Fermi-Pasta-Ulam Problem

Lachlan Kan

March 2025

1 Introduction

Consider $N+1$ unit masses connected by springs of unit strength. We can write down the following Hamiltonian.

$$H = \frac{1}{2}\sum_{i=0}^{N} p_i^2 + \frac{1}{2}\sum_{i=0}^{N}(x_{i+1} - x_i)^2 \tag{1}$$

The boundary conditions are such that the "virtual points" $x_{N+1} = x_{-1} = 0$. We will use this boundary condition for the remainder of this project. Notice that since we start counting from 0, our index goes up until N and thus we end up with $N+1$ masses. Notice that for any i, there are two terms in the potential that concerns it. They are as follows

$$(x_{i+1} - x_i)^2 + (x_i - x_{i-1})^2 \tag{2}$$

With this information, the equations of motion can be easily found via Hamilton's equations, where taking the derivative with respect to x_i amounts to focusing only on the segment from (2) that concerns the mass.

$$\frac{\partial p_i}{\partial t} = -\frac{\partial H}{\partial x_i} = x_{i+1} + x_{i-1} - 2x_i \tag{3}$$

$$\frac{\partial x_i}{\partial t} = \frac{\partial H}{\partial p_i} = p_i \tag{4}$$

Which leads to the following second order differential equation.

$$\ddot{x}_i = x_{i+1} + x_{i-1} - 2x_i \tag{5}$$

For each mass, we can ansatz a solution $x_i = A_i e^{j\omega t}$, where $j \equiv \sqrt{-1}$ and we assume that they all share the same angular frequency ω. Clearly this will only be the case when the system is oscillating in a normal mode. Hence we are solving for the normal modes. Using this ansatz, (5) becomes

$$-\omega^2 A_i = A_{i+1} + A_{i-1} - 2A_i \tag{6}$$

Writing out all the differential equations for $i = \{0, 1, 2, ..., N\}$ and getting all the terms to one side, we obtain the following

$$\begin{cases} 0 &= (\omega^2 - 2)A_0 + A_1 \\ 0 &= A_0 + (\omega^2 - 2)A_1 + A_2 \\ 0 &= A_1 + (\omega^2 - 2)A_2 + A_3 \\ \vdots & \\ 0 &= A_{N-1} + (\omega^2 - 2)A_N \end{cases} \tag{7}$$

This is now a linear system of equations with $N + 1$ equations. To solve this, system, we must call upon linear algebra. Consider a vector with its component being that of the amplitudes.

$$\mathbf{a} = \begin{bmatrix} A_0 \\ A_1 \\ \vdots \\ A_{N-1} \\ A_N \end{bmatrix} \tag{8}$$

Now we must construct the matrix. Notice that all the entries on the diagonal are $\omega^2 - 2$, since the coefficients on A_i is $\omega^2 - 2$. On their neighbours A_{i-1} and A_{i+1}, the entries are all 1. Hence, the matrix is

$$\mathbf{M} = \begin{bmatrix} \omega^2 - 2 & 1 & 0 & 0 \dots 0 & 0 & 0 \\ 1 & \omega^2 - 2 & 1 & 0 \dots 0 & 0 & 0 \\ 0 & 1 & \omega^2 - 2 & 1 \dots 0 & 0 & 0 \\ & & \vdots & \vdots \quad \vdots & & \\ 0 & 0 & 0 & 0 \quad \dots 1 & \omega^2 - 2 & 1 \\ 0 & 0 & 0 & 0 \quad \dots 0 & 1 & \omega^2 - 2 \end{bmatrix} \tag{9}$$

This is a discrete Laplacian matrix of the tridiagonal form. Now, the system of (7) reduces to the following vector equation

$$\mathbf{Ma} = \mathbf{0} \tag{10}$$

Or, equivalently expanded, we have the following

$$\begin{bmatrix} \omega^2 - 2 & 1 & 0 & 0 \dots 0 & 0 & 0 \\ 1 & \omega^2 - 2 & 1 & 0 \dots 0 & 0 & 0 \\ 0 & 1 & \omega^2 - 2 & 1 \dots 0 & 0 & 0 \\ & & \vdots & \vdots \quad \vdots & & \\ 0 & 0 & 0 & 0 \quad \dots 1 & \omega^2 - 2 & 1 \\ 0 & 0 & 0 & 0 \quad \dots 0 & 1 & \omega^2 - 2 \end{bmatrix} \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{N-1} \\ A_N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{11}$$

