Classical motion in synthetic monopole fields

UPPSALA
UNIVERSITET

Bachelor's thesis 15c
Department of Physics and Astronomy
June 5, 2022

Author: Ola Carlsson
Supervisor: Erik Sjöqvist
Subject reviewer: Patrik Thunström

Abstract

In the Born-Oppenheimer approximation for a quantum system the emergent synthetic magnetic field can be seen as generated by monopoles at points of degeneracy, in full analogue to the synthetic fields generating the geometric phase of adiabatically evolving quantum systems. The plausibility of using these synthetic magnetic monopoles as a means to study magnetic monopole dynamics in the absence of fundamental magnetic monopoles has been explored. A bipartite spin system consisting of a dumbbell translating and rotating through space has been modelled, and full equations of motion in the presence of an external magnetic field have been derived. A collection of scripts for numerical evaluation of these equations of motion were subsequently developed, and further put to use in sample simulations for a small range of parameters. The results demonstrate non-negligible perturbations to the centre of mass motion when compared to motion not considering the Born-Oppenheimer synthetic fields, for dumbbell masses of small but not unrealistic proportions. The problems inherent in this approach to elucidating motion in magnetic monopole fields are discussed, but the method should not yet be dismissed until further investigations have been made.

Sammanfattning

Under Born-Oppenheimer-approximationen för ett kvantsystem kan det emergenta syntetiska magnetfältet ses som alstrat av monopoler vid degenerationspunkter, helt analogt med de syntetiska fält som genererar den geometriska fasen vid adiabatisk utveckling av kvantsystem. Möjligheten att använda dessa syntetiska magnetiska monopoler för att studera dynamiken från verkan av en magnetisk monopol, trots att fundamentala magnetiska monopoler ej observerats, har utforskats. Ett tvådelat spinsystem bestående av en hantel som translaterar och roterar genom rummet har modellerats, och fullständiga rörelseekvationer i närvaron av ett yttre magnetfält har härletts. Kod till ändamålet att utvärdera dessa rörelseekvationer har därpå utvecklats, och vidare nyttjats för att simulera rörelsen för ett stickprov av parametrar. Resultaten visar på ej försumbara perturbationer av masscentrums rörelse vid jämförelse med rörelse utan hänsyn till de syntetiska Born-Oppenheimer-fälten, för hantlar av liten men inte orealistiskt liten massa. Problemen och komplikationerna för det här angreppssättet till att utforska rörelse genom magnetiska monopolers fält diskuteras, men metoden bör ej ännu avvisas innan vidare undersökning har genomförts.

Contents

1 Introduction

Our theory of electromagnetism carries certain asymmetries between the magnetic and electric fields. A well known and often discussed such distinction is the absence of magnetic monopoles, i.e., that any analogue of electric charge is absent for the magnetic field. It is an extension often suggested by theorists, initially by Dirac in 1931 [1], to include magnetic charges and also potentially magnetic currents in Maxwell's equations, but there is as of yet no accepted empirical data to support this cause. Magnetic monopoles do however appear occasionally in a less fundamental sense, as emergent phenomena in many-body systems for example in spin ice [2] or Bose-Einstein condensates [3].

Another area in which magnetic monopoles has appeared is the study of *geometric* phases of quantum systems, first described by Berry in 1984 [4]. Roughly speaking this is the phase acquired by the wave function of a system which is not dynamic in origin. The dynamical phase of a system state is induced by the energy of that state, while the geometric phase is surprisingly enough *in*dependent of the energy values, and rather arises as a function of the path taken by the system through the relevant "parameter-space" parametrising the Hamiltonian. For certain systems this geometric phase evolution resembles the evolution of a charged system in a magnetic vector potential field that lives in parameter space. This field, here named the *synthetic* magnetic field, happens to exhibit non-zero divergence at certain points, the degeneracy points of the state energies, which thus correspond to monopoles of the synthetic field.

Geometrical phase is an interesting field of study in itself, but it is with the appearance of magnetic-type monopoles that the present body of work finds its premise. While the study of fundamental magnetic monopoles remains impossible, we can through construction of a suitable system study the effects of magnetic monopoles through their action on the system state in parameter space. Parameter space can be put into correspondence with real space, and the movement of charged matter through monopole fields becomes measurable, even though fundamental monopoles remain fictitious.

Berry's original 1984 article considers a simple spin-system with a single magnetic dipole moment in an external magnetic field $\vec{B}$. The relevant parameter space is the space of all possible external fields, and the geometric phase contribution takes the form of a synthetic magnetic field purely generated by a monopole sitting at the origin, i.e., at $\vec{B} = \vec{0}$. This field is the simplest example of a monopolar field, so to find more complex behaviour this starting point of a system can be extended to include multiple spin components, i.e., multiple magnetic dipoles, and interactions between those dipoles. The effects of introducing such interactions is roughly that of splitting the origin-centred monopole into smaller constituent parts whose positions in parameter space depend on the exact nature of the spin-spin interactions [5].

This splitting is desired, and so the system studied will be composed of two massive spin-$\frac{1}{2}$ components that interact through the so-called Ising interaction described in section 3.2. This is to some extent the simplest spin-spin interaction and is dependent only on spin along a chosen axis, here taken to be the axis connecting the two masses. Movement through parameter space can be mapped to movement of the center of mass through real space given an external inhomogeneous magnetic field, and the movement of the spin components relative to one another is as a simplest case the rotation through polar and azimuthal angles with fixed inter-component distance. Such a setup is reminiscent of a dumbbell translating and rotating through space, with the added complication that each "weight" of the dumbbell acts as a dipole (has spin) interacting with both the dipole at the other weight and an external magnetic field, see figure 1.

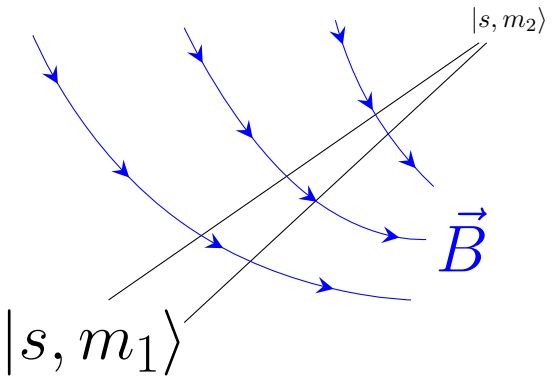


Figure 1: A dumbbell with two spin components moving through a magnetic field.

It is the time evolution of this system, henceforth referred to as the dumbbell, with which this project

is concerned. Approximate equations of motion will be derived and then be put to test in a numerical simulation. Underlying the process is the hope of discerning effects of the synthetic magnetic monopoles on the movement of the dumbbell. In addition to carrying dynamics of interest, the described dumbbell is also appropriate for its potential to model realisable physical systems. Such a realisation of the model herein described opens up the possibility of experimentally measuring the action of the synthetic fields, and by extension the action of synthetic monopoles.

Two paths of realisation spring to mind: Firstly a diatomic molecule with appropriate effective spin of the two constituent atoms could be tested. It would be of importance that the magnetic moment of each atom be large, so that the molecule couples to the external field strongly enough, and it would further be desired that spin-spin interactions between the molecules be strong so as to achieve the more exotic synthetic field texture mentioned in section 1. Preferably the gas phase of such a molecule should be obtainable, for it would be desired to measure the dynamics of single such molecules without inter-molecular interaction. For the dumbbell model to apply reasonably well the interatomic binding would also have to be of such a nature that the interatomic distance would not vary greatly. The plausibility of this approach, and the selection of suitable candidate molecules, is an interesting question in its own right and warrants further research. Something as simple as hydrogen gas is not necessarily unsuited.

Secondly, one might imagine a substantially smaller system consisting of a single atom with non-negligible nuclear and electronic spin. The same considerations concerning the strength of interactions apply as above, but this method appears to carry larger obstructions. Nuclear magnetic moments, while measurable due to resonance effects such as in nuclear magnetic resonance (NMR) measurements, are orders of magnitude smaller than their electronic counterparts [6]. This, together with the disparate masses of electrons and nucleus would necessitate a heavily asymmetric dumbbell model. It is also a well known fact that such classical approximations as the definite position of the electronic part of the system implied by a dumbbell model break down at these length scales. We must consider an atom a quantum thing, for if we do not we will find incorrect results.

2 Background

2.1 The adiabatic approximation

Imagine sitting in a train holding a plate, in which a marble sits at rest much like in picture 2. Consider then what would happen were the train to (de)accelerate. If the conductor brakes forcefully everything in the cabin experiences a fictitious force in the opposite direction of the train acceleration. If this force is large enough, the marble would roll up the side of the rounded plate and fall down to the floor below. If instead the train slows down gradually the fictitious force will be much smaller, it can even be small enough for the marble to never roll up the entire height of the plate. Then the marble would stay confined at the bottom of the porcelain, no matter for how long the train brakes. The marble keeps its state, the position in the plate, even though its environment, the train, undergoes change.

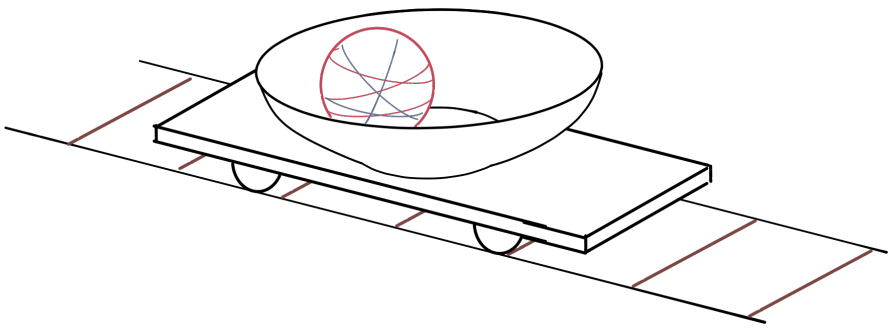


Figure 2: A marble on a plate on top of a ”train”.

This thought experiment if translated to the quantum realm, where states are kets or wavefunctions and the environment is encapsulated in the system Hamiltonian, captures the essence of the *adiabatic theorem* finalized by Born and Fock [7]. Concisely stated, the original version reads in translation to English:

> A physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian's spectrum.

In addition to the classical analogue given above, where the speed of the change of environment was required to be "slow", we note an additional requirement on the state: the energy of the state must not lie close to the energy of any other state available. This facet will be of great importance in the discussion of geometric phase, and is in a sense the origin of the synthetic magnetic monopoles. The assumption that the conditions for this theorem are all satisfied is often referred to as the *adiabatic approximation*, which will be done also in this text.

As an illustrative example consider a particle of spin s in some external magnetic field $\vec{B}$. The total energy, and thus Hamiltonian, for such a system is

$$\mathcal{H} = -\gamma \vec{B} \cdot \vec{S}.$$

Here, $\vec{S}$ is the spin operator and γ is a constant determining the magnetic dipole moment per unit of angular momenta, i.e., it looks typically as $\gamma = \frac{g\mu}{\hbar}$. Here, g is a dimensionless *g-factor* and μ is an appropriate magneton. The eigenstates of this Hamiltonian are the same as the spin eigenstates $|\vec{n}, m\rangle$ in the direction $\vec{n}$ of $\vec{B}$ with the quantum number m describing their directional spin eigenvalues. The eigenvalues E_m of the Hamiltonian, the energies of the states, become

$$E_m = -\gamma B \hbar m.$$

Here B is the magnetic field strength. The adiabatic theorem states for this example that any "slow" change of the Hamiltonian, that is any slow change of the magnetic field $\vec{B}$, will not change the quantum number m of the state occupied if the system was originally put into an eigenstate $|\vec{n}, m\rangle$. The only things that change in the state occupied are possibly the direction $\vec{n}$ and the energy level E_m. The criterion for the theorem to hold will here translate into, apart from the speed of change, that B be nonzero. If B were zero the energy levels would be degenerate and the theorem would not be applicable.

2.2 The geometric phase

It was the evolution of systems satisfying the adiabatic theorem that concerned Berry as he demonstrated the nature of the geometrical phase in the eighties [4], although non-adiabatic extensions have been made since [8]. As mentioned in section 1 the adiabatic geometric phase is a contribution to the phase of a system undergoing adiabatic changes which is *independent* of the energies during the adiabatic change [9]. Instead it is a contribution dependent only on the path traversed through parameter space, which in the above example is the three-dimensional space of possible external magnetic fields. The speed by which the path, often considered to be a loop $\mathcal{C}$, is traversed matters not for the geometric phase. Note however that this speed cannot become arbitrarily large, as it would then eventually break the adiabatic approximation. It is also relevant to the discussion to remind oneself that the accumulation of phase in quantum mechanics compromises all of dynamics, i.e., all forms of time evolution can be phrased as changes in the phase of states.

In less abstract terms, it can be shown that an arbitrary quantum energy eigenstate $|n\rangle$ (labelled after its energy E_n) affected by the Hamiltonian $\mathcal{H}$ will under adiabatic evolution along some loop $\mathcal{C}$ accumulate a geometric phase η_n. This means that if the loop is traversed in time T the state will after one revolution end up as $e^{-\frac{i}{\hbar}\int_0^T \langle n|\mathcal{H}|n\rangle dt} e^{i\eta_n} |n\rangle$. Note the inclusion of the dynamical phase with the first factor, owing to the "standard" time evolution due to energy. The value of η_n can be calculated from the following integral [4]:

$$\eta_n(\mathcal{C}) = i \oint_{\mathcal{C}} \left\langle n \middle| \vec{\boldsymbol{\nabla}}_{\vec{R}} n \right\rangle \cdot d\vec{R}. \tag{1}$$

Here, $\vec{R}$ denotes the parameters at some point in parameter space, so the line integral is appropriately carried out over that space. Note also that the Hamiltonian and therefore also all states depend on $\vec{R}$.

By Stokes' theorem this integral can be transformed into a surface integral over any enclosed area in parameter space [4]. If parameter space is three dimensional as in the example in section 2.1 above the cross product can additionally be utilized, and the geometric phase can be recast as:

$$\eta_n(\mathcal{C}) = -\iint_{\mathcal{C}} d\vec{S} \cdot \vec{G}_n(\vec{R}), \tag{2}$$

where

$$\vec{G}_n(\vec{R}) = \mathrm{Im} \sum_{l \neq n} \frac{\langle n| \vec{\boldsymbol{\nabla}}_{\vec{R}} \mathcal{H} |l\rangle \times \langle l| \vec{\boldsymbol{\nabla}}_{\vec{R}} \mathcal{H} |n\rangle}{(E_l - E_n)^2}. \tag{3}$$

It is here that the monopole field makes its entrance. While the integrand of equation 1 is only defined up to the gradient of an arbitrary differentiable function, owing to an arbitrary phase factor of the eigenstate basis, the field $\vec{G}_n$ is but the curl of this integrand and thus independent of this arbitrary term. The phase evolution due to the geometric phase has then taken the form of the time evolution through a magnetic field $\vec{G}_n$ derived from the vector potential

$$\vec{D}_n = i \langle n| \vec{\boldsymbol{\nabla}}_{\vec{R}} |n\rangle . \tag{4}$$

We call $\vec{G}_n$ the synthetic magnetic field, or the synthetic gauge field, and consider it the field the action of which results in the geometric phase. The monopole structure so often mentioned previously is present in precisely this field, by which it is meant that the singularities of this field which occur at points of energy degeneration may infer a non-zero divergence $\vec{\boldsymbol{\nabla}} \cdot \vec{G}_n$. Since the field is not defined at these points, divergence should here be interpreted in the rather loose sense of a closed flux integral about some point divided by the enclosed volume. A nonzero divergence of a magnetic field is not allowed for standard Maxwellian magnetic fields and would imply some form of magnetic charge or monopole. Even though our synthetic field does not follow Maxwell's equations it shares the magnetic field property of being the curl of a vector potential, and as will be clear in section 4.2 also magnetic properties in how it establishes dynamics. For these reasons the monopoles present at the degeneracies allow us to study, in a sense, the action of Maxwellian magnetic monopoles. It may be worthwhile to emphasize that the singular nature of the field at the energy degeneracies, that the field is there undefined, is crucial to the nonzero divergence. This mimics the model of electrical point charges, but also makes the dependence on the adiabatic approximation abundantly clear.

2.3 Regarding the monopoles

The system outlined in the final paragraph of section 2.1 forms a sort of minimal working example of a synthetic field. The energies of different states are there degenerate only for an external magnetic field of zero magnitude, that is at the origin of parameter space, so the synthetic field has a monopole precisely at the origin. The action from the synthetic field, that is the accumulation of geometrical phase, could be achieved either from slowly varying the external field magnitude and direction or likewise from the slow movement of the particle through an inhomogeneous external field yielding the same effect. It is further the case for this simple example that the synthetic field not only contains monopoles, but that it is *purely* monopolar in origin. This is to say that a "charge" placed at the origin emanating a spherically symmetric field that falls off as the inverse square of the distance yields precisely the synthetic field in question, the field is only due to "charges" [4]. It is important to note that this must not always be the case, the synthetic magnetic field may, depending on the Hamiltonian, take a form which is not possible to describe only through inverse-square falloff from charges. It could even be the case that no such charges are present at all, what the synthetic field simply does is to allow for them.

