ or $1\nu_3 + 1\nu_1$. In real life the vibrational energies are not that rational, but still the line widths of the vibrational levels might allow such transitions.

Here's how we generalized this: Take a linear combination of modes, where all coefficients are non-negative integers; What is the probability that this combination of modes will approximate another such combination of energies? More importantly, given ν_n, what is the probability that such a linear combination of several $\{\nu_i\}_i^k$?

To answer this we computed all of these combinations, up to a certain point, and plotted their *density* as a function of energy. This is essentially a histogram, we call *IVR histogram.* The histogram for our target $CHDBrI^+$ molecule, along with dashed lines marking the input vibrational modes is depicted in figure C.1.

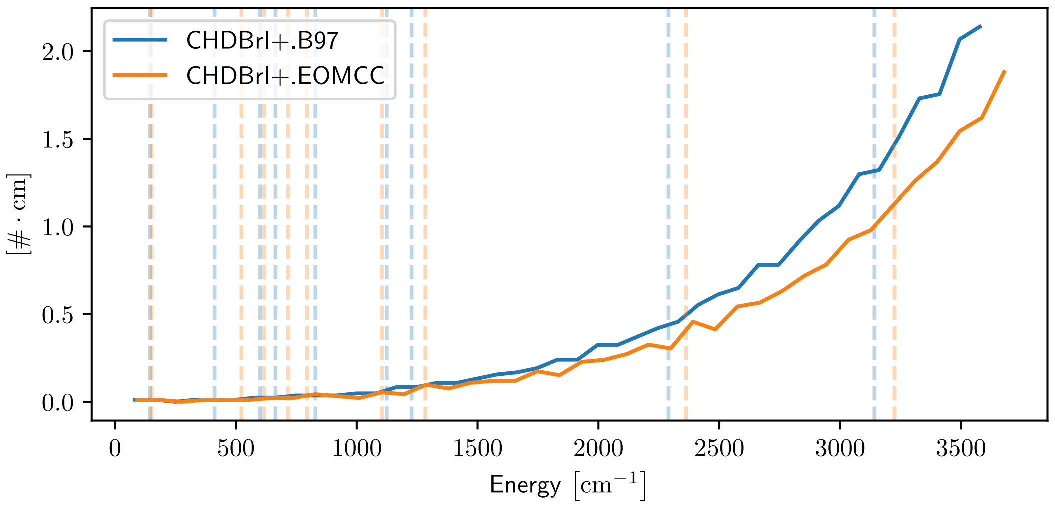


Figure C.1: IVR histogram computed for CHDBrI^+, given the eigen modes also given in table 1.1a. The different colors represent two different computational methods [LEB+23]. Dashed lines represent the input vibrational modes, and their colors match the colors of the ascending histogram lines.

We compared with the same method the IVR histograms of different molecular ions candidates for our Parity Violation measurement, and found outstanding differences in favor of CHDBrI^+, as depicted in figure C.2. It seems that using a heavier atom in combination with CH and BrX is increasing the density of states and thus probably increases the chances for IVR.

