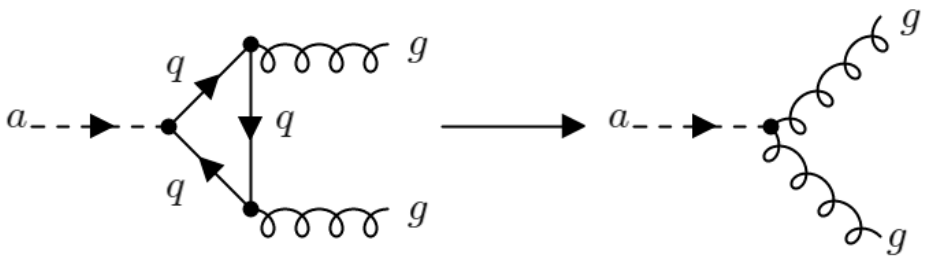
Building an easy accessible Schrödinger-Poisson solver for FDM investigations

Kavermann, Michel

Building an easy accessible Schrödinger-Poisson solver for FDM investigations



THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
PHYSICS

Author :	Michel Kavermann
Student ID :	s3363511
Supervisor :	Dr. Matthieu Schaller
Second corrector :	Dr. Subodh P. Patil

Leiden, The Netherlands, July 28, 2023

Building an easy accessible Schrödinger-Poisson solver for FDM investigations

Michel Kavermann
Lorentz Institute for Theoretical Physics, Leiden University
Niels Bohrweg 2, 2333 CA Leiden, The Netherlands

July 28, 2023

Abstract

The open question what the true nature of dark matter (DM) in the ΛCDM cosmological standard model is, could find its answer due to the naturally appearing QCD-axions in the solution of the strong-CP problem of QCD, which are generalized by string theory to axion-like particles (ALPs). Despite spanning a huge mass range that covers several order of magnitudes, especially ultra-light axions (ULAs) match extraordinary well with the DM properties known from precision cosmology. Due to their small masses, ULAs appear to be Fuzzy DM (FDM) rather than the familiar CDM by manifesting their wavelike character, what intuitively leads to the idea of modelling FDM as a non-relativistic wavefunction, ψ that then could be evolved in a Newtonian gravitational potential, Φ, via the Schrödinger-Poisson equation in a flat, expanding Universe governed by the RW-metric. Building on a strong theoretical framework we worked out in [1] already, this work aims at building an easy accessible numerical Schrödinger-Poisson solver that can be used to investigate FDM and any other wavefunctions in the given framework. After recapping on the most important theoretical lines from our previous work, we first present how our solver is built up and why the used algorithms were chosen, before tackling different systems in one- and three dimensions for various tests as well as first real-world FDM applications.

To Ellie - "No one can say what we get to be. So why don't we rewrite the stars?"

Contents

Chapter 1

Introduction

As was already stated in [1], the ΛCDM cosmological standard model is a well-described model in nowadays precision cosmology that truly establishes itself as the current standard model. However, it faces challenges of which one is the question what the true nature of dark matter (DM) is. Although, we know the most important DM properties very well due to the instruments used in precision cosmology, we still seek after the particle fitting in the picture. As we have discussed in lengths in the previous work, the standard model of particle physics naturally gives the QCD-axion as a well-suited DM-candidate even though it does not appear in the rows of the standard model particles at first, but introduces itself by solving the strong-CP problem of QCD. In the context of GUTs, e.g. string theory, the QCD-axion is generalized to a whole class of particles, called axion-like particles (ALPs) with a common set of properties that match the DM properties frightening well, where a subclass of ALPs are the ultralight axions (ULAs) that span a mass range

$$10^{-33}\,\mathrm{eV} \lesssim m_a \lesssim 10^{-18}\,\mathrm{eV}. \tag{1.1}$$

In the previous work [1] we ended our discussion after realizing that the impact of ULAs on large-scale structure cannot be described by cosmological perturbation theory throughout the whole history of the Universe since the density perturbations grow over time leading to values breaking the prerequisites that are necessary for perturbation theory, eventually leading to the necessity of non-linear evolution equations. By noting that the extremely low ULA-masses justify a wavelike description, i.e. Fuzzy Dark Matter (FDM), for non-relativistic velocities, it was natural to take

the Schrödinger-equation at a starting point. Combined with Newtonian gravity on cosmological scales with sufficiently large distances to black holes, we ended up with the Schrödinger-Poisson equation that shall govern the evolution of FDM in the Universe as soon as the density perturbations have grown big enough. Hence, this work is dedicated to build a numerical Schrödinger-Poisson solver in an educational, but still efficient, way in order to tackle non-relativistic wavefunctions in Newtonian gravity on cosmological scales. The basic idea and the approach is heavily influenced by [2].

We are going to use the Robertson-Walker (RW) metric

$$ds^2 = -dt^2 + a^2[dr^2 + S_\kappa^2(r)d\Omega^2], \tag{1.2}$$

where we typically assume a flat ($\kappa = 0$) Universe geometry, so that $S_{\kappa=0} = r$. Furthermore, we are going to work in the following system of units

$$[M] = 10^{10}\, M_\odot, \quad [L] = \text{Mpc} \quad \text{and} \quad [v] = \frac{\text{km}}{\text{s}}, \tag{1.3}$$

where $M_\odot \approx 1.99 \cdot 10^{30}\,\text{kg}$ is one solar mass and $1\,\text{Mpc} \approx 3.086 \cdot 10^{19}\,\text{km}$. We will always set $c = 1$ if necessary and keep $\hbar$ in all equations. For Fourier-transformations we usually transform the coordinates x to k, where the 2π's are placed below the dk's. G is Newton's gravitational constant and finally, in flat space, we will work with the mostly positive metric signature, i.e. $\eta_{\mu\nu} = (-1, +1, +1, +1)$. New words that are defined or further explained are written in *italic* letters.

Chapter 2

Brief review of axion theory

This chapter is devoted to present very briefly the main considerations that naturally led to the idea of axions as the DM question solving particle as well as a quick overview over the important theoretical framework that is applied in the work. Readers of [1] can safely skip this chapter, although section 2.4 contains a few additional information about the Schrödinger-Poisson equation.

2.1 How axions naturally appear in the known picture

Instead of guessing a particle that has the DM properties we were able to observe and deduce so far and then fitting it into the known pictures, we would like to follow a more natural approach by looking at what we already know. The standard model of particle physics is well-established, so it appears obvious to search for the DM particle in the rows of the extensively studied fundamental theories, but the result is that there is no appropriate DM candidate. However, one building block, i.e. Quantum Chromodynamics (QCD), of the standard model of particle physics suffers the so-called *strong-CP problem*.

The QCD-Lagrangian appears to be invariant under axial transformations if and only if the quark masses vanish, which is apparently not the case. However, the resulting Noether-current gives rise to a chiral anomaly, which in turn introduces an additional term to the QCD Lagrangian, namely

$$\delta S = \alpha \frac{N_f g_s^2}{32\pi^2} \int d^4x \, G^{\mu\nu,a} \tilde{G}^a_{\mu\nu}, \tag{2.1}$$

where

$$G^{\mu\nu,a} \tilde{G}^a_{\mu\nu} = \partial_\mu \left(\varepsilon^{\mu\nu\rho\sigma} A^a_\nu \left[F^a_{\rho\sigma} - \frac{g_s}{3} f^{abc} A^b_\rho A^c_\sigma \right] \right), \tag{2.2}$$

so that the integral seems to become a surface integral which vanishes under the assumption that $A^{\mu,a} = 0$ at spatial infinity, which is nothing else than the vacuum[3]. Although, it seems like there is nothing more to it, t'Hooft found out that one shall not pass the complexity of the vacuum structure, which is in fact crucial to investigate at this point[4]. Interestingly, he realized that one can easily rotate one vacuum state into another what then gives rise to the so-called *θ-vacua*. The main consequence is that the integral above does not vanish trivially and the chiral anomaly is still present, so that the axial transformations cannot be a true symmetry of QCD[4]. The θ-vacua introduce an additional term

$$\mathcal{L} = \frac{\theta}{32\pi^2} \int d^4x \mathrm{Tr}[G_{\mu\nu} \tilde{G}^{\mu\nu}] \tag{2.3}$$

to the QCD-Lagrangian, which violates CP symmetry and thus, gives rise to an electric dipole, which is constrained by experiment[5]

$$\theta \cdot 10^{-16} \, e\,\mathrm{cm} \approx |d_n| \lesssim 3 \cdot 10^{-26} \, e\,\mathrm{cm} \quad \Rightarrow \theta \lesssim 3 \cdot 10^{-10}. \tag{2.4}$$

This is a fine-tuning problem, what is problematic since theoretically, θ can be any value in $[0, 2\pi]$. The situation exacerbates because whilst analyzing the θ-vacua structure in the context of the electroweak (EW) theory, t'Hooft found that the used chiral transformations introduce an additional angle term due to the quark masses to the θ-value, i.e.

$$\theta_{\mathrm{QCD}} = \theta + \arg[\det[M]], \tag{2.5}$$

which should counteract the original angle in order to satisfy the fine-tuning problem and in principal to set $\theta = 0$. One refers to these problems as the strong-CP problem.

In order to solve the problem, we impose an additional $U(1)_{PQ}$-symmetry[1] and assume that at least one fermion of the theory acquires its mass through Yukawa-coupling whose corresponding vacuum expectation value is nonzero

[1] Instead of renaming it later, we already call the new $U(1)$-symmetry with the subscript PQ to honor Peccei and Quinn[6] who solved this problem.

then[6]. The solution then goes as follows. The best case scenario is that the total angle $\bar{\theta} = 0$. The Yukawa coupling gives rise to at least one (pseudo-)scalar field, ϕ,

$$\mathcal{L} \subset \bar{\psi}\left[g_Y\phi\frac{1}{2}(1+\gamma^5) + g_Y^*\phi^*\frac{1}{2}(1-\gamma^5)\right]\psi, \tag{2.6}$$

where g_Y is the Yukawa-coupling-constant and ϕ has the nonzero vacuum expectation value $\langle\phi\rangle = \lambda e^{i\beta}$. In order to minimize the potential, $V(\phi)$, that depends on the field itself and the angle $\bar{\theta} = \theta + \beta$, one finds $\beta = 0$. The corresponding fermion mass term then reads

$$\lambda\bar{\psi}\left[g_Y e^{i\beta}\frac{1}{2}(1+\gamma^5) + g_Y^* e^{-i\beta}\frac{1}{2}(1-\gamma^5)\right]\psi, \tag{2.7}$$

which must be made real again. Now, the new $U(1)_{PQ}$ symmetry comes in handy since we can simply perform the rotation $\exp\{i\gamma^5\theta\}$ for $\theta = -\beta$, so that in the end we get

$$\bar{\theta} = \theta + \beta = \theta - \theta = 0, \tag{2.8}$$

what is precisely the best case scenario we were hoping for. The new $U(1)_{PQ}$-symmetry is said to set $\bar{\theta}$ dynamically to zero, what restores CP invariance on the one hand and solves the strong-CP problem on the other hand. The most important point for us is the introduction of a new pseudoscalar field, ϕ, which couples to the $G\tilde{G}$-term in (2.3), so that we can set

$$\theta_{\text{QCD}} = C\frac{\phi}{f_a}, \tag{2.9}$$

so that ϕ, which is the canonically normalized *axion* field, is clearly part of QCD. Note, that C describes the color anomaly and f_a is the *axion decay constant*.

2.2 The most important axion properties

Historically, the axion was introduced as the QCD-axion like presented in the previous section. We are going to use the term axion more generally for *Axion-Like Particles (ALPs)* that share a small set of common properties which we would like to investigate briefly in the following.

Recall that the strong-CP problem originated from t'Hooft's observation that the vacuum structure is not trivial and one has to consider transitions

between different θ-vacua to which the so-called *instantons* correspond. These in turn come along with a very important property that is crucial for the description of axions, namely that it is a purely non-perturbative effect. We saw in the previous section that it is necessary for θ_{QCD} to bet set to zero, so that we demand a shift-symmetry of the axion field, $\phi \to \phi + \text{const.}$, and that only derivatives of the axion field enter the action. The fact that the instanton-effects are non-perturbative effects come in useful now because the shift-symmetry must be protected from quantum corrections to all orders in perturbation theory which is automatically given by the instantons. For all possible quantum corrections we demand that they are suppressed by powers of f_a[7]. If we now face additional contributions to θ_{QCD} we can simply absorb them via the shift-symmetry in the axion field, ϕ. The result is, that the by the instanton-effects induced action and potential can only depend on the overall axion field. We can reformulate the protection of the shift-symmetry from quantum corrections as the violation of a classical symmetry by quantum effects, i.e. an anomaly, which justifies the color anomaly C in (2.9). The notion of C is the number of distinguishable θ-vacua in $[0, 2\pi f_a]$. Along with the shift-symmetry, we can now state $\phi \to \phi + 2\pi f_a$ and since ϕ is an angular variable, it is safe to say that $C \in \mathbb{Z}$.[2]

Finally, we can investigate the mass range of axions, especially for QCD-axions, by investigating the interaction between axions and quarks. Note, that after QCD-confinement at $T \sim \Lambda_{\text{QCD}}$, we can effectively replace $\bar{q}q$-terms in the interaction by their corresponding vacuum expectation values and note additionally, that interactions between axions and standard model particles are suppressed by powers of f_a, so that by considering only large f_a-values, i.e. small m_a-values, it is sufficient to neglect all but up- and down-quarks in the interaction. Without going into detail, one ends up with

$$m_{\text{a,QCD}} \approx 6 \cdot 10^{-6}\text{eV} \cdot \left(\frac{10^{12}\,\text{GeV}}{\frac{f_a}{C}} \right). \tag{2.10}$$

For $T < \Lambda_{\text{QCD}}$, instanton-effects break the shift-symmetry explicitly to a discrete symmetry,

$$\phi \to \phi + 2\pi \frac{f_a}{C}, \tag{2.11}$$

which coincides with the earlier results that $C \in \mathbb{Z}$. Additionally, the

[2]Please refer for details to [1] and references therein, especially to appendix C for this section.

symmetry breaking induces the QCD-axion potential[3]

$$V(\phi) = m_u \Lambda_{\mathrm{QCD}}^3 \left[1 - \cos\left(C\frac{\phi}{f_a}\right)\right], \tag{2.12}$$

where m_u is the up-quark mass. The cosine potential results from the vacuum energy, we do not want to investigate further here.

Let us summarize the four important shared ALP properties.

1. The classical action has a global PQ-symmetry, i.e a $U(1)_{\mathrm{PQ}}$-symmetry.
2. The spontaneous symmetry breaking scale f_a leads to an angular degree of freedom, ϕ/f_a, that contains a shift-symmetry.
3. The PQ-symmetry is anomalous and thus, explicitly broken at quantum level by non-perturbative instanton-effects that protect the classical shift-symmetry.
4. The protected shift symmetry, $\phi \to \phi + 2n\pi f_a$ with $n \in \mathbb{Z}$, manifests the axion as a pseudo-Goldstone boson that obtains a periodic potential, $V(\phi/f_a) \equiv V(\phi)$, when the non-perturbative quantum effects switch on at some scale, Λ_a. The same effects induce the axion mass, m_a, which is proportional to Λ_a^2/f_a.

To highlight the first connections to DM, note, that DM is non-baryonic, what is given for axions since they are manifested as bosonic particles, i.e. they are pseudo-Goldstone bosons. Further, DM should be stable or at least long-lived compared to the age of the Universe, what is given since we demand the axion decay constant, f_a, to be very large in order for the coupling strengths to standard model particles, that scale with negative powers of f_a, to be very small since observations reveal that DM is basically not interacting[4] with ordinary matter. The missing DM property, i.e that DM particles are cold, is left to show in the next section.

[3] We consider QCD-axions here to give a notion for the general axion properties, but as string-theory dictates, the general results from the QCD-axion investigations are transferable to ALPs in general. For a generic ALP, $V(\phi) = \Lambda_a^4 \left[1 \pm \cos\left(C\frac{\phi}{f_a}\right)\right]$ is the typical potential[7].

[4] Even more appropriate would it be to state, that DM is at most interacting gravitationally with ordinary matter, what underlines observations in which one can explicitly makes ordinary matter visible via X-rays and notices the presence of DM only via gravitational lensing.

2.3 Axion production in the early Universe

One can raise the question now, how a relevant axion population was produced in the early Universe, which shall be the scope of this section. Instead of considering different production mechanisms as in [1], we will immediately discuss the proper one here, namely the non-thermal production via misalignment.

