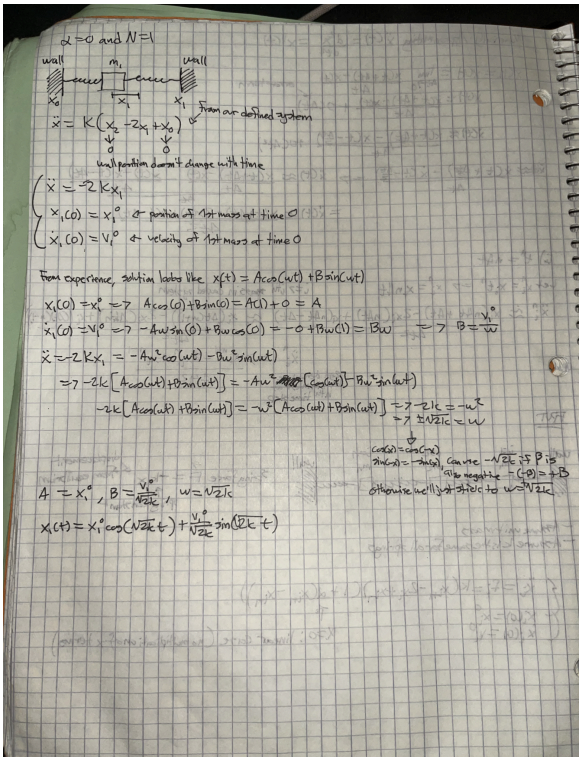
Javier Sanchez

AM129 Fermi-Pasta-Ulam-Tsingou Problem

Background:

The FPUT problem models the behavior of coupled masses in a spring system, specifically, how their displacement changes with time when given an initial position and initial velocity. For a basic understanding we set up a single unit mass attached to two stationary walls by springs with equivalent spring constants. If we perturb the mass from its equilibrium position or give it some initial velocity (and no friction), we expect it to oscillate until it reaches its resting point.

This single-mass system can be modeled with a linear 2nd order equation, which turns into an ODE system when more masses get involved. We have a nonlinearity factor α which we use to turn nonlinearity "on and off" in our model, which influences the displacement of our masses.



Fortran Implementation:

ProblemSetup: This file was initialized for us in the assignment. It creates parameters we can adjust later on (K,C,alpha) and reads in data from our input file. I defined formulas for setting timesteps and spring constants here, as well as established our initial conditions of the masses including the boundary conditions of the walls on each end having no displacement or velocity throughout. At the very end of the project, I noticed that higher numbers of masses led to more lines of data being written (like the number of timesteps increased, but the final time stayed the same). I learned this was due to each iteration of our acceleration computation being dependent on K, the spring constant, which we defined to be dependent on the number of masses. This behavior made sense to me afterwards.

Simulation.init: This is the input file. I can adjust alpha, the number of masses, the size of a factor C (which affects timestep size for oscillation visibility) and through Sean's help I added a parameter to change the final time (multiplied by pi). This helped me better understand how Fortran reads variables from other files, which basically lets me create settings for whatever projects I work on. Changes to this file also don't have to be recompiled which is a bonus.

LeapFrog: This file is the bulk of the math for the problem, and is also the part I easily spent the most time on. I had to heavily reference orbits.f90 and boundryValueProblem.f90 from HW2 and HW3 to figure out how to initialize a Fortran file on my own. I worked with a friend to plan/figure out what the assignment was asking and what we really needed in these files, and we deduced that we'd need two subroutines to actually calculate our solution to the problem. The main difficulty I had with this file was implementing all the indices for our masses. The problem description encouraged us to label our masses according to its number and timestep, which confused me since I'm used to incrementing through lists using just one index (like i). After

staring at the problem for a long time I realized that we only needed 3 time steps for ANY mass at any given iteration: the immediate past, present, and future. This meant I could use 3 variables for time, each indexed according to an arbitrary number of masses i, and helped me finish writing the formulas for finding our first future step.

Fput.f90: This is the main driving file. Implementing my subroutines was pretty straightforward from here. I used boundaryValueProblem.f90 again to learn how to call and use subroutines (very similar to functions in python). The main challenge here was learning about allocatable arrays and how to write data to a file. I was doing the latter wrong for a while, as I realized I was creating an entirely new "outFile" in my directory. I learned from a friend that I should've been writing to a data file in a separate "data" directory, which was a hurdle I had to figure out. Sean helped me out while I was referencing boundaryValueProblem.f90 again, and we decided to not use a do loop to write our data. Instead, I wrote out the function by hand which ended up neglecting the purpose of the **output.f90** file the assignment requested. There was simply no need to make another file entirely for this purpose, and in the end it made the project a little cleaner.