One case for which the synthetic field is *not* purely monopolar is two spin constituents interacting with both an external magnetic field and each other as shown by Eriksson and Sjöqvist [5], much like the system outlined in section 1. These authors further note that the nonzero curl of a synthetic field that is not purely monopolar can be used to extend the allegory between synthetic and Maxwellian fields through a synthetic electrical "current" defined through the curl of the synthetic magnetic field. The main effect of spin-spin interaction of composite spin systems is however the "splitting" or movement of magnetic charge away from the origin of parameter space yielding more exotic fields. This movement of magnetic monopoles is continuous with respect to the spin-spin interaction parameters, and will be fully determined by these parameters. It is further important to the splitting of monopoles that the

spin-spin interaction is of such nature that the total system is not spherically symmetric. Were the system spherically symmetric, the synthetic field would have to be likewise in parameter space, and any monopoles would be restricted to sit at the origin. Since the system outlined in section 1 contains precisely an Ising interaction along some selected axis the synthetic field is in this case not fully spherically symmetric, but must have a rotational symmetry determined by said axis.

For the sake of this body of work we note that the synthetic field of the dumbbell model will be nontrivial in nature, carries distinct synthetic monopoles and further may exhibit nonzero curl, so that it is not purely monopolar.

3 System description

3.1 Coordinates and quantities

Here follows a complete model of the scenario outlined in section 1, with the purpose of simulating the system numerically to gain insights into the dynamics of synthetic magnetic fields. Consider a dumbbell-like system consisting of two equal masses at a distance l from one another, and let m be the total mass. The system can be freely translated and rotated throughout space, so let x, y, z be the position of the centre off mass, and θ_r, φ_r be the polar and azimuthal angle respectively of the axis connecting the two masses. Notate these coordinates compactly as the vector

$$\vec{r} = \begin{pmatrix} x \\ y \\ z \\ \vartheta_r \\ \varphi_r \end{pmatrix}.$$

Fix the angles such that a polar angle of $\vartheta_r = 0$ implies a dumbbell parallel to the z-axis and so that an azimuthal angle of $\varphi_r = 0$ implies that the dumbbell axis lies in the xz-plane.

Consider also each of the masses of the dumbbell to carry spin, intrinsic angular momentum, of size $\frac{1}{2}$ each. The state of the spin components must be described quantum mechanically, so let $|s, m'\rangle$ denote the state of the system with *total* spin magnitude squared $s(s+1)\hbar^2$ and *total* spin measured along the z-axis $\hbar m'$. Note that the spin quantum number s will be 1 for the composite system, and so values of m' will range from -1 to 1. An external field $\vec{B}$ is present, which we can describe by its magnitude B and its angular direction ϑ_B, φ_B in analogue with the angles defined above.

3.2 The Hamiltonian

The time evolution of such a system is governed in both classical and quantum mechanics by its Hamiltonian. Since spin is the epitome of a phenomena demanding a quantum mechanical interpretation we have no choice but to model the whole system quantum mechanically. The Hamiltonian which will be assumed for the system is:

$$\mathcal{H} = \sum_{i=1}^{5} \frac{\vec{p_i}^2}{2m_i} + \frac{4J}{\hbar} S_\mu^{(1)} S_\mu^{(2)} - \gamma \vec{B}(\vec{r}) \cdot \vec{S}. \tag{5}$$

The first sum is over the five degrees of translational and rotational coordinates in $\vec{r}$. Their conjugate momentum operators are taken to be $p_i = i\hbar\partial_i$ with ∂_i as the derivative with respect to the corresponding coordinate. Note that it is not a priori clear that the effective masses m_i for all degrees of freedom are the same, but we can until later note that at least the first three are equal to m.

For the potential energy the spin-spin interaction is taken to be of Ising form, which is the first term after the sum, while the Zeeman interaction between spin and magnetic field is considered in the final term. An Ising interaction requires a preferred axis, which for symmetry reasons of the system has been chosen to be the direction μ of the dumbbell axis, the axis connecting the two masses. The necessity for selecting an axis is the reason for choosing precisely an Ising interaction, as it breaks the spherical symmetry of the system and allows for more exotic synthetic field textures as described in section 2.3. The parameters J and γ are the strengths of both of these interactions, while the operators $\vec{S}$ and $S_\mu^{(n)}$ are respectively the one related to the total spin of the system and the spin in the μ-direction of the nth system component. Note that the parameter γ much like the example of section 2.1 typically looks like $\gamma = \frac{g\mu_f}{\hbar}$, where g is a g-factor and μ_f is some appropriate magneton.

Similar systems as the one considered here has been studied before in the context of geometric phase, in particular a bipartite spin-$\frac{1}{2}$ system with coordinate-fixed Ising axis by Yi and Sjöqvist [10]. If the rotational degrees of the present system are ignored and the coordinate ϑ_r is set to 0 whenever present the system of [10] will be matched in full, save for that the external field is there varied directly instead of through centre of mass motion.

3.3 Effective mass

To clearly see the values of the effective masses paired with the rotational momenta a quick derivation of the kinetic part of the Hamiltonian is in order. The kinetic energy related to rotation is of the form

$$K_{rot} = \frac{m}{2}\left(\frac{l}{2}\right)^2\left(\dot{\vartheta}_r^2 + \dot{\varphi}_r^2\right),$$

which is the same as the relevant terms of the Lagrangian. The quantum mechanical momenta correspond to the momenta received from differentiating the classical Lagrangian, and as of such we have in the classical picture that

$$p_4 = \frac{\partial K}{\partial \dot{\vartheta}_r} = \frac{ml^2}{4}\dot{\vartheta}_r,$$
$$p_5 = \frac{\partial K}{\partial \dot{\varphi}_r} = \frac{ml^2}{4}\dot{\varphi}_r.$$

Performing the Legendre transform from the Lagrangian to the Hamiltonian yields:

$$\mathcal{H}_{rot} = p_4\dot{\vartheta}_r + p_5\dot{\varphi}_r - K_{rot} = \frac{p_4^2 + p_5^2}{2}\frac{4}{ml^2}.$$

It is then clear that the effective masses to be used in equation 5 are:

$$m_i = \begin{cases} m, & i = 1,2,3, \\ \frac{ml^2}{4}, & i = 4,5. \end{cases}$$

The rotational effective "masses" are of course moments of inertia, but will be referred to as masses such that all five degrees of freedom are treated equally.

3.4 Rotation matrices

The potential energy operators will be of great use in some matrix form, so let the spin state of the entire system be described in the total spin basis $(|0,0\rangle, |1,-1\rangle, |1,0\rangle, |1,1\rangle)$ with the coordinate z-axis as the spin measurement direction. In the special case where the axis of the dumbbell (henceforth "Ising axis") and the magnetic field $\vec{B}$ are parallel to the z-axis, it is clear that the operators take the form:

$$\vec{B}\cdot\vec{S} = B\hbar\begin{pmatrix} 0 & & & \\ & -1 & & \\ & & 0 & \\ & & & 1 \end{pmatrix} \tag{6}$$

and

$$S_\mu^{(1)}S_\mu^{(2)} = \frac{\hbar^2}{4}\begin{pmatrix} -1 & & & \\ & 1 & & \\ & & -1 & \\ & & & 1 \end{pmatrix}. \tag{7}$$

The second matrix follows from the well known representation of a two-component spin-$\frac{1}{2}$ system as singlet and triplet states: Let $|m_1\rangle \otimes |m_2\rangle$ be the state with spin-z number m_1 for the first spin and m_2

for the second spin. Then

$$
\begin{aligned}
|0,0\rangle &= \frac{1}{\sqrt{2}}\left[\left|\frac{1}{2}\right\rangle \otimes \left|-\frac{1}{2}\right\rangle - \left|-\frac{1}{2}\right\rangle \otimes \left|\frac{1}{2}\right\rangle\right] \\
|1,-1\rangle &= \left|-\frac{1}{2}\right\rangle \otimes \left|-\frac{1}{2}\right\rangle \\
|1,0\rangle &= \frac{1}{\sqrt{2}}\left[\left|\frac{1}{2}\right\rangle \otimes \left|-\frac{1}{2}\right\rangle + \left|-\frac{1}{2}\right\rangle \otimes \left|\frac{1}{2}\right\rangle\right] \\
|1,1\rangle &= \left|\frac{1}{2}\right\rangle \otimes \left|\frac{1}{2}\right\rangle .
\end{aligned}
$$

Both matrices above assume that the basis is aligned with $\vec{B}$ and the Ising axis respectively. Therefore some rotation operator must be found that can describe a state given in the z-axis basis in a basis aligned with $\vec{B}$ or the Ising axis.

Consider therefore first a rotation of the *state* vectors, which can be inverted to receive the forward transformation also necessary for the transformation of operator matrices. The inversion process is but a complex conjugation since the operator in question is unitary. It can be shown that the rotation about three Euler angles α, β, δ of a state is given by the matrix with elements as [11]:

$$
\mathcal{U}_{m'm''} = \langle s, m'| \, e^{\frac{-iS_\mu \alpha}{\hbar}} e^{\frac{-iS_y \beta}{\hbar}} e^{\frac{-iS_\mu \delta}{\hbar}} \, |s, m''\rangle \, .
$$

Here, s, m', m'' are spin quantum numbers of the system, which in the more general case can be replaced by angular momentum quantum numbers. The rotations α, β and δ are done about the $z-$, $y-$ and then $z-$ body axes of the system. Since the spin states considered here are symmetric about their body z-axes the final rotation δ is superfluous and thus will be discarded. Identifying the angles $\alpha = \varphi$ and $\beta = \vartheta$ for rotation to some spherical coordinates it can further be shown that the exponential operators amount to:

$$
\mathcal{U} = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & \frac{e^{-i\varphi}}{2}(1+\cos(\vartheta)) & \frac{e^{-i\varphi}}{\sqrt{2}}\sin(\vartheta) & \frac{e^{-i\varphi}}{2}(1-\cos(\vartheta)) \\
0 & -\frac{1}{\sqrt{2}}\sin(\vartheta) & \cos(\vartheta) & \frac{1}{\sqrt{2}}\sin(\vartheta) \\
0 & \frac{e^{i\varphi}}{2}(1-\cos(\vartheta)) & -\frac{e^{i\varphi}}{\sqrt{2}}\sin(\vartheta) & \frac{e^{i\varphi}}{2}(1+\cos(\vartheta))
\end{pmatrix} .
$$

An operator matrix A transforms under rotation as $A_{rot} = \mathcal{U} A \mathcal{U}^\dagger$, so the operator of equation 6, which is expressed in terms of a basis rotated by angles ϑ_B and φ_B, can be written in the z-axis basis as:

$$
\vec{B} \cdot \vec{S} = B\hbar \begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & -\cos(\vartheta_B) & \frac{e^{-i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) & 0 \\
0 & \frac{e^{i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) & 0 & \frac{e^{-i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) \\
0 & 0 & \frac{e^{i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) & \cos(\vartheta_B)
\end{pmatrix} . \tag{8}
$$

Analogously, the matrix of equation 7 is expressed in a basis rotated through angles ϑ_r and φ_r, so in the z-axis basis it can be written:

$$
S_\mu^{(1)} S_\mu^{(2)} = \frac{\hbar^2}{4} \begin{pmatrix}
-1 & 0 & 0 & 0 \\
0 & \cos^2(\vartheta_r) & -\frac{e^{i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & e^{-2i\varphi_r}\sin^2(\vartheta_r) \\
0 & -\frac{e^{-i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & -\cos(2\vartheta_r) & \frac{e^{-i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) \\
0 & e^{2i\varphi_r}\sin^2(\vartheta_r) & \frac{e^{i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & \cos^2(\vartheta_r)
\end{pmatrix} . \tag{9}
$$

In equations 8 and 9 it is readily visible that the spin singlet state $|0,0\rangle$ is unaffected by the external magnetic field, as could be concluded even without the explicit Hamiltonian. As a result the total Hamiltonian for the singlet state is but the sum of two terms dependent on different sets of variables. This is to say that separation of variables can be used to solve the eigenstate problem, so the qualities presently at interest are lost. For this reason henceforth only the non-singlet, that is triplet, states are considered, and matrices will subsequently be reduced to the relevant three-dimensional subspace for simplicity's sake.

At last all parts of the potential energy are expressed in a single basis, such that the potential part of the Hamiltonian takes the form:

$$\mathcal{H}_f = \gamma B\hbar \begin{pmatrix} \xi\cos^2(\vartheta_r) + \cos(\vartheta_B) & -\frac{e^{-i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) - \xi\frac{e^{-i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & \xi e^{-2i\varphi_r}\sin^2(\vartheta_r) \\ -\frac{e^{i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) - \xi\frac{e^{i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & -\xi\cos(2\vartheta_r) & -\frac{e^{-i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) + \xi\frac{e^{-i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) \\ \xi e^{2i\varphi_r}\sin^2(\vartheta_r) & -\frac{e^{i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) + \xi\frac{e^{i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & \xi\cos^2(\vartheta_r) - \cos(\vartheta_B) \end{pmatrix}. \tag{10}$$

Here, $\xi = \frac{J}{\gamma B}$ is a proportionality factor between the spin-spin and spin-field interactions. This together with the kinetic part of the Hamiltonian

$$\mathcal{H}_s = \sum_{n=1}^{5} \frac{p_i^2}{2m_i}. \tag{11}$$

will determine the time evolution of the system.

4 The Born-Oppenheimer approximation

4.1 Derivation

Solving for the eigenstates of such a Hamiltonian as described above is a mighty task. Note in particular that the contribution from potential energy to the Hamiltonian, equation 10, is heavily dependent on the position and orientation of the dumbbell through all parameters ϑ_r, ϑ_B, φ_r, φ_B and B. This couples all degrees of freedom for the system, which complicates the problem greatly.

An approximation is therefore in order. If the position and orientation, henceforth the "slow" parameters, are more or less static in comparison with the spin degrees of freedom, henceforth "fast" parameters, the Born-Oppenheimer approximation is applicable. A version of the adiabatic approximation described in section 2.1, it assumes that a "fast" subsystem, dependent on the fast parameters, can be described by eigenstates of a Hamiltonian parametrised by the slow parameters. A fast system in such an eigenstate can be considered to remain in the same eigenstate as the associated "fast" Hamiltonian slowly changes, changing its eigenvalue as the slow parameters evolve. So far this is just the adiabatic approximation. In the terminology used for geometric phases the slow parameters then correspond to the parameter space of the fast system.

The Born-Oppenheimer approximation involves the extension of this to also consider how the "slow" system evolves, in practice finding an effective Hamiltonian to the slow system as well. Originally an approximation used in molecular physics proposed in 1927 [12], it applies also to the present situation. The full system is considered to be described by the product of a wave function to the slow system Ψ_s and some eigenstate of the fast Hamiltonian $|n\rangle$, i.e.,

$$|\Psi_{full}\rangle = \Psi_s |n\rangle .$$

The aforementioned fast and slow Hamiltonians are for the system in consideration the previously found $\mathcal{H}_f$ and $\mathcal{H}_s$, respectively. The full solution of the fast system is assumed to be known, i.e., that

$$\mathcal{H}_f |n\rangle = E_n |n\rangle$$

is solved. The Schrödinger equation implies, since $\frac{\partial}{\partial t}|n\rangle = 0$,

$$\begin{aligned} i\hbar\frac{\partial}{\partial t}(\Psi_s|n\rangle) &= (\mathcal{H}_f + \mathcal{H}_s)\Psi_s|n\rangle \\ i\hbar\frac{\partial\Psi_s}{\partial t} &= \langle n|(\mathcal{H}_f + \mathcal{H}_s)|n\rangle\Psi_s = (\langle n|\mathcal{H}_s|n\rangle + E_n)\Psi_s. \end{aligned}$$

This can be interpreted as an effective Hamiltonian $\mathcal{H}_{eff}^{(n)} = \langle n|\mathcal{H}_s|n\rangle + E_n$ governing the slow wave function. The inner product term can be further manipulated in our system as follows:

$$\langle n|\mathcal{H}_s|n\rangle\Psi_s = \sum_{i=1}^{5}\left[\langle n|\frac{p_i^2}{2m_i}|n\rangle\Psi_s + \langle n|\frac{p_i}{m_i}|n\rangle p_i\Psi_s + \frac{p_i^2}{2m_i}\Psi_s\right].$$

Here, it is to be understood that the momentum operators in $\mathcal{H}_s$ act on *both* the spin ket and and the slow wave function. An operator to the left of a ket and wave function product will however be understood to act on the ket only, if no clarifying parentheses are written out explicitly. Since all p_i are Hermitian operators and can thus be acted on bras to the left without conjugation the most troublesome term can also be written as:

$$\begin{aligned}\langle n|\, p_i^2\, |n\rangle &= \langle p_i n|p_i n\rangle \\ &= \langle p_i n|n\rangle \langle n|p_i n\rangle + \langle p_i n|\left(\mathbb{1} - |n\rangle\langle n|\right)|p_i n\rangle \\ &= \langle n|p_i n\rangle^2 + \langle p_i n|\left(\mathbb{1} - |n\rangle\langle n|\right)|p_i n\rangle\,.\end{aligned}$$

An identity relation was used in the second step, where $\mathbb{1}$ is the identity operator. Inserting the derivative form of the momentum operator as seen in section 3.2 and rearranging terms with some convenient notation we arrive to the Hamiltonian providing the interesting properties sought after:

$$\mathcal{H}_{eff}^{(n)} = \sum_{i=5}^{5} \frac{(p_i - A_i^{(n)})^2}{2m_i} + \Phi^{(n)} + E_n, \tag{12}$$

$$A_i^{(n)} = i\hbar\langle n|\partial_i n\rangle\,, \tag{13}$$

$$\Phi^{(n)} = \sum_{i=1}^{5} \frac{\hbar^2}{2m_i}\langle \partial_i n|\left(\mathbb{1} - |n\rangle\langle n|\right)|\partial_i n\rangle\,. \tag{14}$$

4.2 Interpretation

Equation 12 has been aptly written on a form which suggests the physics to be studied. Note that the sum over i looks precisely like the Hamiltonian of a magnetic field with vector potential $\vec{A}^{(n)} = i\hbar\left\langle n\middle|\vec{\boldsymbol{\nabla}} n\right\rangle$ on a particle of charge 1 and momentum $\vec{p} = (p_1, \ldots, p_5)$. Note also that this magnetic field is the same as the synthetic magnetic field outlined in section 2.2, and as of such carries precisely the same properties. In particular the field will carry a monopolar dependence, as desired. A difference present to Maxwellian magnetic fields is that both field and momentum are here five dimensional, and furthermore that the masses of the two final degrees of freedom are rather moments of inertia. The dynamics of this term is the main interest of this discussion, but we note also a scalar field $\Phi^{(n)}$, analogously called the synthetic electric field or the synthetic scalar field. Roughly speaking however this field is a factor $\hbar$ smaller than the synthetic magnetic field and will in most cases be negligible.