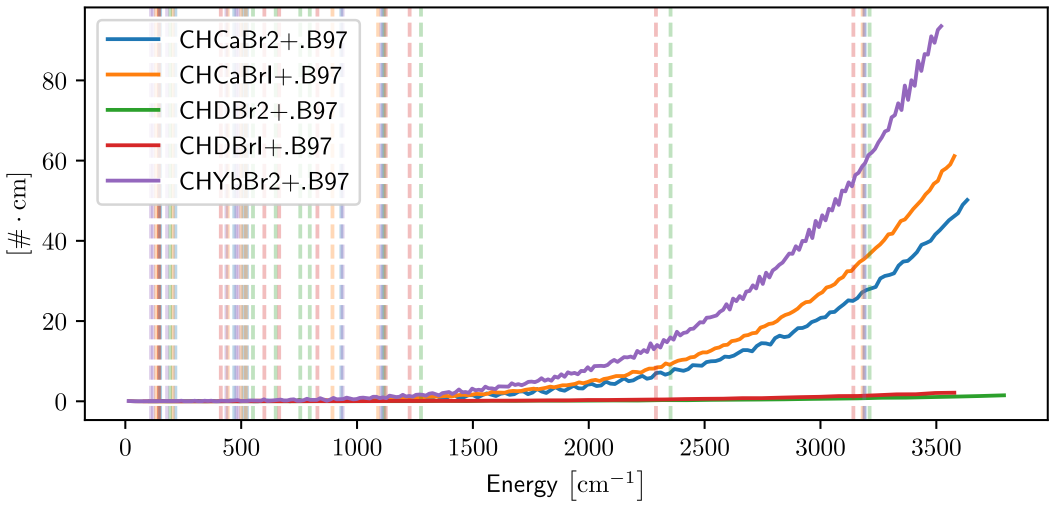


Figure C.2: A comparison of IVR histograms for all molecules mentioned in our ab-initio calculations [LEB+23]. All eigen modes were computed with the `B97` method.

C.1 Technical Notes

The source code for computing all possible energy combinations and plotting them, is available at https://gitlab.com/doronbehar/IVR-calculator. It consists mainly of 2 Python scripts, one for generating the data, and the other for plotting it. The main Python dependencies are Numpy [HMvdW+20] & Matplotlib [Hun07] and for generating IVR data, `progressbar2`[1] is required too.

C.1.1 `./calc.py`

Must be given a simple text file with a list of numbers, each corresponding to an eigen energy in any units you wish to use. In all calculations above and as usually in vibrational modes energy units are cm^{-1}, and these are the units the plotting script `./plot.py` uses too. The way it iterates the combinations looks like this:

```
[prompt] $ ./calc.py <your-energies-file>.txt
Energies are [132.0, 144.7, 205.0, 442.0, 500.0, 520.0, 894.0, 1091.0, 3182.0]
Cutoff energy (cumputed) used will be 3578.0
Will save all combinations to example.omegas.combinations.npz
maximal coefficients are:                    = (28, 25, 18, 09, 08, 07, 05, 04, 02)
 [Elapsed Time: 0:00:00] |# |combination: (00, 00, 03, 06, 00, 00, 00, 00, 00)
```

The last line is a progress bar where `#` are used to mark progress in the calculation, and takes the full terminal width. The coefficients calculated at any given time are printed there as well. The maximal coefficients are calculated such that each the maximal coefficient of any eigen energy alone doesn't go beyond the cutoff energy declared at the 2nd line. The cutoff energy is computed somewhat heuristically to avoid calculating too much beyond the last energy, such that the graphs stop just a bit after the highest eigen energy.

This computation takes a long time, and unfortunately is not parallelized as of yet. When this finishes, it creates a `.npz` file as declared in the 3rd line printed above, and this file can be given as an argument to `./plot.py`.

C.1.2 `./plot.py`

Plotting is fairly easy - you just give it as arguments `.npz` suffixed files that contain calculations generated by `./calc.py`. The command contains several aesthetic command line options that are self explanatory when you run `./plot.py --help`. However `--bins` is of worth mentioning: Since we are calculating density of states combinations as a function of energy, we need to actually calculate a histogram. `--bins` simply specifies the number of bins used for the histograms. It defaults to the $\sqrt{}$ of the number of total combinations, which comes out best.

[1] https://pypi.org/project/progressbar2/

C.1.3 `./density-printer.py`

If you wish to calculate the densities of states exactly near the eigen energies, you can use this script. It will print these in the format of a table in different formats: (1) ASCII to stderr and (2) LaTeX to stdout. This can be useful if you wish to put this information in an article. The density of states of the molecules in figure C.2 at their eigen energies is printed in table C.1.