Makefile: My Makefile was the first step I said I was gonna do in my initial assignment summary, but almost immediately realized I didn't actually know the order of the file dependencies until I'd started writing them. At the start I was only able to imagine what the order probably was, but I didn't really know until I was like 1 or 2 files to the end. Sean helped me make a generalized Makefile which I will probably be using for any other projects later down the line (all I have to do is change the name of my executable and the dependent files and it should work with anything).

massplotter.py (1 and 2): These were my matplotlib files to plot the analytical solution compared to the numerical, and the specific kind of graph asked for in **b)**. I had to refresh a little on matplotlib's settings/structure. I used Claude to look at what a plot would even look like, and then trimmed it down to what I thought was important and changed lines to make sure I understood them.

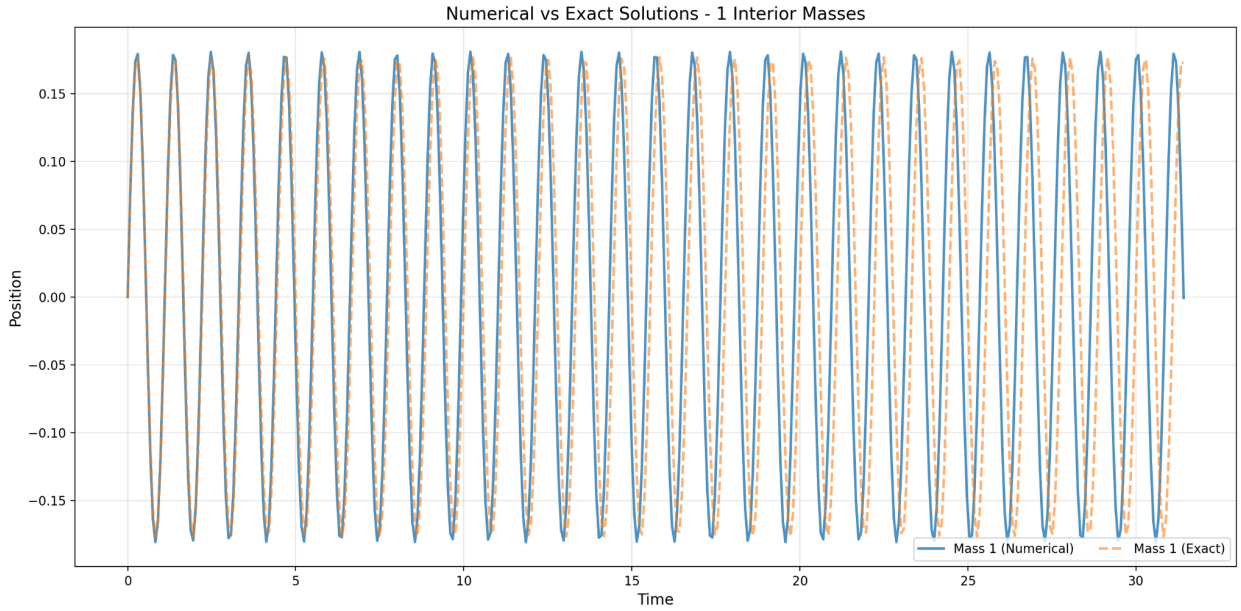
Conclusion: Using 3 main Fortran files and python to plot results, I modeled an N mass spring system and how each mass's displacement changes in 1D over time. **problemSetup.f90** set my initial conditions, **leapfrog.f90** defined a forward time step and computed the acceleration of the system, and **fput.f90** was the main driving file that called my subroutines and wrote data to a file within a data directory. **simulation.init** was my initializing file, where I could edit parameters without recompiling. The system has a nonlinear factor α which we can set to on or off by setting it equal to 0 or not, which changes the amplitude of oscillations with time. Motion of masses mimics sine and cosine functions as time moves forward.

