Numerical Analysis of Nonlinear Localized Modes in Vibrational and Magnetic Lattices

Hieu Le
Dickinson College

Follow this and additional works at: https://scholar.dickinson.edu/student_honors

Part of the Physics Commons

Recommended Citation

This Honors Thesis is brought to you for free and open access by Dickinson Scholar. It has been accepted for inclusion by an authorized administrator. For more information, please contact scholar@dickinson.edu.
Numerical Analysis of Nonlinear Localized Modes in Vibrational and Magnetic Lattices

Submitted in partial fulfillment of honors requirements for the Department of Physics and Astronomy, Dickinson College,

by

Hieu Le

Advisor: Professor Lars Q. English
Reader: Professor David Jackson
Reader: Professor Brett Pearson

Carlisle, PA
May 13, 2019
Abstract

In this project, I used various numerical techniques to investigate the existence of nonlinear, spatially localized modes for vibrational and magnetic lattices. I started by using Newton-Raphson algorithm to solve the well-known 1-D Fermi-Pasta-Ulam (FPU) lattice for nonlinear, stable and localized solutions, also known as intrinsic localized modes (ILMs). Using this approach, I found lattice ILMs at the nonlinear frequency just above the minimum nonlinear threshold, then input them into Newton-Raphson to obtain more solutions at higher frequencies via continuation. I then evolved these solutions in time with Runge-Kutta (RK4) algorithm to evaluate their stability.

Afterwards, I turned to the 1-D ferromagnetic and antiferromagnetic lattices, and tried a similar approach to the FPU lattice. Due to the huge increase in complexity of the magnetic lattices and possibly other complications, the Newton-Raphson algorithm failed to converge even when backtracking line search and trust-region constraint variations were implemented. Thus I used the shooting method to solve for ILMs in the lattice, and successfully found ILMs in the antiferromagnetic lattice. Afterwards, I also explored both the formation of nontrivial localized modes in 1-D and 2-D ferromagnetic lattice from small perturbations from the uniform mode and their long term stability.
Acknowledgements

I would like to thank Professor Lars English, who functioned both as the advisor to this project and support throughout my time at Dickinson. Without his guidance and constant encouragement, I don’t think I would have started studying physics in the first place, let alone getting this far in my academic career. Also, I deeply appreciate the entire Dickinson physics department and fellow students, in particular William Boyes, Amanda Baylor, Julia Huddy and Sophie Kirkman for their constant support and camaraderie in the face of hardship (read: graduate school applications and honors project). I am glad that I got to experience the whole four years of college with you all.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Theory</td>
<td>4</td>
</tr>
<tr>
<td>2.1 ILMs</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Algorithms</td>
<td>8</td>
</tr>
<tr>
<td>2.2.1 Newton-Raphson</td>
<td>8</td>
</tr>
<tr>
<td>2.2.2 RK4</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Lattices</td>
<td>14</td>
</tr>
<tr>
<td>2.3.1 Fermi-Pasta-Ulam</td>
<td>14</td>
</tr>
<tr>
<td>2.3.2 Magnetic</td>
<td>16</td>
</tr>
<tr>
<td>3 Numerical Setup</td>
<td>21</td>
</tr>
<tr>
<td>4 Results and Analysis</td>
<td>25</td>
</tr>
<tr>
<td>4.1 Fermi-Pasta-Ulam ILM</td>
<td>25</td>
</tr>
<tr>
<td>4.2 Antiferromagnetic ILM</td>
<td>32</td>
</tr>
<tr>
<td>4.3 Ferromagnetic spontaneous localizations</td>
<td>36</td>
</tr>
<tr>
<td>5 Conclusion</td>
<td>39</td>
</tr>
<tr>
<td>A Appendix</td>
<td>40</td>
</tr>
<tr>
<td>A.1 RK4.py</td>
<td>40</td>
</tr>
<tr>
<td>A.2 lattices.py</td>
<td>41</td>
</tr>
<tr>
<td>A.3 newton.py</td>
<td>48</td>
</tr>
<tr>
<td>A.4 shoot.py</td>
<td>52</td>
</tr>
</tbody>
</table>
List of Figures

1Plane wave mode and localized mode for an antiferromagnetic lattice. 1
2Demonstration of the degeneracy of information outside of the first Brillouin zone. 5
3Dispersion relation for a hard nonlinear system (FPU) and a soft nonlinear system. 6
4Newton-Raphson algorithm. 8
5Examples of Newton-Raphson failing to converge. 9
6RK4 algorithm. 12
7Dispersion relation for the FPU lattice. 16
8Uniform mode for a ferromagnetic lattice. 18
9Dispersion relation for the antiferromagnetic lattice. 20
10Summing over neighboring sites for a lattice with periodic boundary and non-periodic boundary. 22
11Different trajectories for the shooting method. 25
12Even- and odd-parity ILMs for the FPU lattice. 28
13Graph of amplitude at each site vs frequency for even- and odd-parity ILMs. 29
14Even- and odd-parity ILMs for an FPU lattice with only nonlinear interactions. 30
15Time evolution of the even-parity ILM at $\omega = 2.1$ with RK4. 31
16Instability of ILMs over long periods of time due to approximation and numerical errors. 31
17Example of an ILM with nonzero phase. 32
18Even-parity ILM for an antiferromagnetic lattice. 34
19Difference between absolute amplitudes of ILM associated with $s_0 = 0.7$ vs. $s_0 = 0.65$ and $s_0 = 0.6$. 34
20Time evolution of the even-parity antiferromagnetic ILM at $s_0 = 0.7$. 35
21Time evolution of an uniform 1-D ferromagnetic with small excitations. 37
22Time evolution of an uniform 1-D ferromagnetic with small excitations. 38
1 Introduction

Linear systems and linearization of nonlinear systems are well-studied in physics, due to the simplicity and predictability of the problem. However, the majority of systems in nature are nonlinear, and thus warrant close investigations despite their complexity. One such problem that has been gaining attention in condensed matter physics is the dynamics of intrinsic localized modes (ILMs) in perfect lattices (in contrast to stable localized modes in lattices with defects). An ILM is a stable oscillatory localization of a lattice, as opposed to a plane wave/normal mode in which the lattice energy is spread out evenly on each lattice site. An example of an ILM can be seen in Fig. 1. In this thesis, I will use various numerical analysis techniques to investigate ILM behaviors in a vibrational Fermi-Pasta-Ulam (FPU) lattice and magnetic lattices (antiferromagnetic and ferromagnetic).

![Figure 1: (a) Plane wave mode and (b) localized mode for an antiferromagnetic lattice in the $xy$-plane. The position of each vector represents the spin’s actual position in the 1-D lattice, while the length represents the $xy$ projection of the 3-D spin. Note that the dots in (b) are spins that are aligned with the $z$-axis (out of the page). All the energy in the (b) lattice is concentrated in the middle.](image)

The process is as follow: I start with differential equations that describe the dynamics of the system in question (e.g. $m\ddot{x} = -kx$), then use ansatzs, approximations, etc. to reduce the differential equations to a system of algebraic equations. The roots
to these algebraic equations are configurations that are exact numerical solutions (or solutions obtained numerically that are close enough to the real solution), and thus are stable oscillatory modes for this system. For this purpose, I can use root finding algorithms such as Newton-Raphson [1] in order to look for ILM solutions to the system at oscillation frequencies where the nonlinearity of the lattice begins to manifest. After finding one solution, I can use it as input to Newton-Raphson to find more energetic solutions at higher frequencies. Then, I use RK4 to evaluate the solution’s stability over time. All the above algorithms are implemented using Python, mainly with an IPython (in particular Jupyter Notebook) environment due to its extreme versatility, ease of use and the nature of the project as a test trial for the numerical process.

The above process doesn’t specify a particular lattice; indeed, ILMs have great genericity and can appear in many perfect lattice [6–10], as long as the dynamical equations for lattice is nonlinear and the lattice is discrete. The lattice’s discrete structure induces a nonlinear region above the dispersion curve at the Brillouin zone boundary [2]. The lattice nonlinearity gives rise to a relationship between the oscillation frequency and amplitude, which allows the lattice frequency to move to nonlinear regions. Together, the two conditions satisfy the ILM existence criterion: the frequencies at which it is possible to observe ILMs must avoid all linear frequencies on the plane wave spectrum.

In this thesis, I will look for ILMs in the 1-D version of the FPU lattice [15] and magnetic lattices and begin to extend this to 2-D. The FPU lattice is a 1-D analog of atoms in a 1-D crystal lattice, in which the atoms are linked by nonlinear springs, given by the addition of a nonlinear quartic term in the lattice potential. This problem is similar to that of multiple coupled spring-mass system, but instead of a normal linear spring, I use nonlinear springs that change their hardness with amplitude. The FPU lattice is relatively simple, and in a way serves as a proof-of-concept for the application of my numerical routine to the magnetic lattices.