The action, S, for a minimally coupled real scalar field, ϕ, in the theory of general relativity is given by[8]

$$S_\phi = \int d^4x\sqrt{-g}\left[-\frac{1}{2}\partial_\mu\phi\partial^\mu\phi - V(\phi)\right]. \tag{2.13}$$

Note, that this holds only after symmetry breaking in order for the axion to be established as a pseudo-Goldstone boson and it is only valid after non-perturbative effects switch on since this is necessary for the axion to acquire mass. The latter is no instantaneous process, which is why there will be a time-dependence on the ϕ-fields[5]. By varying the action with respect to ϕ we obtain the equations of motion (EOM)

$$0 = \frac{1}{\sqrt{-g}}\partial_\mu(\sqrt{-g}g^{\mu\nu}\partial_\nu)\phi - \frac{\partial V}{\partial\phi} =: -\frac{\partial V}{\partial\phi}. \tag{2.14}$$

Recall, that we work with the RW-metric (1.2)

$$ds^2 = -dt^2 + a^2[dr^2 + S_\kappa^2(r)d\Omega^2], \tag{2.15}$$

where we will assume a flat Universe with $\kappa = 0 \Rightarrow S_\kappa(r) = r$, so that

$$ds^2 = -dt^2 + a^2[dr^2 + r^2d\Omega^2] \tag{2.16}$$

plugged into the EOM (2.14) results in

$$\ddot{\phi} + 3H\dot{\phi} + m_a^2\phi = 0, \tag{2.17}$$

which is the equation of a harmonic oscillator with an additional friction term, that in this case expresses the expansion of the Universe with the Hubble parameter, $H(t) := \frac{\dot{a}}{a}$. We know, that the scale-factor, a, scales as a power-law, $a \sim t^p$, what allows for an analytical solution

$$\phi(t) = a^{-\frac{3}{2}}\left(\frac{t}{t_i}\right)^{\frac{1}{2}}[C_1J_n(m_at) + C_2Y_n(m_at)] \tag{2.18}$$

[5]Note, that this can be converted, as usual, to a temperature-dependence.

for the EOM (2.14), where $n = (3p-1)/2$, $J_n(x)$ and $Y_n(x)$ are Bessel functions of the first and second kind, respectively. In order to solve for the coefficients, $C_{1,2}$, we need proper initial conditions. First, $\dot{\phi}_i = 0$ appears reasonable because by looking at the EOM (2.14), one sees that ϕ is overdamped for $H \gg m_a$. Second, by setting the color anomaly in (2.9) to unity, after rearranging we get $\phi(t_i) = f_a\theta_i$ as the second initial condition, which basically states that the vacuum was initially misaligned, i.e. ϕ not being at its potential minimum.

By varying the action (2.13) with respect to the metric we get the energy-momentum tensor

$$T^{\mu}_{\nu} = g^{\mu\alpha}\partial_\alpha\phi\partial_\nu\phi - \frac{\delta^{\mu}_{\nu}}{2}\left[g^{\alpha\beta}\partial_\alpha\phi\partial_\beta\phi + 2V(\phi)\right]. \tag{2.19}$$

Note, that we assume f_a to be very large, so that $m_a \sim f_a^{-1}$ is very small. Hence, a perfect fluid description for the axions is valid and the energy-momentum tensor has the components

$$T^0_0 = -\rho, \quad T^0_i = (\rho + P)v_i \quad \text{and} \quad T^i_j = P\delta^i_j + \Sigma^i_j, \tag{2.20}$$

where ρ is the energy density, P is the pressure, v_i is the velocity and Σ^i_j is the anisotropic stress. Recall, that we use a flat RW-geometry, i.e. the most general metric that satisfies the cosmological principle that states that the Universe is homogeneous and spatially isotropic. Thus, we get

$$\bar{\rho}_a = \frac{1}{2}\dot{\phi}^2 + \frac{1}{2}m_a^2\phi^2 \quad \text{and} \quad \bar{P}_a = \frac{1}{2}\dot{\phi} - \frac{1}{2}m_a^2\phi^2 \tag{2.21}$$

as the background energy density and pressure, respectively. Due to (2.18), we know that $\phi \sim a^{-\frac{3}{2}}$ and $\dot{\phi}_i = 0$, so that $\bar{\rho}_a \sim \phi^2 \sim a^{-3}$, meaning that axions behave as ordinary matter. However, the EOM (2.14) describe a harmonic oscillator, meaning that at a certain scale factor, a_{osc}, the condition of overdamping, $H \gg m_a$, must be violated since for ongoing time, the scale factor is increasing and thus, $H := \frac{\dot{a}}{a}$ is decreasing. As we have just shown, $\rho_a(a) \sim a^{-3}$ holds, so that we can approximate

$$\rho_a(a)a^3 \approx \rho_a(a_{\text{osc}})a^3_{\text{osc}} \tag{2.22}$$

for $a > a_{\text{osc}}$ since the energy density must be almost constant in the underdamped oscillation region. Thus along with $\dot{\phi}_i = 0$,

$$\rho_a(a_{\text{osc}}) \approx \frac{1}{2}m_a^2\phi_i^2, \tag{2.23}$$

so that by knowing the axion mass, m_a, and the initial field displacement, ϕ_i, we are able to compute the axion energy density. A well-fitting approximation [9] is $3H(a_{\rm osc}) = m_a$ to get $a_{\rm osc}$. Note, that $H(a_{\rm eq}) \sim 10^{-28}\,\text{eV}$, where $a_{\rm eq}$ is the scale factor at radiation-matter equality. If $m_a > H(a_{\rm eq})$, the axions start oscillating in the radiation-dominated epoch. Assuming this to be the case, one can show [10] that

$$\Omega_a \approx \left\langle \left(\frac{\phi_i}{M_{\rm pl}} \right)^2 \right\rangle \cdot \begin{cases} \frac{1}{6}(9\Omega_\gamma)^{\frac{3}{4}} \left(\frac{m_a}{H_0} \right)^{\frac{1}{2}} & a_{\rm osc} < a_{\rm eq} \\ \frac{9}{6}\Omega_m & a_{\rm eq} < a_{\rm osc} \lesssim 1\,, \\ \frac{1}{6} \left(\frac{m_a}{H_0} \right)^{\frac{1}{2}} & a_{\rm osc} \gtrsim 1 \end{cases} \tag{2.24}$$

where the angular brackets are a necessary average depending on if the PQ-symmetry gets broken during or after the era of inflation. We do not go deeper into this discussion, but will assume that the PQ-symmetry gets broken during the era of inflation in order to produce sufficiently large axion populations[1]. However, for $H \ll m_a$, i.e. the underdamped case of the EOM (2.14), one can make a WKB-approximation with the ansatz $\phi(t) = A(t)\cos(m_a t + \vartheta)$, where ϑ is an arbitrary phase and the amplitude, $A(t)$, is in consistence with the slow-roll inflation model,

$$\frac{\dot{A}(t)}{m_a} \sim \frac{H(t)}{m_a} \ll 1, \tag{2.25}$$

so that via the EOM (2.14) one gets $A(t) \sim a^{-\frac{3}{2}}$ and hence, we once again get

$$\rho_a \overset{(2.21)}{\sim} \phi_i^2 \overset{\text{WKB}}{\sim} A^2 \sim a^{-3}, \tag{2.26}$$

meaning that axions still behave as ordinary matter and thus, contribute to the total matter content of the Universe. Finally, note that (2.24) in order to produce a significant contribution to the amount of dark matter, $\Omega_{\rm DM}h^2 \approx 0.12$, the axions should have $f_a \gtrsim \phi_i > 10^{14}\,\text{GeV}$, what agrees with our previous results that build on the assumption that f_a is very large. This motivates us to focus on the study of the so-called *ultralight-axions (ULAs)* in the mass range

$$10^{-33}\,\text{eV} \lesssim m_a \lesssim 10^{-18}\,\text{eV}. \tag{2.27}$$

The lower bound is of the order of the Hubble constant, H_0, and the upper bound is related to the baryon Jeans scale to reflect the role of ULAs in structure formation[7].

Note, that the whole production mechanism is solely based on the theory of general relativity, i.e gravitational effects, so that it clearly is a non-thermal process on the one hand, but, most importantly, on the other hand, is a very natural mechanism to provide an explanation for a relevant axion population. Additionally, the non-thermal production apparently produces cold axions, so that the fourth DM property, that was left at the end of the previous section, is also fulfilled. This clearly establishes the axion as a viable DM candidate.

2.4 The Schrödinger-Poisson equation

In order to describe the evolution of initial density perturbations in the early Universe, we extensively discussed cosmological perturbation theory in [1], mostly based on the recommendable review article [11]. However, at later times, the perturbations grow far beyond the perturbation theory requirements, so that non-linear theory is in need, which is what we would like to briefly describe now.

Let us first note, that based on the the previous sections, we know that the axion mass should be very small. In fact, it should be small enough that the pure particle description of the axion can be loosened a bit, i.e. by the wave-particle-dualism, it should be valid to describe the axion as a wave in general. This justifies why the literature typically speaks of *Fuzzy*-DM (FDM). Since virial velocities in the Universe are small compared to the speed of light, $v_{\text{vir}} \ll c$, and the Newtonian gravitational potential is small compared to unity, $\Phi \ll 1$, everywhere except in the vicinity of a black hole (BH), a good starting point seems to be the Schrödinger equation

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\psi + V\psi, \tag{2.28}$$

where ψ is the wavefunction of the particle and m its mass and V the potential it is located in as usual. From [7] we know that

$$\rho_a = \frac{1}{2}\left[(1-2\Psi)\left(\frac{\partial\Phi}{\partial t}\right)^2 + m_a^2\phi^2 + \frac{1}{a^2}(1+2\Psi)(\nabla\phi)^2\right] \tag{2.29}$$

and by making a WKB-approximation[6] for the axion field, ϕ, i.e.

$$\phi = \sqrt{\frac{\hbar^3}{2m_a}} \left(\psi e^{-i\frac{m_a}{\hbar}t} + \psi^* e^{i\frac{m_a}{\hbar}t} \right), \tag{2.30}$$

where ψ is a complex scalar field, one can show that $\rho_a = |\psi|^2$ holds in Newtonian limit working up to leading order in Ψ if we only consider axion-wavelengths above their Compton wavelength. Since we work in Newtonian limit, ρ_a should source the gravitational potential via the Poisson equation

$$\Delta\Phi = 4\pi G\rho_a = 4\pi G|\psi|^2. \tag{2.31}$$

Based on this insight and a the work done in [1, 11] under the same assumptions, one ends up with the non-linear Schrödinger-Poisson equation

$$i\hbar\frac{\partial\psi}{\partial t} + \frac{3}{2}i\hbar H\psi = -\frac{\hbar^2}{2m_a}\Delta\psi + m\Phi\psi, \tag{2.32}$$

which is the EOM of ψ. Note, that the gravitational potential, Φ, is given in dimensions of energy per unit mass, so that in order to insert Φ in V in the Schrödinger (2.28) equation, we must multiply it with an additional mass factor, m. With the replacements[7]

$$\psi \to a^{-\frac{3}{2}}\psi,\ \Delta \to a^{-2}\Delta \text{ and } \Phi \to a^{-1}\Phi, \tag{2.33}$$

where $\Delta = \nabla^2$ should be kept in mind, we transform the Schrödinger-Poisson equation to comoving coordinates, so that it reads

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m_a a^2}\Delta\psi + \frac{m_a}{a}\Phi\psi. \tag{2.34}$$

Likewise, the Poisson equation (2.31) is transformed, but the additional factors a^{-3} on both sides cancel each other, so that it is invariant under this transformation.

[6] We are keeping factors of $\hbar$ whilst setting $c = 1$ in the equations since $\hbar$ is going to be relevant in the further discussion unlike c.

[7] Recall the RW-metric (1.2), then you immediately know, that coordinates transform as $x \to a \cdot x$, so that the replacements for $\nabla_i = \partial/\partial x_i$ and ψ with dimensions $M^{1/2}L^{-3/2}$ follow immediately. For Φ with dimensions EM^{-1}, note that $E \to a^{-1}E$ in an expanding Universe.

Originally, the Schrödinger-Poisson[8] equation was derived by Ruffini and Bonazzola [12] in 1969. Instead of equation based on the perturbed RW-metric in the non-relativistic limit or replacing the potential, V, in the Schrödinger equation by the gravitational potential, they kept a non-gravitational potential inside the equation, i.e.

$$i\hbar\frac{\partial\psi}{\partial t} = -\frac{\hbar^2}{2m}\Delta\psi + V\psi + m\Phi\psi, \tag{2.35}$$

where V is the non-gravitational potential and Φ is the gravitational potential. They ran into some conflicts with interpretation of their results, so we do not want to follow their track here.

Following our initial path, we have to be very careful about the correct interpretation of the wavefunction.

First of all, recall, that $\rho = |\psi|^2$ should hold, but we tacitly assumed all units to work out just fine. It is crucial to note that the wavefunction, ψ, must be normalized. By following Born's rule, $|\psi|^2(\vec{x}, t)$ is the probability to find the system that is described by ψ to be in $\vec{x}$ at time t, so that after integration over the whole space for a fixed time yields

$$\int |\psi|^2 \, d^3x = 1 \tag{2.36}$$

in order to fulfill the probability interpretation. Since the dimension of dx is length, L, $|\psi|^2$ must have dimensions L^{-3}. However, since ρ should be given in dimensions ML^{-3}, $|\psi|^2$ must have the same dimensions, what conflicts with the above normalization. In order to fix this, instead of normalizing the wavefunction to unity, we simply normalize it to the total mass, M_{tot}, in the volume as this is likewise conserved, so that

$$\int |\psi|^2 \, d^3x = M_{\text{tot}}. \tag{2.37}$$

Again, with $[d^3x] = L^3$ and $[M_{\text{tot}}] = M$ we then get $[|\psi|^2] = ML^{-3}$, so that the normalization and hence, the dimensions, are in no conflict with $\rho = |\psi|^2$.

[8] In their original work, they spoke of the Schrödinger-Newton equation due to the applied Newtonian limit, but we will stick to the nowadays common term in the literature.

Second, May and Springel[13] noted that since axions are bosons with low velocities, most of them should be in their ground state, what justifies a mean-field approximation interpretation of our ansatz, so that the whole axion population can in fact be modelled by a single wavefunction, ψ, whose evolution is governed by the non-linear Schrödinger-Poisson equation (2.34).

With the Schrödinger-Poisson equation (2.34) and the Poisson equation (2.31) at hand, we should be able to compute the evolution of any given wavefunction, ψ, that describes a system of particles in a gravitational potential, Φ, with low velocities in non-relativistic as well as Newtonian limit, i.e. being far away from black holes.

Chapter 3

Building a Schrödinger-Poisson solver

The implementation of the Schrödinger-Poisson solver was done in `Python 3.8.10`, for which we strongly made use of NumPy[14] and[1] Matplotlib[16]. Note, that the python files themselves contain short information about the purpose of a specific file and function, respectively, as well as what input is required along with the output that is to be expected.

The solver is structured fairly simple and can be broken down in the following manner. First, the parameters and initial values are defined. Second, the initial wavefunction under consideration gets computed. Third, the main routine iterates through a fixed amount of timesteps of variable length in order to evolve the system in time. For this purpose, per iteration the gravitational potential is computed. Then, the so-called *Schrödinger-velocity* is computed and based on it, the current timestep is computed via the *Courant-condition*. Out of the current wavefunction, the potential, the Schrödinger-velocity and the timestep the wavefunction at the next point in time is computed. In equidistant time intervals, the system is plotted for analysis purposes. Fourth, after going through all timesteps, some tracking variables are plotted and stored along with the simulation parameters for convenience. In the following sections, we would like to go through theses steps in somewhat more detail, explaining how the implementation is done and why we chose certain approaches.

[1] We also used the Simpson-integrator from the SciPy[15] module, but for basically all other computation and data formatting purposes, we used NumPy or the built-in Python functions.

3.1 Parameters, setup and constants

In `constants.py` we have stored Newton's constant of gravity, G, and the reduced Planck constant, $\hbar$. Since we use (1.3) as the system of units we would like to work in, the constants read

$$G = 43.21 \, \frac{\text{Mpc}}{10^{10} M_\odot} \left(\frac{\text{km}}{\text{s}} \right)^2 \tag{3.1}$$

$$\text{and} \quad \hbar = 1.709 \cdot 10^{-100} \, \text{Mpc} \, \frac{\text{km}}{\text{s}} \, 10^{10} M_\odot. \tag{3.2}$$

The first step of the solver is to define basic parameters (see Listing A.1) . The output directory is defined, where all the simulation output is stored during and after the simulation[2]. The *boxsize* is set to be 10 Mpc and is meant to be the length of each space-coordinate axis of the to be simulated cubical Universe under consideration. The *number_axis_points*, N_x, then sets the number of points per axis, so that in one dimension, for instance, you simulate $L = 10^3$ axis points along the x-axis. We have tested different values for L, but since $L = 10^4$ is unnecessarily detailed and requires way too much memory, $L = 10^2$ are not enough axis points in order to get a satisfying result, i.e. the resolution is too low to get correct results. We will later see how L significantly affects the runtime of the simulation.

The `number_time_steps` defines how many iterations the main routine has to go through. Apparently, the more timesteps you choose, the longer the simulation needs in order to finish, but also the bigger is the time interval you are able to simulate. However, for testing purposes, it is sufficient to choose one just to check that the code runs without errors and afterwards, values of $10^2 - 10^3$ are appropriate to check if the evolution of ψ is computed correctly, for example. In the real simulations we chose way higher numbers of timesteps to get significant results on the hand but also to check if there is an upper limit at which the simulation breaks and gives wrong results or to check when the time interval gets too large, so that the numerical error made is scaled to too large values.