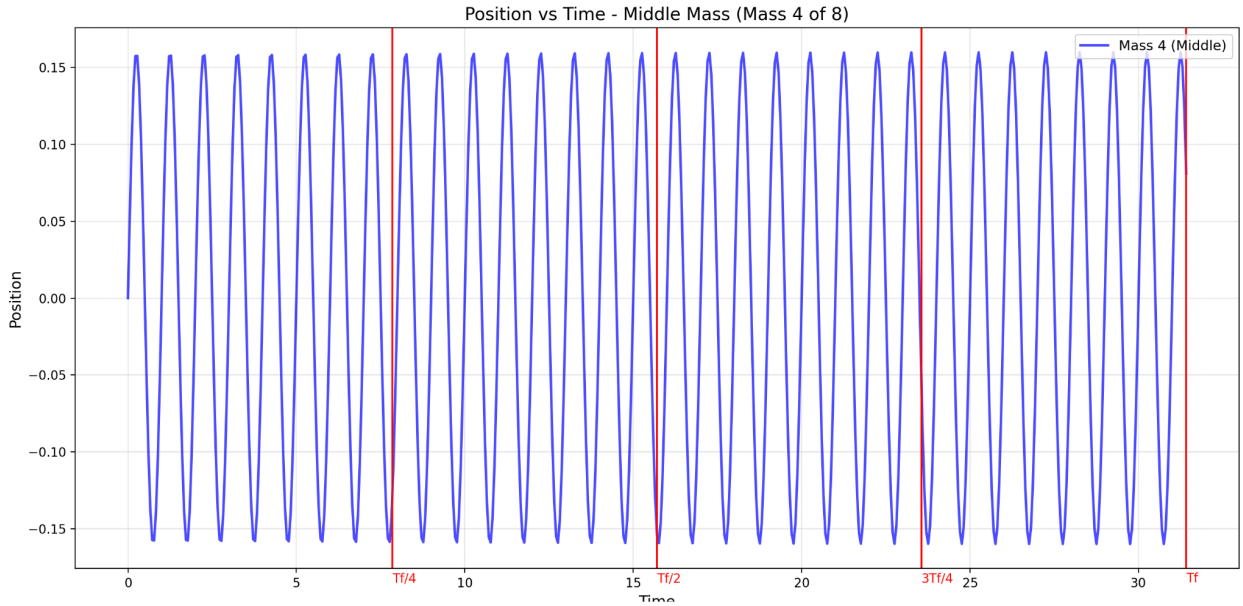
Questions:

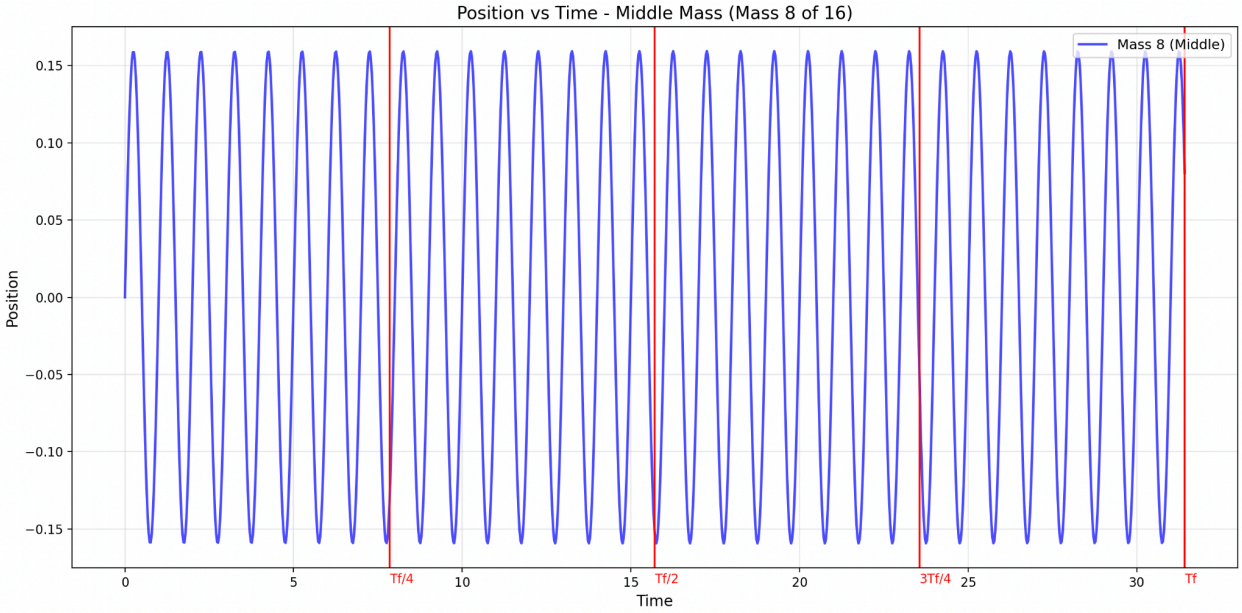
a) b) c) e) are in the images below.