The magnetic lattices represents an ionic crystal lattice, are emulated by a variation of the classical Heisenberg model [17]. There are two types of magnetic lattices: ferromagnetic, where neighboring spins tend to align and produce a net magnetic field; and antiferromagnetic, where neighboring spins tend to antialign and produce a net zero magnetic field. The Heisenberg exchange term represents this interaction. Furthermore, classically a dipole under a magnetic field undergoes precession about
the direction of the applied magnetic field. Here, the dipoles are valence electrons of the lattice ions and the magnetic field is directed along the lattice’s intrinsic easy axis (an axis with which magnetic spins tend to align), in this case the $z$-axis. This interaction is represented by the intrinsic anisotropy term, and is essential to the existence of ILM in the magnetic lattice. I will also investigate the spontaneous formation of localized modes due to perturbation to an eigenstate of the magnetic lattice, in order to get a closer look at the effect of the lattice’s nonlinearity in terms of energy distribution in the system.
2 Theory

2.1 ILMs

In a pure lattice, vibrational phenomena are often linear/planar in nature due to their periodicity and linearity according to the Bloch theorem [2]. With the introduction of defects such as impurities [3], the loss of symmetry gives rise to oscillatory localizations. However, it was theoretically shown that localized modes can be observed in a very simple nonlinear [4] and discrete lattice [5]. These intrinsic localized modes (ILMs), also known as discrete breathers, are stable oscillatory spatial localizations in a lattice maintained by nonlinear vibration phenomenon. Their stability means ILMs are exact solutions (with special properties, such as translational asymmetry) to the system’s equations of motion. ILMs are very generic, and have been found across a variety of lattices including photonic crystals [6], atomic lattices [7] and anti-ferromagnets [8], as well as macroscopic lattices such as electrical transmission lines [9] and pendulum chains [10]. They can exist if the lattice’s dynamical differential equations are nonlinear enough to initiate energy localizations, and if the lattice is discrete enough to stabilize that mode.

Nonlinearity

For a linear lattice, such as a simple harmonic coupled mass-spring system, the frequencies of the plane wave modes (or linear modes) are independent of oscillation amplitudes (e.g. \( \omega \) for a simple harmonic oscillator). In a ‘hard’ nonlinear system, these frequency values increase with amplitudes, while the opposite is true for ‘soft’ ones [3]. One can think of these two cases as a hard spring that gets stronger the longer it is pulled, and a soft spring that gets weaker the longer it is pulled. Thus by changing the oscillation amplitudes, we can shift the frequency relative to the system’s linear dispersion curve.

Discreteness

But nonlinearity alone is not enough to entice the lattice to redistribute its normal mode energy. The frequency of the lattice can shift around, but if there exists a wavenumber \( k \) for every \( \omega \) (as is the case for a wave in a continuous medium [11]), the system will always find a stable plane wave mode regardless of frequency. This is where the discreteness of the lattice comes in, and its role is best demonstrated by an example.
Consider a linear coupled mass-spring system with \( N \) masses, lattice spacing \( a \), spring constant \( c \) and equations of motion

\[
m \ddot{x}_n = c(x_{n+1} - x_n) - c(x_n - x_{n-1}),
\]

for \( n = 1, 2, \ldots, N \). Plugging in a plane wave ansatz of the form \( x_n(t) = A_n e^{i(kna + \omega t)} \) where \( k \) is the wavenumber and \( \omega \) is the wave frequency, we get \( \omega^2 = 2c/m[1 - \cos(ka)] \) and consequently the dispersion relation

\[
\omega(k) = 2\sqrt{\frac{c}{m}} \left| \sin \left( \frac{ka}{2} \right) \right|.
\]

We see that \( \omega(k) = \omega(k + 2\pi/a) \), so \( \omega(k) \) is periodic with a period of \( 2\pi/a \). Thus an interval of, for example, \(-\pi/a \leq k \leq \pi/a\) completely characterizes all plane wave solutions of the lattice, and is also called the first Brillouin zone \([2]\). In fact, as shown in Fig. 2, we can see that the Brillouin zone is generic to all discrete lattices, and not just this specific example of a coupled mass spring system. Since \( \omega \) in a discrete lattice is periodic and differentiable (no funny infinities!), the dispersion relation is bounded by a maximum frequency as seen in Fig. 3. Above this cutoff frequency, there are no plane wave normal modes for the system, which is exactly where we should be looking for interesting behaviors as the lattice now no longer has a well-defined wavenumber.

Figure 2: Demonstration of the degeneracy of information outside of the first Brillouin zone. Although the underlying waves are different, as far as the discrete lattice is concerned there is no difference between waves with \( k = \pi/a \) and \( k = 3\pi/a \) because every lattice site is located at a node of both waves.
Figure 3: Dispersion relation/plane wave spectrum for a (a) hard nonlinear system (FPU) and a (b) soft nonlinear system (antiferromagnet). Note the gap above $k = \pi/a$ at the zone boundary for (a), and the gap below $k = 0$ for (b). These are the regions with frequencies that can sustain ILMs.
**ILM existence criterion**

In this section, I will explore a hard nonlinear system (the same logic applies for a soft nonlinear system). As we recall, the lattice’s nonlinearity gives its frequencies an amplitude dependence, and its discreteness breaks the monotonicity of the lattice’s dispersion curve and introduces a gap above the plane-wave spectrum (as seen in Fig. 3(a)). Thus we can manipulate the system’s frequency either by increasing or decreasing the oscillation amplitude. Nonlinear systems, however, approach linear systems at low amplitude limits as they can now be approximated by a first order Taylor approximation, and will not display interesting nonlinear behaviors. Consequently, in order to observe nonlinear behaviors, we have to increase the lattice oscillation amplitude.

For a hard nonlinear system, increasing the amplitude raises the frequency, and eventually near the top of the band/spectrum, the plane wave mode starts to destabilize and localize, initially forming a wide ILM with slightly higher amplitude than the plane wave mode at the top of the band. We can see that the lattice energy has been redistributed and localized on the ILM [3]. In order for ILMs to exist, the oscillatory frequencies at which it is possible to observe ILMs have to avoid the linear modes on the plane-wave spectrum [12]. For the lattice in question, its discreteness induces the nonlinear region of the dispersion relation, while its nonlinearity helps the lattice frequency breaks free from the band and enter the nonlinear regime by introducing the amplitude dependence.

For a soft nonlinear system, increasing the oscillation amplitude decreases the frequency. Thus, it is impossible for a soft system to avoid the plane wave spectrum unless there is a nonlinear gap under the plane wave spectrum for the frequency to descend into. This can be the case when the particle is excited enough to be in the optical band of a diatomic system [2], or when the lattice Hamiltonian admits an onsite energy term. An onsite energy term gives every lattice site an energy term (e.g. gravity) that’s independent of other nearby sites, and is also the component that allows the soft magnetic lattice that we will investigate to display ILM behaviors.
2.2 Algorithms

2.2.1 Newton-Raphson

Newton-Raphson is one the most basic, popular, and still rather powerful root finding algorithms. To begin, I will describe the general idea of Newton-Raphson in 1-D then extend it to multiple dimension. Consider a point $x_i$ and an objective function $f(x)$ whose root we want to find. Recall the Taylor series expansion of a function around a point

$$f(x_i + \delta) = f(x_i) + f'(x_i)\delta + \frac{f''(x_i)}{2}\delta^2 + \cdots \quad (3)$$

Within a small enough $\delta$ neighborhood of $x$, we can ignore higher order terms and obtain the linearization $f(x_i + \delta) \approx f(x_i) + f'(x_i)\delta$. If $f'(x_i) \neq 0$, the tangent line $f(x_i) + f'(x_i)\delta$ passes the $x$-axis at some point. Let $(x_{i+1}, 0)$ be this intersection for $x_{i+1} = x_i + \delta$; then we have

$$f(x_i) + f'(x_i)(x_{i+1} - x_i) = 0, \quad (4)$$

or in a more suggestive way,

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \quad (5)$$

Figure 4: Illustration of the Newton-Raphson algorithm. The algorithm approaches the root with every iteration in this scenario. Adapted from *Applied Numerical Methods with MATLAB* by S. Chapra (2018)
Equation (5) is an iterative function: given a guess for the root \( x_0 \), we can follow the tangent line at this point to the \( x \)-axis for the next point \( x_1 \), then repeat the process to find \( x_2, x_3, \ldots \). Our eventual goal is to compute numerically a value \( c \) that is the true root of the function such that \( f(c) = 0 \). Equation 3 also highlights the importance of picking a beginning guess \( x_0 \) close enough to the true root: since every iteration we hope that the successive point \( x_i + \delta \) would be \( c \), if the initial guess \( x_0 \) was far away from \( c \), the higher order terms in the Taylor expansion can no longer be ignored, and we cannot guarantee a straight route from \( x_0 \) to \( c \) e.g. there might exist a local minimum or maximum between \( x_0 \) and \( c \). If the next iteration returns an \( x_1 \) close to this point, the slope is approximately 0 and the algorithm shoots off into the wild blue yonder. Of course, how close is ‘close enough’ entirely depends on the function we are dealing with: the less complicated and more convex the function is, the further \( x_0 \) can go without the algorithm failing.

![Graph](image1.png)

**Figure 5:** Two examples of Newton-Raphson failing to converge when the initial guess is not sufficiently close to the root. Adapted from *Applied Numerical Methods with MATLAB* by S. Chapra (2018)

Newton-Raphson is also easily generalizable to multiple dimensions. Given a system of \( N \) equations represented by the vector \( \mathbf{F}(\mathbf{x}) \) where \( \mathbf{F} = (F_1(\mathbf{x}), \ldots, F_N(\mathbf{x})) \)
and \( \mathbf{x} = (x_1, \ldots, x_N) \), we would like to find a root \( \mathbf{c} \) such that \( \mathbf{F}(\mathbf{c}) = \mathbf{0} \). Similar to the 1-D version, performing a multidimensional Taylor approximation yields an expression similar to Eq. (4)

\[
\mathbf{F}(\mathbf{x}_i) + \mathbf{J} \cdot (\mathbf{x}_{i+1} - \mathbf{x}_i) = \mathbf{0},
\]

(6)

where \( \mathbf{J} \) is the Jacobian matrix of \( \mathbf{F} \), defined as

\[
\mathbf{J} = \begin{bmatrix}
\frac{\partial}{\partial x_1} F_1 & \frac{\partial}{\partial x_2} F_1 & \cdots & \frac{\partial}{\partial x_N} F_1 \\
\frac{\partial}{\partial x_1} F_2 & \frac{\partial}{\partial x_2} F_2 & \cdots & \frac{\partial}{\partial x_N} F_2 \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial}{\partial x_1} F_N & \frac{\partial}{\partial x_2} F_N & \cdots & \frac{\partial}{\partial x_N} F_N
\end{bmatrix}
\]

Newton-Raphson has a variety of benefits and drawbacks. The main attraction of Newton is that the algorithm converges quadratically for a simple root (nonzero slope at root), which means the number of significant digits doubles with every iteration [1]. This places Newton-Raphson in the upper echelon of fast numerical algorithms, and makes it the algorithm of choice for functions with efficient derivatives that are continuous and nonzero near the root. The drawbacks of Newton-Raphson is the need for an efficient derivative of the function, and the above problem of the algorithm possibly not being able to converge if the initial guess \( x_0 \) is too far away from the true root. If the derivative or Jacobian is inconvenient or expensive to evaluate, it can be approximated using finite-difference method [14]. However, this additional approximation makes Newton-Raphson slower than many of its variations, and thus should be used with care.