Finally, the `mass_axion` is, as it suggests, the axion mass, m_a, for which we chose a value at the lower boundary of the ULA masses, i.e. $m_a = 10^{-33}$ eV, which converts in our units (1.3) to

$$m_a \approx 9 \cdot 10^{-100} \cdot 10^{10} \, M_\odot \approx 10^{-99} \cdot 10^{10} \, M_\odot. \tag{3.3}$$

[2]For the explicit implementation of the `output_directory`-method, see Listing A.9

Recall, that in the Schrödinger-Poisson equation (2.34), a factor $\frac{\hbar}{m_a}$ appears in the first term and its reciprocal, $\frac{m_a}{\hbar}$, in the second term. If $\hbar$ and m_a have values that are several magnitudes of orders apart from each other, both terms diverge quickly from each other, so that either the kinetic or the gravitational term dominate. In order to compensate this as best as possible, apart from the selection of the wavefunction itself, we chose a mass at the lower boundary, so that

$$\frac{\hbar}{m_a} \sim \frac{10^{-100}}{10^{-99}} = 10^{-1} \text{ and } \frac{m_a}{\hbar} \sim 10^{1}, \tag{3.4}$$

what is completely acceptable.

Now, we define the three coordinate-axes[3] via a NumPy `linspace` in order to get N_x equidistant points with total length L. Then, the `position_step`, Δx, is computed, which is the distance between two neighbouring axis points. Note, that if you have N_x axis points, there are $N_x - 1$ spaces in between, so that

$$\Delta x = \frac{L}{N_x - 1} \tag{3.5}$$

holds. Later, we are going to perform a Fourier transformation[4], for which we need the wave-vector, $\vec{k}$, and its inverse square, k^{-2}. Although Fourier analysis is typcially done in terms of frequency and wavefrequency rather than position and wave number, we are allowed to call NumPy's `fftfreq`-method from the `fft` package since the formula is basically the same going from coordinate- to reciprocal-space. It only needs to know N_x and Δx due to (3.5). The result is given in a somewhat cumbersome ordering, i.e. the null frequency, then the positive and then the negative frequencies, but this does not affect k^{-2} since addition is commutative. k^2 is obviously obtained as the scalar product $k^2 = \vec{k} \cdot \vec{k}$, what is computed via NumPy's `dot`-method.

[3] Note, that we will always present the code that is used by the solver in 3D. However, later we will also present results from the one-dimensional simulations that were the first step to build the solver. The 3D code can be easily brought to 1D by mostly just reducing the number of dimensions in the used arrays.

[4] See section 3.3.

3.2 How to define a wavefunction ψ

Now, we need to initialize the wavefunction, ψ, that we want to evolve in time (see Listing A.2).

First, an empty 3D-array is created with NumPy's `zeros`-method, where we specifically demand the values to be complex since $\psi \in \mathbb{C}$ in general. Second, we must compute the initial wavefunction, ψ_0, at each grid point, which is done in three nested `for`-loops that go through all axes points and store the results in the previously created empty 3D array. One can use basically every wavefunction one desires to simulate by simply inserting your equation in the line

```
1     wavefunction[xi,yi,zi] = ...
```

3.3 The main routine: Evolving ψ in time

3.3.1 Compute the gravitational potential Φ

The first step of the main routine is to compute the current gravitational potential, Φ, what is depicted in the code of Listing A.3. Recall, the Poisson equation (2.31)

$$\Delta\Phi = 4\pi G\rho, \tag{3.6}$$

where $\Delta = \nabla^2$ and $\rho = |\psi|^2$. Hence, we only need to know the wavefunction at the current time, what we do apparently. The idea is to compute the density, ρ, first. Then, we apply a *Fourier transformation* (FT) on the Poisson equation (2.31) to end up with an algebraic equation, i.e.

$$\Delta\Phi \equiv \nabla^2\Phi = 4\pi G\rho \quad \overset{\text{FT}}{\Rightarrow} \quad -k^2\tilde{\Phi} = 4\pi G\tilde{\rho}, \tag{3.7}$$

where a tilde denotes the Fourier transformed variables. Recall, that the FT of a function, $f(x)$, where x is to be understood in coordinate-space, is given by[17]

$$\tilde{f}(k) := \text{FT}[f(x)] = \int f(x)e^{-ikx}dx \tag{3.8}$$

in one dimension and in three dimensions trivially after the replacements $x \to \vec{x}$, $k \to \vec{k}$ and $dx \to d^3x$. Thus, one can easily check the two properties

$$\text{FT}\left[\frac{d^n f(x)}{dx^n}\right] = (ik)^n \cdot \text{FT}[f(x)] \quad \text{and} \quad \text{FT}[C \cdot f(x)] = C \cdot \text{FT}[f(x)], \tag{3.9}$$

where C =const. and $n \in \mathbb{N}_0$. Note, that both properties transform to their 3D pendants via the same replacements as above. Note additionally, that k is the pendant of x in the reciprocal Fourier space and that the inverse FT (FT^{-1}) is given by

$$f(x) := \mathrm{FT}^{-1}[\tilde{f}(k)] = \int \tilde{f}(k) e^{ikx} dk. \tag{3.10}$$

The Fourier transformed Poisson equation (3.7) can be rearrangend to

$$\tilde{\Phi} = \frac{4\pi G}{-k^2} \tilde{\rho}, \tag{3.11}$$

so that the inverse transformation (3.10)

$$\Phi = \mathrm{FT}^{-1}[\tilde{\Phi}] = \mathrm{FT}^{-1} \left[\frac{4\pi G}{-k^2} \tilde{\rho} \right] \tag{3.12}$$

should give the desired potential, Φ. The ever constant k^{-2} appearing in the equation was already computed in the beginning of the simulation in section 3.1, so that we can immediately apply NumPy's `fftn`-method for the Fast Fourier transformation[5] (FFT) in n-dimensions and NumPy's `ifftn`-method for the inverse Fast Fourier transformation in n-dimensions.

3.3.2 Computing second derivatives of ψ

The second step in the main routine is to compute $\Delta\psi$ that appears in the Schrödinger-Poisson equation (2.34). This is done via a *centered finite-differencing* method[6] as can be seen in Listing A.4. The idea is to consider Taylor expansions $f(x + 2\Delta x)$, $f(x + \Delta x)$, $f(x - \Delta x)$ and $f(x - 2\Delta x)$ of a continuous function, $f(x)$, up to a certain order, so that one can then compute a linear combination of the results in order to solve the resulting

[5] The FFT is an extremely important upgrade to the standard discrete Fourier transformation in terms of computation time. However, we do not want to go into detail about the algorithm here as there is a huge variety of literature on the topic. An educational approach can be found in [18], for instance.

[6] Again, there is a wide variety of literature on that topic. One example is the book [19] by Milne who is one of the pioneers of the method.

equation for the desired derivative. For example,

$$f(x+2\Delta x) \approx f(x) + 2\Delta x f^{(1)}(x) + 4\Delta x^2 \frac{f^{(2)}(x)}{2!} + 8\Delta x^3 \frac{f^{(3)}(x)}{3!} \tag{3.13}$$

$$f(x+\Delta x) \approx f(x) + \Delta x f^{(1)}(x) + \Delta x^2 \frac{f^{(2)}(x)}{2!} + \Delta x^3 \frac{f^{(3)}(x)}{3!} \tag{3.14}$$

$$f(x-\Delta x) \approx f(x) - \Delta x f^{(1)}(x) + \Delta x^2 \frac{f^{(2)}(x)}{2!} - \Delta x^3 \frac{f^{(3)}(x)}{3!} \tag{3.15}$$

$$f(x-2\Delta x) \approx f(x) - 2\Delta x f^{(1)}(x) + 4\Delta x^2 \frac{f^{(2)}(x)}{2!} - 8\Delta x^3 \frac{f^{(3)}(x)}{3!}, \tag{3.16}$$

where $\approx$ instead of a true equality comes from cutting of higher order expansion terms. Now, the first derivative, $f'(x) \equiv f^{(1)}(x)$, is given by

$$f'(x) \approx \frac{f(x-2\Delta x) - f(x-\Delta x) + 8f(x+\Delta x) + f(x+2\Delta x)}{12\Delta x}. \tag{3.17}$$

Likewise, one can derive the second derivative, $f''(x) \equiv f^{(2)}(x)$, to be

$$f''(x) \approx \frac{f(x-2\Delta x) - 16f(x-\Delta x) - 30f(x) + 16f(x+\Delta x) + f(x+2\Delta x)}{12\Delta x^2}. \tag{3.18}$$

Note, that both, $f'(x)$ and $f''(x)$ given above, are $\mathcal{O}\left(\Delta x^4\right)$ approximations. These results are easily transferable to higher orders. However, a fourth-order approximation is sufficient for our solver. Since ∇ contains partial derivatives for each spatial direction, we can simply apply the same second derivative formula (3.18) to each spatial direction in a nested for-loop over all grid points. Since this requires to address each grid point three times, going from one to three dimensions decreases the computation speed from $\mathcal{O}\left(N_x\right)$, where N_x is the number of data points per axis, down to $\mathcal{O}\left(3N_x^3\right)$ and since we need to address five values of the `wavefunction`-array, we need $15N_x^3$ array-accesses per simulation step in order to compute the second derivative. It is safe to say, that this is clearly the bottleneck of the simulation in terms of runtime. Note additionally, that one typically uses a centered finite-differencing method only for the grid points between the end points and a forward (backward) algorithm for the leftest (rightest) end. This does not apply to our solver since it applies periodic boundary conditions to the simulated box, making it theoretically infinitely large, so that we do not have to distinguish between the algorithms.

The code itself (see Listing A.4) is precisely the formula (3.18) in a nested for-loop, first, in order to compute the derivative, $\Delta\psi$, in a three-dimensional array `vel_schr`[7]. Then, the scale-factor, a, in a flat, matter-dominated Universe is computed[8] to be

$$a(t) = \left(\frac{t}{t_0}\right)^{\frac{2}{3}}. \tag{3.19}$$

Its necessity results from the transformation of the Schrödinger-Poisson equation to comoving coordinates. Finally, the seperate pieces are put together to compute the Schrödinger-velocity `vel_schr`. Note, that we must shift the time in $a(t)$, so that it starts at some initial time, t_i, that we would like to be different from $t_i = 0$, i.e. the Big Bang, in general. For this purpose, it is sufficient to make a time translation, so that

$$a(t) = \left(\frac{t_i + t}{t_0}\right)^{\frac{2}{3}} \tag{3.20}$$

is going to be the scale-factor equation in the solver. For t_i we chose the time at redshift $z_i = 1000$, what we actually can use (3.19) with the definition $a := (1+z)^{-1}$, so that we get

$$t(z) = t_0(1+z)^{-\frac{3}{2}} \;\Rightarrow\; t_i \equiv t(z=1000) \overset{t_0 \approx 13.7\,\mathrm{Gyr}}{\approx} 330000\,\mathrm{yr}. \tag{3.21}$$

Note, that since we are working units (1.3), we need to modify the unit of time correspondingly, so that

$$[T] = \frac{[L]}{[v]} = \frac{1\,\mathrm{Mpc}\cdot\mathrm{s}}{\mathrm{km}} = 3.086\cdot 10^{19}\,\mathrm{s} = 0.978\cdot 10^{12}\,\mathrm{yr} \tag{3.22}$$

is the unit of time we are using, meaning that if the simulation, that is fully working in our chosen units, giving back a time of 1 it actually means $0.978\cdot 10^{12}$ yr. Accordingly, the initial time, i.e. at $z = 10^3$, is now $t_i \approx 4.43\cdot 10^{-7}$ and the age of the Universe is $t_0 \approx 14.11\cdot 10^{-3}$ in our new time unit.

[7] Instead of having two separate arrays for the second derivative and the Schrödinger velocity, respectively, we spare some memory by just using one for the Schrödinger-velocity directly since we do not need $\Delta\psi$ apart from that.

[8] Start with the Friedmann equation $\left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G}{3}\rho(a)$, where $\rho(a) = \rho_0 a^{-3}$ for matter. Then solve for $a(t)$ to get $a \sim t^{2/3}$, so that you can take $a(t)/a_0$ with $a_0 = 1$, as usual.

3.3.3 Compute the Schrödinger-velocity and the timestep

Let us now justify retroactively what we mean in terms of the so-called Schrödinger-velocity. Recall, that the Schrödinger-Poisson equation in one dimension reads

$$i\hbar\frac{\partial\psi}{\partial t} = \left[-\frac{\hbar^2}{2m_a a^2}\frac{\partial^2}{\partial x^2} + \frac{m_a}{a}\Phi\right]\psi. \tag{3.23}$$

Dividing by the prefactor of the time-derivative, one gets an equation for the *Schrödinger velocity*, v_S, i.e.

$$v_S = i\left[\frac{\hbar}{2m_a a^2}\frac{\partial^2}{\partial x^2} - \frac{m_a}{\hbar a}\Phi\right]\psi. \tag{3.24}$$

We call this equation a velocity because we will use it to evolve the wave-function ψ as

$$\psi_{n+1} = \psi_n + v_S \cdot \Delta t_n, \tag{3.25}$$

where Δt_n is the step size of the n-th iteration (see Listing A.6). Note, that the term in square brackets in (3.23) is the Hamiltonian, which has the dimension of energy, which in turn has dimensions $[\text{Energy}] = E = ML^2T^{-2}$. One can check that this indeed turns out since $[\hbar] = ML^2T^{-1}$, mass trivially has dimensions of mass, $[a] = 1$, $[\partial x^2] = L^2$ and as noted in the previous chapter, $[\Phi] = EM^{-1}$. Using these information now in the Schrödinger velocity (3.24), one finds that both terms in the square brackets have dimensions

$$1^{\text{st}}\text{ term: } \left[\frac{\hbar}{m_a a^2}\frac{\partial^2}{\partial x^2}\right] = \frac{ML^2}{TM}\frac{1}{L^2} = \frac{1}{T} \quad \text{and} \tag{3.26}$$

$$2^{\text{nd}}\text{ term: } \left[\frac{m_a}{\hbar a}\Phi\right] = \frac{T}{ML^2}\frac{ML^2}{T^2} = \frac{1}{T} \quad , \tag{3.27}$$

so that the Schrödinger velocity in fact describes the change of the wave-function ψ with respect to time. Recalling, that $[\psi] = M^{1/2}L^{-3/2}$, v_S must have dimensions

$$[v_S] = \frac{[\psi]}{T} = \frac{M^{1/2}}{L^{3/2}T}, \tag{3.28}$$

which are not the dimensions of a classical velocity we are familiar with, especially since $v_S \in \mathbb{C}$ in general.

The third step in the main routine (see Listing A.5) is to compute `time_step`, Δt, that is not fixed, but adjusted to v_S, so that the evolution is stable. Think

about a moving point mass on a one dimensional axis. If the velocity of the point mass is high, you would like to choose a sufficiently small timestep because then the change in position is not conflicting with your chosen resolution, i.e. if the timestep is chosen too high you might increase the error of the result significantly. For example, you choose a timestep, so that the point mass travels by $2\Delta x$ in Δt, but after $1\Delta x$ it rapidly decelerates to half its previous velocity and in total just travels to $1.5\Delta x$, meaning that you have an error of $0.5\Delta x$ in the position of the point mass. However, for our simulation we follow the idea of Courant, Lewis and Lewy[20] that can broken down to

$$C \leq v \cdot \frac{\Delta t}{\Delta x} \quad \Leftrightarrow \quad \Delta t \leq C \cdot \frac{\Delta x}{v}, \tag{3.29}$$

where C is the so-called *CFL-number*[9] and can be regarded as a free parameter, basically. By considering the inequality, it gives the amount of steps in terms of Δx an object with velocity v in time Δt makes, so that $C = 1$ means that it moves precisely Δx. For our simulations[10], we chose $C = 0.1$.

Now, the question is what velocity should be taken to compute $v_{\max}$. There are several approaches to define a velocity and even more conditions on the timestep could be applied, see for instance [13], but we just want to briefly touch upon the following three. First, if simply take v_S we must deal with $v_S \in \mathbb{C}$ in general, so that we first have to extract some real quantity out of it. There are various ways, but let us focus on the two most simple ones, namely taking the absolute value, $|v_S|$, or simply adding the real and imaginary part of v_S. They are related as

$$|v_S|^2 = \Re^2[v_S] + \Im^2[v_S] \tag{3.30}$$

$$\leq \Re^2[v_S] + 2Re[v_S]\Im[v_S] + \Im^2[v_S] = (\Re[v_S] + \Im[v_S])^2, \tag{3.31}$$

so that after taking the square root

$$|v_S| \leq |\Re[v_S] + \Im[v_S]| \tag{3.32}$$

holds. This suggests that taking the absolute value of the sum of the real and imaginary part of the Schrödinger velocity is always greater than the

[9] Note, that it is common in literature to just speak of the *Courant-condition* and the *Courant-number*, but we stick to CFL.