To solve this, we take the determinant of $\mathbf{M}$ and set it to 0. Then we can solve for ω directly. This method is easily implementable on python. However, it is computationally expensive and the computation time gets much longer with every mass added to the chain.

2 Nonlinear Perturbation

We now perturb the spring potential by a nonlinear term. Let α be a small positive number. Applying the perturbation, the Hamiltonian is thus given by

$$H = \frac{1}{2}\sum_{i=0}^{N} p_i^2 + \frac{1}{2}\sum_{i=0}^{N}[(x_{i+1}-x_i)^2 + \alpha(x_{i+1}-x_i)^\gamma] \tag{12}$$

Where $\gamma = 2$ or 3, depending on whether the perturbation is quadratic or cubic. Applying the same Hamilton equations, we arrive at the following equations of motion

$$\dot{p_i} = (x_{i+1} + x_{i-1} - 2x_i) + \alpha[(x_{i+1}-x_i)^\gamma - (x_i - x_{i-1})^\gamma] \tag{13}$$
$$\dot{x_i} = p_i \tag{14}$$

Here, we are interested in solving directly for the equations of motion. For computational purposes, it is best to condense all the momenta and position equations of the particles into one state vector

$$\mathbf{v} = \begin{bmatrix} p_0 \\ x_0 \\ p_1 \\ x_1 \\ \vdots \\ p_N \\ x_N \end{bmatrix} \tag{15}$$

If we denote $\mathbf{v} = v^\mu$ where $\mu = \{0, 1, 2, ..., 2N+1\}$, then an even μ denotes momenta and an odd μ denotes position. From (15), we can specify the initial conditions by specifying $\mathbf{v}(0)$. Similarly, we can construct a vector that encodes the time evolution of the state vector

$$\dot{\mathbf{v}} = \begin{bmatrix} \dot{p}_0 \\ \dot{x}_0 \\ \dot{p}_1 \\ \dot{x}_1 \\ \vdots \\ \dot{p}_N \\ \dot{x}_N \end{bmatrix} \tag{16}$$

Where the time derivatives $\dot{p}_i$ and $\dot{x}_i$ can be found in (13) and (14) respectively. In Python, we can solve the system by creating a function that generates the evolution vector, and then passing the function (not the evolution vector itself, but the function that generates it), as well as the initial conditions $\mathbf{v}(0)$ on to scipy for solving. Since there is a nonlinear term to the differential equations, it is best to choose the Backwards Differentiation Formula (BDF) method to solve the system. A way to check for correctness of the code is to use only two masses and set $\alpha = 0$. We can then displace the masses identically and release from rest. If this results in a normal mode, then the code is likely to be correct.

3 Modal Spectra

For any given system, it should be well known that there exists the same number of normal modes as its degrees of freedom. The mode spectra of the FPU oscillator can be found as the spatial Fourier sine transform of the displacement. The discrete Fourier coefficient ϕ_{ik} for the $i-$th particle in the k-th mode is well known to be

$$\phi_{ki} = \sqrt{\frac{2}{N}} \sin \frac{ik\pi}{N} \tag{17}$$

We can thus construct the matrix ϕ_{ki} where the rows represent modes and columns represent particles. If we extract only the displacement from the state vector and write out all its time evolution on the rows, then we obtain

$$d_{it} = \begin{bmatrix} x_0(0) & x_0(t_1) & x_0(t_2) & \cdots & x_0(t_f) \\ x_1(0) & x_1(t_1) & x_1(t_2) & \cdots & x_1(t_f) \\ \vdots & \vdots & \vdots & & \vdots \\ x_N(0) & x_N(t_1) & x_N(t_2) & \cdots & x_N(t_f) \end{bmatrix} \tag{18}$$