It can however be shown that the scalar field acts as a repulsive inverse square force near degeneracies in the fast Hamiltonian[13]. The inverse square dependence to the distance of a degeneracy point means that the scalar field will have appreciable effects if the slow parameters are close enough to the degeneracy, and furthermore the repulsive nature actually leads to a strengthening of the Born-Oppenheimer approximation as the adiabatic approximation loses validity at points of degeneracy.

4.3 Dynamics

Having found an effective Hamiltonian to the slow system the application of this Hamiltonian to the dynamics of the system remains to be performed. One could proceed with the quantum mechanical methods used so far, solving for eigenstates of $\mathcal{H}_{eff}$. It is however now practical to consider the slow subsystem to effectively lie in the classical domain, and the Hamiltonian derived by quantum mechanical means will be utilized in the role of the Hamiltonian for classical mechanics.

Hamilton's canonical equations indicate the time evolution of $\vec{r}$:

$$\begin{aligned}\frac{d\vec{r}}{dt} &= \frac{\partial \mathcal{H}_{eff}^{(n)}}{\partial \vec{p}} = \frac{\vec{p} - \vec{A}^{(n)}}{m} \\ \frac{d\vec{p}}{dt} &= -\frac{\partial \mathcal{H}_{eff}^{(n)}}{\partial \vec{r}} = \left(\frac{\partial \vec{A}^{(n)}}{\partial \vec{r}}\right)^T \frac{\vec{p} - \vec{A}^{(n)}}{m} - \frac{\partial \Phi^{(n)}}{\partial \vec{r}} - \frac{\partial E_n}{\partial \vec{r}} = \left(\frac{\partial \vec{A}^{(n)}}{\partial \vec{r}}\right)^T \frac{d\vec{r}}{dt} - \frac{\partial \Phi^{(n)}}{\partial \vec{r}} - \frac{\partial E_n}{\partial \vec{r}}.\end{aligned}$$

Note in particular that the first of these equations imply that the canonical momentum $\vec{p}$ is *not* $m\frac{d\vec{r}}{dt}$. The effective force acting on the system can be found, utilizing that the synthetic vector potential does

not depend explicitly on time, i.e., that $\frac{\partial \vec{A}^{(n)}}{\partial t} = 0$:

$$m\frac{d^2\vec{r}}{dt^2} = \frac{d\vec{p}}{dt} - \frac{d\vec{A}^{(n)}}{dt} = \left(\frac{\partial \vec{A}^{(n)}}{\partial \vec{r}}\right)^T \frac{d\vec{r}}{dt} - \left(\frac{d\vec{r}}{dt}\cdot\vec{\boldsymbol{\nabla}}\right)\vec{A}^{(n)} - \frac{\partial \Phi^{(n)}}{\partial \vec{r}} - \frac{\partial E_n}{\partial \vec{r}}. \tag{15}$$

The Jacobian matrix can be treated elementwise, as well as the second term:

$$\frac{1}{i\hbar}\left(\frac{\partial \vec{A}^{(n)}}{\partial \vec{r}}\right)_{ji} = \partial_i \langle n|\partial_j n\rangle = \langle \partial_i n|\partial_j n\rangle + \langle n|\partial_i\partial_j n\rangle$$

$$\frac{1}{i\hbar}\left(\left(\frac{d\vec{r}}{dt}\cdot\vec{\boldsymbol{\nabla}}\right)\vec{A}^{(n)}\right)_i = \sum_{j=1}^{5}\frac{dr_j}{dt}\partial_j \langle n|\partial_i n\rangle = \sum_{j=1}^{5}\frac{dr_j}{dt}(\langle \partial_j n|\partial_i n\rangle + \langle n|\partial_j\partial_i n\rangle).$$

Insertion into equation 15 then yields a higher dimensional analogue to a cross-product based Lorentz-type force:

$$\begin{aligned}\frac{1}{i\hbar}\left[\left(\frac{\partial \vec{A}^{(n)}}{\partial \vec{r}}\right)^T \frac{d\vec{r}}{dt} - \left(\frac{d\vec{r}}{dt}\cdot\vec{\boldsymbol{\nabla}}\right)\vec{A}^{(n)}\right]_i &= \frac{1}{i\hbar}F_i^A = \sum_{j=1}^{5}\frac{dr_j}{dt}[\langle \partial_i n|\partial_j n\rangle - \langle \partial_j n|\partial_i n\rangle] \\ &= \sum_{j=1}^{5}\sum_{l}\frac{dr_j}{dt}[\langle \partial_i n|l\rangle\langle l|\partial_j n\rangle - \langle \partial_j n|l\rangle\langle l|\partial_i n\rangle] \\ &= \sum_{j=1}^{5}\sum_{l\neq n}\frac{dr_j}{dt}[\langle \partial_i n|l\rangle\langle l|\partial_j n\rangle - \langle \partial_j n|l\rangle\langle l|\partial_i n\rangle] \\ &= \sum_{j\neq i}\sum_{l\neq n}\frac{dr_j}{dt}[\langle \partial_i n|l\rangle\langle l|\partial_j n\rangle - \langle \partial_j n|l\rangle\langle l|\partial_i n\rangle].\end{aligned}$$

Here, $|l\rangle$ simply denotes an eigenstate to $\mathcal{H}_f$ of some index l, and the sum over l is over all available states. The exclusion of $l = n$-terms follows as $\langle \partial_i n|n\rangle$ is purely imaginary, which can be seen from differentiating $\langle n|n\rangle = 1$. This rearrangement is highly desirable, for it now so happens that this allows us to take derivatives of the Hamiltonian instead of the rather tricky differentiation of the eigenkets. Differentiating the Schrödinger equation and acting on it with some other eigenbra yields:

$$\begin{aligned}\mathcal{H}_f|n\rangle = E_n|n\rangle &\implies \\ \partial\mathcal{H}_f|n\rangle + \mathcal{H}_f|\partial n\rangle = E_n|\partial n\rangle &\implies \\ \langle l|\partial\mathcal{H}_f|n\rangle = \langle l|\partial n\rangle(E_n - E_l).&\end{aligned}$$

Rearranging, a very useful relation emerges:

$$\langle l|\partial n\rangle = \frac{\langle l|\partial\mathcal{H}_f|n\rangle}{E_n - E_l}. \tag{16}$$

This we can insert into the above:

$$\begin{aligned}\frac{1}{i\hbar}F_i^A &= \sum_{j\neq i}\sum_{l\neq n}\frac{\frac{\partial r_j}{\partial t}}{(E_n - E_l)^2}[\langle n|\partial_i\mathcal{H}_f|l\rangle\langle l|\partial_j\mathcal{H}_f|n\rangle - \langle n|\partial_j\mathcal{H}_f|l\rangle\langle l|\partial_i\mathcal{H}_f|n\rangle] \\ &= 2i\sum_{j\neq i}\sum_{l\neq n}\frac{\frac{\partial r_j}{\partial t}}{(E_n - E_l)^2}\,\mathrm{Im}\,[\langle n|\partial_i\mathcal{H}_f|l\rangle\langle l|\partial_j\mathcal{H}_f|n\rangle]. \qquad (17)\end{aligned}$$

Equation 16 can also be used when evaluating the synthetic electric potential:

$$\Phi^{(n)} = \sum_{i=1}^{5}\sum_{l\neq n}\frac{\hbar^2}{2m_i}\langle \partial_i n|l\rangle\langle l|\partial_i n\rangle = \sum_{i=1}^{5}\sum_{l\neq n}\frac{\hbar^2}{2m_i}\frac{\langle n|\partial_i\mathcal{H}_f|l\rangle\langle l|\partial_i\mathcal{H}_f|n\rangle}{(E_n - E_l)^2} \tag{18}$$

$$= \sum_{i=1}^{5}\sum_{l\neq n}\frac{\hbar^2}{2m_i}\frac{|\langle n|\partial_i\mathcal{H}_f|l\rangle|^2}{(E_n - E_l)^2}. \tag{19}$$

For the last equalities to hold in both contributions we require that the derivative of the Hamiltonian is Hermitian, but thankfully derivatives of Hermitian operators are Hermitian so this holds true. Note however that no simple form to the derivative of the electric potential $\Phi^{(n)}$ has been found, which might not be easily described analytically.

The problem has thus been reduced to evaluating, per equations 17 and 18,

$$m\frac{d\vec{r}}{dt^2} = \vec{F}^A - \frac{\partial\Phi^{(n)}}{\partial\vec{r}} - \frac{\partial E_n}{\partial\vec{r}}. \tag{20}$$

4.4 Differentiation of the Hamiltonian

In order to easily evaluate equations 17 and 18 derivatives of $\mathcal{H}_f$ from equation 10 are to be found. Writing any of the coordinates x, y, z as r the derivatives can be written:

$$\begin{aligned}
\partial_r\mathcal{H}_f &= \gamma B\hbar \begin{pmatrix} \frac{\dot{B}}{B}\cos(\vartheta_B) - \dot{\vartheta}_B\sin(\vartheta_B) & \Omega & 0 \\ \Omega^* & 0 & \Omega \\ 0 & \Omega^* & -\frac{\dot{B}}{B}\cos(\vartheta_B) + \dot{\vartheta}_B\sin(\vartheta_B) \end{pmatrix}, \\
\partial_{\vartheta_r}\mathcal{H}_f &= \gamma B\xi\hbar \begin{pmatrix} -\sin(2\vartheta_r) & -\sqrt{2}e^{-i\varphi_r}\cos(2\vartheta_r) & e^{-2i\varphi_r}\sin(2\vartheta_r) \\ -\sqrt{2}e^{i\varphi_r}\cos(2\vartheta_r) & 2\sin(2\vartheta_r) & \sqrt{2}e^{-i\varphi_r}\cos(2\vartheta_r) \\ e^{2i\varphi_r}\sin(2\vartheta_r) & \sqrt{2}e^{i\varphi_r}\cos(2\vartheta_r) & -\sin(2\vartheta_r) \end{pmatrix}, \\
\partial_{\varphi_r}\mathcal{H}_f &= \gamma B\xi\hbar \begin{pmatrix} 0 & i\frac{e^{-i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & -2ie^{-2i\varphi_r}\sin^2(\vartheta_r) \\ -i\frac{e^{i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & 0 & -i\frac{e^{-i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) \\ 2ie^{2i\varphi_r}\sin^2(\vartheta_r) & i\frac{e^{i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & 0 \end{pmatrix}.
\end{aligned} \tag{21}$$

Here, $\Omega = (-\frac{\dot{B}}{B}\sin(\vartheta_B) + i\dot{\varphi}_B\sin(\vartheta_B) - \dot{\vartheta}_B\cos(\vartheta_B))\frac{e^{-i\varphi_B}}{\sqrt{2}}$ is introduced as a means of compressing the rather lengthy expressions for the derivative with respect to r.

4.5 Solution of the fast subsystem

The usage of the Born-Oppenheimer approximation requires a solution for the fast subsystem, i.e., that the eigenvalues and eigenvectors of $\mathcal{H}_f$ are found. Unfortunately this is not possible analytically for the present system, but note that it is the same as solving the following cubic characteristic equation, which follows from equation 10, for the eigenvalues $\lambda_n = \frac{E_n}{\gamma B\hbar}$:

$$0 = \begin{vmatrix} \xi\cos^2(\vartheta_r) + \cos(\vartheta_B) - \lambda_n & -\frac{e^{-i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) - \xi\frac{e^{-i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & \xi e^{-2i\varphi_r}\sin^2(\vartheta_r) \\ -\frac{e^{i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) - \xi\frac{e^{i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & -\xi\cos(2\vartheta_r) - \lambda_n & -\frac{e^{-i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) + \xi\frac{e^{-i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) \\ \xi e^{2i\varphi_r}\sin^2(\vartheta_r) & -\frac{e^{i\varphi_B}}{\sqrt{2}}\sin(\vartheta_B) + \xi\frac{e^{i\varphi_r}}{\sqrt{2}}\sin(2\vartheta_r) & \xi\cos^2(\vartheta_r) - \cos(\vartheta_B) - \lambda_n \end{vmatrix}. \tag{22}$$

Calculation of eigenvalues and eigenvectors may be left to numerics from the onset. Equation 22 could in principle be differentiated implicitly to receive the derivatives of the energies also needed, but since simulation of the system requires many other quantities to be calculated numerically, the derivatives of the energies will be done likewise for the sake of practicality.

5 Simulation

The equation of motion 20 represents the furthest point that analytical methods have reached for this system. The inherent complexity in the problem now necessitates numerical analysis if further results are to be reached, and for this reason the resultant dynamics of the translating and rotating dumbbell have been simulated for selected parameters in a predetermined external magnetic field.

5.1 The chosen field

It is apparent from the form of the differentiated Hamiltonians of equations 21 that the synthetic fields are set in proportion to the inhomogeneity of the external magnetic field. To maximize the action of the synthetic fields it then becomes important to consider an external field which varies as much as possible

in both magnitude and direction. It is furthermore of interest that the field contains a point or points of zero magnitude. If the spin-spin interaction factor J is zero, these will be the points of synthetic magnetic charge which we wish to study. In general we consider $J \neq 0$, for which these charges will not sit precisely at the points of vanishing fields but will translate as a function of J away from these as described in section 2.3.

As an initial example of such a field consider two simple coils of the same diameter and axis of symmetry placed some distance apart. If a current is run through both coils but the direction of current is different between the two, one running clockwise and the other counter-clockwise, a suitable field is created. Not even the complete field of a single coil has a closed analytic expression, but we may still note some qualitative properties. If the axis of rotational symmetry for both coils is taken to be the z-axis, as will be done in the simulations, the xy-plane at equal distance to both coils will have a field z-component that is zero. This follows from simple symmetry reasons, from which it is also apparent that the total field is precisely zero at the "centre" of this plane, where the distance to all current-carrying wires are the same. The magnetic field in the plane then increases in magnitude as the distance between xy-plane and the nearest coil wire decreases, pointing either directly towards or away from the centre point depending on current direction. In the limit of increasing distance to the centre point the field naturally tends towards zero. If points away from the xy-plane are considered the field will tend in a smooth fashion towards the regular single coil field.

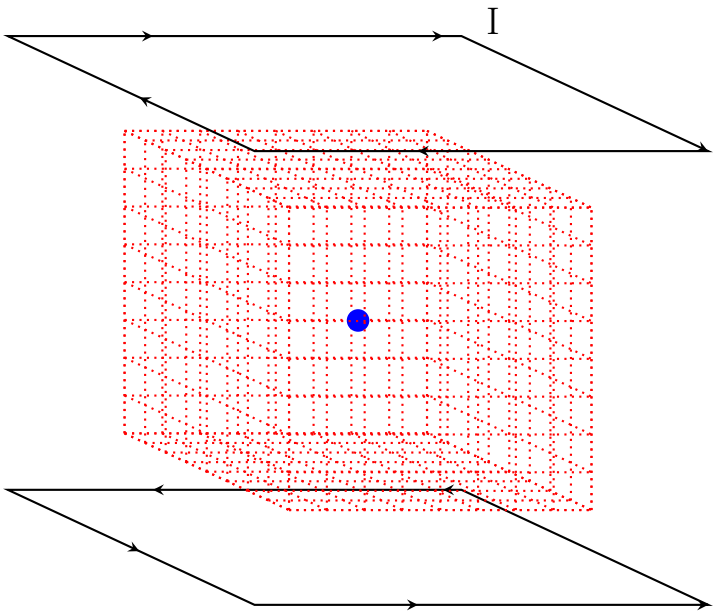


Figure 3: The currents generating the external magnetic field enclosing the simulated "lab" region, shown in red. A blue dot marks the point of zero field strength at the centre.