Considering all these points, I chose Newton-Raphson for this project since the Jacobian matrix can be analytically derived from the lattice’s algebraic system of equations (recall that these equations are obtained with ansatzs and approximations from the system’s dynamical equations). Furthermore, practically every root finding algorithm requires a ‘close enough’ guess, thus we are not compromising in that respect. Moreover, the convergence radius of Newton-Raphson can be improved in a variety of ways, the most common of which are backtracking line search [14] and trust-region constraint [14]. In both cases, the algorithms minimize a proxy function, the minimum of which is the root of the objective function. Finding minima is relatively easy compared to root finding. The backtracking line search algorithm adjusts the Newton-Raphson step size for every iteration to ensure that the algorithm stays on
The trust-region constraint algorithm does the opposite: it first chooses a step size, then maps out a trust-region in function space, then moves in the direction of the best model to the proxy function in that region.

### 2.2.2 RK4

Given an initial state \( x_0 \) and an ordinary differential equation \( x'(x, t) \) describing the time evolution of a physical system, we often would like to investigate the dynamics of such system at some point in the future. This is known as an initial value problem, and the go-to numerical routine for this problem is Runge-Kutta, also known as RK4 (since the 4\(^{th}\) order of the algorithm is standard). A more primitive version of RK4 is the Euler method, in which the state of the system at a point \( x_{i+1} = x(t_i + \Delta t) \) is extrapolated using the derivative \( x'(x_i, t_i) \) at \( x_i = x(t_i) \) and a step size \( \Delta t \) using the formula

\[
x_{i+1} = x_i + x'(x_i, t_i) \Delta t.
\]

The Euler method is not very accurate compared to more advanced algorithms nor is it stable; therefore, it is not fit for practical use. However the method provides a good starting point for an intuitive understanding of initial value algorithms. Similar to the Euler method, RK4 also approximates \( x_{i+1} \) using the derivative at \( x_i \), but also utilizes various derivatives at the midpoint \( t_i + \Delta t/2 \) and endpoint \( t_i + \Delta t \) to provide a better approximation. The formula for RK4 reads

\[
\begin{align*}
  k_1 &= x'(x_i, t_i) \Delta t \\
  k_2 &= x' \left( x_i + \frac{k_1}{2}, t_i + \frac{\Delta t}{2} \right) \Delta t \\
  k_3 &= x' \left( x_i + \frac{k_2}{2}, t_i + \frac{\Delta t}{2} \right) \Delta t \\
  k_4 &= x' \left( x_i + k_3, t_i + \Delta t \right) \Delta t \\
  x_{i+1} &= x_i + \frac{1}{2} (k_1 + 2k_2 + 2k_3 + k_4).
\end{align*}
\]

where \( k_n/\Delta t \) (unrelated to the wavenumber) represents the slope evaluated during the algorithm at various \( x \) and \( t \) values. I will now refer to \( k_n \)'s as 'slopes' for simplicity. Intuitively, we start by evaluating the slope \( k_1 \) at \( (x_i, t_i) \), then follow this slope to the point \( (x_i + k_1/2, t_i + \Delta t/2) \) and evaluate the slope \( k_2 \). We then start over from \( x_i \) then follow \( k_2 \) to evaluate the slope \( k_3 \) at \( (x_i + k_2/2, t_i + \Delta t/2) \). Then we follow \( k_3 \) from \( x_i \) to evaluate slope \( k_4 \) at \( (x_i + k_3, t_i + \Delta t) \). The true estimate of \( x_{i+1} \) is then computed using a weighted average of the four slopes \( k_1, k_2, k_3 \) and \( k_4 \). A visual demonstration of the algorithm is shown in Fig. 6. Although the algorithm looks expensive, RK4’s
error term is of 5th order, which is minuscule compared to Euler’s first order error. Thus RK4 can use a much larger time step $\Delta t$ compared to Euler, and can therefore propagate the system much faster in time.

Figure 6: Illustration of the RK4 algorithm. The final slope $\Phi$ is computed by taking a weighted average of $k_n$’s. Adapted from *Applied Numerical Methods with MATLAB* by S. Chapra (2018)

The above algorithm is used to solve a system of first order ODEs. As we will see, the FPU lattice is described by a system of second order ODEs, and RK4 will have to be slightly modified to accommodate this. We start by splitting each second-order ODE $x''(x', x, t)$ into two coupled first order ODEs

$$\begin{cases} 
  x'(y, x, t) = y(x, t) \\
  y'(y, x, t) = x''(y, x, t)
\end{cases} \tag{9}$$

with initial boundary values $x_i$ and $y_i$. The RK4 algorithm can then be performed normally, but since the two ODEs are coupled, the slope computations are interlinked:
\[
k_1 = x'(y_i, x_i, t_i) \Delta t
\]
\[
l_1 = y'(y_i, x_i, t_i) \Delta t
\]
\[
k_2 = x' \left( y_i + \frac{l_1}{2}, x_i + \frac{k_1}{2}, t_i + \frac{\Delta t}{2} \right) \Delta t
\]
\[
l_2 = y' \left( y_i + \frac{l_1}{2}, x_i + \frac{k_1}{2}, t_i + \frac{\Delta t}{2} \right) \Delta t
\]
\[
k_3 = x' \left( y_i + \frac{l_1}{2}, x_i + \frac{k_1}{2}, t_i + \frac{\Delta t}{2} \right) \Delta t
\]
\[
l_3 = y' \left( y_i + \frac{l_1}{2}, x_i + \frac{k_1}{2}, t_i + \frac{\Delta t}{2} \right) \Delta t
\]
\[
k_4 = x'(y_i + l_3, x_i + k_3, t_i + \Delta t) \Delta t
\]
\[
l_4 = y'(y_i + l_3, x_i + k_3, t_i + \Delta t) \Delta t
\]
\[
x_{i+1} = x_i + \frac{1}{2} (k_1 + 2k_2 + 2k_3 + k_4)
\]
\[
y_{i+1} = y_i + \frac{1}{2} (l_1 + 2l_2 + 2l_3 + l_4).
\]

This gives the points \(x_{i+1}\) and \(y_{i+1}\) that we can use to compute the next iteration.
2.3 Lattices

2.3.1 Fermi-Pasta-Ulam

The FPU lattice is described by the following equations [15]

\[ \ddot{x}_n = (x_{n+1} - 2x_n + x_{n-1}) + \beta \left[ (x_{n+1} - x_n)^3 - (x_n - x_{n-1})^3 \right] \]  

(11)

where \( x_n \) is the displacement at the \( n^{th} \) site where \( n = 1, 2, \ldots, N \) for a lattice of size \( N \), and \( \beta \) is the nonlinear coefficient. The behaviors at the endpoints \( x_1 \) and \( x_N \) depend on the choice of boundary: the lattice can either have fixed boundary, where the endpoints are only coupled to nearby sites, or periodic boundary where the endpoints are also coupled to one another, essentially 'looping' the lattice around. These equations have been non-dimensionalized so that the mass \( m \) and spring constant \( c \) both equal 1 for simplicity. Without the nonlinear term, the system would be identical to a nearest-neighbor simple harmonic coupled mass-spring system as seen in Eq. (1). This additional term necessary to supply the anharmonicity needed to display ILM behavior. The lattice can be thought of as a model describing the vibrational dynamics of a perfect crystal, where each mass is a lattice atom.

Recall that ILMs are exact solutions to a lattice’s equations of motion, in this case Eq. (11). However, for most nonlinear differential equations, solving them analytically is impossible. Thus we can start with an approximation of the true solution, then let a numerical root finding algorithm (in this case Newton-Raphson) do the rest of the work to get the numerically exact solution. We consider the FPU lattice at low amplitudes. Here, the effects of the nonlinear term on the lattice dynamics is minimal, and the lattice is approximately a linear coupled mass-spring system. We can therefore use the ansatz \( x_n(t) = A_n \cos \omega t \), where \( A_n \) is the oscillation amplitude for each lattice atom, and \( \omega \) is the vibrational frequency for the lattice. Note that since the solution has to be stable in time, the ansatz forces all sites to oscillate in phase, but allows them to have different amplitudes. Plugging this into Eq. 11 we have:

\[-A_n \omega^2 \cos \omega t = (A_{n+1} - 2A_n + A_{n-1}) \cos \omega t + \beta \left[ (A_{n+1} - A_n)^3 - (A_n - A_{n-1})^3 \right] \cos^3 \omega t.\]

(12)

Here we introduce another approximation. Since the lattice is in the low amplitude limit, it is almost linear and oscillates at a frequency of \( \omega \) by assumption. Thus terms that oscillate at much higher frequencies in \( \cos^3 \omega t = \frac{1}{4}(3 \cos \omega t + \cos 3\omega t) \) can be
safely ignored. This is known as the rotating wave approximation [16], and the result is $\cos^3 \omega t \approx \frac{3}{4} \cos \omega t$. With this approximation, Eq. (12) becomes:

$$(A_{n+1} - 2A_n + A_{n-1}) + \frac{3}{4} \beta [(A_{n+1} - A_n)^3 - (A_n - A_{n-1})^3] + A_n \omega^2 = 0. \quad (13)$$