	CHCaBr2+	CHCaBrI+	CHDBr2+	CHDBrI+	CHYbBr2+
0	0.101	0.056	0.000	0.000	0.070
1	0.000	0.056	0.019	0.012	0.070
2	0.000	0.000	0.009	0.024	0.000
3	0.000	0.000	0.038	0.036	0.140
4	0.152	0.056	0.019	0.036	0.070
5	0.000	0.112	0.038	0.084	0.070
6	0.253	0.337	0.085	0.108	0.912
7	0.709	0.674	0.275	0.553	0.772
8	28.157	36.291	0.883	1.514	61.385

Table C.1: Density of states of our Δ_{PV} candidates, at each vibrational energy.

Bibliography

[Ake12] Nitzan Akerman. *Trapped Ions and Free Photons.* PhD thesis, Wizmann Institute of Science, Rehovot, Israel, 2012. 143 pages. URL: https://weizmann.primo.exlibrisgroup.com/permalink/972WIS_INST/10fp4af/alma990002769100203596.

[BR97] Dean Boyall and Katharine L. Reid. Modern studies of intramolecular vibrational energy redistribution. *Chem. Soc. Rev.*, 26:223–232, 3, 1997. URL: http://dx.doi.org/10.1039/CS9972600223.

[BW02a] T. Baba and I. Waki. Sympathetic cooling rate of gas-phase ions in a radio-frequency-quadrupole ion trap. *Applied Physics B*, 74(4):375–382, April 1, 2002. URL: https://doi.org/10.1007/s003400200829 (visited on 05/12/2025).

[BW02b] T. Baba and I. Waki. Sympathetic cooling rate of gas-phase ions in a radio-frequency-quadrupole ion trap. *Applied Physics B: Lasers and Optics*, 74(4–5):375–382, April 2002. URL: http://dx.doi.org/10.1007/s003400200829.

[Coh92] C. Cohen-Tannoudji. Atomic motion in laser light. In J. Dalibard, J. M. Raimond, and J. Zinn-Justin, editors, *Fundamental Systems in Quantum Optics (Systèmes Fondamentaux en Optique Quantique)*, chapter I-3, page 29. Elsevier Science Publishers B.V., Amsterdam, 1992. URL: https://www.phys.ens.psl.eu/~cct/articles/cours/atomic-motion-in-laser-light-1992.pdf. Les Houches, Session LIII, 1990.

[Dah00] David A. Dahl. Simion ion optics simulation program, Idaho National Engineering Laboratory, 2000. URL: http://simion.com. Version 8.0.

[Dan20] Derek J. Daniel. Exact solutions of mathieu's equation. *Progress of Theoretical and Experimental Physics*, 2020(4):043A01, April 2020. eprint: https://academic.oup.com/ptep/article-pdf/2020/4/043A01/33114067/ptaa024.pdf.

[Del24] Pol Dellaiera. *Reproducibility in Software Engineering.* PhD thesis, University of Mons, August 2024. URL: https://doi.org/10.5281/zenodo.13208605.

[Dol06] Eelco Dolstra. *The Purely Functional Software Deployment Model.* PhD thesis, Utrecht University, 2006. URL: https://dspace.library.uu.nl/handle/1874/7540. Available from Utrecht University Repository.

[DT] Eelco Dolstra and The Nix contributors. Nix. URL: https://github.com/NixOS/nix.

[Ere23] Itay Erez. *Towards a Measurement of Parity Violation in Chiral Molecular Ions: The Advantage of Using a Mixed Handedness Sample.* MSc, Technion Institute of Technology, Haifa, Israel, 2023.

[ESL$^+$23] Eduardus, Yuval Shagam, Arie Landau, Shirin Faraji, Peter Schwerdtfeger, Anastasia Borschevsky, and Lukáš F. Pašteka. Large vibrationally induced parity violation effects in chdbri+. *Chem. Commun.*, 59:14579–14582, 98, 2023. URL: http://dx.doi.org/10.1039/D3CC03787H.

[FN22] Francis Filbet and Claudia Negulescu. FOKKER-PLANCK MULTI-SPECIES EQUATIONS IN THE ADIABATIC ASYMPTOTICS. July 2022. URL: https://hal.science/hal-03573120 (visited on 03/06/2025).