For computational practicality the coils for the simulation are taken to be as described above, with the axis of symmetry being the z-axis, but being of square form instead of the regular circular shape. This perturbs the field described above slightly, which after all does nothing but increase the desired inhomogeneity. The side length of the coils is taken to be the distance between the coils, so that they form edges of a cube, see figure 3.

5.2 Code outline

Simulation of the system is in essence nothing more complicated than solving the ordinary differential equation, the ODE, 20. Some difficulties arise since the fast Hamiltonian is not analytically diagonalisable, all scalar fields must be calculated and in addition several quantities need to be numerically differentiated. The dumbbell is given an initial velocity and is positioned close to the boundary of a box of side length $\frac{2}{3}$ times the distance between the coils, centred about the zero-field point described in section 5.1. All simulation is done in the box. That is to say, that the dumbbell is only allowed inside of this box and that field values are exclusively calculated within the box, which will be at times referred to as the "lab".

For complete Python scripts, see appendix A. A few notable choices of methods will be here mentioned, but further details should be discernible from the commented code.

The lab is divided into a discrete lattice of points so that field values can be calculated in advance and saved to file, illustrated by the red grid in figure 3. This has the side-effect of restricting the position of the dumbbell to points in this lattice. The field generation is done per numerical integration of the Biot-Savart law using the same step size as the lab lattice.

Solution of the differential equation is done by means of the *scipy* function solve_ivp() which is a somewhat sophisticated tool containing error estimation and flexible termination conditions, here used to stop the algorithm if the dumbbell leaves the lab. As is often done the ODE is transformed into first order by extension of the five positional coordinates to a ten-dimensional position and velocity vector. All acceleration contributions are summed up in a single function called by the ODE solver. As part of this process the diagonalization of the fast Hamiltonian is performed by the *scipy* funtion eigh(). It here becomes crucial to consider which eigenstate of the fast Hamiltonian the dumbbell is in. A predetermined eigenstate is selected for each simulation, indexed by $n = 0$, 1, 2 in increasing order of energies. Since each eigenstate is indexed by the value of its energy it is crucial that the energies never cross, which would lead to eigenstates changing index. This problem is of course resolved by the adiabatic approximation assumed, and any trajectories of dumbbells traversing points of energy degeneracy are discarded.

At many stages in the process derivatives of fields are needed, as well as derivatives of the fast energies. Since no closed forms of these quantities are available the derivatives of some quantity $\Lambda(\omega)$ depending on coordinate ω has been approximated as

$$\frac{\partial \Lambda}{\partial \omega} \approx \frac{\Lambda(\omega + s_\omega) - \Lambda(\omega - s_\omega)}{2s_\omega},$$

where s_ω is the step size of ω implied by the lattice. This is sometimes called the three-point centred difference formula. No error estimation scheme has been implemented for these parts of the algorithm, so care must be taken that the step size does not become so small that rounding errors of the floating point operators dominate. A rule of thumb is that a step size of $s_\omega \approx \sqrt[3]{\epsilon}\omega_c$ is close to the optimal point of good precision without leading to large rounding errors [14]. Here ϵ is the machine epsilon in the order of 10^{-16}, and ω_c is the typical scale of ω, which is taken to be the lab side length, so it is clear that any lattice with less than 10^5 sites along each cube side leads to step sizes well above this limit.

Finally the result is plotted using appropriate functions, as of yet a simple *pyplot* implementation has been done.

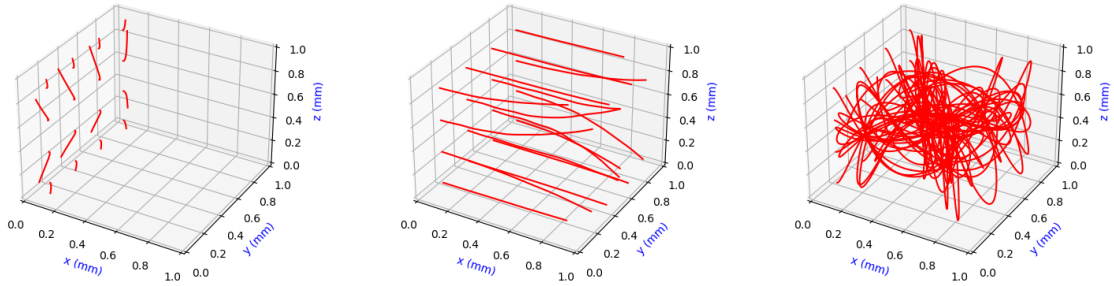
6 Results

Sample executions of the simulation outlined in section 5 have been performed. As a starting point a lab side length of 1 mm was chosen, the lattice granularity was set to 101^3 sites and the current running through each coil was selected as 10 A. All graphs presented are drawn with metres as axis units. In light of the typical Stern-Gerlach experiment the total mass of the dumbbell was initially assumed to be 3.58×10^{-25} kg, the mass of two silver atoms, and additionally the distance between the two dumbbell masses was set to 5×10^{-5} m. At first a distance in the order of Ångström was used, but nonzero spin-spin coupling J then led to rotational accelerations too large for the ODE solver to handle. The velocity by which the dumbbell enters the lab was 1 cm/s in the positive x direction for all simulations. A maximum simulation time of 0.1 s was assumed, but this matters only for the high energy state since the other trajectories exit the lab before then. The graphs shown have been "swarmed", by which it is meant that several simulations of equal parameters have been placed at equidistant initial positions along the yz plane. Note also that the linearity of the external magnetic field strength B with respect to the current I means that any multiplication of that variable will yield the same effect as an equal change in the spin-field coupling γ, since the field strength and coupling parameter always appear as a single product. For this reason the current of the magnetic field will in general be held fixed and γ will be varied instead, as this does not require additional field generation.

The scripts print average force magnitudes for the different terms in equation 20. For sake of brevity, these are not displayed in full here, but will be quoted when appropriate. Note also that the rotation of the dumbbell, while simulated and while it does affect the centre off mass trajectory, is not plotted.

First all three possible eigenstates were tested for the given parameters, and with $\gamma = 10^{10}$ J/($\hbar$T), $J = 10^5$ J/$\hbar$, see figure 4. For all of these simulations the gradient of the fast Hamiltonian energy was the dominating term by a factor $\sim 10^5$, but note the large differences in behaviour of this non-synthetic

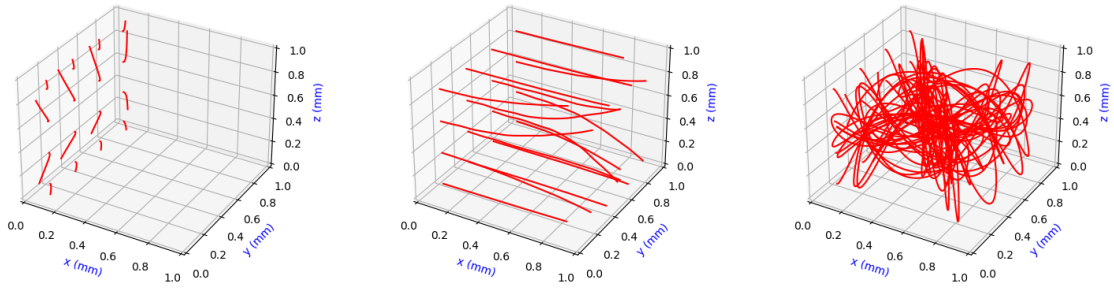
action. This is the expected behaviour, as will be discussed in section 7.1 , but note for now that the low energy state is repelled away from the lab and is thus not very useful. The high energy state is "caught" between the coils while the middle state achieves but a perturbed straight motion through the lab. Motion for the same parameters but excluding synthetic field effects were simulated, see figure 5.



(a) $n = 0$ (b) $n = 1$ (c) $n = 2$

Figure 4: All three eigenstates simulated for $\gamma = 10^{10}$ J/($\hbar$T), $J = 10^5$ J/$\hbar$.

Note that the nonsynthetic dynamics dominate completely. While small differences can be found they are miniscule, and may very well lie within the error of the simulation.



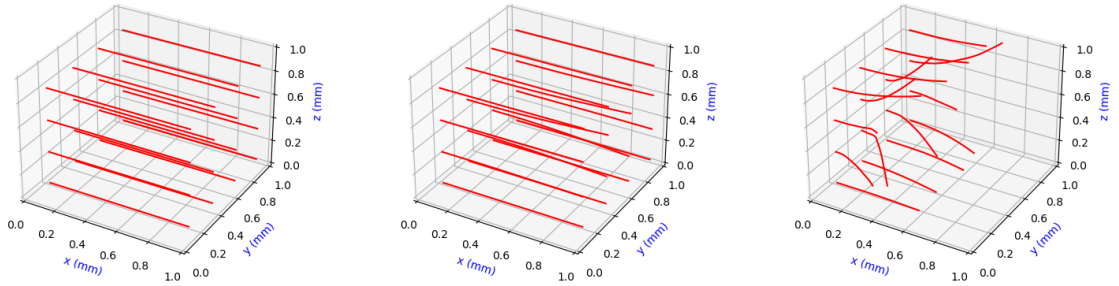
(a) $n = 0$ (b) $n = 1$ (c) $n = 2$

Figure 5: All three eigenstates simulated for $\gamma = 10^{10}$ J/($\hbar$T), $J = 10^5$ J/$\hbar$, without synthetic field effects.

6.1 The middle state

Let now attention be turned to the middle state ($n = 1$) case. If the nonsynthetic term is to be reduced in size, such that the synthetic field effects become visible, a reduction in fast Hamiltonian energy is desired. Note that for the special case $J = 0$ the fast Hamiltonian diagonalizes, and the middle state will simply have zero energy. This change in parameter yields a graph such as the one to the left in figure 6. As predicted the fast energy is now identically zero, up to a numerical error in the order of $\sim 10^{-44}$ J, and since that was previously the dominating term motion is simply rectilinear. While hardly a useful result in itself, it is now notable that the synthetic contributions to the dynamics are generally many orders of magnitude larger than the fast energy gradient, which is but a numerical residue in the order of $\sim 10^{-37}$ N. The magnitude of the synthetic forces, in the order of $\sim 10^{-31}$ N for the magnetic field and $\sim 10^{-33}$ N for the scalar field, are simply too small in comparison to the dumbbell mass to yield appreciable acceleration.

Any simple tool for increasing the action of the synthetic fields is however absent. As can be noted both by parallel to the geometric phase and directly from equations 17 and 18, the synthetic potentials are generally not dependent on the magnitude of the fast energy. As of such direct increases of parameters J and γ will typically have the effect of increased nonsynthetic action while the synthetic fields are not strengthened. Also relevant is the dependence of the synthetic magnetic force on the velocity of the dumbbell. However, much like the motion through a classical magnetic field the impulse from a Lorentz-type force over a fixed distance is independent of the velocity, as the travel time scales reciprocally to



(a) $J = 0$, $n = 1$, $m = 3.58 \times 10^{-25}$ kg

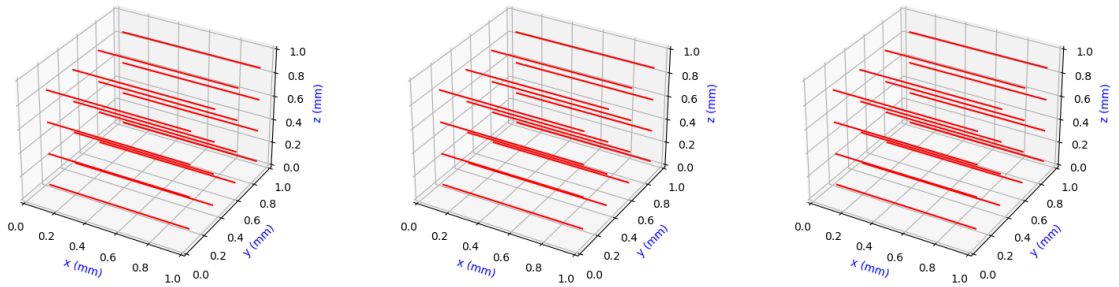
(b) $J = 0$, $n = 1$, $m = 3.58 \times 10^{-28}$ kg

(c) $J = 0$, $n = 1$, $m = 3.58 \times 10^{-29}$ kg

Figure 6: Three different masses for $n = 1$, $J = 0$

the force magnitude. The lab length could be adjusted, but here too would the effects on the synthetic magnetic field action cancel out. Imagine that the velocity were kept constant in relation to the lab side length. Multiplying the side length by any factor would then scale the velocity equally. However the derivatives present in equation 17 would scale reciprocally, and nothing would be achieved.

All that is left is to adjust the mass of the dumbbell, as this affects nothing but the resultant acceleration. Reduction of mass yields perceptible effects first in the order of $\sim 10^{-28}$ kg, and in $\sim 10^{-29}$ kg, as visible in figure 6. The dumbbell is clearly repelled from the point of energy degeneration, as is the predicted behaviour of the synthetic scalar field per section 4.2. That this behaviour is due to the synthetic fields only is clear from simulations run without synthetic dynamics, see figure 7. That the scalar synthetic field now dominates even though the numbers cited earlier for the "high" mass case would indicate otherwise can be traced to that the printed force magnitudes are averaged out over the trajectory, so a difference in trajectory will affect the average force values. Specifically if the scalar field repels the dumbbell away from the lab a larger proportion of the trajectory will be close to the centre and thus the average scalar force will increase. This is visible in the printouts, for the simulation to the right hand panel in figure 6 the average synthetic scalar force was in the order of $\sim 10^{-30}$ N.



(a) $J = 0$, $n = 1$, $m = 3.58 \times 10^{-25}$ kg

(b) $J = 0$, $n = 1$, $m = 3.58 \times 10^{-28}$ kg

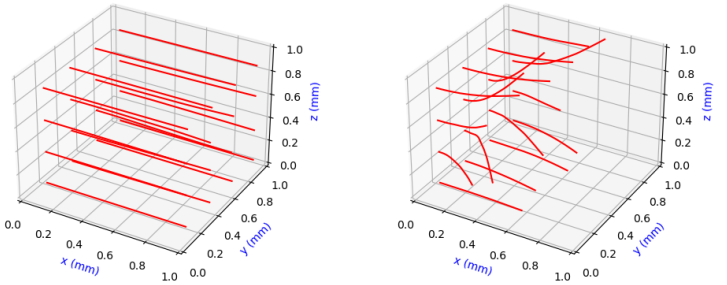
(c) $J = 0$, $n = 1$, $m = 3.58 \times 10^{-29}$ kg

Figure 7: Three different masses for $n = 1$, $J = 0$, without synthetic field effects.

That it is indeed the synthetic scalar field repelling the trajectories from the centre point can be confirmed through running simulations without either the synthetic scalar or magnetic field, as has been done in figure 8.

Note that the impact of the synthetic magnetic field as in the leftmost graph is not completely negligible, but does constitute a visible perturbation to the rectilinear motion. Further reduction of the mass to increase the synthetic magnetic dynamic lead however to no interesting patterns.

It should be noted that such large reductions of mass make the results rather unstable to numerical errors. It can be noted in many graphs above that the trajectories are not fully symmetric as one would expect in the perfect case, but that numerical perturbations propagate. It would be a stretch to call the behaviour 'chaotic', but the exact trajectories should not be taken for more than examples of qualitative behaviour.

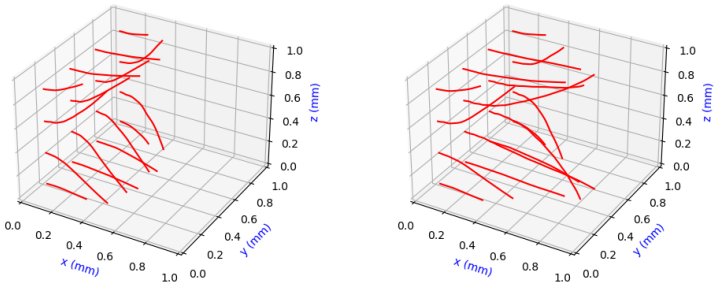


(a) $J = 0$, $n = 1$, $m = 3.58 \times 10^{-29}$ kg, no synthetic scalar field

(b) $J = 0$, $n = 1$, $m = 3.58 \times 10^{-29}$ kg, no synthetic magnetic field

Figure 8: The final simulation of figure 6 without the synthetic scalar or magnetic fields

The more exotic synthetic field given by a nonzero value of J would be of interest to consider, but regrettably leads to a sharp increase in the fast eigenvalues. This puts a limit to the size of the parameter which can be simulated before the dumbbell is quickly expelled from the lab, but sample simulations of $\gamma = 10^{10}$ J/($\hbar$ T) and $J = 100$ J/$\hbar$ were performed in figure 9. Note the dependence on whether synthetic field effects are included. Once again it is the synthetic scalar effects that dominate.