Finally, recall that Newton-Raphson requires the Jacobian matrix of the system of equations. The Jacobian can be approximated using a finite-difference method [14], but in this case we can solve for the Jacobian in closed form, which results in a faster algorithm. From Sect. 2.2.1, let the right hand side of Eq. (13) be $f_n$, then the Jacobian matrix of the FPU lattice has the form

$$J = \begin{bmatrix}
\partial A_1 f_1 & \partial A_2 f_1 & \cdots & \partial A_N f_1 \\
\partial A_1 f_2 & \partial A_2 f_2 & \cdots & \partial A_N f_2 \\
\partial A_2 f_3 & \partial A_3 f_3 & \cdots & \partial A_N f_3 \\
\partial A_3 f_4 & \partial A_4 f_4 & \cdots & \partial A_N f_4 \\
\vdots & \vdots & \ddots & \vdots \\
\partial A_1 f_N & \cdots & \partial A_{N-1} f_N & \partial A_N f_N
\end{bmatrix}$$

where

$$\partial A_i f_j = \begin{cases} 
0 & |i - j| > 1 \\
1 + \frac{9}{4} \beta (A_i - A_j)^2 & |i - j| = 1 \\
-2 - \frac{9}{4} \beta [(A_{j+1} - A_j)^2 + (A_j - A_{j-1})^2] + \omega^2 & i = j
\end{cases} \quad (14)$$

Since the FPU lattice in this thesis only considers nearest-neighbor interactions, the Jacobian is rather sparse with (partial) tridiagonal symmetry. However, note the special cases of $\partial x_N f_1$ and $\partial x_1 f_N$. In the Jacobian matrix above, these two terms are nonzero since we’re considering periodic boundary conditions, which means the endpoints of the lattice are coupled together. If this is not the case, those two terms would each take on a value of 0.

Now we have a system of $N$ equations and $N+1$ unknowns from Eq. (13), including all $A_n$’s and $\omega$. For every $\omega$ there exists a different set of coefficient $A_n$’s, and by the ILM existence criterion, only frequencies that are not on the plane wave spectrum can return ILMs. To find the approximate dispersion curve for the linearization of the lattice, note that for a plane wave mode, we can safely ignore the nonlinear term in the equation of motion. Then the equation of motion becomes $\ddot{x}_n = x_{n+1} - 2x_n + x_{n-1}$, which is similar to the example we solved in Sec. 2.1. Again, consider a wave
ansatz \( x_n(t) = A e^{i(kn a + \omega t)} \), where \( a \) is the lattice spacing constant. Then we have \( x_{n \pm 1} = x_n e^{\pm i kn} \), and plugging this into the equations of motion gives

\[
\omega(k) = 2 \left| \sin \left( \frac{ka}{2} \right) \right|. \tag{15}
\]
as shown in Fig. 7. At the zone boundary, we find \( \omega(\pi/a) = 2 \) which is also where the plane wave modes start to destabilize. Then to begin, we can start finding roots for Eq. (13) with frequencies slightly larger than 2, for example \( \omega = 2.1 \). Since this frequency is only slightly above the linear spectrum, the FPU lattice can be solved with the rotating wave approximation without significant loss in accuracy.

![Dispersion relation of FPU lattice](image)

Figure 7: Dispersion relation for the FPU lattice. Since the FPU lattice has hard nonlinearity, increasing the peak amplitude of the system allows the frequency \( \omega \) to rise above the plane wave spectrum at \( k = \pi/a \).

### 2.3.2 Magnetic

The time evolution of a magnetic lattice is described by [17]

\[
\frac{dS_n}{dt} = S_n \times H_n^{\text{eff}} \tag{16}
\]
where \( S_n \) is the magnetic moment (for this lattice an atom’s valence electron spin) at site \( n \) and \( H_n^{\text{eff}} \) is the effective magnetic field at site \( n \), represented as

\[
H_n^{\text{eff}} = 2J(S_{n-1} + S_{n+1}) + 2DS_n^z \hat{z}. \tag{17}
\]
This is the effective magnetic field for a variation of the Heisenberg model of a 1-D nearest-neighbor magnetic lattice, where \( J \) is the spin coupling constant and \( D \) is the
lattice anisotropy (read: direction dependent) coefficient. Similar to the FPU lattice, this equation has been non-dimensionalized (i.e. using units where \( \hbar, \gamma, \ldots = 1 \)) for simplicity. The model is actually defined by its Hamiltonian, from which the effective field is obtained by

\[
H_n^{\text{eff}} = -\nabla S_n \mathcal{H} = - \sum_{n=1}^{N} (\partial \mathcal{H} / \partial S_n) \hat{S}_n,
\]

with

\[
\mathcal{H} = -2J \sum_n S_n \cdot S_{n+1} - D \sum_n (S_n^z)^2.
\]  

(18)

In both Eq. (17) and Eq. (18), the first term is called the Heisenberg exchange term, which describes interactions between nearest-neighbor lattice sites, and the other term is the easy axis anisotropy term which describes the interaction that only depends on the current site. The Hamiltonian of a system almost always represent the total energy, and a system tends to minimize its energy. Thus, we can see that the sign of \( J \) signifies whether the lattice is ferromagnetic (aligned neighboring spins) or antiferromagnetic (anti-aligned neighboring spins). If \( J \) is positive, \( S_n \) will tend to align with \( S_{n+1} \) to minimize the overall exchange term and vice versa. Meanwhile, the anisotropy term is minimized if the dipole/spin is aligned with the anisotropy direction (also called the easy axis), in this case the \( z \)-axis. The easy axis is responsible for a gap under the plane wave spectrum on the soft nonlinear magnetic lattice’s dispersion curve, and by proxy the lattice’s ability to display ILM behaviors.

The anisotropy term describes the interaction associated with a magnetic lattice’s easy axis, which is a line through the lattice that is energetically favorable for spontaneous magnetization. From classical electrodynamics, a dipole \( \mu \) in a magnetic field \( B \) with direction \( \hat{S} \neq \pm \hat{B} \) experiences a torque \( \tau = \mu \times B \), and will start precessing in the clockwise direction around the direction of the field. In fact, this would be exactly the same as Eq. (16) if the effective magnetic field \( H_n^{\text{eff}} \) did not have the Heisenberg exchange term - if the lattice dipoles don’t interact with each other at all.

Consider the uniform mode in a ferromagnetic lattice, in which all spins are displaced from the \( z \)-axis by the same amount, and are all in phase as seen in Fig. 8. The wavenumber of this setup is \( k = 0 \), and without the anisotropy term, nothing would happen since all lattice spins would align with its neighboring spins’ current direction, which is the same for all spins and is also the net magnetization of the lattice. For an antiferromagnetic lattice, there would be no net magnetization and thus also no movement. In both cases, since the spin is not precessing, the lattice’s oscillation frequency is \( \omega = 0 \) and we have no band gap. However, with the anisotropy term, all lattice spins would experience a torque and will start to precess, giving rise to a nonzero frequency \( \omega \). And since all spins precess in unison, \( k = 0 \) and we have
a gap underneath the plane wave spectrum, as can be seen in Fig. 9.

Figure 8: Uniform mode for a ferromagnetic lattice. All spins are initiated in the same position and phase, and precess clockwise about the easy axis in an uniform manner with identical frequency and coordinate amplitudes for all time.

Similar to the FPU lattice, we attempt to simplify Eq. (16) into an algebraic system of equations that can be solved numerically. Recall that a dipole under a magnetic field will precess in the clockwise direction around the magnetic field direction. Thus we consider the ansatz

\[
S_n(t) = (s_n \cos \omega t)\hat{x} - (s_n \sin \omega t)\hat{y} + \left(\sqrt{1 - s_n^2}\right)\hat{z}
\]

for the ferromagnetic lattice and

\[
S_n(t) = (s_n \cos \omega t)\hat{x} - (s_n \sin \omega t)\hat{y} + (-1)^n\left(\sqrt{1 - s_n^2}\right)\hat{z}
\]

for the antiferromagnetic lattice, where \(-1 \leq s_n \leq 1\) is the projected spin amplitude on the \(xy\)-plane and \(\omega\) is the precession frequency. From Eqs. (16) and (17) we have

Ferromagnet

\[
\left(\sqrt{1 - s_{n-1}^2} + \sqrt{1 - s_{n+1}^2}\right)s_n - \left(s_{n-1} + s_{n+1} + \frac{D}{J} s_n\right)\sqrt{1 - s_n^2} - \frac{\omega}{2J}s_n = 0 \quad (19)
\]

Antiferromagnet

\[
(-1)^n \left[\left(\sqrt{1 - s_{n-1}^2} + \sqrt{1 - s_{n+1}^2}\right)s_n + \left(s_{n-1} + s_{n+1} - \frac{D}{J} s_n\right)\sqrt{1 - s_n^2}\right] + \frac{\omega}{2J}s_n = 0.
\]

The Jacobian matrix for the magnetic lattice is structurally similar to the Jacobian of the FPU matrix, but instead with \(g_j\) as the left hand side of Eqs. (19) and (20), and the matrix elements are defined as follow
Ferromagnet

$$\partial_{s_i} g_j = \begin{cases} 
0 & |i - j| > 1 \\
-s_j \frac{s_i}{\sqrt{1 - s_i^2}} - \sqrt{1 - s_j^2} & |i - j| = 1 \\
(\frac{1}{\sqrt{1 - s_j^2}} + \frac{1}{\sqrt{1 - s_{j+1}^2}}) + (s_{j-1} + s_{j+1}) \frac{s_j}{\sqrt{1 - s_j^2}} \\
-\frac{D}{J} \frac{1 - 2s_j^2}{\sqrt{1 - s_j^2}} - \frac{\omega}{2J} & i = j 
\end{cases}$$