[Foo05] Christopher J. Foot. *Atomic physics.* In See subsection 7.6.2 for derivation of Rabi frequency vs intensity. Oxford University Press, Oxford, UK, 2005. Chapter 7, pages 142–143.

[Gho95] Pradip K Ghosh. *Ion Traps.* Oxford University Press, November 1995. URL: https://doi.org/10.1093/oso/9780198539957.001.0001.

[HA91] R. W. Hasse and V. V. Avilov. Structure and Madelung energy of spherical Coulomb crystals. *Physical Review A*, 44(7):4506–4515, October 1, 1991. URL: https://link.aps.org/doi/10.1103/PhysRevA.44.4506 (visited on 03/11/2025).

[HFC$^+$22] C. A. Holliman, M. Fan, A. Contractor, S. M. Brewer, and A. M. Jayich. Radium ion optical clock. *Phys. Rev. Lett.*, 128:033202, 3, January 2022. URL: https://link.aps.org/doi/10.1103/PhysRevLett.128.033202.

[HH17] S. Hoyer and J. Hamman. Xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 2017. URL: https://doi.org/10.5334/jors.148.

[HMvdW$^+$20] 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. *Nature*, 585(7825):357–362, September 2020. URL: https://doi.org/10.1038/s41586-020-2649-2.

[Hom13] Jonathon P. Home. Chapter 4 - quantum science and metrology with mixed-species ion chains. In Ennio Arimondo, Paul R. Berman, and Chun C. Lin, editors, *Advances in Atomic, Molecular, and Optical Physics*. Volume 62, Advances In Atomic, Molecular, and Optical Physics, pages 231–277. Academic Press, 2013. URL: https://www.sciencedirect.com/science/article/pii/B9780124080904000049.

[HRK+15] M. Hettrich, T. Ruster, H. Kaufmann, C. F. Roos, C. T. Schmiegelow, F. Schmidt-Kaler, and U. G. Poschinger. Measurement of dipole matrix elements with a single trapped ion. *Phys. Rev. Lett.*, 115:143003, 14, August 2015. URL: https://link.aps.org/doi/10.1103/PhysRevLett.115.143003.

[Hud16] Eric R. Hudson. Sympathetic cooling of molecular ions with ultracold atoms. *EPJ Techniques and Instrumentation*, 3(1):1–21, December 2016. (Visited on 07/23/2025).

[Hun07] J. D. Hunter. Matplotlib: a 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[IUH+93] H. Imajo, S. Urabe, K. Hayasaka, M. Watanabe, and R. Hayashi. Laser cooling of a small number of be+ ions in a penning trap. *Applied Physics B Photophysics and Laser Chemistry*, 57(2):141–144, August 1993. URL: http://dx.doi.org/10.1007/BF00425998.

[JPYM92] Jodie V. Johnson, Randall E. Peddeer, Richard A. Yost, and R. E. March. The stretched quadrupole ion trap: Implications for the Mathieu au and qu parameters and experimental mapping of the stability diagram. *Rapid Communications in Mass Spectrometry*, 6(12):760–764, 1992. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rcm.1290061210 (visited on 07/30/2025).

[JV19] Charles F. Jekel and Gerhard Venter. *pwlf: A Python Library for Fitting 1D Continuous Piecewise Linear Functions*. 2019. URL: https://github.com/cjekel/piecewise_linear_fit_py.

[KKM+00] D. Kielpinski, B. E. King, C. J. Myatt, C. A. Sackett, Q. A. Turchette, W. M. Itano, C. Monroe, D. J. Wineland, and W. H. Zurek. Sympathetic cooling of trapped ions for quantum logic. *Phys. Rev. A*, 61:032310, 3, February 2000. URL: https://link.aps.org/doi/10.1103/PhysRevA.61.032310.

[LEB$^+$23] Arie Landau, Eduardus, Doron Behar, Eliana Ruth Wallach, Lukáš F. Pašteka, Shirin Faraji, Anastasia Borschevsky, and Yuval Shagam. Chiral molecule candidates for trapped ion spectroscopy by ab initio calculations: from state preparation to parity violation. *The Journal of Chemical Physics*, 159(11):114307, September 2023. eprint: https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/5.0163641/18138404/114307_1_5.0163641.pdf.

