AM 129 Final Project Report Deliverables

Project Type B: Fermi-Pasta-Ulam-Tsingou Problem

Owen Morehead

Dec 14 2020

Abstract

This computational project studies the behavior and motion of a system of point masses connected by springs with and without a *quadratic* nonlinear term. This system can be written as a coupled system of second order ordinary differential equations (ODE's). It has been well studied, and is what makes up the Fermi-Pasta-Ulam-Tsingou problem. This is a paradox in chaos theory that for many nonlinear systems they will eventually exhibit periodic behavior, called Fermi-Pasta-Ulam recurrence, amid the chaotic aperiodic behavior. It's been found that indeed a gradual increase of energy in the higher modes is observed, but instead of the expectation of equal distribution of energy among the different modes, the system seems to periodically concentrate most or all of its energy in one or a very small number of modes. We solve the systems ODE through a *leapfrog* method of integration. This generates discretized solutions over time for all the point masses we choose to include in our system. Plots are generated (both for systems that do and don't include the quadratic nonlinear term) which show the solution in time of specific point masses as well as the solution of all masses for specific snapshots in time. These results agree with those originally discovered which demonstrate this paradox in nonlinear systems. We find that this asymmetric energy exchange generates lower modes that are indeed disproportionately excited when there is a nonlinear force in the system. When there is no nonlinear force, all the amplitude in a mode will stay in that mode.

Methods

In this project we numerically solve the coupled, 2nd order ODE's using leapfrog integration. The coupled system of 2nd order ODE's is:

$$\ddot{x}_i = f_i = K(x_{i+1} - 2x_i + x_{i-1})(1 + \alpha(x_{i-1} - x_{i-1}))$$

$$x_i(0) = x_i^0 = 0$$

$$\dot{x}_i(0) = v_i^0 = sin(\frac{i\pi}{N+1}) \tag{1}$$

Where $1 \leq i \leq N$ is the index of the point mass in our system, $K = 4(N+1)^2$ is the spring constant, and α is the strength of the nonlinearity. We assume the first and last masses are connected by springs to rigid walls such that for all $t \geq 0$:

$$x_0(t) = 0$$
$$x_{N+1}(t) = 0 \tag{2}$$

By the finite difference formula, we can discretize our 2nd order time derivative such that:

$$\ddot{x}_i(t^n) = \frac{x_i^{n+1} - 2x_i^n + x_i^{n-1}}{\Delta t^2} + \mathcal{O}(\Delta t^2) \tag{3}$$

Which becomes an approximation after we eliminate the unknown error term $\mathcal{O}(\Delta t^2)$. Inserting this into **Equation 1**, we can now advance the solution through time, producing:

$$x_i^{n+1} = 2x_i^n - x_i^{n-1} + K\Delta t^2(x_{i+1}^n - 2x_i^n + x_{i-1}^n)(1 + \alpha(x_{i+1}^n - x_{i-1}^n)) \tag{4}$$

This generates the future solution in time x_i^{n+1} from the past solutions of the same point mass x_i^n and x_i^{n-1}, as well as from the past solutions of neighboring point masses x_{i+1}^n and x_{i-1}^n. **Equation 4** showcases this *leapfrog* method of integration used to obtain the position solutions through time for each point mass in our system. By linear extrapolation from the initial condition in **Equation 1,** we initialize the problem such that:

$$x_i^1 = x_i^0 + \Delta t v_i^0 \tag{5}$$

then use **Equation 4** to generate the solution starting with x_i^2. For the solutions generated by **Equation 4** to be stable, one can show that the maximum allowed time step size is bounded by $K^{-1/2}$, which means that if we want to evolve the solution to a final time T_f, we need to use M total steps such that:

$$\Delta t = \frac{T_F}{M}$$
$$M \geq \frac{T_f\sqrt{K}}{C} \tag{6}$$

So we set our total amount of time steps to $M = \frac{T_f\sqrt{K}}{C}$. This *safety factor* $C \leq 1$ reduces the step size and hence produces more time steps for the nonlinear case ($C < 1$). For the linear case ($\alpha = 0$) $C = 1$.

This program utilizes both Fortran and Python computing languages. The setup, initialization, and implementation of the leapfrog method is done in Fortran and resulting data for each set of specific initial conditions is saved. Python is then used to implement a run setup, scheduler, and data visualizer. The data is traversed in Python and the relevant portions are extracted to generate the desired plots. I chose to store my solution data, generated in Fortran, in the form of a matrix, where each row represents the time series solution for the ith point mass in the system.