We can thus find the discrete sine transform via the following well known formula

$$a_{kt} = \sqrt{\frac{2}{N}} \sum_i x_i(t) \sin \frac{ik\pi}{N} = \sum_i x_i(t) \phi_{ki} \tag{19}$$

Notice the t in the subscript means that we must apply the transformation across all time. Notice that we sum across all i, the index that the two matrices share. This "summing over repeated index" is equivalent to the matrix product. Since $x_i(t)$ is discretised by d_{it}, the discrete Fourier coefficients for every point in time t can be found by the following product

$$a_{kt} = \phi_{ki} d_{it} \tag{20}$$

This can be easily calculated in Python by the @ operator, and has components of the time evolution of each mode. We can easily find the momenta modes by constructing the same matrix in (18) but with momenta, and projecting that onto the basis matrix ϕ_{ki}, which yields the Fourier sine transform of momenta.

3.1 Excitation of Modes

To excite a certain mode, say the k-th mode, all we have to do is initialise the state vector in a way such that all the masses' the initial displacements matches up exactly with the Fourier coefficient ϕ_{ki} in that k-th mode (Each mass will have a different initial displacement that corresponds to the mode). When the initial conditions are set up this way, the mode spectra will show a large amplitude for the desired mode k and zero amplitude for the other modes. Under nonlinear perturbation, the amplitude will "spread" from the k-th mode to the other modes, resulting in high amplitudes in the other modes over time. This contrasts with the linear case, whereby if the initial conditions are set up perfectly, only one mass will be excited for all time.

4 Results

Here are some results and examples from the Python code, implementing the methods outlined in the previous sections.

4.1 Quadratic Perturbation

Here we will demonstrate the results when $\gamma = 2$, which corresponds to the FPU-α problem.

4.1.1 Testing with a Known Case

To test the code, we first run it with $\alpha = 0$ (no nonlinearity - a known case) with two masses starting with equal displacement. We obtain figure 1. Notice that their displacements are identical. We have reached a normal mode. This is a known case, and we can be more confident that our solver is correct.

4.1.2 Four Masses

We can now examine more interesting cases. Consider 4 masses under quadratic perturbation. Here we have set $\alpha = 1/6$ and displaced them initially with slightly different displacements, as shown in figure 2.

4.2 Fifty Masses

We will test 50 masses displaced identically at the start, released from rest. The plot, shown in figure 3 is beginning to look like that of a continuum. The small plot however, does not do the system justice since its amplitudes of oscillation are somewhat muted due to the sheer volume of masses needed to be fit into the small plot.

4.3 Mode Spectra

Here we will deal with the mode spectra part of the simulation.

4.3.1 Exciting the First Mode

Here, we will initialise the initial conditions such that they match up with the $k = 1$ mode. Again, we will be dealing with 4 masses and testing with no nonlinearity. This results in figure 4. Notice that all other modes are 0 except for the $k = 1$ mode. This is what we expect. We can now see what happens when there exists some nonlineariy, say $\alpha = 1/4$. This results in figure 5. Notice that immediately, some Fourier amplitude gets "transferred" into the second mode. The $k = 1$ mode is gradually decreasing in Fourier amplitude and the $k = 2$ mode is gradually increasing. It may be interesting to run the simulation for a longer period of time to see what this leads to. Perhaps there exists a state where the decrease and increase stops and the system reaches equilibrium.

4.3.2 Arbitrary Initial Conditions

Here we will impose upon the system of 4 masses an arbitrary set of initial conditions, with $\alpha = 1/4$ and observe the behaviour. This results in figure 6. Notice that one mode still dominates over the other, however the oscillations are now a combination of the modes.