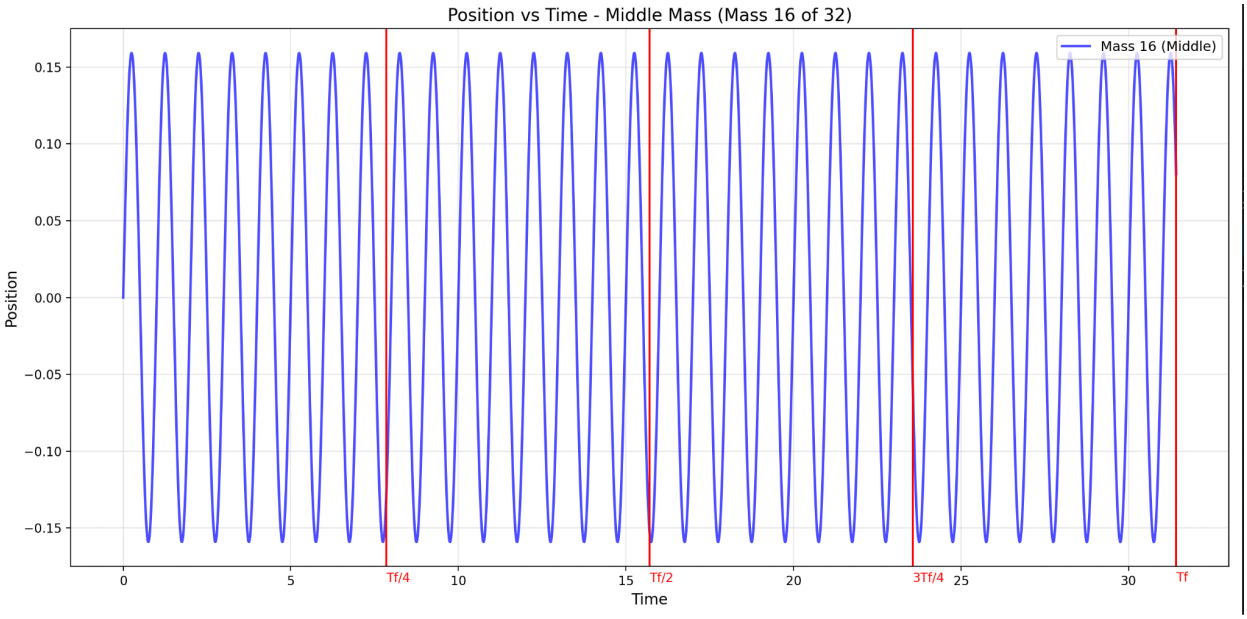
c) When C = 1, I couldn't even see the nonlinear case. When I set it to 0.5 though it looked very similar to the linear case except oscillations started to vary in size as time advanced. It was like the amplitude was changing.

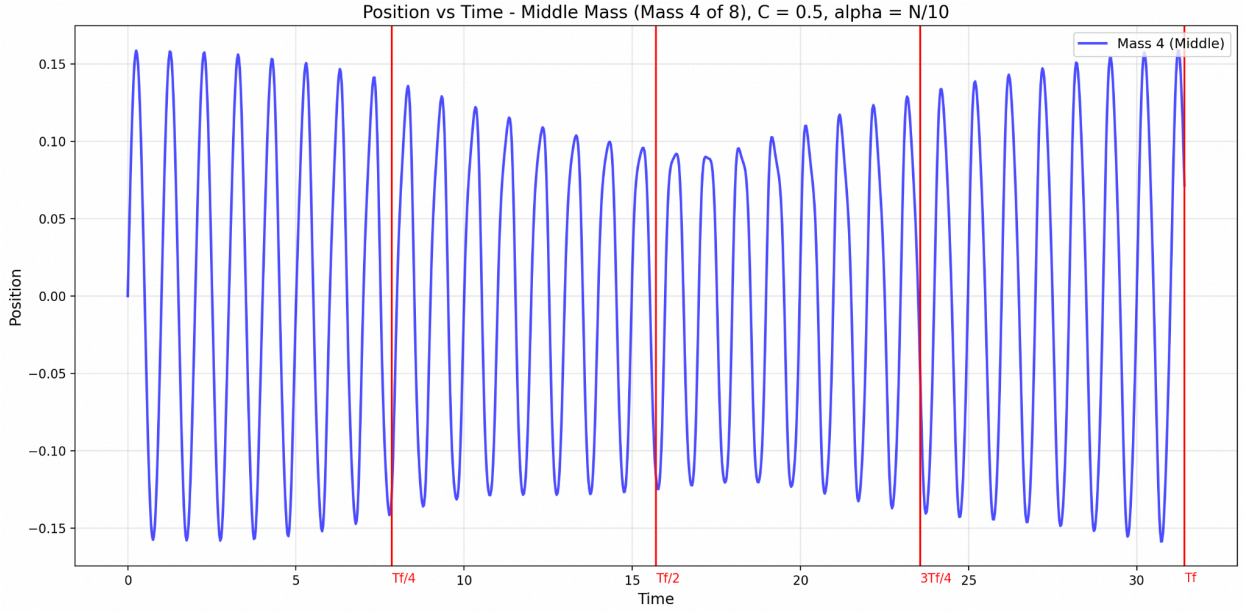
d) The closest C value I could get to 1 before breaking the nonlinear case was 0.9515.