Results / Findings

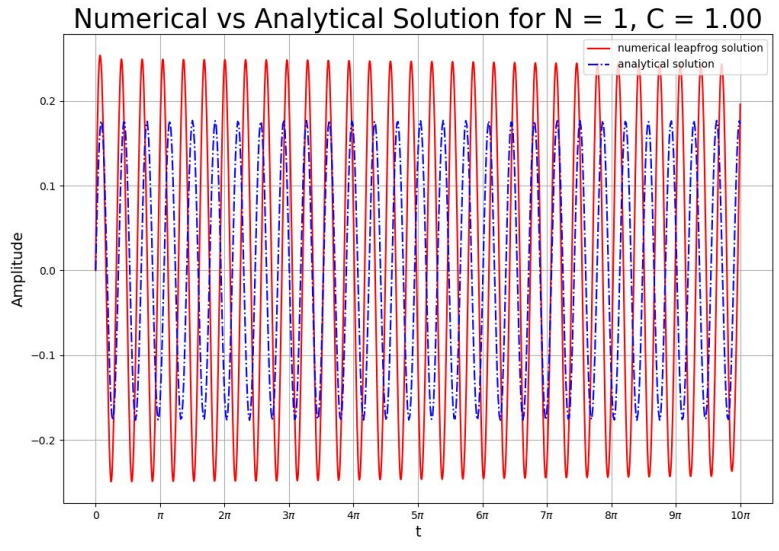
(a) We start with **N = 1**, a system of only one point mass in between two dummy masses. In this case the governing equation reduces to a linear, second order, scalar equation:

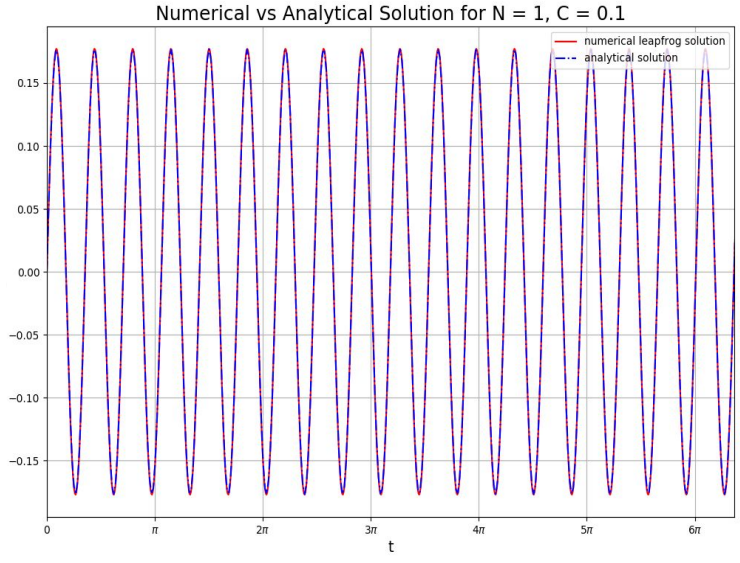
$\ddot{x} = -2kx$ which has a specific solution when we consider our boundary conditions in **Equation 1**. The solution is:

$$x(t) = \frac{1}{\sqrt{2k}} sin(\sqrt{2k}t)$$

Figures 1 and 2 show our numerical solution plotted against this analytical solution for C = 1 and C = 0.1 respectively.