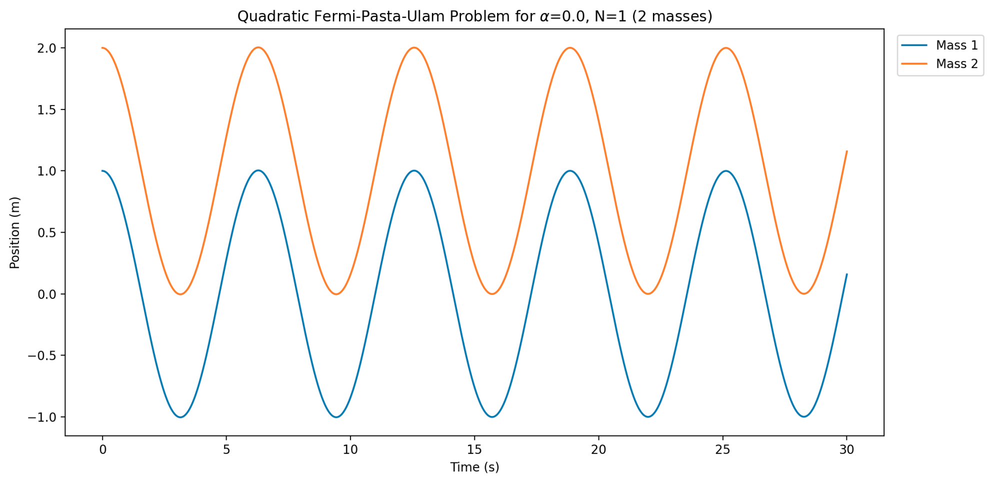
4.4 The Paradox

In the modal spectra plots, we see that the first (excited) mode gradually decreases in Fourier amplitude and the rest has its Fourier amplitudes gradually increase. It may be tempting the think that if we let the simulation run long enough, that there will be a stable point where all the modes balance and the Fourier amplitudes do not change. In other words, the system will thermalise. However, our results shown in figure 7 tells a different story. Here, we run a much longer and large scale simulation, with $\alpha = 0.6$ and 32 masses for a max time of 36000 to observe the long term behaviour of the system. We also start the simulation by imposing initial conditions that excite mode $k = 1$. we observe the classic FPU paradox - instead of equipartitioning (same amplitude for all modes in equilibrium), the modes (major trends) oscillate. Mode 1 starts decreasing and mode 2 increases, but after sufficient time, mode 2 will star decreasing and mode 1 will increase again. This periodic behaviour was the original paradox, discovered by Fermi, Pasta and Ulam. In this simulation, the maximum Fourier amplitude for mode 1 decreases every major cycle. We also compute the spectral energies for the system, shown in figure 8. Here we see the same phenomenon in terms of the energy - the energies gets periodically transferred from mode to mode. In the beginning, mode $k = 1$ (the mode we excited initially) has the most energy. Later, other modes take over as having the highest energy. This "highest energy mode" is transferred between modes $k = 2, 3, 4, ...$ one by one until eventually, mode $k = 1$ gains the most energy again, and the cycle continues. This raises the question of whether or not there is a bigger cycle, or does the system truly thermalise for large enough times. Unfortunately, this simulation was already heavy for the Macbook and the system is nowhere near thermalising. We can only conclude that even if the system thermalises, the timescales required is too large to simulate on a typical computer.