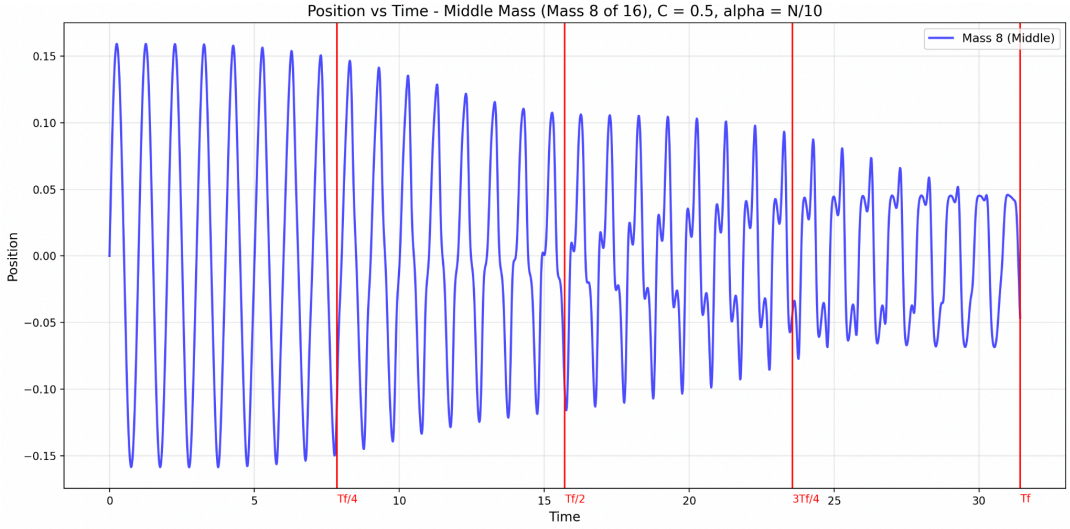

Numerical vs Exact Solutions - 1 Interior Masses
Position
Time
Mass 1 (Numerical)
Mass 1 (Exact)