[Leb10] Eric O. Lebigot. *Uncertainties: a Python package for calculations with uncertainties*. Source code available at https://github.com/lmfit/uncertainties. 2010. URL: https://uncertainties.readthedocs.io/en/latest/.

[Mat36] Émile Mathieu. Mémoire sur le mouvement vibratoire d'une membrane de forme elliptique. *Journal De Mathématiques*:68, 1836.

[MD21] Yansong Meng and Lijun Du. Study on the high-efficiency sympathetic cooling of mixed ion system with a large mass-to-charge ratio difference in a dual radio-frequency field by numerical simulations. *The European Physical Journal D*, 75(1):19, January 21, 2021. URL: https://doi.org/10.1140/epjd/s10053-020-00015-1 (visited on 03/06/2025).

[OMS96] Yoshitaka Oshima, Yoshiki Moriwaki, and Tadao Shimizu. Sympathetic cooling of ions in an rf trap. *Progress in Crystal Growth and Characterization of Materials*, 33(1):405–408, 1996. URL: https://www.sciencedirect.com/science/article/pii/0960897496836801.

[PSHL25] Yuehua Pang, Yi Shen, Jiahao Huang, and Chaohong Lee. High-precision many-body ramsey spectroscopy with composite pulses. *Phys. Rev. A*, 111:042611, 4, April 2025. URL: https://link.aps.org/doi/10.1103/PhysRevA.111.042611.

[QS01] Martin Quack and Jürgen Stohner. Molecular chirality and the fundamental symmetries of physics: influence of parity violation on rovibrational frequencies and thermodynamic properties. *Chirality*, 13(10):745–753, 2001. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/chir.10025.

[Qua02] Martin Quack. How important is parity violation for molecular and biomolecular chirality? *Angewandte Chemie International Edition*, 41(24):4618–4630, 2002. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/anie.200290005.

[Ran20] Anthony Michael Ransford. *Old Dog, New Trick: High Fidelity, Background-free State Detection of an Ytterbium Ion Qubit*. PhD thesis, University of California, 2020. 136 pages. URL: https://campbellgroup.physics.ucla.edu/papers/AnthonyRansfordThesis.pdf.

[RDH+20] Vyas Ramasubramani, Bradley D. Dice, Eric S. Harper, Matthew P. Spellings, Joshua A. Anderson, and Sharon C. Glotzer. Freud: a software suite for high throughput analysis of particle simulation data. *Computer Physics Communications*, 254:107275, 2020. URL: http://www.sciencedirect.com/science/article/pii/S0010465520300916.

[SBD10] E. S. Shuman, J. F. Barry, and D. DeMille. Laser cooling of a diatomic molecule. *Nature*, 467(7317):820–823, October 2010. URL: https://www.nature.com/articles/nature09443 (visited on 07/28/2025).

[SKH+16] Nicolas Sillitoe, Jean-Philippe Karr, Johannes Heinrich, Thomas Louvradoux, Albane Douillet, and Laurent Hilico. $\Bar\textrmH^+$ Sympathetic Cooling Simulations with a Variable Time Step. In *JPS Conference Proceedings*, volume 18, page 011014, Kanazawa, Japan, March 2016. URL: https://hal.science/hal-01669539 (visited on 03/17/2025).

[SL03] U. Shumlak and J. Loverich. Approximate Riemann solver for the two-fluid plasma model. *Journal of Computational Physics*, 187(2):620–638, May 2003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0021999103001517 (visited on 03/06/2025).

[SRKK12] Kenichiro Saito, Peter T.A. Reilly, Eiko Koizumi, and Hideya Koizumi. A hybrid approach to calculating coulombic interactions: an effective and efficient method for optimization of simulations of many ions in quadrupole ion storage device with simion. *International Journal of Mass Spectrometry*, 315:74–80, 2012. URL: https://www.sciencedirect.com/science/article/pii/S1387380612001236.