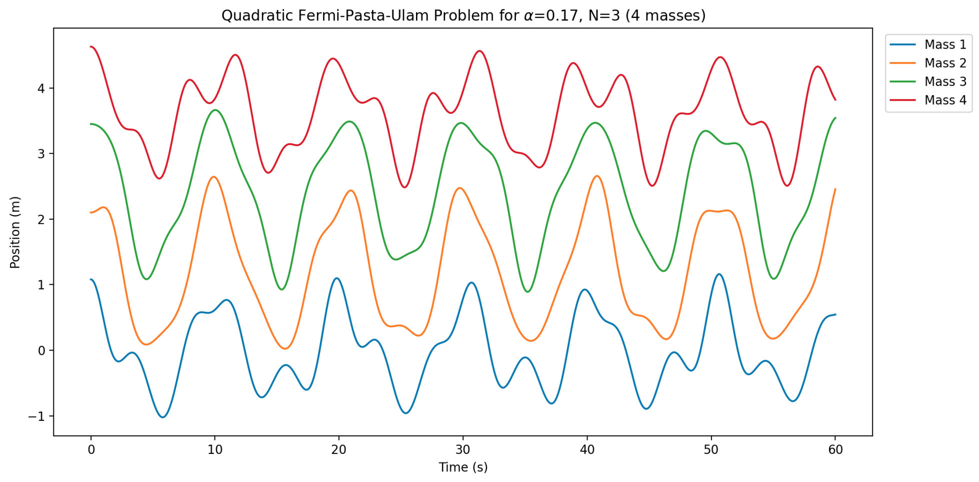
4.5 Analogous Autoparametric Systems

The traversal between modes is analogous to an autoparametric oscillator. Consider a mass attached to a spring, where the spring is hung from a ceiling and is free to swing in a plane, resulting in a mix between spring and pendulum motion. Such a system is "autoparametric", meaning that the mass will first trace out the spring motion, bobbing up and down, and then soon become pendulum-like, swinging side to side. Over time, the spring motion transforms into the pendulum motion and back, indefinitely. The spring and pendulum motion can be thought as two different modes of which the system travels between.

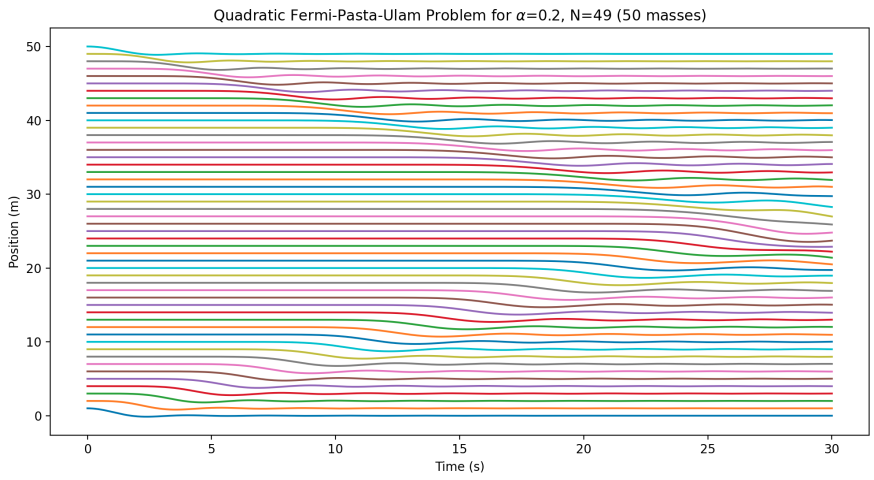
5 Figures



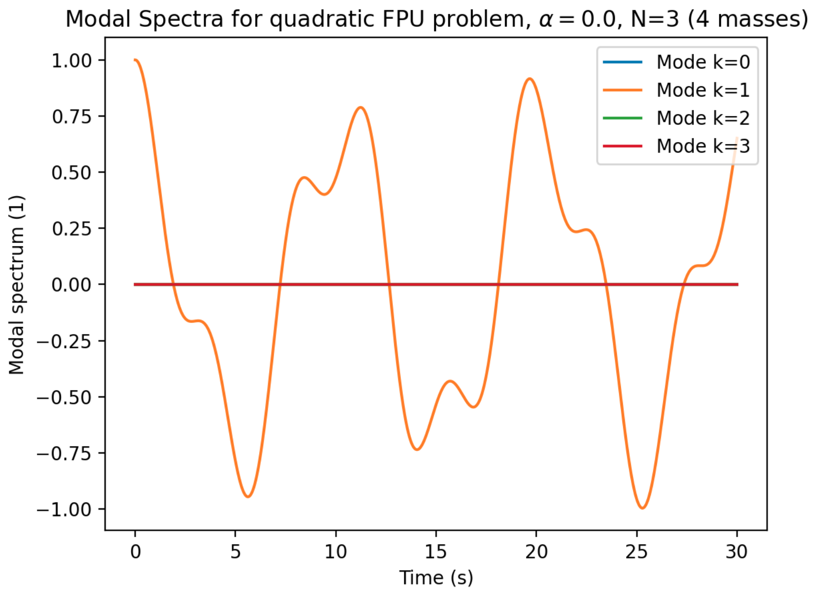
(Fig. 1)