[10] The chosen value of C often depends on the algorithms that are used to compute v and typically $0 < C \leq 1$, where smaller values of C apparently give more accurate results in trade for a higher runtime if one desires to simulate a fixed amount of time rather than a fixed amount of timesteps.

absolute value itself if $\Re \neq \Im \neq 0$, what we assume to be true in general. Second, considering the Ehrenfest-Theorem, the velocity operator in Quantum Mechanics (QM) reads

$$\hat{v}_{\text{Ehre}} = \frac{1}{i\hbar}[\hat{x}, \hat{H}], \tag{3.33}$$

where the hat indicates an operator and the square brackets the commutator $[A, B] = AB - BA$, as usual. From the Schrödinger-Poisson equation (2.34) we read off $\hat{H}$ with the QM-typical replacement $-i\hbar\nabla \rightarrow \hat{p}$, so that we obtain

$$\hat{v}_{\text{Ehre}} = \frac{1}{(i\hbar)^2}\left[\hat{x}, \frac{1}{2m_a a^2}\hat{p}^2 + \frac{m_a}{a}\Phi(\hat{x})\right], \tag{3.34}$$

where we factored one $(i\hbar)^{-1}$ out of the commutator. Now,

$$\begin{aligned}
\hat{v}_{\text{Ehre}} &= \frac{1}{(i\hbar)^2}\left(\left[\hat{x}, \frac{1}{2m_a a^2}\hat{p}^2\right] + \left[\hat{x}, \frac{m_a}{a}\Phi(\hat{x})\right]\right) && (3.35)\\
&= \frac{1}{(i\hbar)^2}\left\{\frac{1}{2m_a a^2}\left(\hat{p}\,[\hat{x}, \hat{p}] + [\hat{x}, \hat{p}]\,\hat{p}\right) + \frac{m_a}{a}\left[\hat{x}, \Phi(\hat{x})\right]\right\} && (3.36)\\
&= \frac{1}{(i\hbar)^2}\left\{\frac{1}{2m_a a^2}\left(\hat{p}i\hbar + i\hbar\hat{p}\right) + \frac{m_a}{a}\cdot 0\right\} && (3.37)\\
&= -\frac{1}{m_a a^2}\nabla, && (3.38)
\end{aligned}$$

where in the first line we used $[A + B, C] = [A, C] + [B, C]$, in the second line we used $[AB, C] = A[B, C] + [A, C]B$, in the third line used the canonical commuation relation $[\hat{x}, \hat{p}] = i\hbar$ as well as the fact that the last term vanishes since the commutator is taken of an observable and a function of the same observable, which commutes always, and in the last line we inversely made the replacement $\hat{p} \rightarrow -i\hbar\nabla$. However, applying the Ehrenfest-velocity operator to ψ, we get the Ehrenfest-velocity

$$\vec{v}_{\text{Ehre}} = \hat{v}\psi = -\frac{1}{m_a a^2}\vec{\nabla}\psi, \tag{3.39}$$

but since this would force us to compute the first partial derivatives of ψ as well, we discard this ansatz as the numerical computation of the second derivative is already the solver's bottleneck in terms of computation time and we do not want to make things worse. We do not need to worry too much about the actual interpretation here since we are solely interested in getting an appropriate $v_{\max}$ to compute a proper timestep. The third

ansatz we could think of, is to check the values of $|\dot{\psi}/\psi|$. Assume that $\psi \neq 0 \forall\, t, \vec{x}$, then it should be safe to state

$$\left|\frac{\dot{\psi}}{\psi}\right| = \left|\frac{1}{\psi}\frac{\partial\psi}{\partial t}\right| \overset{(2.34)}{=} \left|\frac{1}{\psi}v_S\right|, \tag{3.40}$$

but since we assumed $\psi \neq 0$, what implies $|\psi| > 0$, we get

$$\left|\frac{v_S}{\psi}\right| < |v_S|, \tag{3.41}$$

so that we basically end up with our first ansatz. Taking all this into account, we adapt our first result and define

$$v_{S,\max} = \max\left\{|\Re[v_S] + \Im[v_S]|\right\}, \tag{3.42}$$

where max indicates that the biggest value of the resulting Schrödinger-velocity vector should be taken. Note, that we tacitly dropped the vector arrows for the Schrödinger-velocity and assumed the absolute value of the sum to be taken element-wise. We can use the maximum then to finally compute

$$\Delta t = C \cdot \frac{\Delta x}{v_{S,\max}}. \tag{3.43}$$

3.3.4 Update ψ

The fourth and final step of the main routine is to evolve ψ according to (3.25). The code is trivially just incrementing `wavefunction`, ψ, by the product of `timestep`, Δt, and `vel_schr`, v_S, as we have discussed above. As you can see in Listing A.6, before evolving ψ, we make a 3D scatterplot at all 10^3 timesteps (for plotting see Listing A.10 and A.11) to be able to track the evolution of $\rho = |\psi|^2$.

3.4 Finishing the simulation run

As a fourth and last step, the solver makes plots (for plotting see again Listing A.10 and A.11) from different variables (see Listing A.7).

In fact, we make three important final plots after the main routine. The first one is a 3D scatter plot of the final density. The second plot shows the maximum velocities, $v_{\mathrm{max,i}}$ against their corresponding timestep index and the third plot shows the timestep values against their corresponding timestep index. Due to the Courant condition (3.43), one expects that the latter two plots are somewhat reciprocal to each other, which is a good check that they were computed correctly.

Finally, we store the ψ-values, `wavefunction`, as well as the $v_{\mathrm{max,i}}$- and Δt-values, respectively, in .npz-files via NumPy's `savez`-method (see Listing A.8). The constants G and $\hbar$ as well as the boxsize, L, the number of axis points, N_x, the number of timesteps, N_t, the position step, Δx along k^{-2}, the simulation- and run-time are stored in a .txt-file for an easy access by the user. This is done for convenience in order to match simulation outputs with the most important parameters and variables.

Chapter 4

Using the Schrödinger-Poisson solver

In the following we would like to test the solver we have built in the previous chapter with known examples, so that we should be able to tackle real world applications like the deeply desired axion wavefunction. However, we start by investigating the one-dimensional solver in section 4.1 with simple examples like Gaussian pulses (subsection 4.1.1) and a point mass (subsection 4.1.2). Then we check if the simulation runs stable for a large amount of timesteps (subsection 4.1.3) and take a look on the open problems for the one-dimensional solver (subsection 4.1.4). The next step is to consider the three-dimensional solver in section 4.2, where we again take a look at a Gaussian (subsection 4.2.1) as well as a point-mass (subsection 4.2.2). Then we finally making first steps in simulating the axion wavefunction (subsection 4.2.3) and discussing open problems (subsection 4.2.4).

4.1 In one dimension

4.1.1 Evolving Gaussian pulses

First of all, let us try to simulate two Gaussian pulses, where each pulse is described by

$$\psi_0(x) = \frac{c}{\sqrt{2a}} \cdot \exp\left\{ik_0 x - \frac{x^2}{4a}\right\}, \tag{4.1}$$

where

$$a = \frac{\sigma}{2}, \quad c = \left(\sqrt{2\sqrt{\pi}}\right)^{-1} \tag{4.2}$$

with $\sigma = 0.5$ and $k_0 = (-1)^n m\pi/L$ with $m = 10^6$. The latter was chosen because the group velocity of a wave is defined as

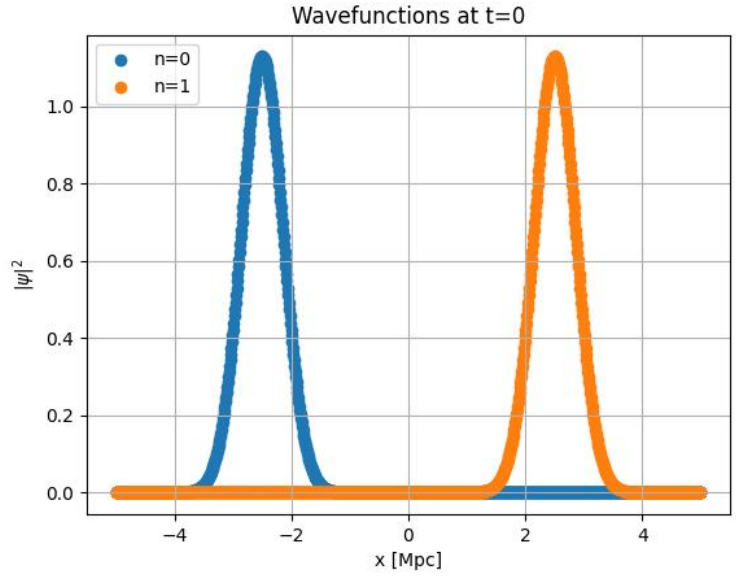
$$v_G := \frac{\partial\omega(k)}{\partial k} = \frac{\hbar k}{m} \tag{4.3}$$

since $2m\omega(k) = \hbar k$ holds for the Gaussian pulse, so that by choosing m in k_0 accordingly, we can influence the direction in which a pulse is going to propagate over time. However, in any case this is going to be a very small velocity compared to the size of the simulated box, so that in practice we will not be able to really observe such a behavior.

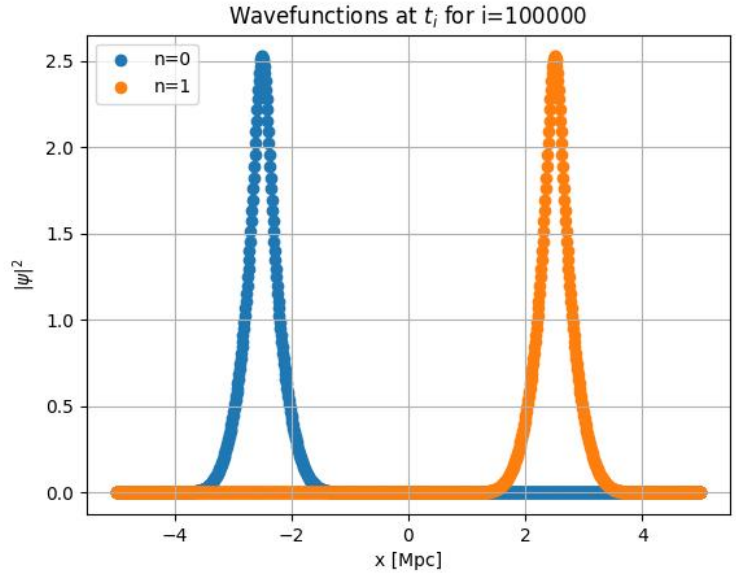
Note, that the blue ($n = 0$) pulse was shifted 3.25 Mpc along the negative x-axis and the orange ($n = 1$) pulse by the same distance along the positive x-axis. You see the initial densities qualitatively in 4.1(a). We have let them evolve for $t_{\max} = 10^5$ timesteps, what you can qualitatively see in 4.1(b). One can clearly see that the pulses are sharpening, i.e. their variances decrease, and the pulses seem to become more concentrated. In order for the normalization to be invariant under time translation it appears logical that the amplitudes of the peaks seem to increase as well. However, we will see in subsection 4.1.4 that this turns out to be not quite exact. Note, that the two Gaussian pulses are a representative test choice rather than depicting an interesting physical system, which is why we restrain ourselves on qualitative investigations for now.

In figure 4.2 we plotted the maximum Schrödinger-velocity, $v_{\mathrm{S,max}}$, and the corresponding timesteps, Δt, against the timestep indices, i, to visualize their evolution. As one could expect from (3.43), we observe the $\Delta t \sim v_{\mathrm{S,max}}^{-1}$ behavior very clearly, so that it is sufficient in the following to focus on one of them. If one considers the timestep (figure 4.2(b)) one can see that the timesteps are decreasing over time. It also seems to oscillate heavily, what indicates some instability that we will investigate further in subsection 4.1.4.

We can summarize that it is no problem to simulate two Gaussian pulses in our Schrödinger-Poisson solver. Of course, one can just simulate one Gaussian or arbitrarily many more than just two, but one has to keep in mind that the wavefunction simulation points must be stored, which should not become a problem in the three-dimensional solver, but still, one can easily overshoot the available memory capacity.

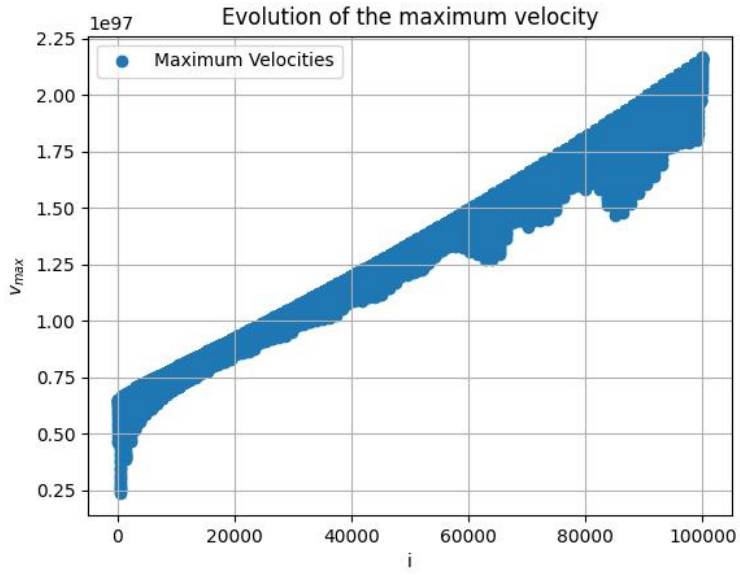


(a) *Density of two Gaussian pulses (4.1) at initial timestep index $i = 0$.*

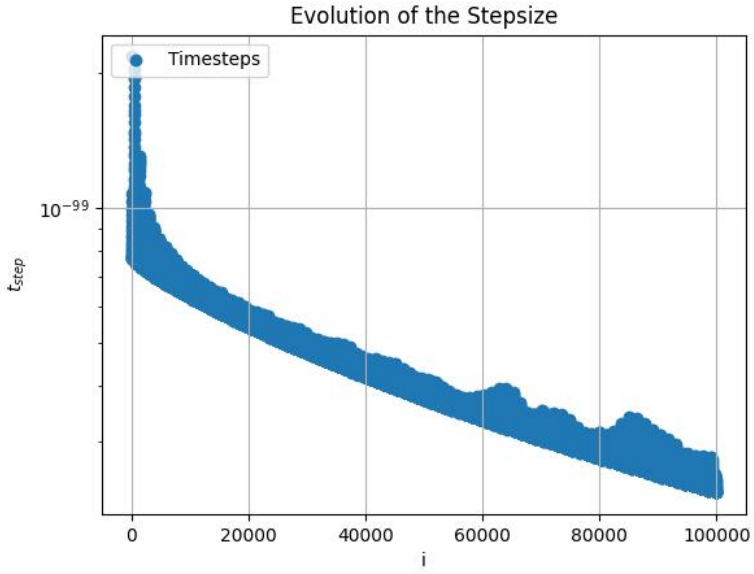


(b) *Density of two Gaussian pulses (4.1) at final timestep index $i = 10^5$.*

Figure 4.1: *Two Gaussian pulses. The left (blue, $n = 0$) pulse has a group velocity that points to the positive x-direction, whereas the right (orange, $n = 1$) pulse has a group velocity that points to the negative x-direction. Ordinate values are to be understood qualitatively, hence, no units were given.*



(a) *Maximum Schrödinger velocity, $v_{S,max}$, against timestep index, i.*



(b) *Timestep, $t_{step} \equiv \Delta t$, against timestep index, i.*

Figure 4.2: *Time-evolution of the two important evolution equation (3.25) parameters, namely the timestep Δt and the maximum Schrödinger-velocity, $v_{S,max}$. Note, that the timestep-axis is given logarithmically, which actually makes no difference in the visualization since $v_{S,max} \sim \Delta t^{-1}$ due to (3.43) and by considering the velocity axis in (a).*

4.1.2 Testing the Poisson-solver

A standard test to check that the gravity part, i.e. the Poisson-solver, in the solver works correctly is to compute the potential of a point mass. Say, we want to model a point mass, m, at position x_0, then

$$\psi(x) = \delta(x - x_0), \tag{4.4}$$

where δ is the Dirac-Delta distribution, so that

$$|\psi(x)|^2 = \delta^2(x - x_0) \tag{4.5}$$

is to be expected, then for $x_0 = 0$ we indeed have a the to a point mass corresponding wavefunction in figure 4.3(a). Note, that we multiplied the Dirac-Delta with a factor of 10 in ψ, so that the amplitude of δ^2 is 100 as displayed, what we have done solely to get better displayable values on the potential axis in figure 4.3(b). In the same figure you can easily recognize the typical $1/r$-behavior of the point mass potential. Note, that we manually capped the potential values in an infinitesimal vicinity around $x = 0$ in order to make the $1/r$-behavior clearly visible.