(21)

Antiferromagnet

$$\partial_{s_i} g_j = \begin{cases} 
0 & |i - j| > 1 \\
(-1)^n \left[-s_j \frac{s_i}{\sqrt{1 - s_i^2}} + \sqrt{1 - s_j^2}\right] & |i - j| = 1 \\
(-1)^n \left[(\frac{1}{\sqrt{1 - s_j^2}} + \frac{1}{\sqrt{1 - s_{j+1}^2}}) - (s_{j-1} + s_{j+1}) \frac{s_j}{\sqrt{1 - s_j^2}}
\right. \\
\left. -\frac{D}{J} \frac{1 - 2s_j^2}{\sqrt{1 - s_j^2}} + \frac{\omega}{2J}\right] & i = j 
\end{cases}$$

(22)

Now we would like to find the cutoff frequency for the nonlinear regime of the lattice’s dispersion relation. For a ferromagnetic lattice, we can assume the plane wave solution by setting all $s_n = s_0$ for some $|s_0| < 1$. Because the nonlinear frequencies in magnetic lattices lie below the linear spectrum, we have $k = 0$. Then a uniform mode ansatz of the system would be $S_n(t) = (s_0 \cos \omega t) \hat{x} - (s_0 \sin \omega t) \hat{y} + (\sqrt{1 - s_0^2}) \hat{z}$. From Eqs. (16) and (17) we have

$$\frac{dS_n}{dt} = S_n \times H_n^{\text{eff}} = -2Ds_0 \sqrt{1 - s_0^2} (\sin \omega t \ \hat{x} + \cos \omega t \ \hat{y}).$$

From the ansatz, $dS_x/dt = -\omega s_0 \sin \omega t$ and $dS_y/dt = -\omega s_0 \cos \omega t$. In both cases, we obtain

$$\omega(0) = 2D \sqrt{1 - s_0^2}. \quad (23)$$

In the low amplitude limits, the system is approximately linear with $\omega(0) \approx 2D$. For the purpose of this project, setting $D = 1$ for simplicity we get $\omega(0) = 2$ and consequently $\omega/2J = 1$ which is also the threshold at which the plane wave modes begin to break down.
For the antiferromagnetic lattice, deriving the dispersion relation is complicated due to interactions between the anisotropy field and exchange field in the lattice. Therefore I will not derive the relation here, but the nonlinear threshold frequency is given in Ref. [17] as

\[ \omega(0) = 2J \sqrt{\left(2 - \frac{D}{J}\right)^2 - 4}. \] (24)

and the full dispersion relation is demonstrated in Fig. 9. Letting \( D = 1 \) and \( J = -1 \), we have \( \omega(0) = -2\sqrt{5} \) and for the purpose of Eq. (19), \( \omega/2J = \sqrt{5} \). Below this value lies the nonlinear regime, where we can begin our search for the antiferromagnetic ILM.

Figure 9: Dispersion relation for the antiferromagnetic lattice. The anisotropy term is responsible for the gap underneath the plane wave spectrum. Increasing the peak amplitude of the system lowers the frequency \( \omega \) to the nonlinear regime at \( k = 0 \).
3 Numerical Setup

For the project, I used Python due to its accessibility and relative simplicity. Although Python’s interpreted nature makes it slower than compiled languages like C/C++, it is worth trading speed for ease of use since a large amount of this project is exploring, testing concepts and troubleshooting. To that purpose, I also use the IPython environment, in particular Jupyter Notebook. The IPython environment allows me to modularize my script, and as such is incredibly useful for debugging and testing, as I can simply execute and troubleshoot the section of code that’s currently being dealt with.

For the simulation itself, I made use of the numerical package numpy [19] and occasionally the scientific computing package scipy [20]. The package numpy is mainly used for matrix manipulation while scipy is used for numerical routines such as root finding, Fourier analysis, etc. In particular, scipy includes both Newton-Raphson and RK4 algorithms, but due to the elaborateness of the package’s algorithm and the increased complexity of the magnetic lattices, I resorted to using my own version of the numerical routines except for when the situation calls for a more robust implementation of Newton-Raphson. Both packages are optimized for speed and robustness, and numpy is especially efficient when dealing with matrices compared to the traditional method of iterating over rows and columns using for loops. Thus a goal of the project is to utilize numpy as much as possible, as simulating large lattices can take an extremely long time.

In order to keep the code as simple as possible without losing its flexibility and readability, I put all related and often used pieces of code (such as functions, algorithms, parameters) into separate Python modules, then import them and use as needed. As of now, the main modules that I have are RK4.py, which contains the RK4 algorithms for both first and second degree ODE; lattices.py, which contains functions that generate various lattice configurations; newton.py, which contains implementations of Newton-Raphson; and shoot.py, which contains all the pieces needed for a crude implementation of the shooting method (described later). These code pieces are meant to be used as needed in an IPython environment, but can also be implemented in an executable .py file if needed. All codes are available in Appendix A, and also on GitHub at https://github.com/pband1256.

For the lattices, I employed periodic boundary conditions although fixed boundary cases are also included in lattices.py. For ILMs in large lattices however, the
oscillation amplitudes at the boundary are so small that the effects of each type of boundary on the system and consequently ILM formation is minimal. For localized modes that form spontaneously from deviation from an eigenstate, this is the most efficient way to obtain a semi-infinite lattice without long computational time. To sum neighboring sites, I can simply use a loop and iterate over every row and column, adding every site with its neighbors. For periodic boundary, I use the index modulo the lattice size instead of simply using the array index to ensure the lattice 'loops' around as expected. For fixed boundary conditions, I pad the lattice borders with zeroes so that the neighbor sum stops at the boundary. However, a lot of computing time can be cut by summing neighboring sites for the entire lattice at once (regardless of lattice dimension) using np.roll over each axis except the one containing the actual spin components, then sum over all such lattices. For fixed boundary, I again have to pad the lattice border with zeroes, then sum with np.roll similar to the periodic boundary case. Both cases are illustrated in Fig. 10.

Figure 10: Summing over neighboring sites for a lattice with (a) periodic boundary and (b) non-periodic boundary. The neighbor sum for all lattice sites can be computed simultaneously by 'rolling' the lattice to the left and right by 1 and summing down the 3 lattices. For (b), padding with zeroes ensures the sum only includes directly adjacent sites. In 2-D, this can be accomplished by rolling the lattice up, down, left, right by 1, and so on.

A major part of my computational manipulations within the project is related
to the magnetic lattice. To begin finding ILMs for the lattice, I attempted to use the Newton-Raphson method on a 1-D antiferromagnetic lattice of 128 spins with $\omega = 0.95\sqrt{5}$, right below the nonlinear threshold; however, Newton-Raphson was unsuccessful. Recall that Eq. (19) contains multiple terms with square roots. To get closer to the solution, Newton-Raphson takes the fastest descending direction across solution space, and along the way there are many lattice solutions with at least one spin $s_n > 1$. Evaluating this solution returns a square root error and stops the algorithm in its track. The added complexity of handling a system of 128 nonlinear equations exacerbates the problem by making Newton-Raphson go through more solutions before converging.

A large amount of my time with the project was spent on error-proofing the lattice equations, attempting various maneuvers and more robust variations of Newton-Raphson. This includes Newton with backtracking line search and trust-region constraint, the details of which are rather extensive but were briefly discussed in Sect. 2.2.1. For line search backtracking, after a certain point the altered step size shrinks to the order of $10^{-16}$, which suggests that the algorithm has come to a halt in a local minimum. I used `scipy.optimize.minimize(method='trust-constr')` [20] for the trust-region constraint method since implementing the algorithm from scratch is very time consuming. The `scipy` version takes a long time to converge (upwards of an hour), but does converge most of the time except for initial conditions with $s_n$ values close to 1. However, the algorithm always converges to the trivial solution where all $s_n \approx 0$, and combined with the long computational time, this method is not realistically feasible.

With the apparent failure of Newton-Raphson, I used the symmetry of the ILM mode and turned to the shooting method. The shooting method aims to solve two-point boundary value problems for ODEs, unlike RK4, which solves ODEs with all boundary conditions at the starting point (initial value problem). Here, we would like to find the solution to $N$ coupled ODEs given $n_1$ boundary conditions at a point $x_1$ and $N - n_1$ boundary conditions at another point $x_2$. The boundary conditions at $x_1$ do not uniquely determine a solution; thus we let the boundary conditions at $x_2$ be arbitrary free parameters, and modify them (launching 'shots') until we reach a solution that satisfies both boundary conditions. A full-fledged shooting method seeks to zero the discrepancy between a guessed trajectory and the true trajectory; however, due to time constraints I have only been able to minimize this discrepancy manually. Recall that the lattice has $N$ equations and $N + 1$ unknowns, including
all \( s_n \)'s and \( \omega \). For the magnetic lattice, the first \( n_1 \) boundary conditions is only the peak amplitude \( s_0 \). Then \( \omega \) and the spin \( xy \)-amplitudes \( s_n \)'s for \( n = 0, \pm 1, \ldots, \pm N \) are free parameters. However, since the \( s_n \)'s are restricted by Eq. (19), only \( \omega \) can freely be modified. We also know that a true ILM is localized, meaning it terminates at the lattice boundary so that \( s_{\pm N} = 0 \). Then we can launch 'shots' starting from various values of \( s_0 \) and \( \omega \), until we find a trajectory that gives us the desired localized boundary conditions \( s_{\pm N} = 0 \). It is also noteworthy that all the discarded trajectories are still solutions to the system with different free parameters. These solutions might have multiple localized wave packets, as seen in Fig. (11) instead of one localized packet similar to an ILM, but they are still stable modes provided that the lattice boundary conditions are satisfied.