(a) $\gamma = 10^{10}$ J/($\hbar$ T) and $J = 100$ J/$\hbar$, $n = 1$, $m = 3.58 \times 10^{-29}$ kg, with synthetic field effects

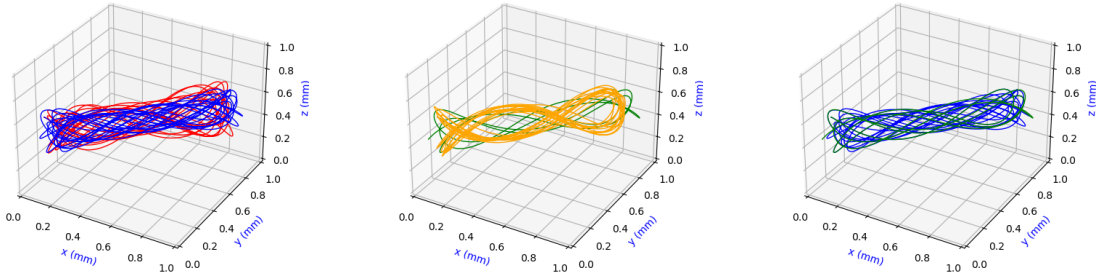
(b) $\gamma = 10^{10}$ J/($\hbar$ T) and $J = 100$, $n = 1$, $m = 3.58 \times 10^{-29}$ kg, without synthetic field effects

Figure 9: Simulations for the middle state with nonzero J, both with synthetic field effects and without.

6.2 The high energy state

The behaviour of the nonsynthetic fields to "trap" the dumbbell between the coils if placed in the high energy state ($n = 2$) is promising. Even if the fast energy gradient can be assumed to dominate the behaviour, the prospect of the synthetic fields acting on the dumbbell for extended periods of time suggests perhaps notable perturbations to the nonsynthetic motion. As seen in figure 4 the graphs are quickly cluttered when several starting positions are considered, and so a different "swarming" procedure was adopted. Choosing a single starting position and velocity trajectories owing to different choices of fields contributing were performed simultaneously, and subsequently coloured thereafter for visibility. Furthermore the simulation time was increased to half a second, as to propagate the synthetic perturbations as much as possible without obstructing visibility too much. This then infers a choice of starting position. Several such choices were sampled, here are presented the most interesting ones found as of yet. Further reducing the number of starting positions and parameters possible is the fact that some trajectories cross the degeneration at the centre, prohibiting field action comparison. Such choices of parameters are not shown here.

First simulations of the same parameters as in figure 4 were performed for a starting position close to $y = 0$ mm and $z = 0.75$ mm, see figure 10. Note the clearly visible perturbation of the synthetic field on the trajectory, and note further the large influence of the synthetic magnetic field on the trajectory



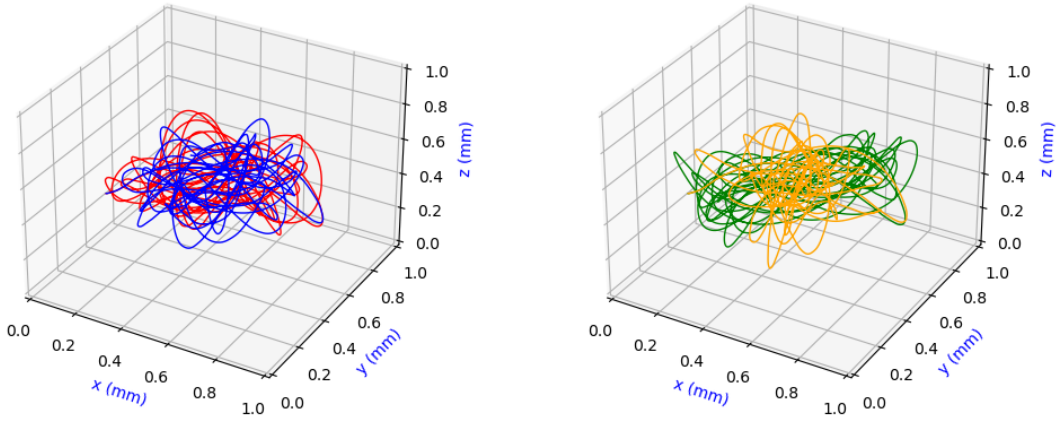
(a) With synthetic fields in red, without in blue

(b) Without synthetic magnetic field in green, without synthetic scalar field in yellow

(c) Without any synthetic fields in blue, without synthetic magnetic field in green

Figure 10: Simulations of the high energy state at a fixed starting position for $\gamma = 10^{10}$ J/(ħT), $J = 10^5$ J/ħ.

compared to the synthetic scalar field. Direct comparison between the trajectory without synthetic fields at all and the trajectory without the synthetic magnetic field show large similarities, but still perturbation by the synthetic field is noticeable. It can be concluded that the synthetic magnetic field plays a major role in repelling the trajectory from the slim cylinder traversed otherwise as seen in the left panel of figure 10. This is also supported by the average force values, which for the present simulation were in the order of $\sim 10^{-29}$ N for the synthetic magnetic field, $\sim 10^{-34}$ N for the synthetic scalar field and $\sim 10^{-24}$ N for the fast energy gradient.

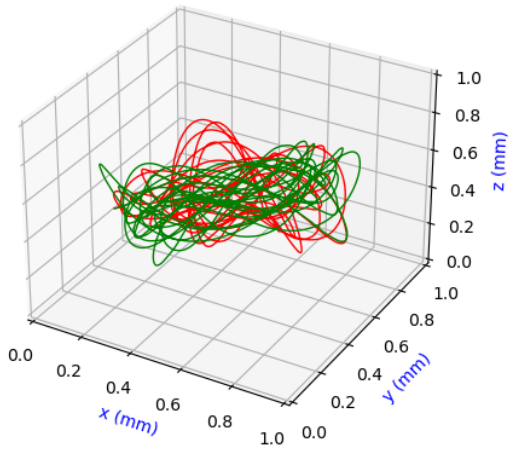


(a) With synthetic fields in red, without in blue

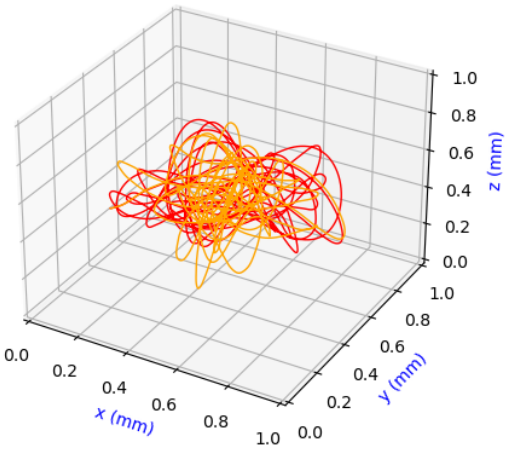
(b) Without synthetic magnetic field in green, without synthetic scalar field in yellow

Figure 11: Simulations of the high energy state at a fixed starting position for $\gamma = 10^8$ J/(ħT), $J = 10^5$ J/ħ, $m = 3.58 \times 10^{-27}$ kg.

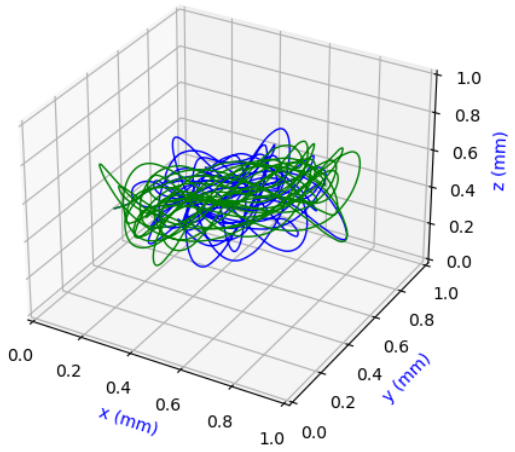
Just like the discussion in section 6.1 concluded reduction of mass is the main tool for increase of synthetic magnetic field effects. To this end simulations were run for a mass of 3.58×10^{-27} kg, where the spin-field coupling constant was adjusted accordingly to $\gamma = 10^8$ J/(ħT) such that the fast energy acceleration remained approximately the same, see figure 11. The starting position was close to $y = 0.25$ mm and $z = 0.25$ mm. Here, the synthetic effects can easily be said to play a significant role in the dynamics of the system. Qualitative descriptions of the behaviour are not from the onset very clear however, with the exception that the synthetic scalar field appears to somewhat confine the motion, or in the very least reduce the rotation about the system z-axis. That all contributions to the dynamics play noticeable roles can be seen from the pairwise comparisons of figure 12. For these simulations the magnitude of the synthetic scalar and magnetic forces were in the same order of $\sim 10^{-28}$ N, while the fast energy gradient was in the order of 10^{-26} N.



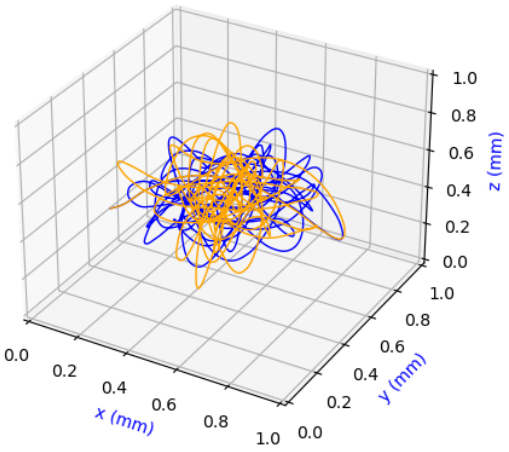
(a) With synthetic fields in red, without synthetic magnetic field in green



(b) With synthetic fields in red, without synthetic scalar field in yellow



(c) Without synthetic magnetic field in green, without both synthetic fields in blue



(d) Without synthetic scalar field in yellow, without both synthetic fields in blue

Figure 12: Simulations of the high energy state at a fixed starting position for $\gamma = 10^8$ J/($\hbar$T), $J = 10^5$ J/$\hbar$, $m = 3.58 \times 10^{-27}$ kg.

7 Discussion

7.1 On the nonsynthetic behaviour of the eigenstates

Why is there such a sharp distinction between the nonsynthetic forces on the different fast Hamiltonian eigenstates as seen in figure 4? This can be understood in the limit $J \ll \gamma$. For such parameters, which is often the case considered, the eigenstates of the fast Hamiltonian will tend to the simple cases of $|\vec{n}, 1\rangle$, $|\vec{n}, 0\rangle$ and $|\vec{n}, -1\rangle$, where the integer designates the directional spin along the axis $\vec{n}$ of the external magnetic field. This follows from that the fast Hamiltonian is diagonalizable at $J = 0$, where the high and low energy states in addition can be seen as the spin pointing in antiparallell or parallel fashion, respectively, to the external field. The fast energies of those states will tend towards $-B\gamma\hbar$, 0 and $B\gamma\hbar$ respectively. The only quantity therein dependent on the position or angle of the dumbbell is the external field strength B, and since the nonsynthetic force acts in opposite direction to the fast energy gradient the *only* behaviour of the dumbbell in this limit is to seek or avoid higher external field strengths, depending on whether the low or high energy state is assumed.

It is also the case that within the volume enclosed between the two coils, the lab, external field strength is at its highest further away from the centre, close to the coils, and at its lowest at the point of zero field magnitude at the centre. By this simple reasoning it becomes clear that the high energy state will be attracted to the centre of the lab, while the low energy state is quickly pushed away from that very point. The middle state has no energy gradient in the limit of low J, and so passes through the lab without any nonsynthetic force acting on it.

Of course, a different lab setup can change this behaviour. If the coils are for example put *inside* of the simulated lab, the low energy state becomes available to study as it will be attracted to the coils now present in the lab. This is a possible extension when considering further simulations done with the developed code. Whether it is an interesting one however remains to be seen, and it should be noted that such a setup would repel the high energy state from the lab instead.

7.2 On the topic of mass

It is a peculiar feature of the synthetic magnetic field that its strength is hardly affected by the parameters of the setup. As discussed in section 6.1 for a fixed magnetic field configuration this means adjustment of the dumbbell mass is a major tool for exploring synthetic dynamics. For the middle state however these dynamics became significant first when the mass was reduced to values in the order of $\sim 10^{-29}$ kg, a most displeasing result. The atomic mass unit is approximately 1.66×10^{-27} kg, dumbbell masses but a hundredth of this magnitude are far from feasible for any system where the movement can be considered "slow" and classical. Consequently the results of section 6.1 may be considered of value mostly due to the appearance of the predicted repulsive nature of the synthetic scalar field.

The case of the high energy states however bring great promise when it comes to domains of mass. Not only is the synthetic effects visible for the dumbbell mass of two silver atoms, rather heavy an element, but strong synthetic action appears already at masses in the order of $\sim 10^{-27}$. Diatomic dumbbells of light elements such as hydrogen lie within this mass region, and as of such could constitute candidates for realisations of the model if ever that enterprise is undertaken.

Adjustments of mass lead, at least in the high energy case, to corresponding adjustments of the spin-field coupling strength γB as was done in section 6.2. There, adjustments were only done to γ, since field strength adjustments are computationally more demanding, but it is of course the case that any real life realisation of the system would carry with it some rather fixed value of γ. It is therefore fortunate that the field strength in a realisation is highly adjustable by means of the current flowing through the coils. It remains unknown whether the currents needed are of reasonable order, but this could be checked by estimating plausible dumbbell parameters.

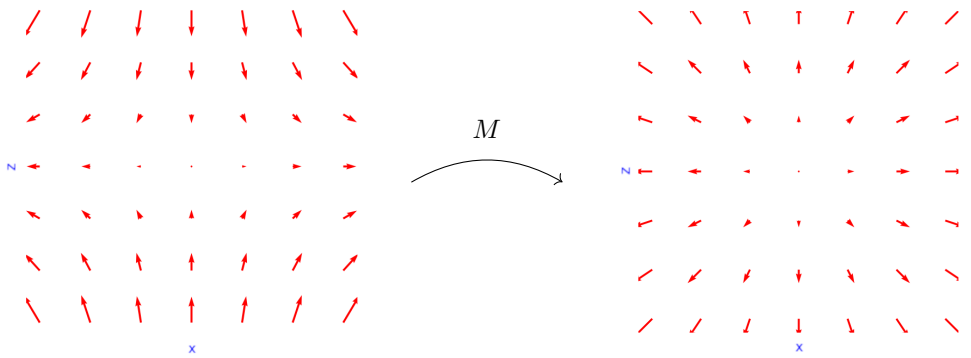
Is mass the *only* tool available for strengthening the synthetic effects? This need not be the case. Recalling the link between the Born-Oppenheimer synthetic magnetic field and the geometric phase, it is clear that traversing larger swathes of parameter space is connected to larger synthetic magnetic field action. This geometric line of thinking can elucidate the independence of the synthetic fields on energy magnitudes. Increasing the energy magnitudes, say through increasing B, does not change the path taken through parameter space more than multiplying its distance to the origin. Typical geometric phases, such as Berry's initial consideration of a system like that discussed in section 2.1, are independent of the path's distance to the origin, and by these means the energy magnitudes can be seen to not affect the synthetic magnetic field effects.

Here, another way of strengthening the synthetic effects become clear: by complicating the path taken through parameter space. The capability of the dumbbell to rotate already constitute a great achievement in this regard, as the relative direction of the magnetic field to the dumbbell axis may vary much more than what would otherwise be the case. Something not done here is however letting the dumbbell pass through some much more complicated external field. Many options are here available with nothing but one's creativity setting the limits. A more complicated field does however also complicate the study of the monopoles in the synthetic field, as the complexity of the mapping between real space and parameter space increases.

7.3 On the topic of monopoles

It was an explicit objective of this project to study the dynamics arising from magnetic monopole fields, as these are made available by the synthetic magnetic field. While non-negligible synthetic field action has been demonstrated for both the synthetic scalar and magnetic field, most prominently in the simulations of figure 11, the interpretation of these fields as arising from monopole action is a difficult one to consider.

One reason for such difficulties is simply that the monopoles to the synthetic magnetic field "live" in the appropriate parameter space, in the present case the space of external fields, both when it comes to their locations as well as the field they generate. The map between this space and real space cannot however be the identity map, such an external field is not allowed by Maxwell's equations, instead the map will be rather nontrivial in general. Any motion through real space must therefore first be "translated" by means of this map if that motion is to be seen as motion in a monopole field. Luckily the present choice of external magnetic field inadvertently implies a not too complex such map, see figure 13. As described in section 5.1 the external field from the two opposing coils increase in magnitude with increasing distance to the zero-point at the centre. The field additionally points towards the xy-plane intersecting the centre, and points either away from or towards the axis of symmetry depending on the choice of current direction. Therefore a simple reflection in the mentioned xy-plane followed, if the current direction requires it, by inversions of the x and y coordinates about the centre is almost the map between real space and parameter space required. Some scaling of all dimensions must also be performed, the details of which are potentially quite complex.



External field in real space. External field in parameter space.

Figure 13: An illustration of the map M between real space and parameter space, which must map every point of external field $\vec{b}$ to the parameter space point with the same external field $\vec{b}$. The planes shown are the xz-planes through the zero-field centre point of the lab and the origin, respectively.

It can also be noted, in contrast with the established goal, that the modification of the pure monopolar magnetic synthetic field by the introduction of the spin-spin coupling breaks the monopolar structure in unwanted ways. While the splitting of magnetic charge as described in section 2.3 is desired and interesting, it is also associated with a conversion of the field from purely monopolar to a field of nonzero curl. If we wish to study the effects of the monopole action this will obscure the results with dynamics arising from the non-monopolar part of the field, and separating the two effects may be impossible.