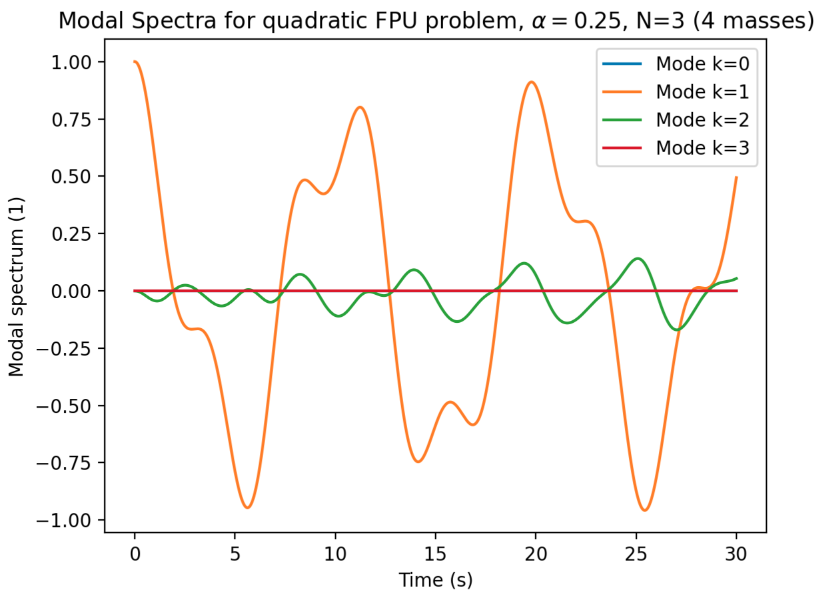
(Fig. 2)


Quadratic Fermi-Pasta-Ulam Problem for α=0.2, N=49 (50 masses)
Position (m)
Time (s)

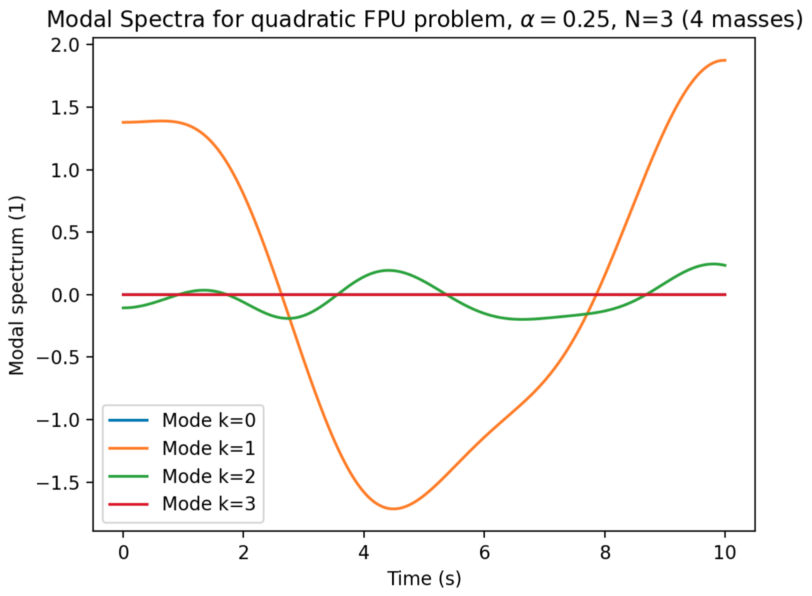
(Fig. 3)


Modal Spectra for quadratic FPU problem, α = 0.0, N=3 (4 masses)
Mode k=0
Mode k=1
Mode k=2
Mode k=3
Modal spectrum (1)
Time (s)