Figure 11: Different trajectories for the shooting method. As \( \omega \) approaches the frequency for an ILM with peak amplitude \( s_0 = 0.7 \), the trajectory goes from (a) to (b), pushing other energy localizations away until they disappear from the lattice, which is the desired ILM solution.
For the even-parity mode, let the middle site be $s_0$ so that we have the symmetry $s_n = s_{-n}$. Starting with a certain value of $s_0$ and $\omega$, I can solve for $s_{\pm 1}$, then use $s_0$ and $s_{\pm 1}$ to solve for $s_{\pm 2}$ and so on. Thus, the problem is converted into solving a single nonlinear equation 64 times in succession instead of solving 128 nonlinear equations simultaneously, and was easily handled by `scipy.optimize.fsolve` [20]. This method can also be likened to a 1-D iterated map, where each successive value represents the amplitude of the next node. Since $\omega$ has an amplitude $s_0$ dependence, every ILM with a certain peak value $s_0$ will have exactly one $\omega$ value associated with it. Thus, I can vary the frequency and go through different solutions of the system until a certain value returns a localized solution. Due to the limited timeframe of the project, I have not been able to implement an automated and robust algorithm for the method and have only been able to find a few ILMs that are rather close together in terms of $\omega$ and $s_0$; nevertheless, the algorithm successfully returned ILM solutions for the lattice. Similarly, I assume that the odd-parity mode can also be solved by assuming $s_n = -s_{-n}$. 
4 Results and Analysis

4.1 Fermi-Pasta-Ulam ILM

The two possible ILM solutions to various nonlinear frequencies \( \omega \) of a FPU lattice with 20 sites and periodic boundary are shown in Fig. 12. Figure 12(a) shows the even-parity (Sievers-Takeno mode) and 12(b) shows the odd-parity (Page mode) solution [15]. The \( y \)-axis shows the amplitudes \( A_n \) as seen in Eq. (13) (independent of mass \( m \) and spring constant \( c \)) and the \( x \)-axis denotes the lattice site. Recall in Sec. 2.3.1 that the frequency threshold for ILM formation is at \( \omega = 2 \). I therefore started the Newton-Raphson algorithm with \( \omega = 2 \) using two different initial conditions for the two different solutions. For the even-parity solution, I started with a delta function at site \( n = 10 \), and for the odd parity solution, I added an additional delta peak at site \( n = 9 \) in the opposite direction. In both cases, Newton-Raphson successfully converged to the two solutions found at \( \omega = 2 \). Afterwards, I proceeded to find solutions at higher frequencies iteratively, with the previous solution as initial guesses for (almost) guaranteed convergence.

Figure 13 shows a plot of absolute amplitude for every lattice site versus frequency for each solution parity. Each slice of a frequency value on the \( x \)-axis is one ILM as shown in Fig. 12. Here, I plotted the absolute amplitude to reveal trends in the wave packet; since the scale of the \( y \)-axis is exceedingly small, if the axis denoted raw amplitudes, all intricacies would be swallowed up by sign differences. Furthermore, I only plotted half of all lattice sites since using absolute amplitudes gives the ILMs vertical symmetry across the middle site. It’s important to note that site \( n = 0 \) in Fig. 12 corresponds to \( A_1 \) in Fig. 13 and so on.

Additionally, I also included ILMs for a purely nonlinear system. Investigating this system is expected to reveal the relationship between nonlinearity and the shape of the ILM across different frequencies. In reference to Eq. (11), the lattice is now described as

\[
m \ddot{x}_n = \beta \left[ (x_{n+1} - x_n)^3 - (x_n - x_{n-1})^3 \right].
\]

From Fig. 12, we can see that as the frequency increases, the wave packet becomes more localized with higher peak amplitudes and a sharper packet. This is also illustrated in Fig. 13. We can see that aside from the middle two amplitudes \( A_{10} \) and \( A_{11} \), which increase with frequency, all other amplitudes decrease. This is
expected for the FPU lattice, since the FPU lattice in this thesis is a hard nonlinear system: increasing the frequency also increases the amplitude, and in turn increases the nonlinear behaviors of the lattice and drives the lattice away from normal plane wave behaviors. Figure 14 shows the ILM for an FPU lattice with only nonlinear behaviors. Comparing the shape of the ILM to that in Fig. 12, we can see that at higher amplitudes, the nonlinear modes completely dominate the lattice.

Afterwards, I evaluated the stability of the solution via RK4 to ensure that Newton-Raphson returned an actual ILM, and not just an unstable localized solution that will dissipate over time. The lattice’s time evolution is partly illustrated in Fig. 15, and I did in fact find that the solution is stable in time. Note that in this figure, $t$ does not designate time in seconds, and merely signifies the $n^{th}$ time step in the RK4 algorithm. However, high frequency ILMs break down suddenly after a while as seen in Fig. 16, which shows the middle site amplitude for the even-parity mode over time. Figure 16 shows that the ILM is stable for a longer period of time with a smaller RK4 step and lower frequencies, suggesting that this instability is due to both numerical approximations and the rotating wave approximations; nevertheless, the solution is still stable for upwards of over 30 periods, and thus can be considered a localized eigenstate of the system.
Figure 12: (a) Even-parity ILM and (b) odd-parity ILM for the FPU lattice. Note the increasingly sharper and narrower wave packet as frequency increases.
Figure 13: Graph of amplitude at each site vs frequency for (a) even-parity ILM and (b) odd-parity ILM, with $A_1$ at the bottom and $A_{11}$ at the top (overlapping with $A_{10}$ in (a)). Aside from the amplitudes at the middle sites which increases with frequency, all other sites have amplitudes that decrease with frequency. This again suggests the increasing localization of energy in the lattice, represented by a higher peak amplitude and narrower wave packet.
Figure 14: (a) Even-parity ILM and (b) odd-parity ILM for an FPU lattice with only nonlinear interactions. The energy in this lattice is highly localized at all frequencies, and at high frequencies the pure nonlinear ILM closely resembles that of the normal FPU lattice.
Figure 15: Time evolution of the even-parity ILM at $\omega = 2.1$ with RK4. The ILM solution proves to be very stable. The ILM changes in time only by a phase factor, without any change to the localization of the wave packet. Note that the figure depicts the ILM at each successive RK4 time step, rather than time in seconds.

Figure 16: Instability of ILMs over long periods of time due to approximation and numerical errors. The graphs depict the middle node ($n = 10$) of the even-parity ILM with parameters (a) $\omega = 3.65$, $\Delta t = 0.1s$ (b) $\omega = 2.1$, $\Delta t = 0.1s$ (c) $\omega = 3.65$, $\Delta t = 0.01s$ (d) $\omega = 3.65$, $\Delta t = 0.001s$ where $\Delta t$ is the RK4 time step.
4.2 Antiferromagnetic ILM

Figure 17 shows an example of an antiferromagnetic ILM with a nonzero phase, while Fig. 18 demonstrates two possible ILMs for an antiferromagnetic lattice of 128 spins with periodic boundary and no phase factor. The two ILMs have frequencies of $\omega/2J = 2.155$ and 2.124, and was obtained using the shooting method. Note that since $s_n$ is the $xy$-projected amplitude, it is positively correlated to how much the spin deviates from the easy axis and consequently how much energy the spin has. Figure 19 illustrates the subtle differences between the two ILMs by showing the absolute amplitude difference, obtained by subtracting from the absolute amplitudes $|s_n|$ of the ILM associated with $s_0 = 0.7$ the amplitudes $|s_n|$ of the ILMs for various other $s_0$ (consequently $\omega$) values. Observe that the difference at the middle sites is positive and increases in magnitude as $\omega$ increases, which suggests that compared to $\omega/2J = 2.124$, ILMs of higher frequencies have lower peaks, which agrees with the soft nonlinearity of the antiferromagnetic lattice. Conversely, the difference at non-middle sites are negative and also increase in magnitude, which shows that as frequency increases, the ILM’s non-middle sites increases in magnitude. Both these results are consistent with theory, in that for a soft nonlinear system such as the antiferromagnet, frequency should decrease as amplitude increases. Thus the ILM becomes more localized as its frequency dives deeper into the nonlinear regime, the opposite of the ILM we had with a hard nonlinear system (FPU lattice).

I then evolved the ILM at $s_0 = 0.7$ in time with RK4 starting at an uniform phase of 0 with $S_x = s_n$, $S_y = 0$ and $S_z = \sqrt{1 - s_n^2}$, and the results are shown in Fig. 20. The ILM is stable, and we can see from Fig. 20(b) that the $xy$-projection maintains its shape, only changing its phase with respect to $S_x$ and $S_y$ and precesses at a constant
value $S_z$ that depends on the site. Additionally, note that the ILM obtained from the shooting method with a set frequency $\omega$ was input as is into the RK4 algorithm to evolve in time with no designation of its inherent oscillation frequency. Regardless, the oscillation frequency was measured to be $\omega = 4.407$, consistent with our values from the shooting method.
Figure 18: Even-parity ILM for an antiferromagnetic lattice. In contrast to the FPU lattice, the wave packet becomes sharper and narrower as frequency decreases.

Figure 19: Difference between absolute amplitudes of ILM associated with $s_0 = 0.7$ vs. $s_0 = 0.65$ and $s_0 = 0.6$. The positive difference in the middle and negative difference at other sites also suggest that the wave packet becomes narrower and sharper as frequency decreases.
Figure 20: Time evolution of the even-parity antiferromagnetic ILM at $s_0 = 0.7$. The entire ILM precesses at stable $S_z$ as seen in (b), and oscillates in phase between $S_x$ and $S_y$. 