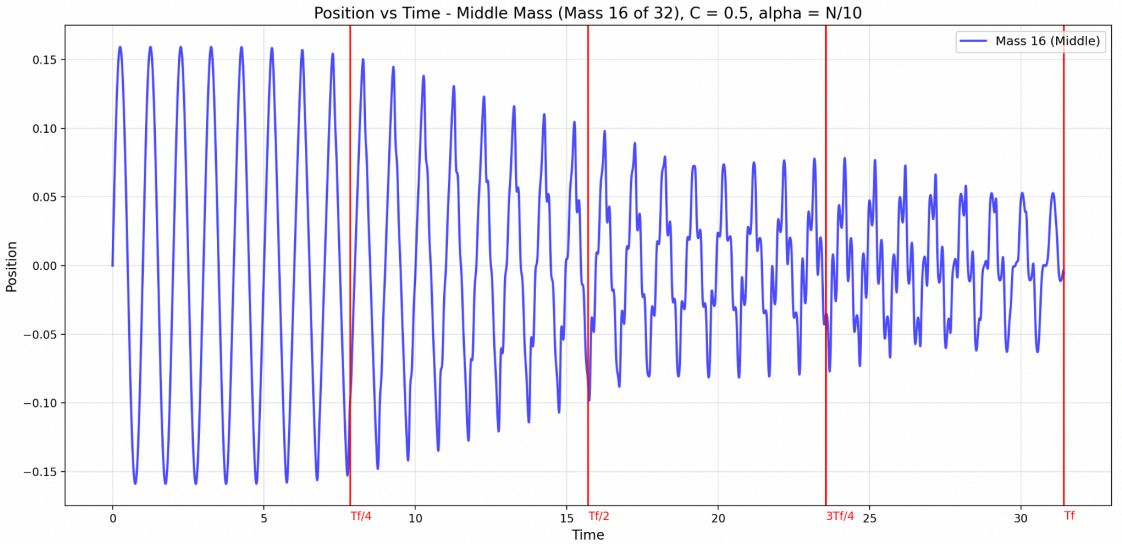

Position vs Time - Middle Mass (Mass 4 of 8)
Position
Time
Mass 4 (Middle)
Tf/4
Tf/2
3Tf/4
Tf

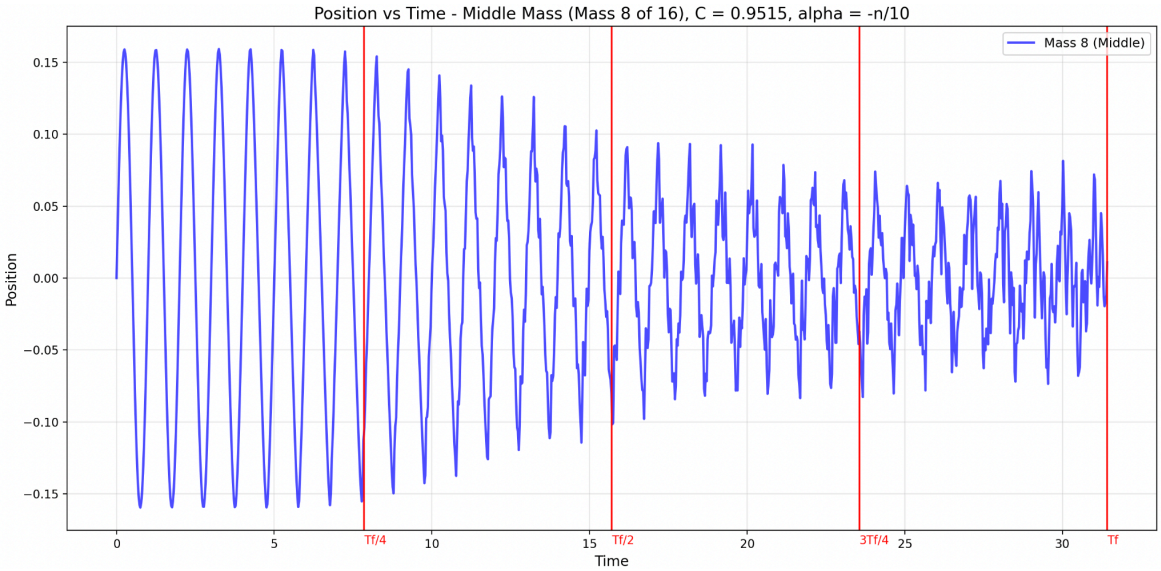

Position vs Time - Middle Mass (Mass 8 of 16)
Mass 8 (Middle)
Position
Time
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
5
10
15
20
25
30
Tf/4
Tf/2
3Tf/4
Tf


Position vs Time - Middle Mass (Mass 16 of 32)
Mass 16 (Middle)
Position
Time
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0
5
10
15
20
25
30
Tf/4
Tf/2
3Tf/4
Tf


Position vs Time - Middle Mass (Mass 4 of 8), C = 0.5, alpha = N/10
Mass 4 (Middle)
Position
Time
Tf/4
Tf/2
3Tf/4
Tf


Position vs Time - Middle Mass (Mass 8 of 16), C = 0.5, alpha = N/10
Mass 8 (Middle)
Position
Time
Tf/4
Tf/2
3Tf/4
Tf


Position vs Time - Middle Mass (Mass 16 of 32), C = 0.5, alpha = N/10
Mass 16 (Middle)
Position
Time
Tf/4
Tf/2
3Tf/4
Tf


Position vs Time - Middle Mass (Mass 8 of 16), C = 0.9515, alpha = -n/10
Mass 8 (Middle)
Position
Time
Tf/4
Tf/2
3Tf/4
Tf