As a second test case, we have chosen a mass distribution in form of a cosine,

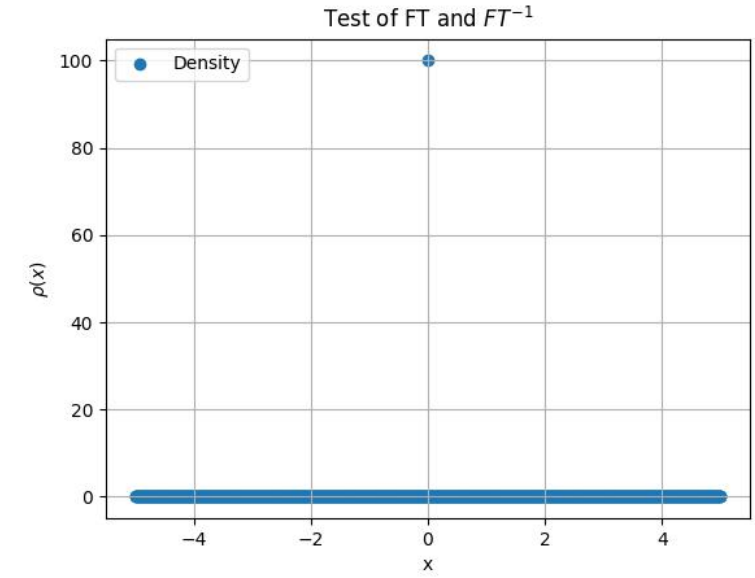
$$\psi(x) = \cos(x) \quad \Rightarrow \quad |\psi(x)|^2 = \cos^2(x), \tag{4.6}$$

visible in figure 4.4(a). Intuitively, we expect to find potential wells at the same positions as mass density peaks. This behavior should be recovered also because by solving the Poisson equation (2.31) via the method described in subsection 3.3.1, we basically expect nothing else than

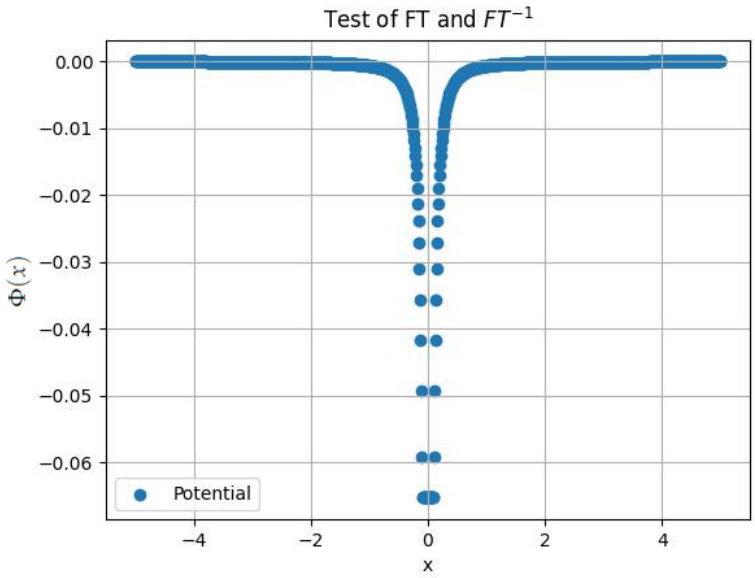
$$\Phi \sim -\cos^2(x). \tag{4.7}$$

The resulting potential (figure 4.4(b)) shows, as expected, the $-\cos^2(x)$-behavior very clearly and one can also see that the positions of the potential minima match very well with the maxima of the density distribution.

Both results give us confidence that the gravity part, i.e. the Poisson-solver, works correctly, what is obviously a crucial component of the Schrödinger-Poisson solver.

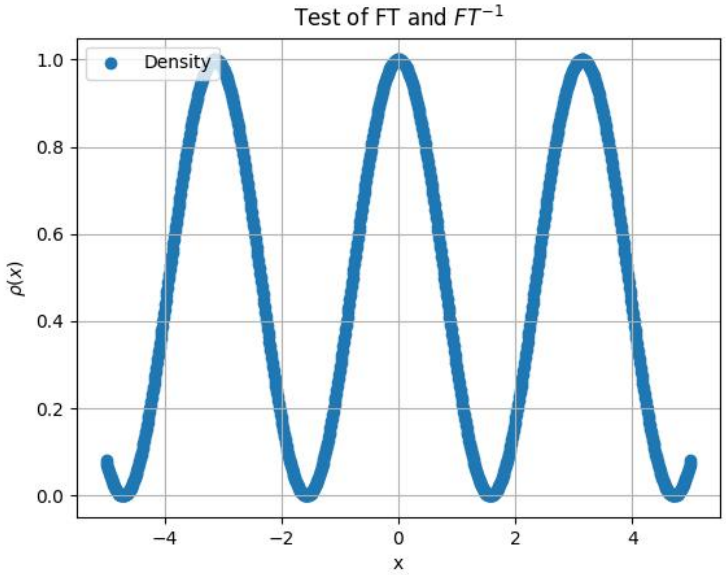


(a) *Density field of a point mass at $x = 0$.*

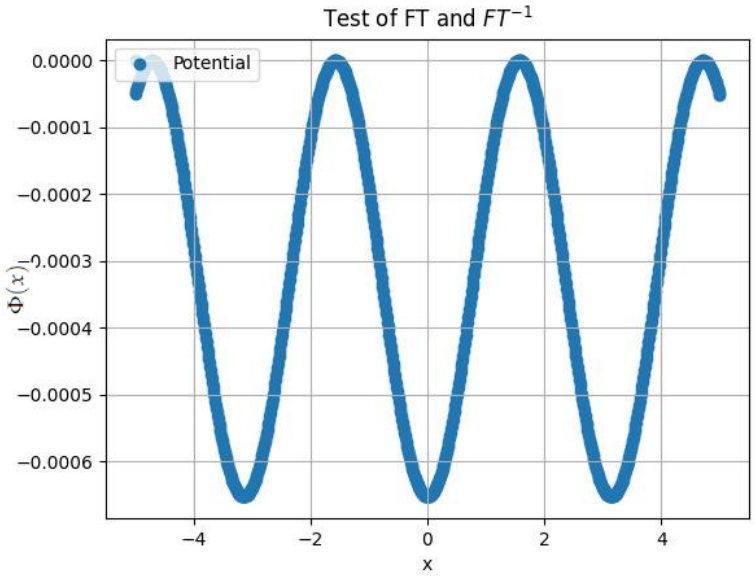


(b) *Potential of a point mass at $x = 0$.*

Figure 4.3: *A point mass is modeled in its simplest form by $\psi = \delta(x)$ in the center, so that one indeed recovers the expected $1/r$-behavior of the gravitational potential of the point mass. To avoid the diverging values in the vicinity of $x = 0$, an infinitesimal vicinity was manually set to a finite value in order to make the $1/r$-behavior clearly visible. Additionally, ψ was rescaled, so that the potential labels are better readable.*



(a) *Density field of a cosine-mass distribution with $\psi(x) = \cos(x)$.*



(b) *Potential of a cosine-mass distribution with $\psi(x) = \cos(x)$ showing the expected $\Phi(x) \sim -\cos^2(x)$ behavior.*

Figure 4.4: *A mass distribution in form of a cosine is modeled by $\psi = \cos(x)$, so that one can expect potential wells in the denser regions apparently or from the algorithm presented in subsection 3.3.1 to solve the Poisson equation (2.31) that indicates a resulting $-\cos^2(x)$ behavior.*

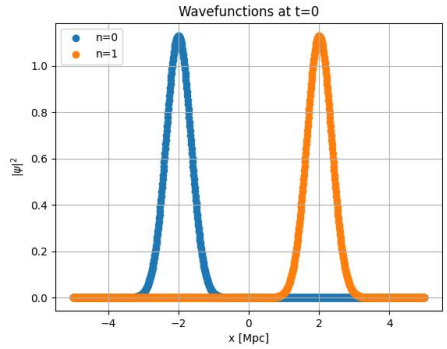
4.1.3 Check of the simulation's stability

In order to test our Schrödinger-Poisson solver's stability, we again used two Gaussian pulses as before, with the slight difference that we only shift both by 2 Mpc in each direction seen from the center. Now, we let the simulation run for $8.5 \cdot 10^6$ timesteps and observe the simulation regularly. In figure 4.5 we have shown three different timestep indices, to represent the evolution of the density fields behavior. For this purpose, we additionally plot the real- and imaginary parts, $\Re[\psi_{n=0}]$ and $\Im[\psi_{n=0}]$, respectively, for the $n = 0$ pulse.

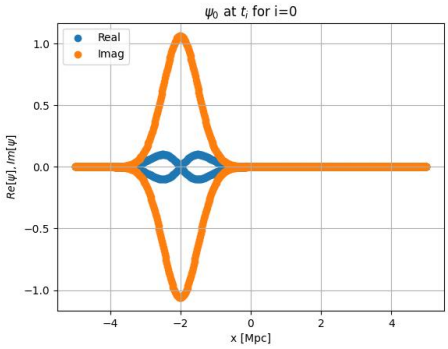
In subsection 4.1.1 we already mentioned that after 10^3 timesteps a sharpening of the density fields can be presumed. By looking at the density field after 10^6 timesteps in figure 4.5(c) this assumptions seems to be approved and strengthened even later after $8.5 \cdot 10^6$ timesteps in figure 4.5(e), when the density field is basically centered on two density peaks at the mean positions of the initial Gaussian pulses. Hence, it seems like the simulation is running stable in the sense that there are no divergences that we are hitting in the first $8.5 \cdot 10^6$ timesteps, but the density field's evolution shows a strong clustering of the modeled matter distributions.

When observing the real and imaginary part of $\psi_{n=0}$, one can surmise basically the same behavior as for the density fields, but it is not quite simple as it seems. The typical behavior of the real and imaginary part is an oscillatory one around the mean of the matter distributions, what could be guessed by looking at figure 4.5(b), but it becomes clear when, for example, making a movie of these splitted plots for the first 10^3 timesteps. We will discuss this in the following subsection in somewhat more detail as it could explain an important open problem of the solver. However, one clearly sees that the smooth real and imaginary part curves are not recoverable after 10^6 timesteps in figure 4.5(d). Both parts seem to be chaotically distributed around the mean of the pulse, but also the width of the curves appears smaller, what fits to the sharpened density peak in figure 4.5(c). This could also be a result of the visualization in the sense that the amplification of the density peaks is also visible in the real and imaginary part. However, the amplification and sharpening goes on, so that after $8.5 \cdot 10^6$ the resulting figure 4.5(f) fits to the the corresponding density field.

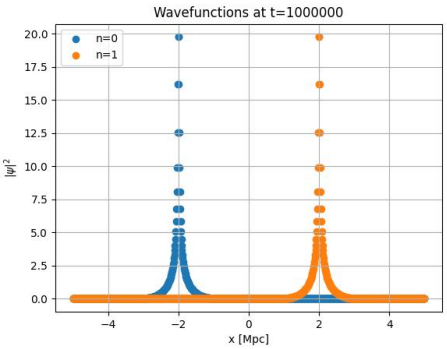
We can safely say that the solver is not running into some form of divergences, but unfortunately, it seems like there could be some form of uncertainty still unrevealed, that amplifies the wavefunctions over time or that we simply observe a strong clustering of apart mass distributions.



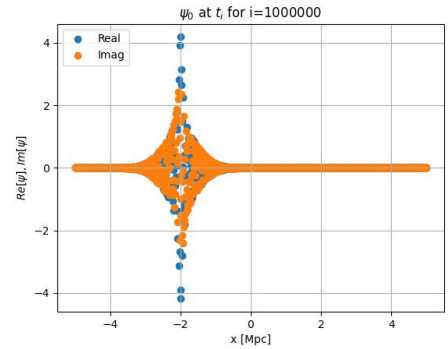
(a) *Density field for two Gaussians at timestep index $i = 0$.*



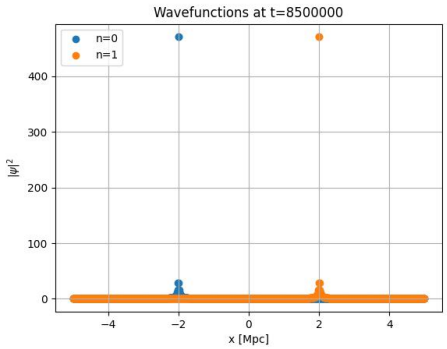
(b) *Real and imaginary part at timestep index $i = 0$.*



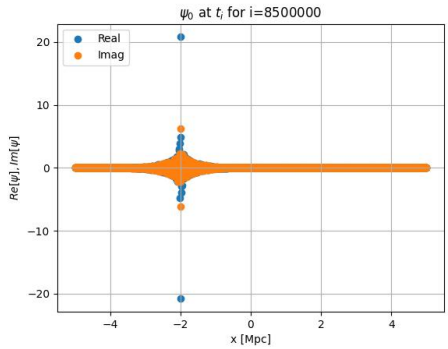
(c) *Density field of two Gaussians at timestep index $i = 10^6$.*



(d) *Real and imaginary part at timestep index $i = 10^6$.*



(e) *Density field of two Gaussians at timestep index $i = 8.5 \cdot 10^6$.*



(f) *Real and imaginary part at timestep index $i = 8.5 \cdot 10^6$.*

Figure 4.5: *For three timestep indices $i \in \{0, 10^6, 8.5 \cdot 10^6\}$, the density field was plotted as well as the real (blue) and imaginary part (orange) of the left (blue, $n = 0$) Gaussian pulse. Whilst the pulses seem to sharpen over time, the behavior of the real and imaginary part gets chaotic, resulting in a somewhat collapsed situation that fits well to the strongly amplified density field.*

4.1.4 Open problem: Non-conserved $|\psi|^2$

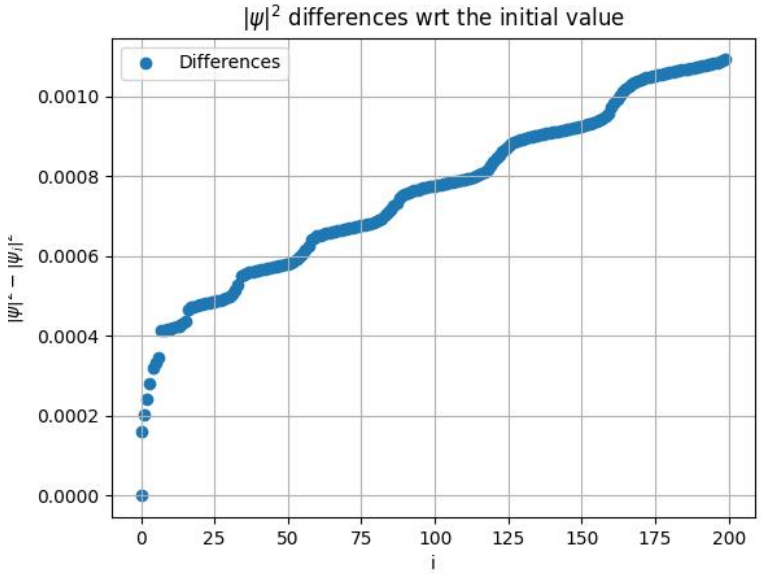
As was presumed in subsections 4.1.1 and 4.1.3, it seems like there could still be some hidden error that causes an ongoing amplification of the density fields. In order to check if there is in fact an error instead of just the observation of clustering of the modeled matter distributions, recall, that

$$\int |\psi(x)| dx \tag{4.8}$$

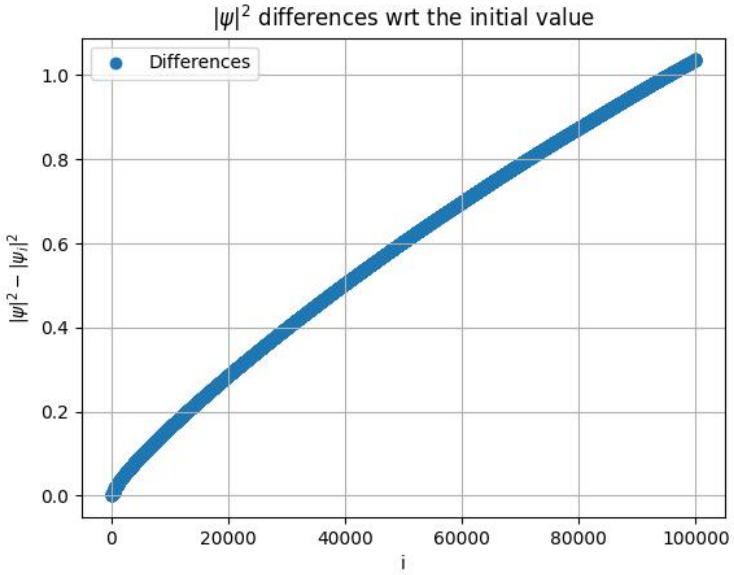
should be a conserved quantity, which is why we track its value at each timestep. This uncovers the following. In the figures in 4.6 we track the just defined integral values subtracted by their initial values for the $n = 0$ Gaussian pulse in figure 4.5(a) from the previous subsection because if it is conserved, it remain zero at all times. Regarding the fact that we are never fully freed from roundoff and truncation errors in numerics, the total error made over 10^5 timesteps (figure 4.6(b)) could be reasonable. However, if one considers only the first 200 timesteps, one sees that in ever increasing time index intervals[1] small jumps in the difference occur that are simply smoothened over large amount of timesteps and hence, not visible in the plot anymore. Hence, we uncovered a systematic error in the solver.

In order to check if the error is made in the quantum or gravitation part of the solver, we cut off the gravitation part in the Schrödinger-Poisson equation (2.34) and redo the simulations for 200 (figure 4.7(a)) and 10^3 (figure 4.7(b)) timesteps. Indeed, we find one and two jumps, respectively. To further investigate the jumps, we again track the real and imaginary part of the wavefunction for the first 10^3 timesteps, which are not given in this thesis, but by making a movie out of the 10^3 plots, one can observe the following. As we suggested in the previous subsection, the real and imaginary part show an oscillatory behavior. For example, let us start in figure 4.5(b), then the real part is increasing until it is approximately the now imaginary curve and the imaginary curve is then the now real curve. Afterwards, this oscillation is repeated, so that we end up with the same picture as at $i = 0$. However, all the time, the real part is expanding towards its amplitude, it shows some kind of rapid acc- and deceleration in the vicinity of its amplitude, so that it overshoots this position a bit what matches exactly with the jumps in figures 4.6 and 4.7. Despite intensive research, it remains an open problem in the solver what causes this overshooting behavior solely in the real part of the wavefunction.