Figure 20: Time evolution of the even-parity antiferromagnetic ILM at $s_0 = 0.7$. The entire ILM precesses at stable $S_z$ as seen in (b), and oscillates in phase between $S_x$ and $S_y$. 
4.3 Ferromagnetic spontaneous localizations

In addition to finding ILMs in the antiferromagnetic lattice, I also investigated the formation of localized energy patterns and redistribution of energy in a ferromagnetic lattice. The ferromagnetic lattice was chosen due to the ease of establishing a uniform mode in this lattice. Starting with an unstable eigenstate of the lattice (in this case the uniform mode), I introduce a noise field that excites every spin by a small amount, as shown in Fig. 21(b). We can see from Fig. 21(a) that the system stays synchronized for a period of time, then suddenly redistributes energy in the system, forming localized 'pockets' of energy. This can be seen from Fig. 21(c), where energy in the system is localized in certain sites while leaving other sites with much lower energy than they started out with. We can also see that these localizations, although not an eigenstate of the system, still stay stable for a long period of time. Recall from the shooting method that a stable eigenstate of the system can have multiple localizations. The long term stability of some localized areas suggests that the local spin configurations around some of these 'pockets' resemble that of a true eigenmode of the system. Furthermore, at \( t = 25 \text{s} \), note that although the noise field was entirely random, the initial localizations are rather equally spaced. This can suggest that the nonlinearity of the lattice prefers a certain global wavenumber that drives the perturbations to the initial localizations as seen in Fig. 21(c).

Furthermore, I also investigate this behavior in a 2-D ferromagnetic lattice of size 50 \( \times \) 50 with periodic boundary conditions and found a similar trend as seen in Fig. 22. Starting with a uniform mode and a minor noise field, we see that the system stays coherent for a while, then abruptly redistributes energy to form localized areas. These localizations are also very stable and, for the most part, are confined to certain locations. However, we also see that they can still move around slightly and merge unhindered by their boundaries. Also note in Fig. 22 that the spins are aligned entirely along the \( +\hat{z} \) direction outside of localized areas (yellow) and \( -\hat{z} \) within localized areas (blue), thus minimizing their energy. This is not the same situation as Fig. 21, where the spins lie directly in the \( xy \)-plane within localized areas. Since the transition from \( +1 \) to \( -1 \) spins has to be continuous, the lattice energy is then forced to be concentrated at the boundary of each localized area.
Figure 21: Time evolution of an uniform 1-D ferromagnetic with small excitations. Figure (a) shows the evolution of the entire lattice in time. Figure (b) shows the initial perturbations to the uniform mode, which in time coalesce into pockets of energy as seen in (c).
Figure 22: Time evolution of an uniform 2-D ferromagnetic with small excitations. At $t = 10$ s, there is no obvious pattern to the lattice; however, soon after the lattice abruptly forms localized areas of spins aligned in the opposite direction to the rest of the lattice in a manner similar to the 1-D lattice.
5 Conclusion

In this thesis, I have found and examined ILMs in various types of nonlinear and discrete lattices, namely vibrational and magnetic lattices, using numerical routines. I successfully utilized Newton-Raphson to find ILMs for the 1-D Fermi-Pasta-Ulam lattice, and although Newton-Raphson failed to converge properly for the 1-D magnetic lattices, I was able to find ILMs for the 1-D antiferromagnetic lattice using the shooting method. The ILMs found were stable in time, suggesting that the numerical algorithms returned true ILM eigenmodes of the lattice. Additionally, the ILMs also exhibit expected behaviors for both a hard nonlinear lattice and a soft nonlinear lattice.

I also observed spontaneous localizations of energy in 1-D and 2-D ferromagnetic lattices due to nonlinear interactions. With small initial perturbations to an eigenmode of the system, the system evolves and abruptly forms localizations of energy that stays stable in time. Additionally, for the 2-D system, all the lattice energy is focused on the boundary of localized areas, shielding the interior of the area from destabilizing but still allow them to merge with other localized areas unhindered. Additionally, these localized areas resemble an eigenstate of the system locally.
A Appendix

A.1 RK4.py

def RK4(x0, func, h, args=[]):
    
    RK4 algorithm for first order ODE

    x0 = initial state
    func = slope function
    h = step size
    args = functiona arguments

    (note: higher dimensional output can be saved with pickle package)

    k1 = h*func(x0,*args)
k2 = h*func(x0 + 0.5*k1,*args)
k3 = h*func(x0 + 0.5*k2,*args)
k4 = h*func(x0 + k3,*args)
return x0 + 1/6*(k1+k4+2*(k3+k2))

def RK4nd(x0, func, h, args=[]):
    
    RK4 algorithm for second order ODE

    x0 = initial state
    func = slope function
    h = step size
    args = functiona arguments

    (note: higher dimensional output can be saved with pickle package)

c1 = x0[1]
k1 = func(x0[0], x0[1],*args)
c2 = x0[1] + h/2*k1
k2 = func(x0[0] + h/2*c1, c2,*args)
c3 = x0[1] + h/2*k2
k3 = func(x0[0] + h/2*c2, c3,*args)

c4 = x0[1] + h*k3

k4 = func(x0[0] + h*c3, c4,*args)

x = np.array([x0[0] + h/6*(c1 + 2*c2 + 2*c3 + c4), x0[1] + h/6*(k1 + 2*k2 + 2*k3 + k4)])

return x

A.2 lattices.py

import numpy as np

def fpu(A, per=True, omega=2.1, beta=1):
    
    1D FPU lattice
    per: True = periodic boundary, False = non-periodic
    omega: frequency
    beta: nonlinear coefficient

    f = np.array([])
    A = np.asarray(A)
    n = len(A)

    if per == False:
        A = np.pad(A, (1, 1), 'constant')

    Aplus1 = np.roll(A, 1, axis=0)
    Aminus1 = np.roll(A, -1, axis=0)

    f = Aplus1-2*A+Aminus1+0.75*beta*((Aplus1-A)**3-(A-Aminus1)**3)+omega**2*A

    if per == False:
        return(f[1:len(f)-1])
    return(f)

def Jfpu(A, per=True, omega=2.1, beta=1):
    

1D FPU lattice Jacobian

**per**: True = periodic boundary, False = non-periodic

**omega**: frequency

**beta**: nonlinear coefficient

```python
Jf = np.empty([n,n])
A = np.asarray(A)
n = len(A)

if per == False:
    A = np.pad(A, (1, 1), 'constant')
else:
    A = np.pad(A, (1, 1), 'wrap')

for i in range(1, n+1):
    row = np.array([])
    for j in range(1, n+1):
        if j == i:
            row = np.append(row, -2-2.25*beta*((A[j+1]-A[j])**2
                                            +(A[j]-A[j-1])**2)+omega**2)
        elif j%n == (i-1)%n:
            row = np.append(row, 1+2.25*beta*(A[j+1]-A[j])**2)
        elif j%n == (i+1)%n:
            row = np.append(row, 1+2.25*beta*(A[j]-A[j-1])**2)
        else:
            row = np.append(row, 0)

    Jf[i-1, :] = row
return(Jf)
```