Figures 1 and 2



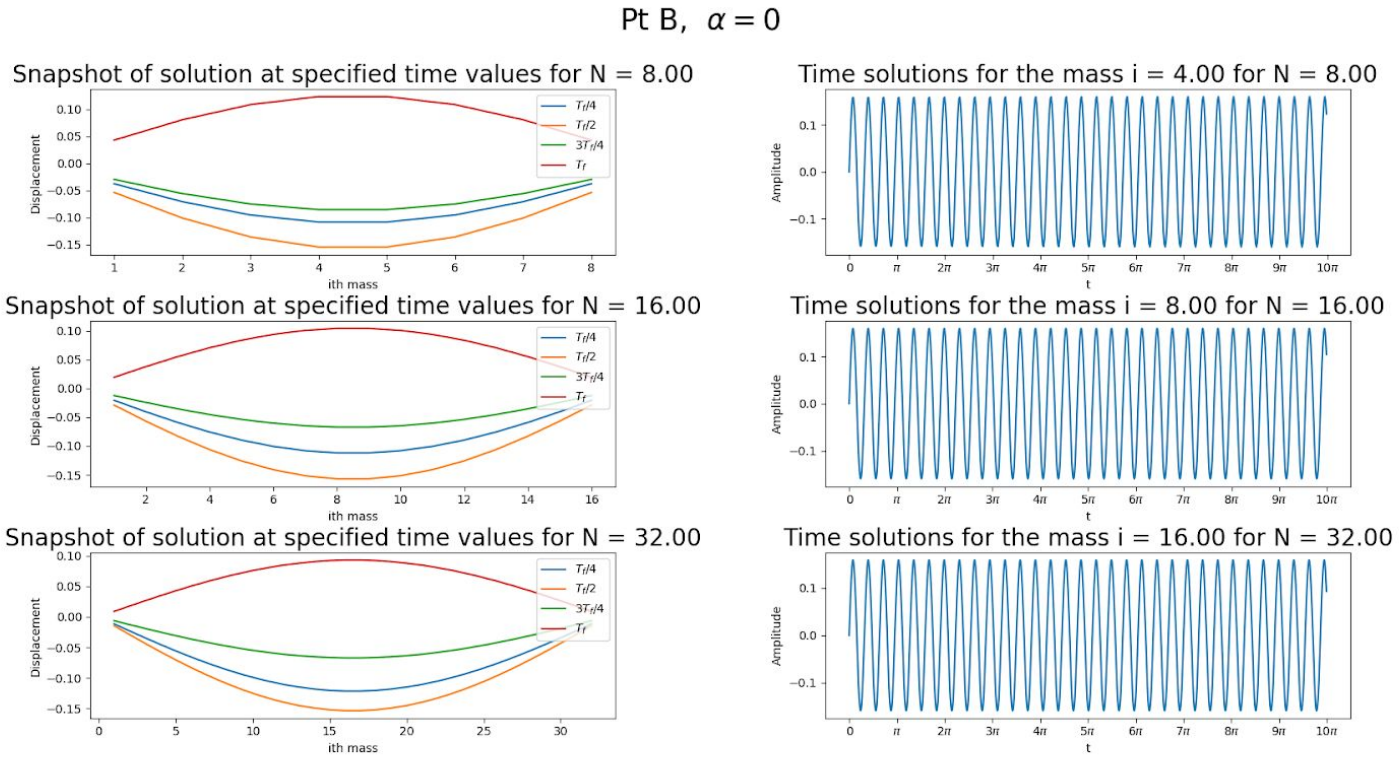


We see both solutions compare well and resemble sinusoidal motion with constant amplitude through time. If C is made smaller, which increases the amount of steps we take for our time data, our numerical solution compares much better to the analytical solution. This makes our

data from the leapfrog integration method more accurate as one would assume. In both cases our solution compares nicely to the expected solution and we can visualize the oscillatory behavior.

(b) We now run our code for values of $N = 8, 16, 32$. Plotted below in Figure 3: the left column is snapshots of the solution at certain values of t, and the right column is the time history solution for the mass $i = N/2$, all for C = 1.