[Ste24a] Daniel A. Steck. *Quantum and atom optics*. In Revision 0.16.2, 15 November 2024. 2024. Chapter 5, pages 248–249. URL: http://steck.us/teaching.

[Ste24b] Daniel A. Steck. *Quantum and atom optics*. In Revision 0.16.2, 15 November 2024. 2024. Chapter 1, pages 48–49. URL: http://steck.us/teaching.

[TAB+22] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Comp. Phys. Comm.*, 271:108171, 2022.

[Tan11] O. Tange. Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36(1):42–47, February 2011. URL: http://www.gnu.org/s/parallel.

[The] The HDF Group. Hierarchical Data Format, version 5. URL: https://github.com/HDFGroup/hdf5.

[TP00] R. C. Thompson and J. Papadimitriou. Simple model for the laser cooling of an ion in a Penning trap. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 33(17):3393, September 2000. URL: https://dx.doi.org/10.1088/0953-4075/33/17/317 (visited on 08/03/2025).

[VC21] Ghislaine Vantomme and Jeanne Crassous. Pasteur and chirality: a story of how serendipity favors the prepared minds. *Chirality*, 33(10):597–601, 2021. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/chir.23349.

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[VHA+22] Nathaniel B. Vilas, Christian Hallas, Loïc Anderegg, Paige Robichaud, Andrew Winnicki, Debayan Mitra, and John M. Doyle. Magneto-optical trapping and sub-Doppler cooling of a polyatomic molecule. *Nature*, 606(7912):70–74, June 2022. URL: https://www.nature.com/articles/s41586-022-04620-5 (visited on 07/28/2025).

[vMHG+22] Martin W. van Mourik, Pavel Hrmo, Lukas Gerster, Benjamin Wilhelm, Rainer Blatt, Philipp Schindler, and Thomas Monz. Rf-induced heating dynamics of noncrystallized trapped ions. *Phys. Rev. A*, 105:033101, 3, March 2022. URL: https://link.aps.org/doi/10.1103/PhysRevA.105.033101.

[Wal10] Kim Walisch. *primesieve: Fast prime number generator*. 2010. URL: https://github.com/kimwalisch/primesieve.

[WAMS12a] Jannes B. Wübbena, Sana Amairi, Olaf Mandel, and Piet O. Schmidt. Sympathetic cooling of mixed-species two-ion crystals for precision spectroscopy. *Phys. Rev. A*, 85:043412, 4, April 2012. URL: https://link.aps.org/doi/10.1103/PhysRevA.85.043412.

[WAMS12b] Jannes B. Wübbena, Sana Amairi, Olaf Mandel, and Piet O. Schmidt. Sympathetic cooling of mixed-species two-ion crystals for precision spectroscopy. *Phys. Rev. A*, 85:043412, 4, April 2012. URL: https://link.aps.org/doi/10.1103/PhysRevA.85.043412.

[WBGS08] Stefan Willitsch, Martin T. Bell, Alexander D. Gingell, and Timothy P. Softley. Chemical applications of laser- and sympathetically-cooled ions in ion traps. *Physical Chemistry Chemical Physics*, 10(48):7200, 2008. URL: http://dx.doi.org/10.1039/b813408c.

[WSoT24] Eliana Ruth Wallach, Yuval Shagam, and Technion - Israel Institute of Technology. Faculty of Physics degree granting institution. Novel ion trap with resolved quantum state detection by velocity separation - toward first probe of parity violation in chiral molecules / eliana ruth wallach ; [supervision: yuval shagam]. eng, Haifa, 2024.

[WvdBG+08] A. L. Wolf, S. A. van den Berg, C. Gohle, E. J. Salumbides, W. Ubachs, and K. S. E. Eikema. Frequency metrology on the $4s\,^2S_{12} - -4p\,^2P_{12}$ transition in $^{40}\mathrm{Ca}^+$ for a comparison with quasar data. *Phys. Rev. A*, 78:032511, 3, September 2008. URL: https://link.aps.org/doi/10.1103/PhysRevA.78.032511.