7.4 Numerical limitations

While the scripts written appear to work as intended no real analysis of their errors and stability has been performed. The apparent instability of the results presented in section 6.1 in particular motivate such

considerations. The division of the lab into a discrete lattice, as well as the discretization of dumbbell rotation angles, can be assumed to limit the accuracy of the result. The selection of 101^3 sites in the lattice was merely due to performance limitations, that number would have been increased had the setup allowed it, but the external field generation function quickly ran out of available resources if this was trialled. The discussion of section 5.2 shows that the used granularity is far below the optimal selection for the performance of the numerical differentiations, so increasing the amount of sites would be of high priority were the code to be optimized further. In addition to code optimization more powerful computing resources could further be employed to increase performance. The simulations here displayed were run but on a low-end personal laptop unit, so access to high-performance resources such as *UPPMAX* [15] could play a decisive role in further studies.

It was also encountered that the ODE solver simply would not converge in any reasonable amount of time for many choices of parameters. Being highly undesirable since this narrows the parameter ranges available for study, the precise reasons for this behaviour warrants further investigation. As an example large values of J or reductions of the dumbbell length l would often not integrate properly as the rotational accelerations and velocities became exceedingly large. Fast rotation of the dumbbell is perhaps not unphysical for the scenario considered, but appeared to not behave well in the numerical simulation. In addition it will be noted that exceedingly quick rotation could violate the adiabatic approximation, undermining the model. A possible correction to the model, if such rotations are deemed physical, could be to consider the rotational degrees of freedom part of the fast system instead of the slow. A hypothesis considering the origin of these non-convergent parameters is that the ODE solver does not behave well in conjunction with the lab discretization. The solver wishes to keep certain error estimates below a threshold value, here set to depend on the granularity of the lattice, and will reduce the step size of the integration until this is satisfied. It could be the case however that reductions of step size do not yield the intended effect for the solver, as the fitting of all positions to points in the lattice puts a lower bound on the effective step size attainable. Further investigation would be desirable.

8 Conclusions

The plausibility of synthetic field effects from the Born-Oppenheimer approximation playing a part in the dynamical evolution of the dumbbell model has been explored. Complete equations of motion, including the rotational degrees of freedom, have been found. These contain a Lorentz-type force involving a five-dimensional generalization of the cross product, which depends on the synthetic magnetic field emanating from the monopoles that are to be studied. Numerical evaluation of these equations of motion has been made possible through the development of Python scripts, one of the main achievements of the project.

Sample simulations by these scripts for a small set of parameters have been performed. The main results of these include a demonstration of the repulsive nature of the synthetic scalar field of the Born-Oppenheimer approximation, albeit for unrealistically small masses. For a much more reasonable mass range, in the order of the atomic mass unit, the simulations have demonstrated noticeable impact of both synthetic scalar and magnetic fields on the resultant dynamics for the high energy eigenstate of the spin system. The nature of these contributions has not been explored in any qualitative nor quantitative manner, but investigations into such matters have been facilitated by the construction of the scripts.

The promise of observing precisely the monopole effects of the synthetic magnetic field has been discussed with its limitations and difficulties. In addition to theoretical complications the endeavour is currently limited by the numerical performance of the scripts developed. Thus, optimization and further developments of these are warranted, as is further research into proposing experimental realisations of the system herein modelled.

References

[1] P. A. M. Dirac, "Quantised singularities in the electromagnetic field," Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character **133**, 60–72 (1931).

[2] C. Castelnovo, R. Moessner, and S. L. Sondhi, "Magnetic monopoles in spin ice", Nature **451**, 42–45 (2008).

[3] M. W. Ray, E. Ruokokoski, K. Tiurev, M. Möttönen, and D. S. Hall, "Observation of isolated monopoles in a quantum field", Science **348**, 544–547 (2015).

[4] M. V. Berry, "Quantal phase factors accompanying adiabatic changes", Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences **392**, 45–57 (1984).

[5] A. Eriksson and E. Sjöqvist, "Monopole field textures in interacting spin systems", Physical Review A **101** (2020).

[6] K. Krane, D. Halliday, and J. W. Sons, *Introductory nuclear physics* (Wiley, 1988), p. 606.

[7] M. Born and V. Fock, "Beweis des adiabatensatzes", Zeitschrift für Physik **51**, 165–180 (1928).

[8] Y. Aharonov and J. Anandan, "Phase change during a cyclic quantum evolution", Phys. Rev. Lett. **58**, 1593–1596 (1987).

[9] E. Sjöqvist, "Geometric phases in quantum information", International Journal of Quantum Chemistry **115**, 1311–1326 (2015).

[10] X. X. Yi and E. Sjöqvist, "Effect of intersubsystem coupling on the geometric phase in a bipartite system", Phys. Rev. A **70**, 042104 (2004).

[11] J. J. Sakurai and J. Napolitano, *Modern quantum mechanics*, 3rd ed. (Cambridge University Press, 2020).

[12] M. Born and R. Oppenheimer, "Zur quantentheorie der molekeln", Annalen der Physik **389**, 457–484 (1927).

[13] M. V. Berry and R Lim, "The born-oppenheimer electric gauge force is repulsive near degeneracies", Journal of Physics A: Mathematical and General **23**, L655–L657 (1990).

[14] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: the art of scientific computing*, 3rd ed. (Cambridge University Press, USA, 2007), p. 230.

[15] *Uppsala multidisciplinary center for advanced computational science*, (2022) `https://www.uppmax.uu.se/`.

A Code

Below follow the scripts used for numerical simulation of the described dumbbell system, all written in Python. The code is divided into three files: one containing the tools for constructing the external magnetic fields used, one containing the tools used to integrate the dynamics within some such field and one main script for setting parameters and calling the necessary functions from the two other modules. To execute properly all three .py - files must be placed in the same folder additionally containing a subfolder named ”saves”, and that subfolder must in turn contain three subfolders ”fields”, ”odesols” and ”graphs”.

A.1 magfield.py

```
import pandas as pd
import numpy as np
from scipy.constants import mu_0 as mu
from scipy.constants import pi as pi
from matplotlib import pyplot as plt

def simplewire(nr, lablength, I, overwrite=False):
    ##Returns a placeholder field corresponding to a current I through a
        wire along the
    ##x-axis. Saves the field and won't generate a preexisting field unless
    ##'overwrite=True' is called. Takes nr as the number of points along
        each axis.

    #Initiate variables:
    stepr = lablength/(nr-1) #Length of each lattice site
    generate = False #Whether the field needs to be generated
    wire = np.array([0,0]) #Position in x and y of current

    try: #Try to load pregenerated field
        field = np.load(f'saves/simplewire{I}field{nr},{lablength}.npy')
    except FileNotFoundError:
        generate = True

    #Returned saved field unless not found or overwrite turned on
    if not (generate or overwrite):
        #Return pregenerated field
        print("Loading saved magnetic field")
        return field
    else:
        #Generate field
        print("Generating magnetic field")

        xx, yy, zz = np.mgrid[0:nr, 0:nr, 0:nr]
        distance = [(xx - xx)*stepr, (yy - wire[0])*stepr, (zz - wire[1])*
            stepr]
        #Use Biot-Savart formula to calculate the magnetic field
        Bx, By, Bz = np.zeros([nr, nr, nr]), np.zeros([nr, nr, nr]), np.
            zeros([nr, nr, nr])
        Bx, By, Bz = mu/(4*pi)*2*I*np.cross([1, 0, 0], distance,
                axisb=0, axisc=0)/griddot(distance, distance)
        Br, Bth, Bph = cart_to_sph(np.array([Bx,By,Bz])) #Express as
            spherical coordinates
        field = np.array((Bx, By, Bz, Br, Bth, Bph))

        np.save(f'saves/simplewire{I}field{nr},{lablength}.npy', field)
        return field
```

```
def oppositecoils(nr, lablength, I, overwrite=False):
    ##Returns a field corresponding to two currents I of opposing
        directions through square coils
    ##placed orthogonally to the z-axis centred 1/4th from the edges of the
         lattice. Saves the field and won't generate a preexisting field
        unless
    ##'overwrite=True' is called. Takes nr as the number of points along
        each axis.

    #Initiate variables:
    stepr = lablength/(nr-1) #Length of each lattice site
    generate = False #Whether the field needs to be generated

    #Positions of all flowing currents below:
    qindex = int(nr/3)
    wire1 = np.array([-qindex,-qindex]) #Current in positive x close to z=0
         and y=0 (pos in y,z)
    wire2 = np.array([-qindex,nr-1+qindex]) #Current in positive y close to
         z=0 and far from x=0 (pos in z,x)
    wire3 = np.array([nr-1+qindex,-qindex]) #Current in negative x close to
         z=0 and far from y=0 (pos in y,z)
    wire4 = np.array([-qindex,-qindex]) #Current in negative y close to z=0
         and close to x=0 (pos in z,x)
    wire5 = np.array([-qindex,nr-1+qindex]) #Current in negative x far from
         z=0 and close to y=0 (pos in y,z)
    wire6 = np.array([nr-1+qindex,nr-1+qindex]) #Current in negative y far
        from z=0 and x=0 (pos in z,x)
    wire7 = np.array([nr-1+qindex,nr-1+qindex]) #Current in positive x far
        from z=0 and y=0 (pos in y,z)
    wire8 = np.array([nr-1+qindex,-qindex]) #Current in positive y far from
         z=0 and close to x=0 (pos in z,x)

    try: #Try to load pregenerated field
        field = np.load(f'saves/fields/oppositecoils{I}field{nr},{lablength
            }.npy')
    except FileNotFoundError:
        generate = True

    #Returned saved field unless not found or overwrite turned on
    if not (generate or overwrite):
        #Return pregenerated field
        print("Loading saved magnetic field")
        return field
    else:
        #Generate field
        print("Generating magnetic field")

        xx, yy, zz = np.mgrid[0:nr, 0:nr, 0:nr]
        #Integrate the field per Biot-Savart along all currents
        Bx, By, Bz = np.zeros([nr, nr, nr]), np.zeros([nr, nr, nr]), np.
            zeros([nr, nr, nr])

        #Currents along x:
        for px in range(-qindex, nr+qindex): #Maybe adjust with a +1
            dx = [stepr, 0, 0]
            #wire1
```

```
        distance = [(xx-px)*stepr, (yy-wire1[0])*stepr, (zz-wire1[1])*
            stepr]
        dBx1, dBy1, dBz1 = (mu*I/(4*pi) * np.cross(dx, distance, axisb
            =0,
                                axisc=0)/(griddot(distance, distance)
                                    **(3/2)))
        Bx += dBx1
        By += dBy1
        Bz += dBz1

        #wire3
        distance = [(xx-px)*stepr, (yy-wire3[0])*stepr, (zz-wire3[1])*
            stepr]
        dBx3, dBy3, dBz3 = -(mu*I/(4*pi) * np.cross(dx, distance, axisb
            =0,
                                axisc=0)/(griddot(distance, distance)
                                    **(3/2)))
        Bx += dBx3
        By += dBy3
        Bz += dBz3

        #wire5
        distance = [(xx-px)*stepr, (yy-wire5[0])*stepr, (zz-wire5[1])*
            stepr]
        dBx5, dBy5, dBz5 = -(mu*I/(4*pi) * np.cross(dx, distance, axisb
            =0,
                                axisc=0)/(griddot(distance, distance)
                                    **(3/2)))
        Bx += dBx5
        By += dBy5
        Bz += dBz5

        #wire7
        distance = [(xx-px)*stepr, (yy-wire7[0])*stepr, (zz-wire7[1])*
            stepr]
        dBx7, dBy7, dBz7 = (mu*I/(4*pi) * np.cross(dx, distance, axisb
            =0,
                                axisc=0)/(griddot(distance, distance)
                                    **(3/2)))
        Bx += dBx7
        By += dBy7
        Bz += dBz7

    #Currents along y:
    for py in range(-qindex, nr+qindex): #Maybe adjust with a +1
        dy = [0, stepr, 0]
        #wire2
        distance = [(xx-wire2[1])*stepr, (yy-py)*stepr, (zz-wire2[0])*
            stepr]
        dBx2, dBy2, dBz2 = (mu*I/(4*pi) * np.cross(dy, distance, axisb
            =0,
                                axisc=0)/(griddot(distance, distance)
                                    **(3/2)))
        Bx += dBx2
        By += dBy2
        Bz += dBz2

        #wire4
```

```
                distance = [(xx-wire4[1])*stepr, (yy-py)*stepr, (zz-wire4[0])*
                    stepr]
                dBx4, dBy4, dBz4 = -(mu*I/(4*pi) * np.cross(dy, distance, axisb
                    =0,
                                    axisc=0)/(griddot(distance, distance)
                                        **(3/2)))
                Bx += dBx4
                By += dBy4
                Bz += dBz4

                #wire6
                distance = [(xx-wire6[1])*stepr, (yy-py)*stepr, (zz-wire6[0])*
                    stepr]
                dBx6, dBy6, dBz6 = -(mu*I/(4*pi) * np.cross(dy, distance, axisb
                    =0,
                                    axisc=0)/(griddot(distance, distance)
                                        **(3/2)))
                Bx += dBx6
                By += dBy6
                Bz += dBz6

                #wire8
                distance = [(xx-wire8[1])*stepr, (yy-py)*stepr, (zz-wire8[0])*
                    stepr]
                dBx8, dBy8, dBz8 = (mu*I/(4*pi) * np.cross(dy, distance, axisb
                    =0,
                                    axisc=0)/(griddot(distance, distance)
                                        **(3/2)))
                Bx += dBx8
                By += dBy8
                Bz += dBz8

            Br, Bth, Bph = cart_to_sph(np.array([Bx,By,Bz])) #Express as
                spherical coordinates
            field = np.array((Bx, By, Bz, Br, Bth, Bph))

            np.save(f'saves/fields/oppositecoils{I}field{nr},{lablength}.npy',
                field)
            return field

def griddot(a, b):
    ##Returns the dot product for each point in the supplied grids a, b.
        Contracts the
    ##first dimension.
    result = np.zeros(a[0].shape) + 1e-20 #Super ugly bodge to fix NaN
    for i in range(len(a)):
        result += a[i]*b[i]

    result = np.where(result == 0, 1e-20, result)
    return result

def cart_to_sph(cart):
    ##Returns spherical coordinates of the form(r, polar, azimuthal) for
        the given cartesian
    ##coordinates of the form (x,y,z) takes an array with coords as the
        first dimension.
    sph = np.zeros(cart.shape) #Initialize array
    xsqysq = cart[0]**2 + cart[1]**2 #Value of x^2 + y^2
```

```
    sph[0] = np.sqrt(xsqysq + cart[2]**2) #Radius r
    sph[1] = np.arctan2(np.sqrt(xsqysq), cart[2]) #Polar angle theta
    sph[2] = np.arctan2(cart[1],cart[0]) #Azimuthal angle phi
    return sph
```