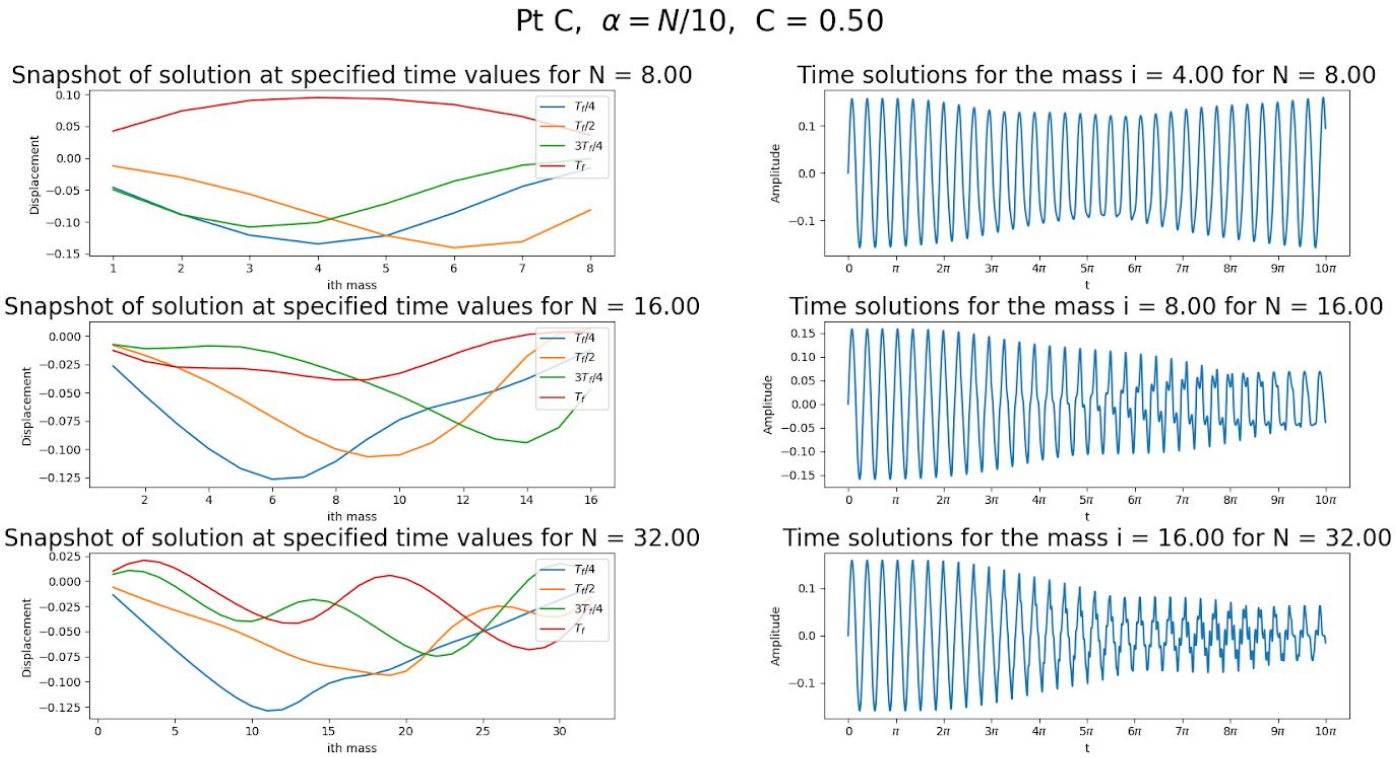
Figure 3



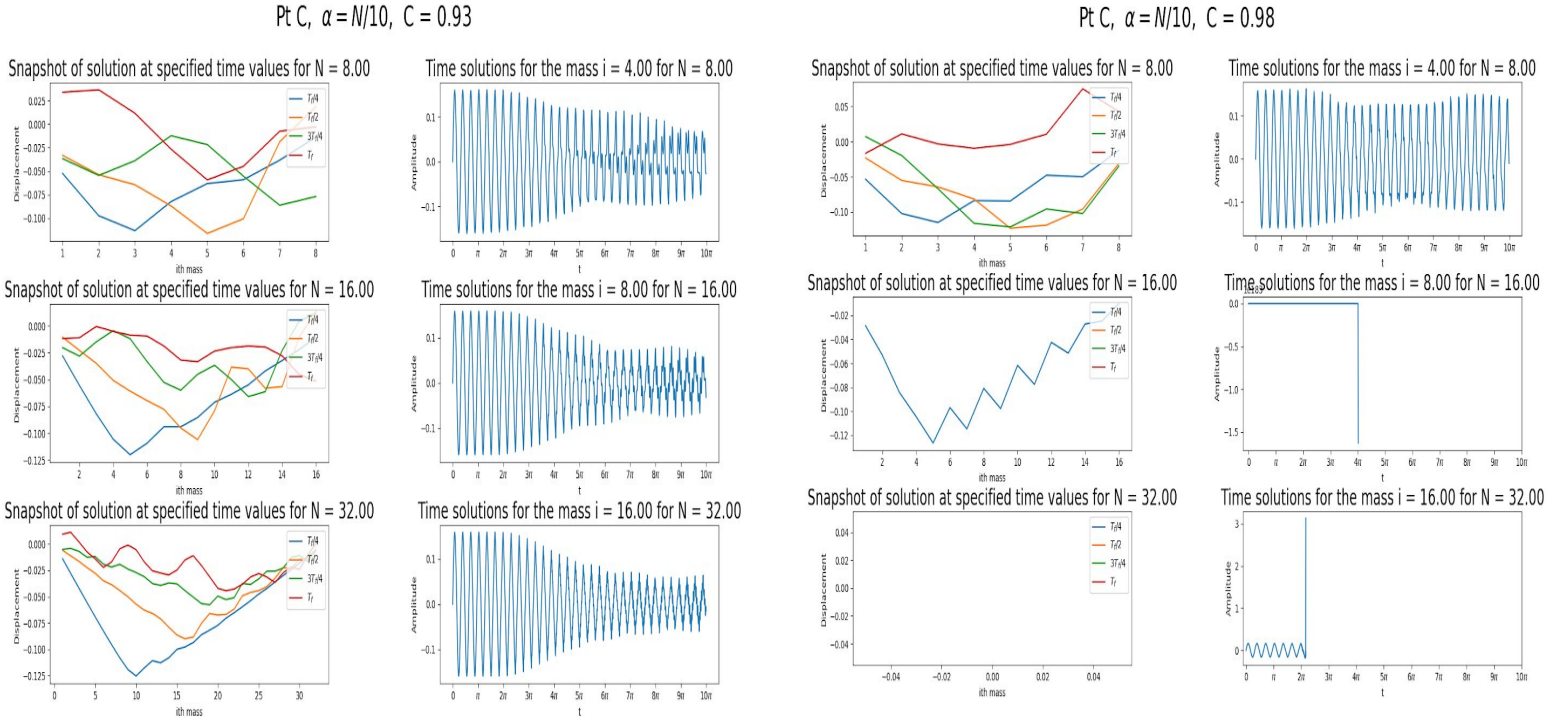
Since there's still zero nonlinear contribution in our system, we expect all the amplitude in a mode to stay in that mode. That is exactly what we see above. All the amplitude is in the first mode for any snapshot in time, and the amplitude can be either positive or negative.

(c) **Nonlinear Oscillators**: Now we set $\alpha = N/10$. This gives a larger nonlinear strength to a system of more point masses. We run the leapfrog integration for C = 0.5, and for the same three values of N as previous. The same type of plots made in Figure 2 are again made below, now with the new parameter values:

Figure 4



Figures 5 and 6

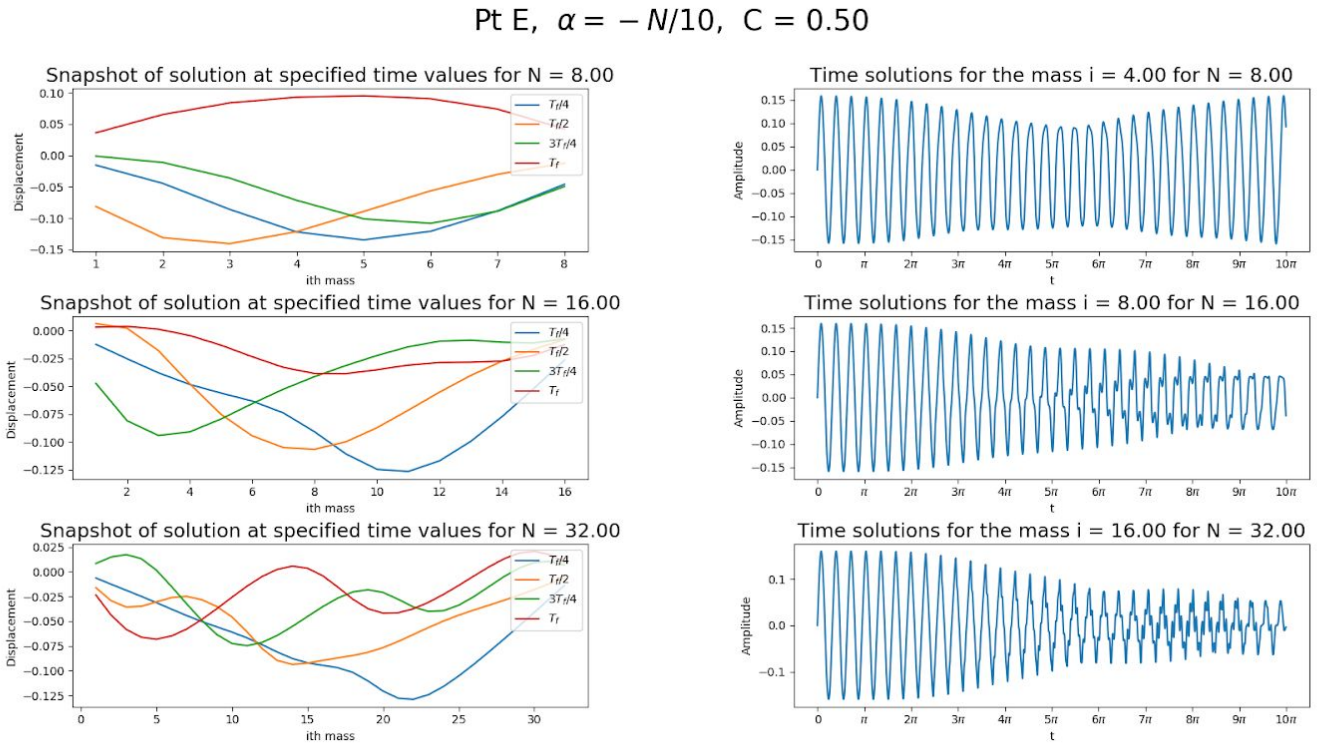


Pardon the distorted figures (trying to make good use of space here). Hopefully zooming in on them will make the labels easier to read. In Figure 4 we can see how this nonlinear contribution affects both the time solution for the mass in the middle of the system, and also the entire system's solution at certain times. We see that the energy is now being spread among multiple modes. At different times, energy is spread among the modes in different quantities as visualized above. The time solutions show how the oscillatory behavior evolves. Instead of staying in one mode throughout all time, the solution now advances into a combination of sine waves of multiple different frequencies - different modes. The amount of modes accessible is related to how many masses make up the system.

(d) As seen in Figures 5 and 6, I was able to make C fairly large before the solution is ruined and blows up. For C = 0.93, the solution is still visible, although data is not smooth, one reason being because there are not quite enough data points. For C = 0.98, the solution is ruined for the second two larger values of N. I found I was able to make C as large as 0.95 or so, before the solution blew up.

(e) Now we investigate what happens for negative values of alpha, $\alpha = -N/10$. As always, the data files are saved in the /fortran/data directory for every pair of parameters chosen to evaluate. We evaluate at the same three values of N, and for C = 0.5.