Nonlinear only

```python
def fpu_non(A, per=True, omega=2.1, beta=1):
    
1D nonlinear FPU lattice
f = np.array([])
A = np.asarray(A)
n = len(A)

if per == False:
    A = np.pad(A, (1, 1), 'constant')

Aplus1 = np.roll(A, 1, axis=0)
Aminus1 = np.roll(A, -1, axis=0)

f = 0.75*beta*(((Aplus1-A)**3-(A-Aminus1)**3)+omega**2*A)

if per == False:
    return(f[1:len(f)-1])
return(f)

def Jfpu_non(A, per=True, omega=2.1, beta=1):
    
    1D FPU lattice Jacobian
    
    per: True = periodic boundary, False = non-periodic
    omega: frequency
    beta: nonlinear coefficient
    
    Jf = np.empty([n,n])
A = np.asarray(A)
n = len(A)

if per == False:
    A = np.pad(A, (1, 1), 'constant')
else:
    A = np.pad(A, (1, 1), 'wrap')
for i in range(1, n+1):
    row = np.array([])
    for j in range(1, n+1):
        if j == i:
        elif j%n == (i-1)%n:
            row = np.append(row, 2.25*beta*(A[j+1]-A[j])**2)
        elif j%n == (i+1)%n:
            row = np.append(row, 2.25*beta*(A[j]-A[j-1])**2)
        else:
            row = np.append(row, 0)
    Jf[i-1,:] = row
return(Jf)

def amag1d(s, omega, per=True, D=1, q=0):
    ""
    1D antiferromagnetic lattice
    ""
    f = np.array([])
    s = np.asarray(s)
    n = len(s)

    if per == False:
        A = np.pad(A, (1, 1), 'constant')
    
splus1 = np.roll(s, 1, axis=0)
sminus1 = np.roll(s, -1, axis=0)
ones = np.empty((len(s),))
ones[::2] = 1
    ones[1::2] = -1

    f = ones*((np.sqrt(1-sminus1**2)+np.sqrt(1-splus1**2))*s
              + ((sminus1+splus1)*np.cos(q)+D*s)*np.sqrt(1-s**2)) - omega*s
if per == False:
    return(f[1:len(f)-1])
return(f)

def Jamag1d(s, omega, per=True, D=1, q=0):
    '''
    1D antiferromagnetic lattice Jacobian
    '''
    Jf = np.empty([n,n])
s = np.asarray(s)
n = len(s)

    if per == False:
        s = np.pad(s, (1, 1), 'constant')
    else:
        s = np.pad(s, (1, 1), 'wrap')

    for i in range(1, n+1):
        row = np.array([])
        for j in range(1, n+1):
            if j%n == i%n:
                row = np.append(row, (-1)**j*((np.sqrt(1-s[j-1]**2)
                +np.sqrt(1-s[j+1]**2))+(s[j-1]+s[j+1])*np.cos(q)
                *(-s[j]/np.sqrt(1-s[j]**2))
                +D*((1-2*s[j]**2)/np.sqrt(1-s[j]**2))) - omega)
            elif j%n == (i-1)%n:
                row = np.append(row, (-1)**j*(s[j]*(-s[j-1]
                /np.sqrt(1-s[j-1]**2)) + np.sqrt(1-s[j]**2)*np.cos(q))
            elif j%n == (i+1)%n:
                row = np.append(row, (-1)**j*(s[j]*(-s[j+1]
                /np.sqrt(1-s[j+1]**2)) + np.sqrt(1-s[j]**2)*np.cos(q)))
            else:
                row = np.append(row, 0)
        Jf[i-1,:] = row
    return(Jf)
def fmag1d(s, omega, s0=0.7, D=1, q=0):
    
    1D ferromagnetic lattice
    
    f = np.array([])
    s = np.asarray(s)
    n = len(s)

    if per == False:
        s = np.pad(s, (1, 1), 'constant')

    splus1 = np.roll(s, 1, axis=0)
    sminus1 = np.roll(s, -1, axis=0)

    f = (np.sqrt(1-sminus1**2)+np.sqrt(1-splus1**2))*s
    -((sminus1+splus1)*np.cos(q)+D*s)*np.sqrt(1-s**2) - omega*s

    if per == False:
        return(f[1:len(f)-1])
    return(f)

def Jfmag1d(s, omega, s0=0.7, D=1, q=0):
    
    1D ferromagnetic lattice Jacobian
    
    Jf = np.empty([n,n])
    s = np.asarray(s)
    n = len(s)

    if per == False:
        s = np.pad(s, (1, 1), 'constant')
    else:
        s = np.pad(s, (1, 1), 'wrap')

    for i in range(1, n+1):
        row = np.array([])
        for j in range(1, n+1):

if j%n == i%n:
    row = np.append(row, (np.sqrt(1-s[j-1]**2) + np.sqrt(1-s[j+1]**2)) + (s[j-1]+s[j+1])*np.cos(q) *
    (s[j]/np.sqrt(1-s[j]**2)) - D*((1-2*s[j]**2)/np.sqrt(1-s[j]**2)) - omega)
elif j%n == (i-1)%n:
    row = np.append(row, s[j]*(-s[j-1] / np.sqrt(1-s[j-1]**2)) - np.sqrt(1-s[j]**2)*np.cos(q))
elif j%n == (i+1)%n:
    row = np.append(row, s[j]*(-s[j+1] / np.sqrt(1-s[j+1]**2)) - np.sqrt(1-s[j]**2)*np.cos(q))
else:
    row = np.append(row, 0)
Jf[i-1, :] = row
return(Jf)

def magnet(s,J=-1,D=1):
    '''
    ODE for a magnet (for RK4)
    '''
    f = []
    s = np.asarray(s)
    n = len(s)
    axis = s.ndim-1
    def neighbor_sum(s):
        sum = s
        for i in range(int(s.ndim-1)):
            sum += np.roll(s, 1, axis=i)
            sum += np.roll(s, -1, axis=i)
        return sum
    def H(s):
        return 2*J*neighbor_sum(s) + 2*D*np.array([0,0,s[..., 2]])
    return np.cross(s, H(s))
A.3 newton.py

import numpy as np

def Newton(f, Jf, x0, args=[], tol=1e-10, maxiter=10000):
    
    Normal multidimensional Newton-Raphson

    f = objective function
    Jf = objective function Jacobian
    x0 = initial guess
    args = function arguments

    i = 0
    x = x0
    while i < maxiter and np.linalg.norm(f(x,*args)) > tol:
        dx = np.linalg.solve(-Jf(x,*args), f(x,*args))
        x = x + dx
        i += 1

    if np.linalg.norm(f(x,*args)) > tol:
        print('Maximum iterations reached without reaching tolerance.
             Not converged.‘)
    return(x)

def Newton_line(func, Jfunc, x, args=[], MAXITS=10000, TOLF=1e-4,
                 TOLMIN=1e-6, TOLX=1e-7, STPMAX=100):

    Newton using backtracking line search method

    func = objective function
    Jfunc = objective function Jacobian
    x = initial guess
    args = function arguments

    x = np.asarray(x)
n = len(x)

def newf(x):
    return 0.5*np.dot(func(x,*args), func(x,*args))
def grad_f(x):
    return np.dot(func(x,*args), Jfunc(x,*args))
def lnsrch(func, xold, fold, g, p, stpmax, bounds=0.999999,
          ALF=1e-2, TOLX=1e-7, maxiter=10000):
    n = len(xold)

    check = 0
    a = lam
    sum1 = bounds-np.amax(np.abs(xold))
    ap = lam*np.amax(np.abs(p))
    if sum1 < ap:
        p *= sum1/ap

    slope = np.dot(g,p)
    if slope >= 0:
        print('Warning(1)')
    for i in range(maxiter):
        x = xold + lam*p
        f = func(x)

        if f <= fold + ALF*lam*slope:  # Sufficient decrease
            print(2, lam)
            return (x,f, check)
        else:  # Backtracking
            if lam == 1:  # First time
                tmplam = -slope/(2*(f-fold-slope))
            else:  # Subsequent times
                rhs1 = f-fold-lam*slope
                rhs2 = f2-fold-alam2*slope
                a = (rhs1/alam**2 - rhs2/alam2**2) / (alam-alam2)
                b = (-alam2*rhs1/alam**2+alam*rhs2/alam2**2)/(alam-alam2)
if a == 0:
    tmplam = -slope/(2.0*b)
else:
    disc = b**2-3*a*slope
    if disc < 0.0:
        tmplam=0.5*alam
    elif b <= 0.0:
        tmplam=(-b+np.sqrt(disc))/(3*a)
    else:
        tmplam=-slope/(b+np.sqrt(disc))
    if tmplam > 0.5*alam:
        tmplam=0.5*alam

alam2=alam
f2 = f;
alam = max(tmplam,0.1*alam)
print(i, alam, tmplam)
return (x,f, check)

fvec = func(x, *args)
fmin = newf(x)

if np.amax(np.abs(fvec)) < 0.01*TOLF: # Test initial guess
    check = 0
    return x

stpmax = STPMAX*max(np.linalg.norm(x), n)

for its in range(MAXITS):
    Jf = Jfunc(x, *args)
gf = grad_f(x)

xold = x
fmin_old = fmin

p = np.linalg.solve(-Jfunc(x,*args), func(x,*args))
x, fmin, check = lnsrch(newf, xold, fmin, gf, p, stpmax)
fvec = func(x, *args)
print(x)

if np.amax(np.abs(fvec)) < TOLF:
    check = 0
    return x
print('Max iter reached.')
return x

def Newton_constr(F, JF, x0, args=[], bounds=(-np.inf, np.inf),
    tol=1e-6, maxiter=10000):
    
    Newton using trust-region constraint method
    Very slow if guess is not close to solution

F = objective function
JF = objective function Jacobian
x0 = initial guess
args = function arguments
bounds = linear bounds on solution

i = 0
x = x0
scale = 0.5
c1 = 0.3

def f(x):
    return 0.5*np.dot(F(x,*args),F(x,*args))
def grad_f(x):
    return np.dot(F(x,*args), JF(x,*args))

if bounds == (-np.inf, np.inf):
    const = ()
else:
    const = optimize.LinearConstraint(np.identity(len(x)),
        bounds[0], bounds[1])
while i < maxiter and np.linalg.norm(F(x,*args)) > tol:
    dx = np.linalg.solve(-JF(x,*args), F(x,*args))

    x = optimize.minimize(f, x, method='trust-constr',
                        jac=grad_f, hess=optimize.BFGS(), constraints=const)['x']
    i += 1
    print(x)
    if np.linalg.norm(F(x,*args)) > tol:
        print('Maximum iterations reached without reaching tolerance. Not converged.

    return(x)

A.4 shoot.py

def spin_init(sl, s0, w, J):
    
    s0 = peak spin amplitude
    sl = s(-1)
    w = frequency
    J = lattice type (pos=ferromagnet, neg=antiferromagnet)

    return 2*np.sqrt(1-sl**2)*s0-np.sign(J)*(2*sl+s0)*np.sqrt(1-s0**2) - w*s0

def spin(sl, s0, sr, n, w, J):
    
    s0 = peak spin amplitude
    sl = s(-1)
    sr = s(+1)
    n = index of lattice site
    w = frequency
    J = lattice type (pos=ferromagnet, neg=antiferromagnet)

    return np.sign(J)**n*((np.sqrt(1-sl**2)+np.sqrt(1-sr**2))*s0
    -np.sign(J)*(sl+s0+sr)*np.sqrt(1-s0**2)) - w*s0

def shoot(s0, w1, w2, wn, n, J):

52
Crude implementation of the shooting method

s0 = peak spin amplitude
w1 = starting frequency
w2 = ending frequency
wn = number of frequencies
n = lattice size
J = lattice type (pos=ferromagnet, neg=antiferromagnet)

```
lat_complete = []
for w in np.linspace(w1, w2, wn):
    print(r'$\omega =$, w)
    s1 = optimize.fsolve(spin_init, 0, args=(s0, w, J))
    lat = [s0,s1]

    for i in range(int(n/2)):
        s = optimize.fsolve(spin, 0, args=(lat[-1],lat[-2],i+1,w))
        lat.append(ok)
    lat = lat[::-1][:-1] + lat
    lat_complete.append(lat)

return lat_complete
```
References