A.2 synfieldtools.py

```
import pandas as pd
import numpy as np
import scipy
from matplotlib import pyplot as plt
from matplotlib.patches import FancyArrowPatch
from mpl_toolkits import mplot3d
from scipy.constants import hbar
from scipy.integrate import solve_ivp
import warnings

##This is a collection of scripts related to the finding of eigenstates and
    thence the
##derivation of the synthetic fields.

#Common variables defined below, are adjusted by synfieldsolver.py:
ntheta = 25 #Number of points of theta
steptheta = np.pi/(ntheta-1) #Step size of theta
nphi = 50 #Number of points of phi
stepphi = 2*np.pi/(nphi) #Step size of phi
lablength = 1e-3 #Side length of environment cube in meters
tmax = 1 #Maximum simulated time in seconds

##In general x,y,z-coordinates are gives as index numbers, angles in
    radians

J = 1e9 #Spin-spin coupling strength
Gamma = 1e9 #Spin-field coupling strength

mass0 = 3.58e-25 #The total mass of the dumbbell in kg, as a placeholder
    this is the mass of two
                #silver atoms
len0 = 5e-10 #The distance between dumbbell edges in m
mass = np.repeat(mass0, 5) #Full mass vector
mass[3] = mass0*len0**2/4
mass[4] = mass[3]

#Define statistics variables
dscalaravg=0
Faavg=0
denergyavg=0
energyavg=0


##Returns the differentiated Hamiltonian w.r.t. the specified par. diffpar=
    r, th_r, ph_r at
##the point given in point=(x, y, z, th_r, ph_r ) in the external field
    field.
##If diffpar=r a list of matrices for derivatives w.r.t. x, y, z are
    returned
diffhamsave = {}
def diffhamiltonian(diffpar, point, field):
```

```
#Check if the differentiated Hamiltonian has already been calculated
    here
global diffhamsave
if (diffpar, point) in diffhamsave:
    #print('diffhamsave used')
    return diffhamsave[(diffpar, point)]

#Find nr and stepr
nr = field.shape[1] #Number of points along each axis of the lattice
stepr = lablength/(nr-1) #Distance between lattice points
#Select derivative to return
if(diffpar == "r"):

    #Initialize return matrices
    diff_hamx = np.zeros([3, 3], dtype='complex_')
    diff_hamy = np.zeros([3, 3], dtype='complex_')
    diff_hamz = np.zeros([3, 3], dtype='complex_')

        #First find coordinate values of all neighbouring sites
    pointx = int(point[0]) #Get point index (this is needed for dtype
        purposes)
    pointy = int(point[1])
    pointz = int(point[2])

    neighgrid = np.mgrid[-1:2,-1:2,-1:2] ##Meshgrid to generate
        neighbours
    neighgrid = np.array([pointx, pointy, pointz])[:,None,None,None] +
        neighgrid ##Uses broadcasting to duplicate
        #x,y,z into each point on the grid. The result has first
            dimension determining which
        #coordinate is given and the remaining specifying position
            related to the point

        #Find neighbouring r_B values
    Bgrid = np.zeros([3,3,3])
    Bgrid[1,1,1] = field[3, neighgrid[0,1,1,1], neighgrid[1,1,1,1],
        neighgrid[2,1,1,1]]
    Bgrid[2,1,1] = field[3, neighgrid[0,2,1,1], neighgrid[1,2,1,1],
        neighgrid[2,2,1,1]]
    Bgrid[0,1,1] = field[3, neighgrid[0,0,1,1], neighgrid[1,0,1,1],
        neighgrid[2,0,1,1]]
    Bgrid[1,2,1] = field[3, neighgrid[0,1,2,1], neighgrid[1,1,2,1],
        neighgrid[2,1,2,1]]
    Bgrid[1,0,1] = field[3, neighgrid[0,1,0,1], neighgrid[1,1,0,1],
        neighgrid[2,1,0,1]]
    Bgrid[1,1,2] = field[3, neighgrid[0,1,1,2], neighgrid[1,1,1,2],
        neighgrid[2,1,1,2]]
    Bgrid[1,1,0] = field[3, neighgrid[0,1,1,0], neighgrid[1,1,1,0],
        neighgrid[2,1,1,0]]
    B = Bgrid[1,1,1] #Magnetic field strength at point

        #Find neighbouring theta_B values
    thetagrid = np.zeros([3,3,3])
    thetagrid[1,1,1] = field[4, neighgrid[0,1,1,1], neighgrid[1,1,1,1],
         neighgrid[2,1,1,1]]
    thetagrid[2,1,1] = field[4, neighgrid[0,2,1,1], neighgrid[1,2,1,1],
         neighgrid[2,2,1,1]]
    thetagrid[0,1,1] = field[4, neighgrid[0,0,1,1], neighgrid[1,0,1,1],
```

```
    neighgrid[2,0,1,1]]
thetagrid[1,2,1] = field[4, neighgrid[0,1,2,1], neighgrid[1,1,2,1],
    neighgrid[2,1,2,1]]
thetagrid[1,0,1] = field[4, neighgrid[0,1,0,1], neighgrid[1,1,0,1],
    neighgrid[2,1,0,1]]
thetagrid[1,1,2] = field[4, neighgrid[0,1,1,2], neighgrid[1,1,1,2],
    neighgrid[2,1,1,2]]
thetagrid[1,1,0] = field[4, neighgrid[0,1,1,0], neighgrid[1,1,1,0],
    neighgrid[2,1,1,0]]
theta = thetagrid[1,1,1] #Value of theta_B at point

    #Find neighbouring phi_B values
phigrid = np.zeros([3,3,3])
phigrid[1,1,1] = field[5, neighgrid[0,1,1,1], neighgrid[1,1,1,1],
   neighgrid[2,1,1,1]]
phigrid[2,1,1] = field[5, neighgrid[0,2,1,1], neighgrid[1,2,1,1],
   neighgrid[2,2,1,1]]
phigrid[0,1,1] = field[5, neighgrid[0,0,1,1], neighgrid[1,0,1,1],
   neighgrid[2,0,1,1]]
phigrid[1,2,1] = field[5, neighgrid[0,1,2,1], neighgrid[1,1,2,1],
   neighgrid[2,1,2,1]]
phigrid[1,0,1] = field[5, neighgrid[0,1,0,1], neighgrid[1,1,0,1],
   neighgrid[2,1,0,1]]
phigrid[1,1,2] = field[5, neighgrid[0,1,1,2], neighgrid[1,1,1,2],
   neighgrid[2,1,1,2]]
phigrid[1,1,0] = field[5, neighgrid[0,1,1,0], neighgrid[1,1,1,0],
   neighgrid[2,1,1,0]]
phi = phigrid[1,1,1] #Value of phi_B at point

#Approximate the derivatives of B
dBdr = np.zeros(3)
dBdr[0] = 0.5*(Bgrid[2,1,1]-Bgrid[0,1,1])/stepr
dBdr[1] = 0.5*(Bgrid[1,2,1]-Bgrid[1,0,1])/stepr
dBdr[2] = 0.5*(Bgrid[1,1,2]-Bgrid[1,1,0])/stepr

#Approximate the derivatives of theta_B
dthdr = np.zeros(3)
dthdr[0] = 0.5*(thetagrid[2,1,1]-thetagrid[0,1,1])/stepr
dthdr[1] = 0.5*(thetagrid[1,2,1]-thetagrid[1,0,1])/stepr
dthdr[2] = 0.5*(thetagrid[1,1,2]-thetagrid[1,1,0])/stepr

#Approximate the derivatives of phi_B
dphdr = np.zeros(3)
dphdr[0] = 0.5*(phigrid[2,1,1]-phigrid[0,1,1])/stepr
dphdr[1] = 0.5*(phigrid[1,2,1]-phigrid[1,0,1])/stepr
dphdr[2] = 0.5*(phigrid[1,1,2]-phigrid[1,1,0])/stepr

#Assign matrix elements
sin = np.sin(theta) #Calculate the sine
cos = np.cos(theta) #Calculate the cosine
omega = (- dBdr*sin + 1j*B*dphdr*sin - B*dthdr*cos)*np.exp(-1j*phi)
   /np.sqrt(2) #Calculates offdiagonals
diff_hamx[0,0], diff_hamy[0,0], diff_hamz[0,0] = (dBdr*cos - B*
   dthdr*sin)
diff_hamx[0,1], diff_hamy[0,1], diff_hamz[0,1] = omega
diff_hamx[1,0], diff_hamy[1,0], diff_hamz[1,0] = np.conjugate(omega
   )
diff_hamx[1,2], diff_hamy[1,2], diff_hamz[1,2] = omega
```

```
        diff_hamx[2,1], diff_hamy[2,1], diff_hamz[2,1] = np.conjugate(omega
            )
        diff_hamx[2,2], diff_hamy[2,2], diff_hamz[2,2] = (-dBdr*cos + B*
            dthdr*sin)

        #Return the matrices with correct prefactors
        diff_hamx = Gamma*hbar*diff_hamx
        diff_hamy = Gamma*hbar*diff_hamy
        diff_hamz = Gamma*hbar*diff_hamz

        diffhamsave[(diffpar, point)] = diff_hamx, diff_hamy, diff_hamz #
            Save the result

        return diff_hamx, diff_hamy, diff_hamz

    elif(diffpar=="th_r"):

        diff_ham  = np.zeros([3,3], dtype='complex_') #Initialize return
            matrix

        #Assign matrix elements
        sin = np.sin(2*point[3]) #Calculate the sine of twice theta_r
        cos = np.cos(2*point[3]) #Calculate the cosine of twice theta_r
        exp = np.exp(1j*point[4]) #Calculate the complex exponential of
            phi_r
        sq = np.sqrt(2) #Calculate the square root of two used
        omega = sq*exp*cos # Calculate an often occuring element

        diff_ham[0,0] = -sin
        diff_ham[0,1] = -1*np.conjugate(omega)
        diff_ham[0,2] = sin/exp**2
        diff_ham[1,0] = -1*omega
        diff_ham[1,1] = 2*sin
        diff_ham[1,2] = np.conjugate(omega)
        diff_ham[2,0] = exp**2*sin
        diff_ham[2,1] = omega
        diff_ham[2,2] = -sin

        #Return the matrix with correct prefactors
        diff_ham = J*hbar*diff_ham

        diffhamsave[(diffpar, point)] = diff_ham #Save the result

        return diff_ham

    elif(diffpar=="ph_r"):

        diff_ham  = np.zeros([3,3], dtype='complex_') #Initialize return
            matrix

        #Assign matrix elements
        sin = np.sin(2*point[3]) #Calculate the sine of twice theta_r
        sin2 = np.sin(point[3])**2 #Calculate the square of sine of theta_r
        exp = np.exp(1j*point[4]) #Calculate the complex exponential of
            phi_r
        sq = np.sqrt(2) #Calculate the square root of two used
        omega = 1j*exp*sin/sq # Calculate an often occuring element
```

```
            diff_ham[0,1] = -np.conjugate(omega)
            diff_ham[0,2] = -2j*sin2/exp**2
            diff_ham[1,0] = -omega
            diff_ham[1,2] = np.conjugate(omega)
            diff_ham[2,0] = 2j*sin2*exp**2
            diff_ham[2,1] = omega

            #Return the matrix with correct prefactors
            diff_ham = J*hbar*diff_ham

            diffhamsave[(diffpar, point)] = diff_ham #Save the result

            return diff_ham

        else:
            raise ValueError("Invalid string for differentiation coordinate
                passed to diffhamiltonian")


##Solves the eigenvalue problem of the fast Hamiltonian at point point for
    field field.
##The point is taken to be of shape (x, y, z, theta_r, phi_r).
##Returns the energies and eigenvectors in pairs with ascending energies.
    The vectors are
##normalized column vectors in the singlet-triplet basis.
def eigensolver(point, field):

    warnings.filterwarnings("error")

    #First calculate the fast Hamiltonian at point
    ham = np.zeros([3,3], dtype='complex_') #Initialize empty matrix
    pointx = int(point[0]) #Get point index (this is needed for dtype
        purposes)
    pointy = int(point[1])
    pointz = int(point[2])

    #Find the values of B, theta_B and phi_B at point
    B = field[3, pointx, pointy, pointz]
    thetaB = field[4, pointx, pointy, pointz]
    phiB = field[5, pointx, pointy, pointz]
    xi = J/(Gamma*B)
    if xi == np.inf or np.isnan(xi):
        print(f'External field is zero at point {point}! Please correct the
            field or the streams')

    #Extract theta_r and phi_r at point
    thetar = point[3]
    phir = point[4]

    #Assign matrix elements
    cosr = np.cos(thetar) #Calculate the cosine of theta_r
    sin2r = np.sin(2*thetar) #Calculate the sine of twice theta_r
    cos2r = np.cos(2*thetar) #Calculate the cosine of twice theta_r
    sinrsq = np.sin(thetar)**2 #Calculate the square of sine of theta_r
    cosrsq = np.cos(thetar)**2 #Calculate the square of cosine of theta_r
    cosB = np.cos(thetaB) #Calculate the cosine of theta_B
    sinB = np.sin(thetaB) #Calculate the sine of theta_B
    expr = np.exp(1j*phir) #Calculate the complex exponential of phi_r
```

```
    expB = np.exp(1j*phiB) #Calculate the complex exponential of phi_B
    sq = np.sqrt(2) #Calculate the square root of two often used

    ham[0,0] = xi*cosrsq + cosB
    ham[0,1] = -sinB / (expB*sq) - xi*sin2r/(expr*sq)
    ham[0,2] = xi*sinrsq/expr**2
    ham[1,0] = -expB*sinB/sq - xi*expr*sin2r/sq
    ham[1,1] = -xi*cos2r
    ham[1,2] = -sinB/(expB*sq) + xi*sin2r/(expr*sq)
    ham[2,0] = xi*expr**2*sinrsq
    ham[2,1] = -expB*sinB/sq + xi*expr*sin2r/sq
    ham[2,2] = xi*cosrsq-cosB

    #Fix matrix prefactors
    ham = Gamma*B*hbar*ham

    #Calculate eigenvalues and eigenvectors
    eigenvalues, eigenvectors = scipy.linalg.eigh(ham)

    #Return the result
    return eigenvalues, eigenvectors

##Calculates the synthetic scalar field at point point for field field for
    state number n.
##The point is taken to be of shape (x, y, z, theta_r, phi_r).
##Returns a tuple of the scalar field value followed by the fast energy.
scalarsave = {}
def scalarcalc(point, field, n):

    point = tuple(point)

    #First check if the scalar field has been calculated here before
    global scalarsave
    if (point, n) in scalarsave:
        #print('scalarsave used')
        return scalarsave[(point, n)]

    #Fix point format
    point = tuple(point)

    #First retrieve the energies and eigenstates
    energies, eigvec = eigensolver(point, field)

    #Differentiate the Hamiltonian w.r.t. each coordinate
    dHam = [0,0,0,0,0]
    dHam[0], dHam[1], dHam[2] = diffhamiltonian('r', point, field)
    dHam[3] = diffhamiltonian('th_r', point, field)
    dHam[4] = diffhamiltonian('ph_r', point, field)

    Phi = 0 #Initialize synthetic scalar

    for i in range(5):
        for l in range(3):
            if not l == n: #Remove diagonals
                braket = np.vdot(eigvec[n], np.dot(dHam[i], eigvec[l])) #
                    Braket for formula
                Phi += (hbar**2 /(2*mass[i]) * #Add up contributions to the
                     synthetic scalar
```

```
                braket*np.conjugate(braket) / (energies[n] - energies[l])
                    **2).real #Note the discard of the imaginary part,
                    numerical errors otherwise arise

    returnlist = (Phi, energies[n])
    scalarsave[(point, n)] = returnlist

    return returnlist

##Calculates the acceleration due to the synthetic magnetic field and
    summarizes all
##acceleration contributions. This is done for the position pos, the
    velocity vel as a tuple
##with the field field for state number n. The position and velocity is
    taken to be of
##shape (x, y, z, theta_r, phi_r). Note that position is here given in m,
    and will be
##fitted to the discrete lattice.
##Returns the velocity in m/s (for integration purposes) followed by the
    acceleration of the
##system in m/s^2.
##Note that the t argument is a dummy.
def acc(t, posvel, field, n, norot, nosyn):
    #Find nr and stepr
    nr = field.shape[1] #Number of points along each axis of the lattice
    stepr = lablength/(nr-1) #Distance between lattice points
    #Extract pos and vel:
    pos = posvel[0:5]
    vel = posvel[5:10]

    #Initialize forces
    Fa = np.zeros(5)
    dscalar = np.zeros(5)
    denergy = np.zeros(5)

    #Fit position to a point
    point = [0,0,0,0,0]
    for i in range(3):
        point[i] = int(round(pos[i]/stepr))
        if point[i] >= nr-2 or point[i] < 2:
            return np.concatenate((vel, np.zeros(5))) ##Sets the
                acceleration to zero if a
                                                      ##point outside the
                                                          grid is sampled
    point[3] = pos[3]
    point[4] = pos[4]

    point = tuple(point)

    if not nosyn == "True":
    ##Make sure the B-field is nonzero
        pointx = int(point[0]) #Get point index (this is needed for dtype
            purposes)
        pointy = int(point[1])
        pointz = int(point[2])

        #Find the values of B at point
        B = field[3, pointx, pointy, pointz]
```

```
    if B == 0.0:
        print(f'Warning, external field of zero encountered at {pos}')
        return np.zeros(10) #Freeze stream

if nosyn == "False" or nosyn == "Noscalar":
    #Find energies and eigenstates at point
    energies, eigvec = eigensolver(point, field)

    #Differentiate the Hamiltonian w.r.t. each coordinate
    dHam = [0,0,0,0,0]
    dHam[0], dHam[1], dHam[2] = diffhamiltonian('r', point, field)
    dHam[3] = diffhamiltonian('th_r', point, field)
    dHam[4] = diffhamiltonian('ph_r', point, field)

    #Calculate the acceleration due to the syn. magnetic field
    for i in range(5):
        for j in range(5):
            if not j == i: #Remove diagonals
                for l in range(3):
                    if not l == n: #Remove diagonals
                        if energies[n] == energies[l]:
                            print(f'Degenerate fast eigenvalues {
                                energies[n]} and {energies[l]}!')
                        Fa[i] += (-2*hbar * vel[j] / (energies[n]-
                            energies[l])**2 *
                                np.imag(np.vdot(eigvec[n], np.dot(dHam[
                                    i], eigvec[l])) *
                                np.vdot(eigvec[l], np.dot(dHam[j],
                                    eigvec[n]))))

    global Faavg
    Faavg = (Faavg + np.linalg.norm(Fa))/2

if nosyn == "False" or nosyn == "Nomag":

    #To get the derivatives of the scalar fields find coordinate values
         of all neighbouring sites

    ##meshgrid to generate neighbours
    neighgrid = np.mgrid[-1:2,-1:2,-1:2, -1:2, -1:2].astype('float')
    neighgrid[3,:,:,:,:,:] *= steptheta #Fix theta and phi step sizes
    neighgrid[4,:,:,:,:,:] *= stepphi
    neighgrid = np.array(point)[:,None,None,None, None, None] +
        neighgrid
    #uses broadcasting to duplicate
    #x,y,z into each point on the grid. the result has first dimension
        determining which
    #coordinate is given and the remaining specifying position related
        to the point

    #Find neighbouring scalar field values and eigenstate energies
    scalar = np.zeros([3, 3, 3, 3, 3])
    energy = np.zeros([3, 3, 3, 3, 3])
    scalar[2,1,1,1,1], energy[2,1,1,1,1] = scalarcalc(neighgrid
        [:,2,1,1,1,1], field, n)
    scalar[0,1,1,1,1], energy[0,1,1,1,1] = scalarcalc(neighgrid
        [:,0,1,1,1,1], field, n)
    scalar[1,2,1,1,1], energy[1,2,1,1,1] = scalarcalc(neighgrid
```

```
        [:,1,2,1,1,1], field, n)
    scalar[1,0,1,1,1], energy[1,0,1,1,1] = scalarcalc(neighgrid
        [:,1,0,1,1,1], field, n)
    scalar[1,1,2,1,1], energy[1,1,2,1,1] = scalarcalc(neighgrid
        [:,1,1,2,1,1], field, n)
    scalar[1,1,0,1,1], energy[1,1,0,1,1] = scalarcalc(neighgrid
        [:,1,1,0,1,1], field, n)
    scalar[1,1,1,2,1], energy[1,1,1,2,1] = scalarcalc(neighgrid
        [:,1,1,1,2,1], field, n)
    scalar[1,1,1,0,1], energy[1,1,1,0,1] = scalarcalc(neighgrid
        [:,1,1,1,0,1], field, n)
    scalar[1,1,1,1,2], energy[1,1,1,1,2] = scalarcalc(neighgrid
        [:,1,1,1,1,2], field, n)
    scalar[1,1,1,1,0], energy[1,1,1,1,0] = scalarcalc(neighgrid
        [:,1,1,1,1,0], field, n)

    #Differentiate the scalar field
    dscalar[0] = (scalar[2,1,1,1,1] - scalar[0,1,1,1,1])/(2*stepr)
    dscalar[1] = (scalar[1,2,1,1,1] - scalar[1,0,1,1,1])/(2*stepr)
    dscalar[2] = (scalar[1,1,2,1,1] - scalar[1,1,0,1,1])/(2*stepr)
    dscalar[3] = (scalar[1,1,1,2,1] - scalar[1,1,1,0,1])/(2*steptheta)
    dscalar[4] = (scalar[1,1,1,1,2] - scalar[1,1,1,1,0])/(2*stepphi)

    global dscalaravg
    dscalaravg = (dscalaravg + np.linalg.norm(dscalar))/2

elif nosyn == "True" or nosyn == "Noscalar":
    #To get the derivatives of the energies find coordinate values of
        all neighbouring sites

    ##meshgrid to generate neighbours
    neighgrid = np.mgrid[-1:2,-1:2,-1:2, -1:2, -1:2].astype('float')
    neighgrid[3,:,:,:,:,:] *= steptheta #Fix theta and phi step sizes
    neighgrid[4,:,:,:,:,:] *= stepphi
    neighgrid = np.array(point)[:,None,None,None, None, None] +
        neighgrid
    #uses broadcasting to duplicate
    #x,y,z into each point on the grid. the result has first dimension
        determining which
    #coordinate is given and the remaining specifying position related
        to the point

    #Find energies at neighbouring points
    energy = np.zeros([3, 3, 3, 3, 3])
    energy[2,1,1,1,1] = eigensolver(neighgrid[:,2,1,1,1,1], field)[0][n
        ]
    energy[0,1,1,1,1] = eigensolver(neighgrid[:,0,1,1,1,1], field)[0][n
        ]
    energy[1,2,1,1,1] = eigensolver(neighgrid[:,1,2,1,1,1], field)[0][n
        ]
    energy[1,0,1,1,1] = eigensolver(neighgrid[:,1,0,1,1,1], field)[0][n
        ]
    energy[1,1,2,1,1] = eigensolver(neighgrid[:,1,1,2,1,1], field)[0][n
        ]
    energy[1,1,0,1,1] = eigensolver(neighgrid[:,1,1,0,1,1], field)[0][n
        ]
    energy[1,1,1,2,1] = eigensolver(neighgrid[:,1,1,1,2,1], field)[0][n
        ]
```

```
            energy[1,1,1,0,1] = eigensolver(neighgrid[:,1,1,1,0,1], field)[0][n
                ]
            energy[1,1,1,1,2] = eigensolver(neighgrid[:,1,1,1,1,2], field)[0][n
                ]
            energy[1,1,1,1,0] = eigensolver(neighgrid[:,1,1,1,1,0], field)[0][n
                ]
            energy[1,1,1,1,1] = eigensolver(neighgrid[:,1,1,1,1,1], field)[0][n
                ]

        else:
            print(f'Incorrect parameter "nosyn" = {nosyn}')

        global energyavg
        energyavg = (energyavg + energy[1,1,1,1,1])/2

        #Differentiate the energies
        denergy[0] = (energy[2,1,1,1,1] - energy[0,1,1,1,1])/(2*stepr)
        denergy[1] = (energy[1,2,1,1,1] - energy[1,0,1,1,1])/(2*stepr)
        denergy[2] = (energy[1,1,2,1,1] - energy[1,1,0,1,1])/(2*stepr)
        denergy[3] = (energy[1,1,1,2,1] - energy[1,1,1,0,1])/(2*steptheta)
        denergy[4] = (energy[1,1,1,1,2] - energy[1,1,1,1,0])/(2*stepphi)

        acc = Fa - dscalar - denergy #Summarize forces

        global denergyavg
        denergyavg = (denergyavg + np.linalg.norm(denergy))/2

        for i in range(5):
            acc[i] = acc[i]/mass[i] #Divide by mass to get acceleration

        if norot: #Freeze rotational axes if norot is turned on
            vel[3:5] = 0
            acc[3:5] = 0

        if acc[3] > 1e10:
            print(f'The rotation is out of control, acc = {acc}')

        return np.concatenate((vel, acc))

    ##Solves the ODE and returns the solution as per sscipy.integrate.solve_ivp
        .
    ##The external magnetic field is given as field. The
    ##dumbbell is placed initially at position pos and with velocity vel of the
         shape (x, y,
    ##z, theta_r, phi_r). Note that cartesian position here is in m.
    ##The spin subsystem is assumed to remain in the fast
    ##eigenstate labeled n. Runs until the time reaches tmax.
    ##The given initial conditions must be tuples.
    def solvedyn(pos, vel, field, n, norot=False, nosyn='False'):
        posvel = pos + vel

        #Reset average force counters
        global Faavg
        global dscalaravg
        global denergyavg
        global energyavg
        Faavg = 0
        dscalaravg = 0
```

```
    denergyavg = 0
    energyavg = 0

    #Find times to require ODE evaluation
    t_eval = np.linspace(0, tmax, 100000)

    #Set error tolerances
    nr = field.shape[1]
    tolr = lablength/(2*(nr - 1))
    toltheta = steptheta/2
    tolphi = stepphi/2
    atol = [tolr, tolr, tolr, toltheta, tolphi, tolr/10, tolr/10, tolr/10,
        toltheta/10,
            tolphi/10]
    edgedistance.terminal = True
    sol = solve_ivp(acc, (0,tmax), posvel, events=edgedistance, args=(field
        , n, norot, nosyn), atol=atol, t_eval=t_eval)

    sol.Faavg = Faavg
    sol.denergyavg = denergyavg
    sol.dscalaravg = dscalaravg
    sol.energyavg = energyavg

    return sol

##Event to terminate integration, returns distance to the closest edge
    minus a small
##correction to avoid hitting the edge
def edgedistance(t, posvel, field, n, norot, nosyn):
    pos= posvel[0:3]
    mindist = np.amin(pos)
    maxdist = np.amax(pos)
    distancetoedge = min(mindist, lablength - maxdist)

    #Find nr and stepr
    nr = field.shape[1] #Number of points along each axis of the lattice
    stepr = lablength/(nr-1) #Distance between lattice points

    return distancetoedge - 2*stepr


##Plotting function, takes a list of solutions sol from solve_ivp, a field
    field and displays an
##interactive 3D swarm plot. Uses matplotlib.
def lineplot(sol, field, I, initvel, swarmnum, n, norot, nosyn,
    alternatestreams):

    #Print average acceleration components
    for stream in sol:
        print(f'Statistics for stream starting at {stream.y[0:3,0]}')
        print(f'Faavg = {stream.Faavg}')
        print(f'dscalaravg = {stream.dscalaravg}')
        print(f'denergyavg = {stream.denergyavg}')
        print(f'energyavg = {stream.energyavg}')

    fig, ax = plt.subplots(subplot_kw={'projection': '3d'})

    ax.set_xlim((0, lablength*1000))
```

```
    ax.set_ylim((0, lablength*1000))
    ax.set_zlim((0, lablength*1000))
    ax.set_xlabel('x (mm)', fontsize=10, color='blue')
    ax.set_ylabel('y (mm)', fontsize=10, color='blue')
    ax.set_zlabel('z (mm)', fontsize=10, color='blue')

    #Find nr and stepr
    nr = field.shape[1] #Number of points along each axis of the lattice
    stepr = lablength/(nr-1) #Distance between lattice points
    #Create grids for the quiver:
    xx, yy, zz = stepr*np.mgrid[0:nr, 0:nr, 0:nr]
    xx = xx[0::int(nr/5), 0::int(nr/5), 0::int(nr/5)]
    yy = yy[0::int(nr/5), 0::int(nr/5), 0::int(nr/5)]
    zz = zz[0::int(nr/5), 0::int(nr/5), 0::int(nr/5)]
    Bx = field[0,:,:,:][0::int(nr/5), 0::int(nr/5), 0::int(nr/5)]
    By = field[1,:,:,:][0::int(nr/5), 0::int(nr/5), 0::int(nr/5)]
    Bz = field[2,:,:,:][0::int(nr/5), 0::int(nr/5), 0::int(nr/5)]


    for stream in sol:
    #Extract pos
        #Set stream colour
        try:
            color = stream.color
        except AttributeError:
            color = 'red'
        pos = stream.y[0:3,:]
        pos = pos*1000 #Plot in mm
        ax.plot3D(pos[0,:], pos[1,:], pos[2,:], color=color) #Plot the
            integrated path
    #Plot magnetic field for testing purposes
    #ax.quiver(xx, yy, zz, Bx, By, Bz, length=0.0001, normalize=True)

    plt.savefig(f'saves/graphs/I{I}nr{nr}lablength{lablength}tmax{tmax}J{J}
        Gamma{Gamma}mass{mass0}len{len0}n{n}vel{initvel}swarmnum{swarmnum}
        norot{norot}nosyn{nosyn}altstream{alternatestreams}.png',
                bbox_inches='tight')
    plt.show()
```