[1] Recall from subsection 4.1.1 that the timestep-size is decreasing over the number of timestep indices, suggesting that the total time interval is still fixed in which the jumps occur.

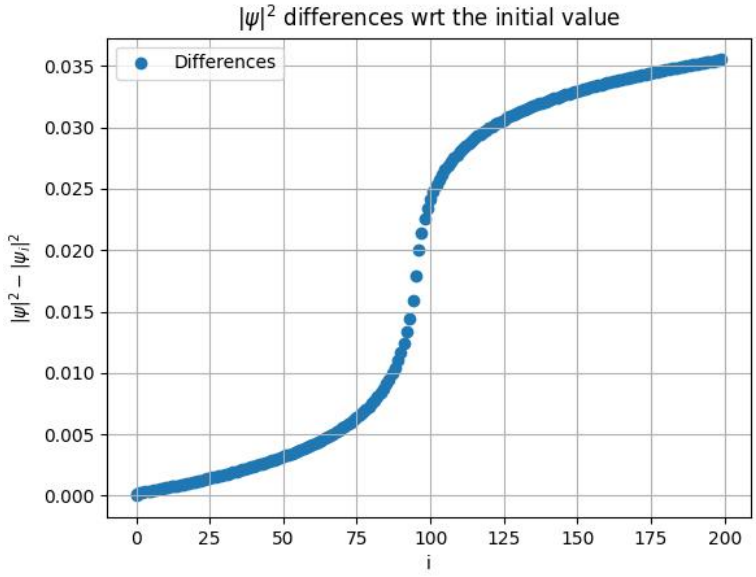


(a) *Differences between the sum of all $|\psi|^2$-values and the sum of the initial $|\psi_0|^2$-values against the timestep indices, i, for 200 timesteps.*

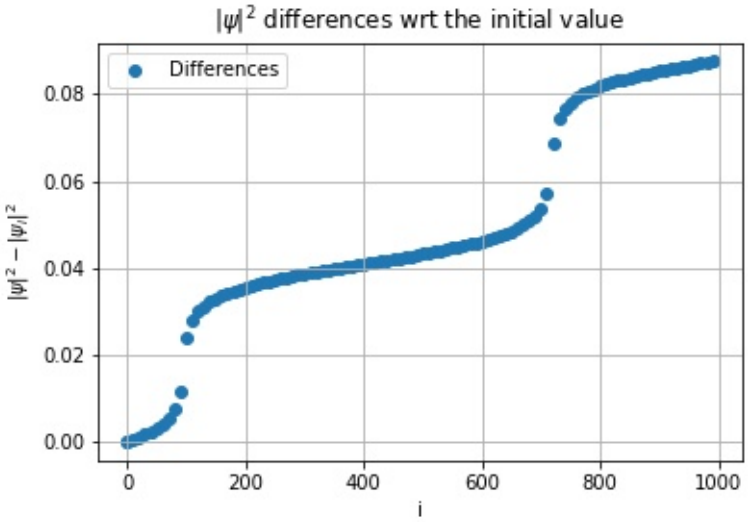


(b) *Differences between the sum of all $|\psi|^2$-values and the sum of the initial $|\psi_0|^2$-values against the timestep indices, i, for 10^5 timesteps.*

Figure 4.6: *Evolution of the total error of the simulation made in the sense of the conserved integral over $|\psi(x)|^2$. Hence, the sum of the $|\psi|^2$-values at a timestep are subtracted by the initial value from which they should not differ significantly over time. One observes a somewhat diverging behavior with little jumps.*



(a) *Differences between the sum of all $|\psi|^2$-values and the sum of the initial $|\psi_0|^2$-values against the timestep indices, i, for 200 timesteps in a pure Schrödinger-solver.*



(b) *Differences between the sum of all $|\psi|^2$-values and the sum of the initial $|\psi_0|^2$-values against the timestep indices, i, for 10^3 timesteps in a pure Schrödinger-solver.*

Figure 4.7: *The same as in the caption of figure 4.6 holds with the important addition that we focus here on the pure Schrödinger-solver, i.e. by neglecting the gravitation term in the Schrödinger-Poisson equation (2.34). The maximum number of timesteps was chosen to visualize one and two jumps, respectively. Note, that a CFL-number of 0.5 was chosen to catch two jumps in the first 10^3 timesteps.*

4.2 In three dimensions

4.2.1 Radially symmetric Gaussian wavefunction

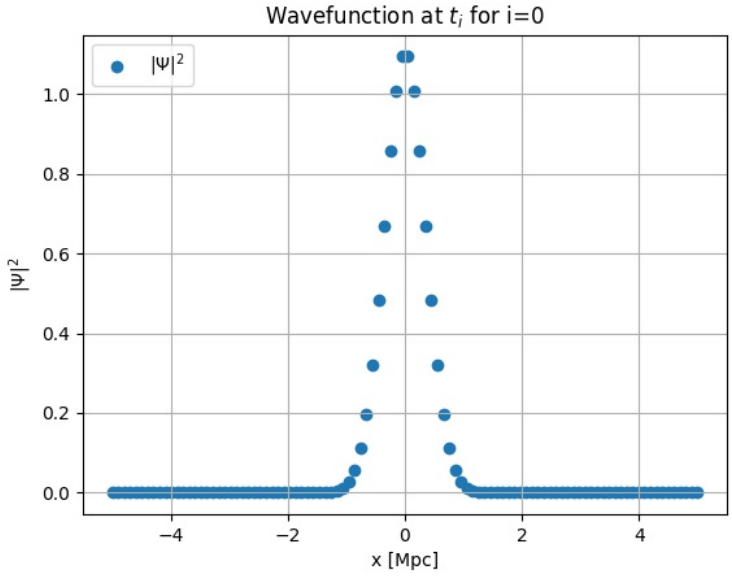
As in the one-dimensional case, we start by testing the solver with a radially symmetric Gaussian wavefunction (4.1), where $x \to \vec{x}$ and $k_0 \to \vec{k}_0$ apparently. We have chosen the same values as before and $n = 0$ in $\vec{k}_0$. However, in contrast to the one-dimensional case, in order to keep simulation runtimes at a minimum, we reduce the number of axis points per axis from $N_x = 10^3$ to $N_x = 10^2$. Note, that whilst it takes just a couple of seconds for the one-dimensional solver to finish for $N_x = 10^2$ and 10^4 timesteps, the three-dimensional solver needed roughly eleven hours to finish.

Instead of showing the three-dimensional scatter plot[2] we present a projection of the $|\psi|^2$-values in x-direction that goes centrally through the simulated box, what you can see in figure 4.8. At the initial timestep (figure 4.8(a)) as well as the final one (figure 4.8(b)) you can clearly see the Gaussian shape, but note, that after 10^4 timesteps one clearly sees the enhancement of the peak compared to the initial one. This indicates again, that the integral over all $|\psi|^2$-values could not be conserved.

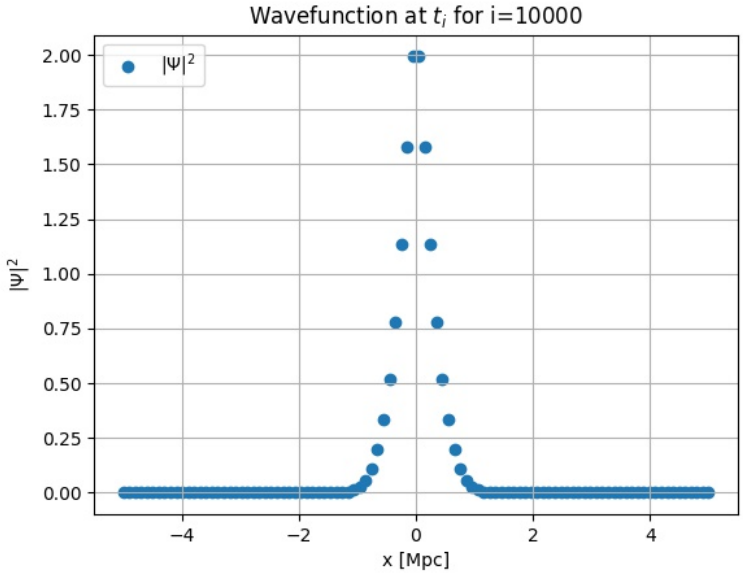
In figure 4.9(a) we plotted again the maximum Schrödinger-velocity, $v_{\mathrm{S,max}}$, and in figure 4.9(b) the timesteps, Δt, against the timestep indices, respectively. Once again, one can clearly see the $\Delta t \sim v_{\mathrm{S,max}}^{-1}$ behavior from the CFL-condition 3.43. One again observes an oscillating behavior in both parameters over time with a clear increasing (decreasing) tendency in the maximum velocities (timesteps) what matches the behavior we observed for the one-dimensional behavior in subsection 4.1.1.

Thus, we conclude again that our solver is capable of simulating, at least, a Gaussian wavefunction properly in time via the non-linear Schrödinger-Poisson equation (2.34). Note, that this is based on a qualitative analysis. However, by choosing $N_x = 10^2$ data points we did not test the limits of the solver, which is also a number of axis points that can be easily simulated on an average desktop computer for educational purposes.

[2] Due to the high number of simulation points that only differ by its colour since the $|\psi|^2$-values are represented by color at each grid point, the outer grid points shield the inner ones. Additionally, only a vicinity of the center has $|\psi|^2$-values that are significantly different from zero, so that both effects in total give three-dimensional representations that are not interpretable.

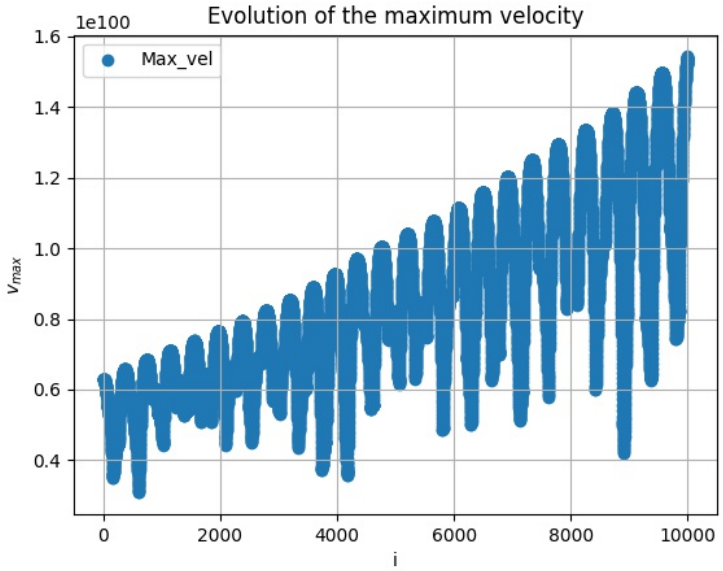


(a) *Density of a radially symmetric Gaussian pulse (4.1) at initial timestep index $i = 0$.*

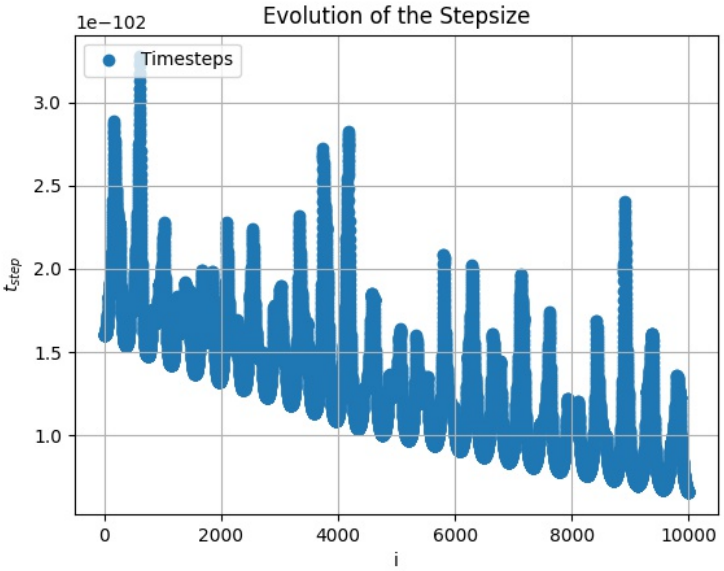


(b) *Density of a radially symmetric Gaussian pulse (4.1) at final timestep index $i = 10^3$.*

Figure 4.8: *One radially symmetric Gaussian pulse. Ordinate values are to be understood qualitatively, hence, no units were given. Note, that in contrast to the one-dimensional discussion, we explicitly have chosen $N_x = 10^2$ instead of $N_x = 10^3$, what does not inflict with the qualitative analysis.*



(a) *Maximum Schrödinger velocity, $v_{S,max}$, against timestep index, i.*



(b) *Timestep, $t_{step} \equiv \Delta t$, against timestep index, i.*

Figure 4.9: *Time-evolution of the two important evolution equation (3.25) parameters, namely the timestep Δt and the maximum Schrödinger-velocity, $v_{S,max}$.*

4.2.2 The density of a point mass

For the three-dimensional solver we once again test the gravity part, i.e. the Poisson-solver, that was described in subsection 3.3.1, with the simplest possible example, namely a point mass in the center of the simulation box. Basically the same equations that were described in subsection 4.1.2 hold with the dimensional replacement $x \rightarrow \vec{x}$ and $\delta \rightarrow \delta^{(3)}$ obviously, where the latter was done to make the dimensionality of the Dirac-Delta distribution explicit. However, we observe an $1/r$-behavior in the potential (figure 4.10) as was expected. Hence, we can safely state, that even in 3D, the Poisson-solver operates as it should.

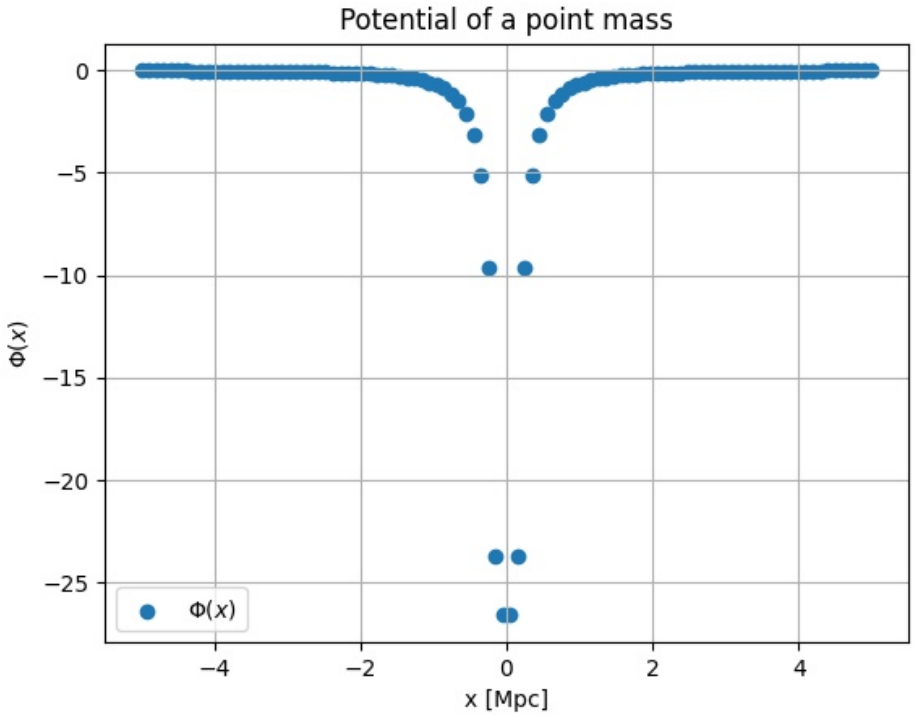


Figure 4.10: *he potential of a point mass in the center of the simulation box after one timestep, solely to show that the expected $1/r$-behavior is clearly visible. The plot shows the potential values along an axis through the center of the simulation box that is parallel to the x-axis.*

4.2.3 An axion wavefunction attempt

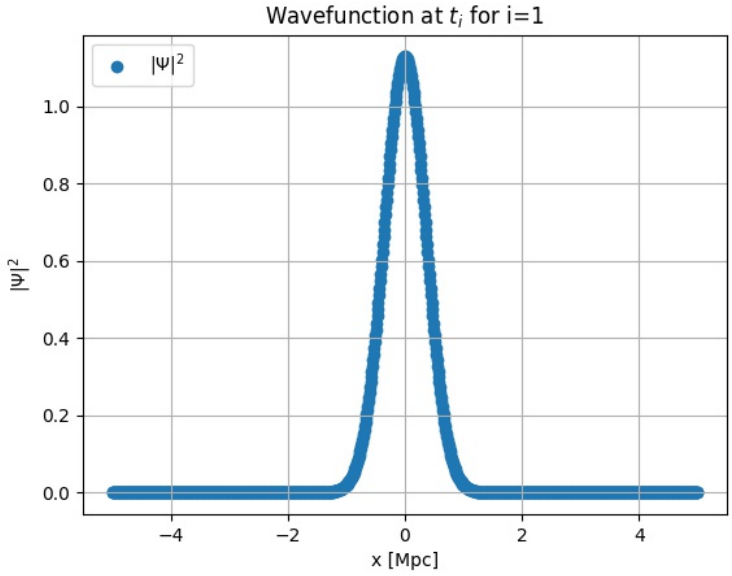
Let us tackle at least a basic example of an axion wavefunction by starting from the WKB-approximation (2.29) for the axion field, ϕ, one notes that the axion wavefunction, ψ, is simply a complex scalar field, so that we could rewrite in polar coordinates as $\psi = |\psi| \exp\{i\alpha\}$, where α is a phase and $|\psi|$ can easily be computed by $\rho_a = |\psi|^2$. If we plug ψ then in the Schrödinger-Poisson equation (2.34), one immediately sees that the exponential drops out, so that α should be a free parameter, which is why we set $\alpha = 0$ for convenience, so that $\psi = |\psi| = \sqrt{\rho_a}$ is a good starting point for the FDM analysis. However, we are now forced to choose an initial density field as the initial condition for the solver to start from. Choosing initial conditions is rather complicated as was discussed by [2, 13] and others. For educational purposes, we simply choose a density that follows a Gaussian-like distribution. By making sure that units work out, we arrive at

$$\psi(\vec{x}) = \sqrt{\frac{M}{V}} \cdot \exp\left\{-\frac{\pi \vec{x}^2}{2V^{2/3}}\right\}, \tag{4.9}$$