Figure 7

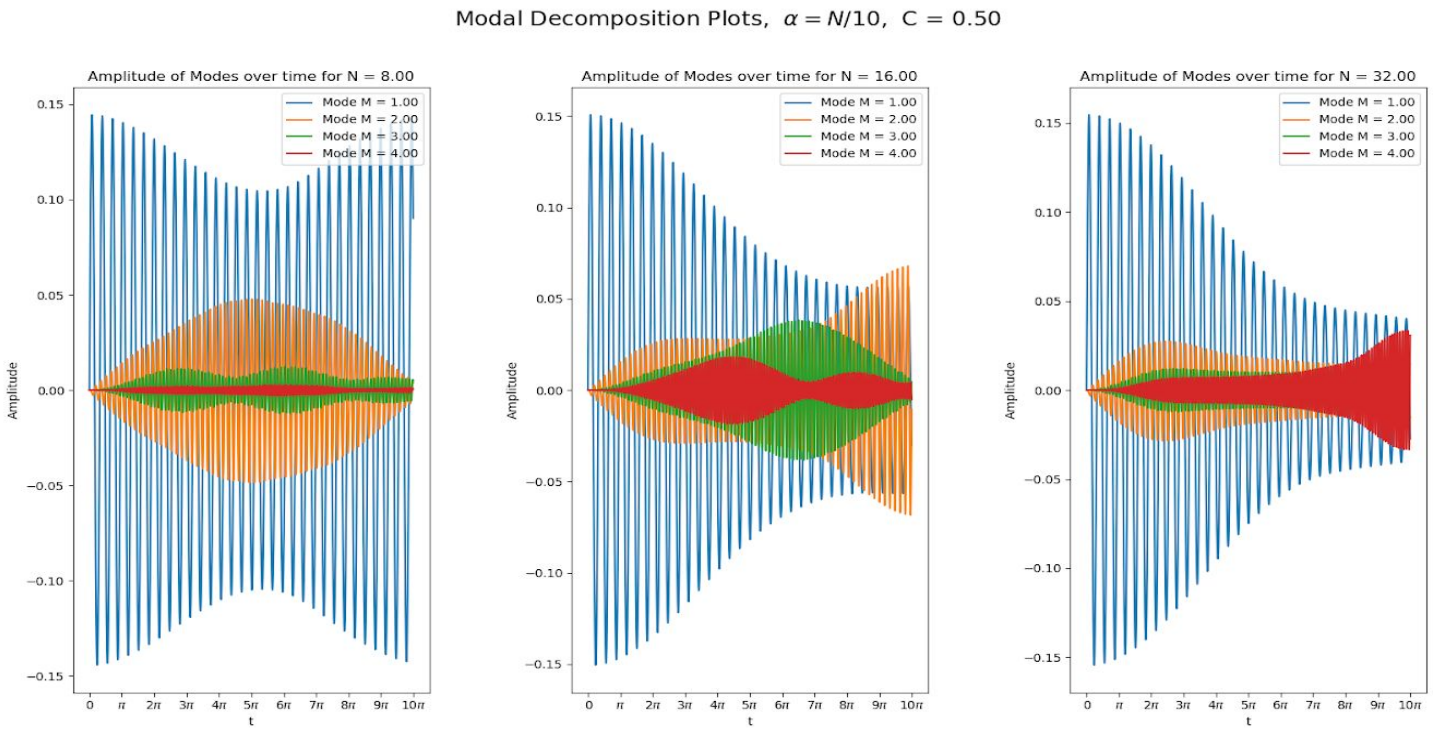


I theorized that since the strength of the nonlinear force is the same, but now applied in the opposite direction, we should see the system solution essentially reversed. That is indeed what we see in the plots of the left column of figure 7 compared to those plots in figure 4 where α is positive. All the solution data for each mass (plots on the left column) is flipped over the *vertical midline* of the system, so at the location of the mass in the middle of the system. We see that the middle mass shows the same trajectory over time for equal and opposite values of α, this makes sense. I also explored the solutions of other masses in the system that are not the center mass (which is the mass we chose to evaluate for all the plots in the right column above). And what I found was that indeed for a positive α, the solution of a mass on one side of the center of the system has the exact same solution as, for a negative value of α with the same magnitude, a mass which is an equal distance from the center of the system but now on the other side. This was a good confirmation.

Modal Decomposition of the Solution (Extra Credit)

My code has been equipped to track the first M = 4 amplitude modes through time. The figure below shows how these amplitudes evolve over time for the three values of N previously analyzed, $N = 8, 16, 32$.

Figure 8



We see that for N = 8, the first mode has a much larger amplitude through all tracked time and there is almost no amplitude to the fourth mode. Whereas for N = 32, the fourth mode

has almost an equal amplitude as the first mode around $t = 10\pi$. We see that indeed for these different values of N, the amplitudes of the first four modes change throughout time as energy is being shifted between modes.

Conclusion

I want to first note that I didn't create a fortran program called *output_module.f90* which was shown as an example in the assignment document. Instead, I wrote the solution out to disk as needed inside the *fput.f90* file. Aside from that, my code has been designed in the modular way as requested in the assignment document. My Fortran files consist of *fput.f90, setup_module.f90, leapfrog_module.f90*, *utility.f90*, and a *makefile*. And my single python program is titled *fputRun.py*.

For the most part, everything went well with this project. It was interesting to visualize the solutions and analyze this Fermi-Pasta ODE problem using the leapfrog integration technique. In learning more about this specific problem online, I realize that plotting up to a time $t = 10\pi$ is not a long enough time to actually visualize the findings of the Fermi-Pasta problem. That is the fact that the amplitude will come back to the original mode after some time. Otherwise said, over longer periods the nonlinear system does indeed thermalize, or reach equilibrium through mutual interaction, but this is not visible in any of the plots I've made above. That's okay though. With the method of data storing used in this program, we would need to store a fairly large amount of data to visualize this system thermalization over a long period of time. Instead we gather data for a smaller time range, and can still visualize how the system evolves in different ways depending on the strength of the nonlinearity and how many point masses our system is made up of.