A.3 synfieldsolver.py

```
import numpy as np
import magfield as mg
import synfieldtools as sn
import pickle
from numpy import pi as pi

if __name__ == '__main__':
    #Set field and modes
    availablefieldtypes = ['simplewire', 'oppositecoils']
    fieldtype = 'oppositecoils'
    I = 10 #Current parameter for the field
    norot = False #Whether to ignore rotational degrees of freedom
    nosyn = 'False' #Whether to ignore synthetic fields, accepts the
        strings 'False',
                    #'True', 'Nomag' and 'Noscalar'
    overwriteresult = False #Whether to overwrite previous ODE results
    alternatestreams = False #Whether to use alternate swarming scheme
```

```
#Define parameters

nr = 101 #Number of points in field lattice
lablength = 1e-3 #Cube side of lab in m
tmax = 0.1 #Trajectory time in s
J = 1e5 #Spin-spin coupling strength
Gamma = 1e10 #Spin-field coupling strength

mass0 = 3.58e-25 #The total mass of the dumbbell in kg, as a
    placeholder this is the mass of
                    #two silver atoms
len0 = 5e-5 #The distance between dumbbell edges in m
mass = np.repeat(mass0, 5) #Full mass vector
mass[3] = mass0*len0**2/4
mass[4] = mass[3]

#Set parameters
sn.nr = nr
sn.lablength = lablength
sn.tmax = tmax
sn.J = J
sn.Gamma = Gamma
sn.mass0 = mass0
sn.len0 = len0
sn.mass = mass

#Set initial position, velocity and the fast eigenstate to consider
step = lablength/nr
initposarray = np.array((10*step, 10*step, 10*step, 0, 0))
swarmnum = 4 #Square root of number of streams in swarm
swarmgrid = np.mgrid[0:lablength-20*step:swarmnum*1j,0:lablength-20*
    step:swarmnum*1j] #Grid to swarm the initial positions
swarmgrid = np.insert(swarmgrid, 2, np.zeros(swarmgrid.shape[1:3]),
    axis=0)
swarmgrid = np.insert(swarmgrid, 2, np.zeros(swarmgrid.shape[1:3]),
    axis=0)
swarmgrid = np.insert(swarmgrid, 0, np.zeros(swarmgrid.shape[1:3]),
    axis=0)
initposarray = initposarray[:,None,None] + swarmgrid
altinitpos = initposarray[:,1,1] #Starting position for alternate
    swarming method
initvel = (1e-2, 0, 0, 0, 0)
eigenstate = 2

try: #Try to load pregenerated result
    with open(f'saves/odesols/resultF{fieldtype}I{I}nr{nr}lablength{
        lablength}tmax{tmax}J{J}Gamma{Gamma}mass{mass0}len{len0}n{
        eigenstate}vel{initvel}swarmnum{swarmnum}norot{norot}nosyn{nosyn
        }altstream{alternatestreams}.bin', 'rb') as file:
        sol = pickle.load(file)
        print(f'Loading result from file resultF{fieldtype}I{I}nr{nr}
            lablength{lablength}tmax{tmax}J{J}Gamma{Gamma}mass{mass0}len
            {len0}n{eigenstate}vel{initvel}norot{norot}nosyn{nosyn}
            altstream{alternatestreams}.bin')
except FileNotFoundError:
    overwriteresult = True

#Generate a field
```

```
if fieldtype == 'simplewire':
    field = mg.simplewire(nr, lablength, I)
if fieldtype == 'oppositecoils':
    field = mg.oppositecoils(nr, lablength, I)

if overwriteresult: #Only perform calculations if result not avaiable
    on save:

    #Integrate paths
    print('Integrating dynamics . . . . Please wait')
    sol = []
    if alternatestreams:
        print(f'Simulating start {tuple(altinitpos)} with synthetic
            fields')
        try:
            stream = sn.solvedyn(tuple(altinitpos[:]), initvel, field,
                eigenstate, norot, 'False')
            stream.color = 'red'
            sol.append(stream)
            print(f'Stream number {len(sol)} has been integrated!')
        except Exception as e:
            print(e)
            print(f'A stream has failed to generate, most probably due
                to crossing the centre')
        print(f'Simulating start {tuple(altinitpos)} without synthetic
            fields')
        try:
            stream = sn.solvedyn(tuple(altinitpos[:]), initvel, field,
                eigenstate, norot, 'True')
            stream.color = 'blue'
            sol.append(stream)
            print(f'Stream number {len(sol)} has been integrated!')
        except Exception as e:
            print(e)
            print(f'A stream has failed to generate, most probably due
                to crossing the centre')
        print(f'Simulating start {tuple(altinitpos)} without the
            magnetic synthetic field')
        try:
            stream = sn.solvedyn(tuple(altinitpos[:]), initvel, field,
                eigenstate, norot, 'Nomag')
            stream.color = 'green'
            sol.append(stream)
            print(f'Stream number {len(sol)} has been integrated!')
        except Exception as e:
            print(e)
            print(f'A stream has failed to generate, most probably due
                to crossing the centre')
        print(f'Simulating start {tuple(altinitpos)} without the scalar
            synthetic field')
        try:
            stream = sn.solvedyn(tuple(altinitpos[:]), initvel, field,
                eigenstate, norot, 'Noscalar')
            stream.color = 'orange'
            sol.append(stream)
            print(f'Stream number {len(sol)} has been integrated!')
        except Exception as e:
            print(e)
```

```
            print(f'A stream has failed to generate, most probably due
                to crossing the centre')

    else:
        for i in range(initposarray.shape[1]):
            for j in range(initposarray.shape[2]):
                print(tuple(initposarray[:,i,j]))
                try:
                    sol.append(sn.solvedyn(tuple(initposarray[:,i,j]),
                        initvel, field, eigenstate, norot, nosyn))
                    print(f'Stream number {len(sol)} has been
                        integrated!')
                except Exception as e:
                    print(e)
                    print(f'A stream has failed to generate, most
                        probably due to crossing the centre')

    #Save the result
    print(f'Saving result to file resultF{fieldtype}I{I}nr{nr}lablength
        {lablength}tmax{tmax}J{J}Gamma{Gamma}mass{mass0}len{len0}n{
        eigenstate}vel{initvel}norot{norot}nosyn{nosyn}altstream{
        alternatestreams}.bin')
    with open(f'saves/odesols/resultF{fieldtype}I{I}nr{nr}lablength{
        lablength}tmax{tmax}J{J}Gamma{Gamma}mass{mass0}len{len0}n{
        eigenstate}vel{initvel}swarmnum{swarmnum}norot{norot}nosyn{nosyn
        }altstream{alternatestreams}.bin', 'wb') as file:
         pickle.dump(sol, file)

else:
    print('Loading previously generated result')
#Extract positions and orientations of the first stream
pos = sol[0].y[0:3,:]
vel = sol[0].y[5:8,:]
ori = sol[0].y[3:5,:]
#Print the result
print('Times sampled:')
print(sol[0].t)
print('Rotation integrated:')
print(ori)

sn.lineplot(sol, field, I, initvel, swarmnum, eigenstate, norot, nosyn,
    alternatestreams)
```