[ZOR+07a] C. B. Zhang, D. Offenberg, B. Roth, M. A. Wilson, and S. Schiller. Molecular-dynamics simulations of cold single-species and multispecies ion ensembles in a linear Paul trap. *Physical Review A*, 76(1):012719, July 30, 2007. URL: https://link.aps.org/doi/10.1103/PhysRevA.76.012719 (visited on 03/17/2025).

[ZOR+07b] C. B. Zhang, D. Offenberg, B. Roth, M. A. Wilson, and S. Schiller. Molecular-dynamics simulations of cold single-species and multispecies ion ensembles in a linear Paul trap. *Physical Review A*, 76(1):012719, July 30, 2007. URL: https://link.aps.org/doi/10.1103/PhysRevA.76.012719 (visited on 03/17/2025).

אנרגיה קינטית.

לבסוף, נציע איך אפשר להתגבר על המגבלה של שימור תנע זוויתי, ועל המשמעות העמוקה של גילוי זה. המשמעות היא למעשה שקצב הקירור תלוי בטמפרטורה ההתחלתית של ענן היונים המתקרר סימפתטית, שכן האנרגיה ההתחלתית מתחלקת בזמן הקירור הסימפתטי לאנרגיה קינטית סיבובית שנשמרת, ולאנרגיה קינטית שמועברת ליוני ה-Yb^+, וכן נאבדת. נציג גם בקצרה מודל שמצאנו בספרות[BW02a], שדווקא חוזה שקצב הקירור אינו תלוי בטמפרטורה ההתחלתית, ולכן התוצאות שלנו למעשה סותרות אותו.

בפרק B, נסביר במפורט איך הסימולציות עובדות במובן הטכני, ואיך התוכנות שמפיקות את הגרפים פועלות. נתאר בפרק זה עוד פרטים טכניים רלוונטיים לסקריפטים נוספים שנכתבו לשם הסימולציות, ולשם הפקת גרפים נוספים שמופיעים בעיקר בפרק 2.

בפרק C, נציג חישוב שאינו תלוי בסימולציות שהוא התוכן העיקרי של המחקר. בפרק זה נסביר על תופעה שנקראת Intra-molecular Vibration Redistribution (IVR), ומדוע היא הייתה רלוונטית למדידה שלנו של המצבים הויבראציונים במולקות קיראליות. התוצאות של חישוב זה הוזכרו באופן מתומצת במאמר של הקבוצה שלנו[LEB$^+$23], וכאן נסביר יותר בפירוט איך החישוב בוצע ונראה יותר תוצאות שלו. כמו כן גם נסביר איך להשתמש בתוכנה שנבניתה לצורך חישוב זה בשביל לבצע חישובים דומים.

תקציר

ישנה תחזית לפיה הכח החלש במודל הסטנדרטי אחראי לשינויים מזעריים בין מולקולות כיראליות, בספקטרום התנודות הויבראציוניות[Qua02]. המטרה של הקבוצה שלנו היא למדוד את הפער האנרגטי הזה, שנקרא גם Parity Violation (PV), ולא נמדד מעולם. חישובים של הפער האנרגטי הזה בכמה זוגות מולקולות הראו שהוא מגיע לכמה Hz בודדים במקרה הטוב, בעוד שהמעברים הויבראציונים עצמם הם בתחום האינפרא אדום האמצעי והרחוק. אנחנו שמנו לעצמנו למטרה למדוד את הפער האנרגטי הזה, ובחרנו ביונים מולקולרים, כי אפשר ללכוד אותם בקלות להרבה זמן, ולקרר אותם.