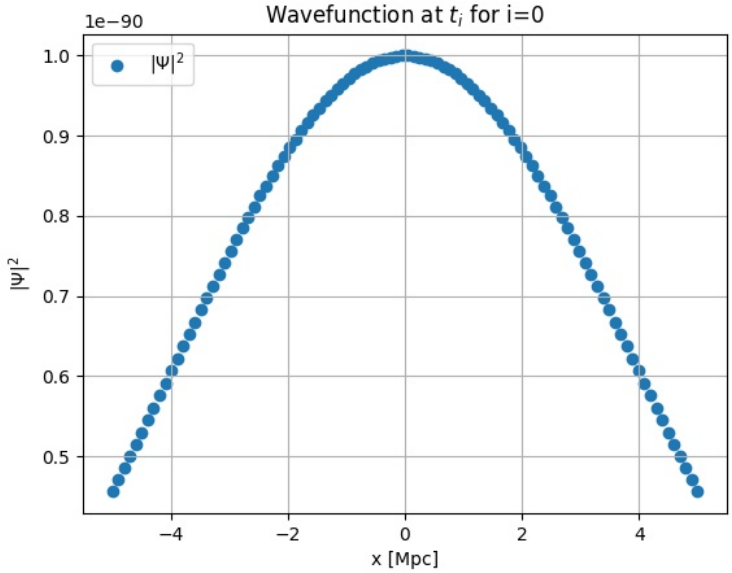
where M is the total mass in the simulated box with volume $V = L^3$. Note, that since the Gaussian integral in the normalization (2.37) is three-dimensional, one gets the Gaussian integral value three times, which then corrects the resulting exponent $2/6$, where the additional factor $1/2$ comes from the standard Gauss-integral solution, so that V cancels out in the normalization and M is achieved as we demand.

We desired to set $N_x = 10^3$ but the current solver simply requires too much computation time in order to compute the second derivatives, $\Delta\psi$, so that have let the solver run for one timestep with $N_x = 10^3$ to show that it is in fact capable of handling 10^9 grid points (see figure 4.11(a)), although it took roughly five hours for one timestep with an additional three hours per plot, so that we decided to let the simulation run for $N_x = 10^2$.

As you can see in figure 4.11(b), the chosen wavefunction (4.9) cannot be simulated by our solver correctly because the $\pi/V^{2/3}$-factor in the exponential widens the Gaussian pulse too much, so that the application of periodic boundary conditions results in wrong second derivatives and thus, in the breakdown of the evolution equation. In future works, one must choose a different wavefunction after a proper initial condition analysis, which is beyond the scope of this project, unfortunately.



(a) *Gaussian pulse after one timestep for $N_x = 10^3$ axis points per axis.*



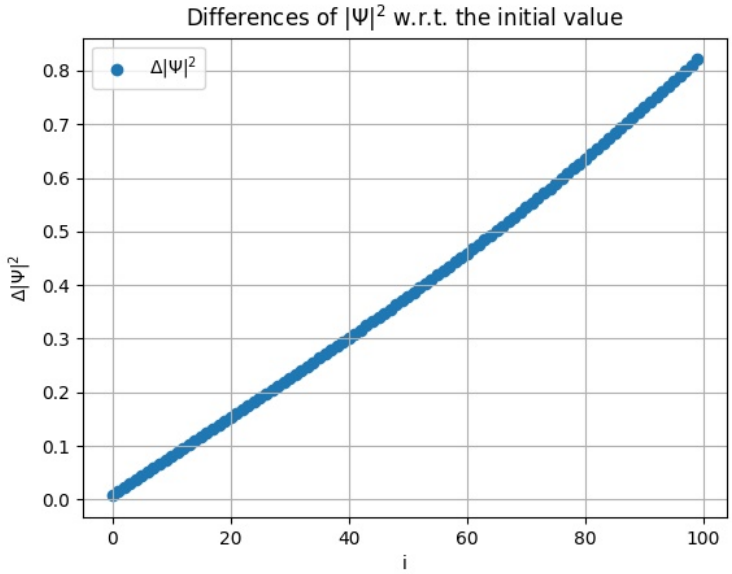
(b) *Axion wavefunction (4.9) for $N_x = 10^2, 10^4$ timesteps and $M = 10^{12} m_a$.*

Figure 4.11: *One radially symmetric Gaussian pulse and one example of a simple axion wavefunction. The Gaussian pulse shows the capability of simulating 10^9 grid points, whereas the axion wavefunction is a simple attempt of solving real-world applications.*

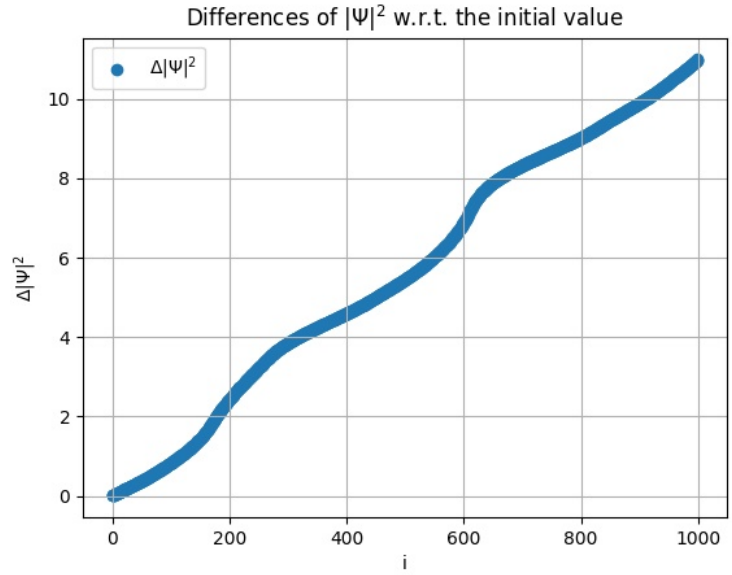
4.2.4 Open problems

The solver currently lacks two significant problems. The first links to the non-conserved integral over all $|\psi|^2$-values that we discussed in 4.1.4. Instead of integrating via SciPy's Simpson integrator along the axis as we did for the one-dimensional solver, we simply sum up all $|\psi|^2$-values of the `wavefunction` array, i.e. replacing the integral in (2.37) with a sum over all grid points. Again, we track the differences of the sums with respect to the initial value. At first, letting the simulation run for 10^2 timesteps, the results shown in figure 4.12(a) suggest that we only deal with numerical standard errors that are summed over more and more timesteps leading to large uncertainties in the final results. However, by letting the simulation run a bit longer, i.e. for 10^3 timesteps (figure 4.12(b)), one sees two jumps in the differences at close to the 200^{th} and around the 600^{th} timestep. By comparing this differences evolution with the corresponding evolutions of the maximal Schrödinger-velocity, $v_{\text{S,max}}$, (figure 4.13(a)) as well as the timestep sizes, Δt, (figure 4.13(b)), one can see that both jumps coincide with sharp peaks in both evolutions, indicating some hidden error in the solver. Like in the one-dimensional case, there must be something wrong with the computation of the real part of the wavefunction, what is still to be solved.

The second open problem we see is the tremendous runtime that comes with larger numbers of axis points. As we have said in subsection 4.2.1 already, letting the simulation run 10^4 timesteps for $N_x = 10^2$ axis points per axis takes roughly five (plus three per plot) hours for the simulation to finish. By using the same wavefunction we tried running the solver for just one timestep with $N_x = 10^3$ axis points per axis and in fact it finished on a computer that was capable of providing enough memory to store the necessary arrays. However, just one timestep took roughly five hours to finish, so that it is beyond the possibilities of this project to let the simulation run with $N_x = 10^3$ axis points per axis for long enough to produce relevant results. It is thus an open problem to improve the computation of the second derivatives of ψ, which is the simulation's biggest bottleneck right now, as we discussed in subsection 3.3.2 already.

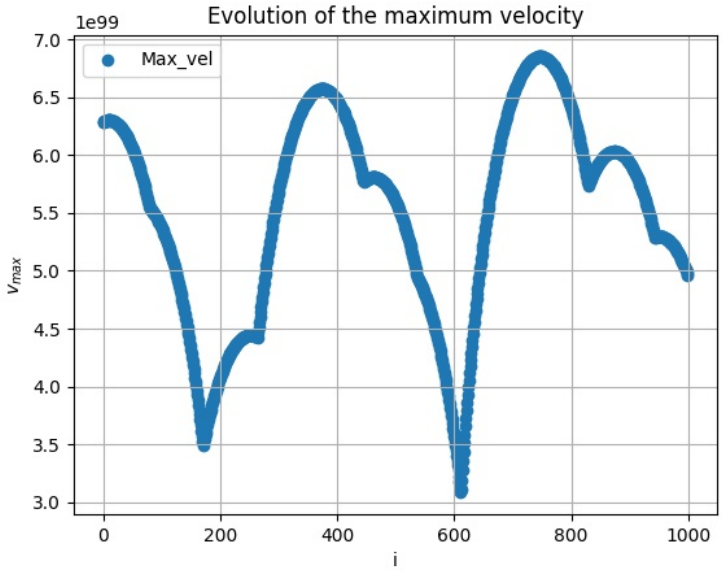


(a) *Differences between the sum of all $|\psi|^2$-values and the sum of the initial $|\psi_0|^2$-value against the timestep indices, i, for 10^2 timesteps.*

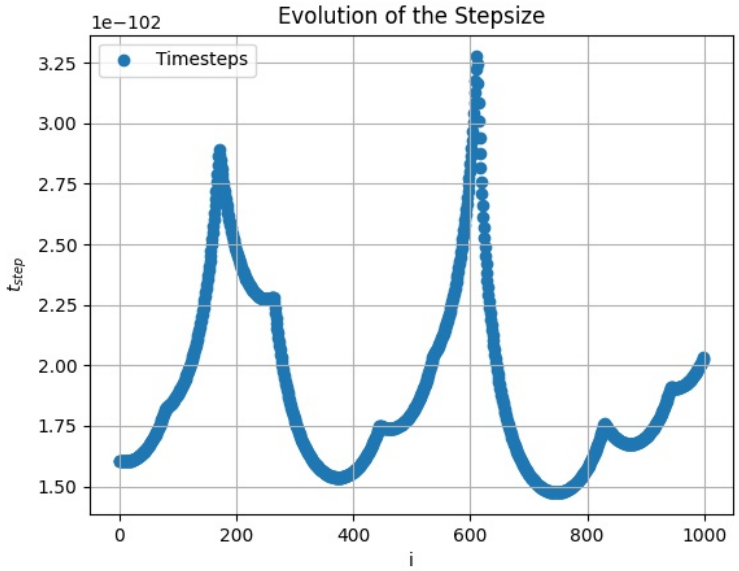


(b) *Differences between the sum of all $|\psi|^2$-values and the sum of the initial $|\psi_0|^2$-value against the timestep indices, i, for 10^3 timesteps.*

Figure 4.12: *Evolution of the total error of the simulation made in the sense of the conserved sum over all $|\psi(x)|^2$-values. Hence, the sum of the $|\psi|^2$-values at a timestep are subtracted by the initial value from which they should not differ significantly over time.*



(a) *Maximum Schrödinger-velocity, $v_{S,max}$, against timestep index, i.*



(b) *Timestep, $t_{Step} \equiv \Delta t$, against timestep index, i.*

Figure 4.13: *Time-evolution of the two important evolution equation (3.25) parameters, namely the timestep Δt and the maximum Schrödinger-velocity, $v_{S,max}$.*

Chapter 5

Conclusion and outlook

In this work, we have built on the theoretical insights we gained through our intensive research on the physics of ALPs and their impact on cosmology, cosmological perturbation theory and non-linear density perturbation evolution in the Universe. We have built an easy accessible Schrödinger-Poisson solver that fulfills not only educational but also scientific purposes in evolving arbitrary non-relativistic wavefunctions in Newtonian gravity in a flat, matter-dominated Universe that is well described by the RW-metric. We were able to check that gravity, at least qualitatively, behaves as we expect it and that it is possible to let the solver run, in a stable way, for very large numbers of timesteps in order to simulate as big time intervals as one desires to.

By changing the equation of the scale-factor (3.19), it is no problem to change the considered Universe's epoch to what one is interested. The same goes for the system under consideration, for one needs only to change the implemented wavefunction, ψ, accordingly. Be aware of following the chosen unit convention (1.3).

However, as we presented in the previous chapter as well, the question remains how we can ensure that the normalization of the wavefunction is ensured over all times. Although, numerical operations come with natural errors, we have clearly shown that in the evolution of the real part of the wavefunction remains a systematic uncertainty that induces a non-negligible error to our computations.

In the future it would be interesting to improve the solver in various manners. First of all, we would like to encourage rewriting the solver in another programming language that in general operates faster, like `C` or even `C++`, although the former should already be an enormous upgrade in terms of runtime. Additionally, it would introduce a pointer-arithmetic that could be used to outsource the creation of the wavefunction, the computation of the second derivatives of ψ and probably all plotting in seperate methods in order to upgrade the `main`-methods style, strongly increasing readability of the solver's basic structure. Second, the computation of the second derivatives of ψ should be rewritten, so that multiprocessing can be applied in an efficient way in order to significantly widen the solver's biggest bottleneck right now. The idea is that the computation takes place in a three-fold nested `for`-loop. Imagine a N_x^3-grid with $N_x = 3$, where we start at the lowest layer at the bottom row and then go through all three columns in order to check all grid points in this row. Then we go up one row in the same layer and repeat this process. After finishing the third row, we go one layer up and repeat the previous steps again to go through all layers. Note, that by doing so, we check each data point three times in total. In a first step, we would assign one processor to each layer, reducing the scaling of the runtime with $3N_x^3$ down to simply $3N_x^2$ as all layers can be checked simultaneously. Note, that for $N_x = 10^3$ what is the desired grid size of our simulation, this already requires 10^3 processors to perform the sped up simulation. However, depending on the resources, one can of course speed up the simulation even more. Third, we highly encourage using a supercomputer in order to have the resources for multiprocessing available more likely and to let the simulations run even quicker.

Furthermore, after improving the solver, we are looking forward tackling CDM wavefunctions as well as different FDM wavefunctions via a proper initial condition treatment as was suggested and presented in [2, 13], for example, for direct comparisons between FDM and CDM in order to draw strong conclusions on the differences that could likely be observable in the near future by precision cosmology. Should the results of upcoming works show that there is no observable differences, one is led to the conclusion that indeed ULAs could fill the knowledge gap in the ΛCDM model. Otherwise, if observable differences are not observed to a sufficiently large precision, one can safely rule out ULAs as proper DM candidates, what shrinks the axion mass-range substantially for other experiments that check the remaining various orders of magnitudes.

Chapter 6

Acknowledgments

I would like to thank my supervisor, Dr. Matthieu Schaller, very much for the guidance, your help and the many discussions throughout the project. Additionally, I would like to thank Dr. Subodh Patil.

Хочу поблагодарить свою подругу, которая вытаскивает меня из всех неудач и сопровождает в счастливые моменты. Я люблю тебя, мой пингвин.

Hiermit gilt mein Dank auch meiner Familie, die mich finanziell sehr im Studium unterstützt haben. Insbesondere möchte ich meinem Bruder danken, mit dem ich viele schöne Momente außerhalb von Physik während der Studienzeit erleben konnte.

Auch gilt mein Dank der Altherrenrunde, die mich auf ihre ganz eigenen Arten und Weisen stets in die Realität zurückholt, aber auch gerne ganz weit von ihr weg trägt. Meister Propper ist Erster! :)

Duizend dank wil ik ook graag aan Hugo zeggen. Ik ben heel blij dat ik met hem vaak een lekker kopje koffie mocht drinken en altijd iemand had om te praten.

I would like to thank the University Leiden for this incredible opportunity to pursue my studies here. It was an ongoing pleasure to gain knowledge and learn from the very best in their fields.

Zum Schluss möchte ich dem Ev. Studienwerk Villigst meinen tiefsten Dank aussprechen, ohne dessen Unterstützung dieses Vorhaben unter keinen erdenklichen Umständen angenehm umsetzbar gewesen wäre. Vielen lieben Dank für das, was ihr mir ermöglicht habt.