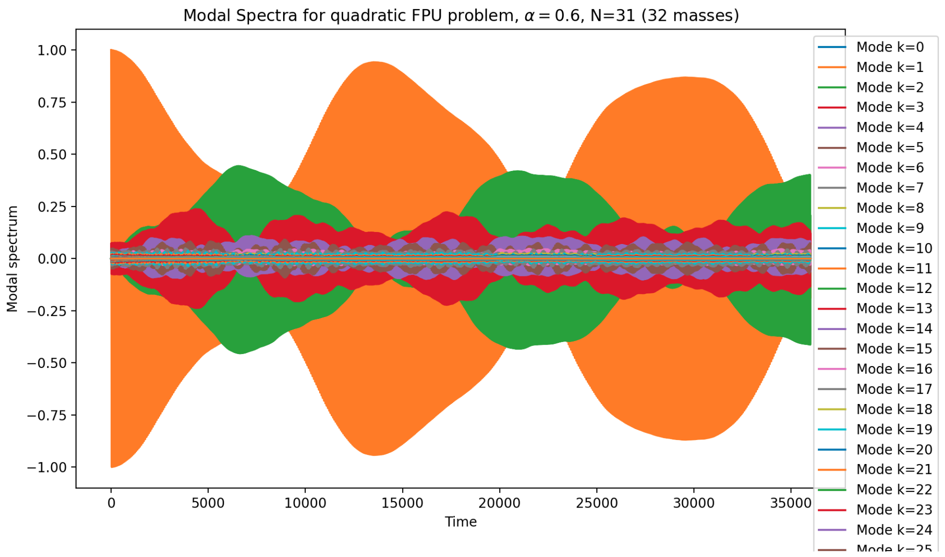
(Fig. 4)


Modal Spectra for quadratic FPU problem, $\alpha = 0.25$, N=3 (4 masses)
Modal spectrum (1)
Time (s)
Mode k=0
Mode k=1
Mode k=2
Mode k=3

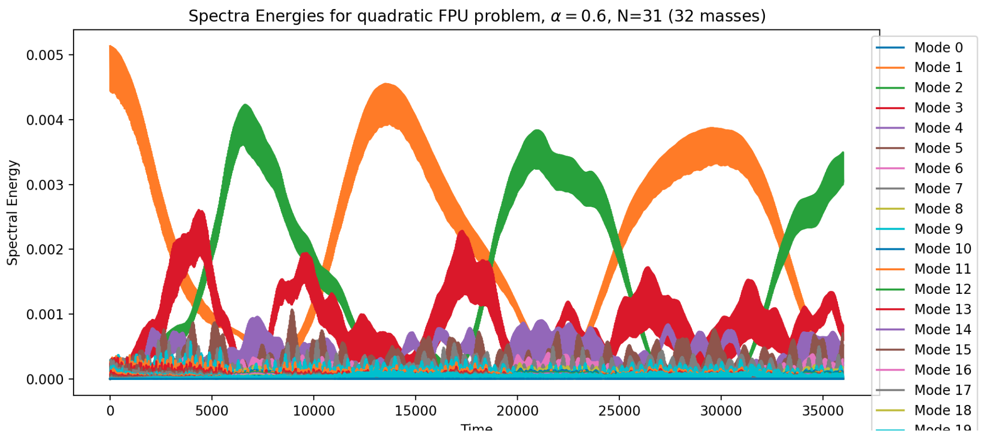
(Fig. 5)


Modal Spectra for quadratic FPU problem, $\alpha = 0.25$, N=3 (4 masses)
Modal spectrum (1)
Time (s)
Mode k=0
Mode k=1
Mode k=2
Mode k=3

(Fig. 6)



(Fig. 7)



(Fig. 8)

6 Comparing with the FPU Paper

The orignal paper by FPU, published in 1955 computed the results for $N = 32$ and $\alpha = 1$. The recurrence time (time measured from the midpoint of the first ”bump” to the midpoint of the second ”bump”) found was slightly over 14000 (the exact value is not known since the results are presented on a hand drawn graph, and we need to infer roughly from that the recurrence time). When running the Python script under these conditions, we obtain a recurrence time of 13500. This gives an error of 3.5%, which is not bad for our simulation.