לקרר מולקולות באופן ישיר זה בדרך כלל קשה מאוד, לכן החלטנו לקרר אותן באופן סימפתטי - כלומר באמצעות לכידה של מולקולה כיראלית יחד עם Yb^+, אותו אנחנו מקררים יחד באופן ישיר באמצעות לייזר. האופן בו אנחנו מודדים את הפער האנרגטי (PV) בין המולקולות מפרק אותן (Photo-dissociation), ולכן אנחנו צריכים לבצע את המדידה באופן חזרתי, ולסנתז מולקולות חדשות כל פעם. מטרת המחקר הזה היא לקרר את המולקולות כמה שיותר מהר בהשוואה לזמן קוהרנטיות, כדי להוריד את הזמן המת בכל חזרה על הניסוי. המאפיינים הייחודיים של הניסוי שלנו, ובמיוחד הצורך שלנו לקרר מהר, הם אלו שעודדו אותנו לבצע סימולציות בהן אנחנו סורקים פרמטרים, ומציגים את התוצאות בדרכים מגוונות המפורטות בפרק 2.

בפרק 2, אני מסביר באופן מפורט את התיאוריה שמאחורי הלכידה והקירור, ומראה שני חידושים קטנים בתחום של לכידה וקירור שהיו חשובים במיוחד למלכודות יונים כמו שלנו. לאחר מכן אני מציג עוד פרטים טכניים לגבי הסימולציות, כמו איך חילקתי את הזמן, ואיך חישבתי את הטמפרטורות בכל נקודת זמן.

בפרק 3, אראה את עיקר התוצאות של מחקר זה, והן הסימולציות כתלות בפרמטרים השונים. ראשית, עוד מבלי להוסיף יונים מולקולרים, נראה שבחירת הזוויות מהן הלייזרים נכנסים למלכודת, משפיעות מאוד על הקירור של ענן ה- Yb^+ - משהו שהיה חשוב מאוד במחקר שלנו בו עוצמת לייזר הקירור שלנו לא גבוהה. אחר כך נוסיף יונים מולקולריים, ונראה איך בטווח העוצמות שלנו, אכן הקירור שלהם יותר יעיל ככל שעוצמת הלייזר יותר גבוהה.

בהמשך הפרק, נראה את החידוש המשמעותי של מחקר זה, והוא המגבלות לקירור סימפתטי שמופיעות במלכודות צילינדריות, ואולי במיוחד במלכודות כמו שלנו, שהתדרים שלהם נמוכים מתדרים אופייניים למלכודות יונים. המגבלה נובעת מתנועה מעגלית של היונים המולקולריים במלכודת, שמתאפשרת במלכודת עם סימטריה צילינדרית. התנועה המעגלית הזו הינה למעשה שימור תנע זוויתי, שמאפשר ליונים המולקולרים להסתובב סביב הקריסטל של יוני ה-Yb^+ מבלי להתנגש בהם וכך לא לאבד

המחקר בוצע בהנחייתם של פרופסור יובל שגם ופרופסור יותם שורק בפקולטה לפיזיקה.

חלק מן התוצאות בחיבור זה פורסמו כמאמרים מאת המחבר ושותפיו למחקר בכנסים ובכתבי-עת במהלך תקופת מחקרו של המחבר. הגרסאות העדכניות ביותר של פרסומים אלו הינן:

Doron Behar. Sympathetic cooling optimization for chiral molecular ions precision spectroscopy. Poster Presented in CCMI 2024 conference, September 2024.

המחקר שבבסיס חיבור זה נערך כולו ביושר, ולפי אמות המידה האתיות המקובלות באקדמיה. בפרט אמורים הדברים בפעילויות איסוף הנתונים, עיבודם והצגתם, התייחסות והשוואה למחקרים קודמים וכו׳, ככל שהיוו חלק מן המחקר. כמו כן, הדיווח על המחקר ותוצאותיו בחיבור זה הוא מלא וישר, לפי אותן אמות מידה.

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמותי.

אופטימיזציה של קירור סימפתטי עבור ספקטרוסקופיה מדויקת של מולקולות כיראליות

חיבור על מחקר

לשם מילוי חלקי של הדרישות לקבלת התואר
מגיסטר למדעים בפיזיקה

דורון בכר

הוגש לסנט הטכניון — מכון טכנולוגי לישראל
אב התשפ"ה חיפה יולי 2025

אופטימיזציה של קירור סימפתטי עבור ספקטרוסקופיה מדויקת של מולקולות כיראליות

דורון בכר