Appendix A

Code of the solver and availability

In this appendix, we would like to present the code of our Schrödinger-Poisson solver that is presented in the main text in chapter 3. Afterwards we present the other functions that were not presented in the main text due to their less relevance. Although, the code present is completely presented here, upon reasonable request, it is possible to get the python-files directly.

```
### Settings
output_directory = setup_output_directory()
boxsize = 10 #Mpc
number_axis_points = 10**3
number_time_steps = 1*10**6
mass_axion = 10**-99 # 10^10 M_solar
time_initial = 4.43*10**-7 # 0.978*10^12 yr


### Initial setup
x = np.linspace(-boxsize/2,boxsize/2,number_axis_points)
y = np.linspace(-boxsize/2,boxsize/2,number_axis_points)
z = np.linspace(-boxsize/2,boxsize/2,number_axis_points)

position_step = boxsize/(number_axis_points-1) #Mpc
k_vec = np.fft.fftfreq(number_axis_points, d=position_step)
k_vec_squared_inv = 1/np.dot(k_vec,k_vec)
```

Listing A.1: *The first part in* `main.py` *is to set the basic parameters of the simulation, i.e. the boxsize L, the number of axis points to be simulated per coordinate-axis, the number of timesteps the simulation should iterate through and the axion mass, m_a. Then, the coordinate-axes are defined, the position step, Δx, is computed and based the number of axis points and Δx, we already compute $\vec{k}$ and k^{-2}, where k is the absolute value of $\vec{k}$, for the Fourier transformation later.*

```
### Create wavefunction
sigma = 0.5
a = 0.5*sigma**2
c = 1/np.sqrt(2*np.sqrt(np.pi))
wavefunction = np.zeros((number_axis_points,
 number_axis_points,number_axis_points),dtype=complex)
k0 = 10**6*np.pi/boxsize
for xi in range(number_axis_points):
  for yi in range(number_axis_points):
    for zi in range(number_axis_points):
      pos = np.sqrt(x[xi]**2+y[yi]**2+z[zi]**2)
      wavefunction[xi,yi,zi] = c*np.exp(1j*k0*pos-pos
 **2/(4*a))/(np.sqrt(2*a))
```

Listing A.2: *The second step in* `main.py` *is to create the wavefunction, ψ, that we would like to evolve in time. First, we initialize an empty 3D array to store the value of the wavefunction at each grid point and then go through all axes points in order to compute the corresponding ψ-value and store it accordingly.*

```
    # Update density and potential
    density = np.real(np.multiply(np.conj(wavefunction),
   wavefunction))
    density_FT = np.fft.fftn(4*np.pi*G*density)
    potential = np.fft.ifftn(-1*k_vec_squared_inv*density_FT)
```

Listing A.3: *The first part of the third step, i.e. the main routine, in* `main.py` *is to compute the current density, ρ, and based on it the current potential, Φ. This is done by using NumPy's FFT implementation and its inverse pendant to quickly solve the Poisson equation for the potential, Φ.*

```
    # Update velocity
    for xi in range(number_axis_points):
      for yi in range(number_axis_points):
        for zi in range(number_axis_points):
          vel_schr[xi,yi,zi] = (-wavefunction[(xi+2)%
   number_axis_points,(yi+2)%number_axis_points,(zi+2)%
   number_axis_points]
                              +16*wavefunction[(xi+1)%
   number_axis_points,(yi+1)%number_axis_points,(zi+1)%
   number_axis_points]
                              -30*wavefunction[xi%
   number_axis_points,yi%number_axis_points,zi%
   number_axis_points]
                              +16*wavefunction[(xi-1)%
   number_axis_points,(yi-1)%number_axis_points,(zi-1)%
   number_axis_points]
                              -wavefunction[(xi-2)%
   number_axis_points,(yi-2)%number_axis_points,(zi-2)%
   number_axis_points])
    vel_schr *= 1/(12*position_step**2)
    time_current = sum(time_steps) # 0.978*10^12 yr
    scale_factor = ((time_initial+time_current)/age_universe)
   **(2/3)
    vel_schr = 1j*(0.5*hbar*vel_schr/(mass_axion*scale_factor
   )-potential*wavefunction/hbar)
```

Listing A.4: *The second part of the third step, i.e. the main routine, in* `main.py` *is to compute the Schrödinger-velocity, which in a first step requires $\Delta\psi$, which is computed via a fourth-order centered finite-differencing formula (3.18) by going through all grid points for each spatial dimension. Afterwards this is plugged in the equation of the Schrödinger-velocity (3.24).*

```
    #Update timestep
    vel_max = np.amax(abs(np.real(vel_schr)+np.imag(vel_schr)
    ))
    max_vels = np.append(max_vels,(vel_max))
    time_step = 0.1*position_step/vel_max
    time_steps = np.append(time_steps,(time_step))
```

Listing A.5: *The third part of the third step in the main routine in* `main.py` *is to compute the maxmimum Schrödinger-velocity, $v_{S,max}$, and based on it the timestep via the CFL-condition (3.43) and to append it to the* `timesteps`*-array in order to visualize it after the simulation.*

```
    # Subplots / Supprints
    if time_index % 100 == 0: print("i=",time_index,"at t=","
    {:.2f}".format(timeit.default_timer()-start),"s with a=","
    {:.4e}".format(scale_factor))
    if time_index % 100 == 0:
      plot_3D(output_directory,"squared3D_"+str(time_index).
    zfill(7),x,"x [Mpc]",y,"y [Mpc]",z,"z [Mpc]",'lin','lin','
    lin',"Wavefunction at $t_i$ for i="+str(time_index),'upper
     left',np.real(np.multiply(np.conj(wavefunction),
    wavefunction)),"$|Psi|^2$")
      plot_1D(output_directory,"squared1D_"+str(time_index).
    zfill(7),x,"x [Mpc]",r"$|\Psi|^2$",'lin','lin',"
    Wavefunction at $t_i$ for i="+str(time_index),'upper left'
    ,np.real(np.multiply(np.conj(wavefunction[:,int(
    number_axis_points/2),int(number_axis_points/2)]),
    wavefunction[:,int(number_axis_points/2),int(
    number_axis_points/2)])),r"$|\Psi|^2$")

    #Update wavefunction
    wavefunction = wavefunction + time_step*vel_schr
```

Listing A.6: *The fourth and final part of the third step in the main-routine in* `main.py` *is to update the wavefunction via (3.25) and eventually to print a status update to the user as well as to make plots in between for some sub-results. Note, that the frequence of these in-between steps should be adjusted to the total amount of timesteps that the simulation goes through in order to not slow the simulation down significantly.*

```
### Plot final densities
plot_3D(output_directory,"final_squared3D",x,"x [Mpc]",y,"y [Mpc]",z,"z [Mpc]",'lin','lin','lin',"Wavefunction at $t_i$ for i="+str(number_time_steps),'upper left',np.real(np.multiply(np.conj(wavefunction),wavefunction)),"$Psi^2$")
plot_1D(output_directory,"final_squared1D",x,"x [Mpc]",r"$|\Psi|^2$",'lin','lin',"Wavefunction at $t_i$ for i="+str(number_time_steps),'upper left',np.real(np.multiply(np.conj(wavefunction[:,int(number_axis_points/2),int(number_axis_points/2)]),wavefunction[:,int(number_axis_points/2),int(number_axis_points/2)])),r"$|\Psi|^2$")

###Analyze max velocities
plot_1D(output_directory,"max_velocities",time_indices,"i","$v_{max}$",'lin','lin',"Evolution of the maximum velocity",'upper left',max_vels,"Max_vel")

###Analyze time steps
plot_1D(output_directory,"time_steps",time_indices,"i","$t_{step}$",'lin','lin',"Evolution of the Stepsize",'upper left',time_steps,"Timesteps")
```

Listing A.7: *The fourth and final step in* `main.py` *is to make some plots and to store the results and simulation parameters. One 1D and 3D scatter plot of the final density is made, respectively, and two 1D plots of the evolution of the maximum velocities,* $v_{max,i}$*, and the timesteps,* $\Delta\tau_i$*, respectively.*

```
### Save important values
with open(output_directory+'settings_and_output.txt', 'w')
 as f:
  f.write("---Constants---\n")
  f.write("G="+"{:.4e}".format(G)+"Mpc*(km/s)^2*(10^10
 M_solar)^-1\n")
  f.write("hbar="+"{:.4e}".format(hbar)+"Mpc*(km/s)*10^10
 M_solar\n")
  f.write("\n")
  f.write("---Settings---\n")
  f.write("Boxsize="+"{:.2f}".format(boxsize)+"\n")
  f.write("Number of axis points="+"{:.4e}".format(
 number_axis_points)+"Mpc\n")
  f.write("Number of time steps="+"{:.4e}".format(
 number_time_steps)+"\n")
  f.write("Position step="+"{:.4e}".format(position_step)+"
 Mpc\n")
  f.write("k^-2="+"{:.4e}".format(k_vec_squared_inv)+"\n")
  f.write("\n")
  f.write("---Output parameters---\n")
  f.write("Simulation time="+"{:.4e}".format(
 time_simulation)+"*3.086e+19s\n")
  f.write("Runtime="+"{:.4e}".format(runtime)+"s\n")
print("Important values are saved in text file
 settings_and_output.txt")
```

Listing A.8: `main.py`. *The 3D wavefunction array* `wavefunction` *as well as the two 1D arrays* `max_vels` *and* `timesteps` *are stored as .npz-files. Finally, the constants and the simulation parameters as well as some variables are stored in a .txt-file for the convenience of the user.*

Methods of less relevance

The method `setup_output_directory` is a very simple, but handy, method (see Listing A.9) in order to create a unique output directory. It simply starts to pretend the current run has the number one and checks if a corresponding directory already exists. If this is not the case, it increments this number by one and repeats to check for its existence. Eventually, it will find the lowest number that does not already exist, makes the corresponding output directory and returns this string for later usage of storing plots and other output files.

```
import os

def setup_output_directory():
  run_number = 1
  path = "run"+str(run_number)+"/"
  while os.path.isdir(path) == True:
      run_number += 1
      path = "run"+str(run_number)+"/"
```

Listing A.9: *In `storage.py` is the method `setup_output_directory` stored, which checks for already existing output directories. Eventually, it finds the newest, non-existing, output directory and simply appends it, so that it can be used in the current run.*

We implemented two methods that are designed to simplify the plotting of various data sets (see Listing A.10 and A.11) in one- and three-dimensions, respectively.

The one-dimensional `plot_1D` requires several input parameters. First, one must pass the output directory, so that the plots are stored correctly, a desired filename, the horizontal axis, x, and its label as well as the label for the vertical axis, y. Then, one must specify if one wants to have one or both axes logarithmic by passing `'log'`[1]. After that, one specifies the plot title and and gives the position of the legend to `legloc`. The most convenient part of the method is that one then can pass an arbitrary amount of data-label-pairs to `*yAndLabel`, where one must be cautious about the ordering "data set" first, "corresponding label" second and that one always passes a label corresponding to a data set, so that it appears in the legend in the plot. From all passed data sets, a scatter plot is made and stored in the given output directory. In fact, the method itself is nothing spectacular, its only purpose is to spare many codelines in `main.py`.

[1] Any other input here gives a linear scale.

```
def plot_1D(output_path,filename,x,xlabel,ylabel,xScale,
    yScale,title,legloc,*yAndLabel):
  plt.xlabel(xlabel)
  plt.ylabel(ylabel)
  plt.title(title)
  plt.grid()

  if xScale == 'log': plt.xscale('log',base=10)
  if yScale == 'log': plt.yscale('log',base=10)

  for i in range(0,len(yAndLabel),2): plt.scatter(x,yAndLabel
    [i],label=yAndLabel[i+1])

  plt.legend(loc=legloc)
  plt.savefig(output_path+filename+".jpg")
  #plt.show()
  plt.clf()
```

Listing A.10: *`plotting.py` contains two different ways of plotting data, i.e a one-dimensional and a three-dimensional case. The former one, `plot_1D` presented here, was implemented in a way that you can add an arbitrary number of data on the vertical axis for the same horizontal axis.*

The three-dimensional `plot_3D` is not as flexible as its one-dimensional pendant, which is also not necessary since we always desire to plot precisely one data set on the whole 3D-grid, i.e. the wavefunction- or the density-values, respectively. However, basically the same input parameters as in the one-dimensional case must be passed with the slight difference that a third axis must be passed along with a third scaling option in order to set one, two or three axes to a logarithmic scaling. In the end, via `vals` and `vals_label` one passes the values on the grid points and the corresponding label, respectively. Then, the plotting is again, nothing spectacular. The only important line to mention is the creation of the grid via NumPy's `meshgrid`-method that takes the three coordinate-axes we defined and passed to the plotting-method and gives the coordinate-values of the grid points back that correspond to the axes-points. Thus, from three axis-arrays of length L, `meshgrid` makes L^3 grid points, so that we are able to make ψ, $|\psi|^2$, ρ or other data, that is defined at each grid point, visible.

```
def plot_3D(output_path,filename,x,xlabel,y,ylabel,z,zlabel,
    xScale,yScale,zScale,title,legloc,vals,vals_label):
  fig = plt.figure()
  ax = fig.add_subplot(projection='3d')

  ax.set_xticks([-5,-2.5,0,2.5,5])
  ax.set_yticks([-5,-2.5,0,2.5,5])
  ax.set_zticks([-5,-2.5,0,2.5,5])

  ax.axes.set_xlim3d(left=-5, right=5)
  ax.axes.set_ylim3d(bottom=-5, top=5)
  ax.axes.set_zlim3d(bottom=-5, top=5)

  ax.set_xlabel(xlabel)
  ax.set_ylabel(ylabel)
  ax.set_zlabel(zlabel)
  ax.set_title(title)
  ax.grid()

  if xScale == 'log': ax.set_xscale('log')
  if yScale == 'log': ax.set_yscale('log')
  if zScale == 'log': ax.set_zscale('log')

  xp, yp, zp = np.meshgrid(x,y,z)

  color_map = plt.get_cmap('Reds')
  scatter_plot = ax.scatter3D(xp, yp, zp, c=vals, cmap=
    color_map, label=vals_label, alpha=0.3)
  plt.colorbar(scatter_plot)

  ax.legend(loc=legloc)

  plt.savefig(output_path+filename+".jpg")
  #plt.show()
  plt.clf()
```

Listing A.11: *`plotting.py` contains two different ways of plotting data, i.e a one-dimensional and a three-dimensional case. The latter one, `plot_3D` presented here, was implemented specifically for showing precisely one data set on each grid point.*

References

[1] M. Kavermann, *Axions as the natural answer to the dark matter question in the ΛCDM cosmological standard model*, Master's thesis, Leiden University, 2023.

[2] M. Schaller, C. Becker, O. Ruchayskiy, A. Boyarsky, and M. Shaposhnikov, *A new framework for numerical simulations of structure formation*, Monthly Notices of the Royal Astronomical Society **442**, 3073 (2014).

[3] R. D. Peccei, *The Strong CP Problem and Axions*, in *Lecture Notes in Physics*, pages 3–17, Springer Berlin Heidelberg, 2008.

[4] G. 't Hooft, *Symmetry Breaking through Bell-Jackiw Anomalies*, Phys. Rev. Lett. **37**, 8 (1976).

[5] V. Rubakov and D. Gorbunov, *Introduction to the Theory of the early Universe - Hot Big Bang Theory*, World Scientific Publishing Co. Pte. Ltd., 2nd edition, 2018.

[6] R. D. Peccei and H. R. Quinn, *CP Conservation in the Presence of Pseudoparticles*, Phys. Rev. Lett. **38**, 1440 (1977).

[7] D. J. Marsh, *Axion cosmology*, Physics Reports **643**, 1 (2016).

[8] S. M. Carroll, *An Introduction to General Relativity - Spacetime and Geometry*, Cambridge University Press, 2020.

[9] R. Hlozek, D. Grin, D. J. Marsh, and P. G. Ferreira, *A search for ultralight axions using precision cosmological data*, Physical Review D **91**, 103512 (2015).

[10] D. J. Marsh and P. G. Ferreira, *Ultralight scalar fields and the growth of structure in the Universe*, Physical Review D **82**, 103528 (2010).

[11] V. F. Mukhanov, H. A. Feldman, and R. H. Brandenberger, *Theory of cosmological perturbations*, Physics reports **215**, 203 (1992).

[12] R. Ruffini and S. Bonazzola, *Systems of Self-Gravitating Particles in General Relativity and the Concept of an Equation of State*, Phys. Rev. **187**, 1767 (1969).

[13] S. May and V. Springel, *Structure formation in large-volume cosmological simulations of fuzzy dark matter: impact of the non-linear dynamics*, **506**, 2603 (2021).

[14] C. R. Harris et al., *Array programming with NumPy*, Nature **585**, 357 (2020).

[15] P. Virtanen et al., *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods **17**, 261 (2020).

[16] J. D. Hunter, *Matplotlib: A 2D graphics environment*, Computing in Science & Engineering **9**, 90 (2007).

[17] G. Arfken and H. Weber, *Mathematical Methods for Physicists*, Elsevier Academic Press, 6th edition, 2005.

[18] E. O. Brigham and R. Morrow, *The fast Fourier transform*, IEEE spectrum **4**, 63 (1967).

[19] L. M. Milne-Thomson, *The calculus of finite differences*, American Mathematical Soc., 2000.

[20] R. Courant, K. Friedrichs, and H. Lewy, *Über die partiellen Differenzengleichungen der mathematischen Physik*, Mathematische Annalen **100**, 32 (1928).