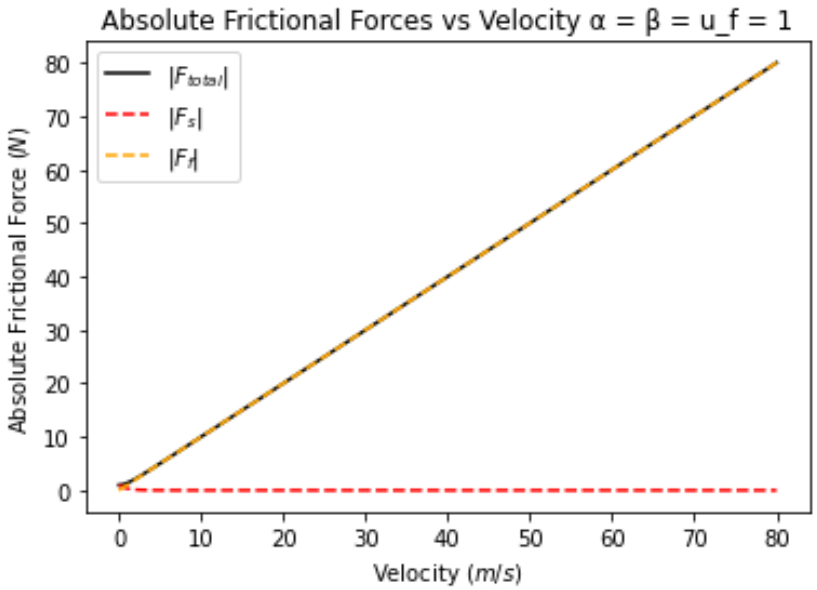
PHY407 Lab 6

Brendan Halliday and Nikolaos Rizos

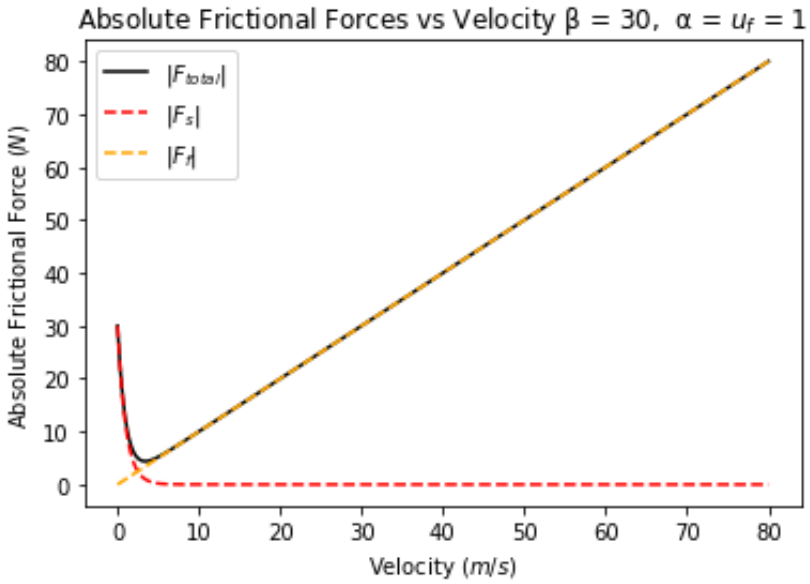
October 22nd. 2021

Q1. (a) (i)- Nikolaos

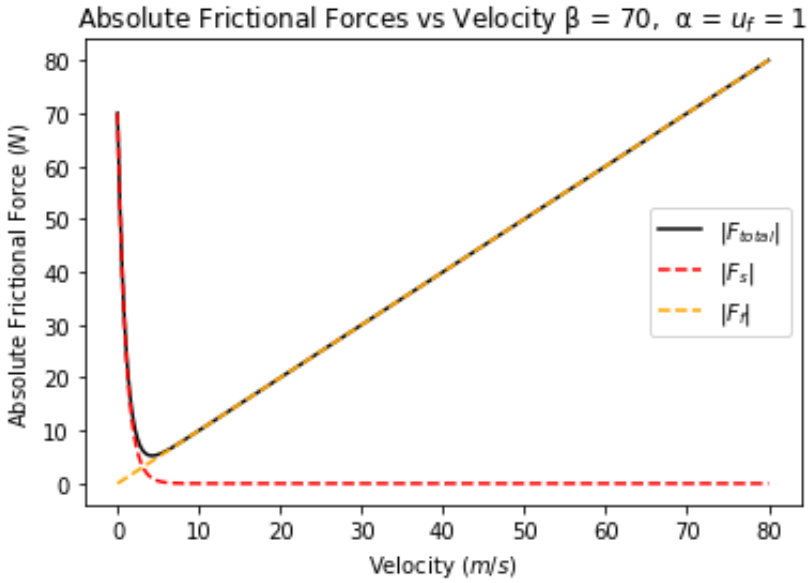
Plotting the total absolute frictional force (TAFF), for the values indicated on the title of each plot, each time:



In this initial case where all the constants have the same value, we notice a linear trend for most of the TAFF values for the given range of velocities. There is a slight curvature on both the static friction force, and the TAFF for small velocity values, which looks slightly like an exponential decay. In order to investigate this further, we increase the proportionality factor β of the static friction force and plot the results:

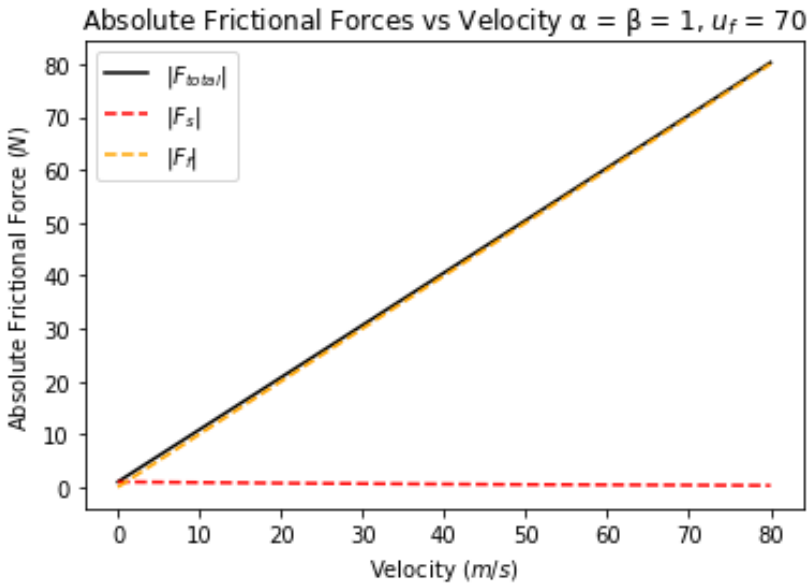


For a higher β value, we notice a clear exponential curve from $0\frac{m}{s}$ to about $5\frac{m}{s}$ for both the static friction force, and the TAFF. The increased scaling factor β appears to have amplified the exponential nature of the static friction for small velocities, making it much more prominent for velocities close to $0\frac{m}{s}$ (where before there was only a slight curvature for velocities close to $0\frac{m}{s}$, now there is a clear and distinct exponential curve, while the static friction at $u = 0\frac{m}{s}$ increased by a large factor). We increase this scaling factor further to observe the results:

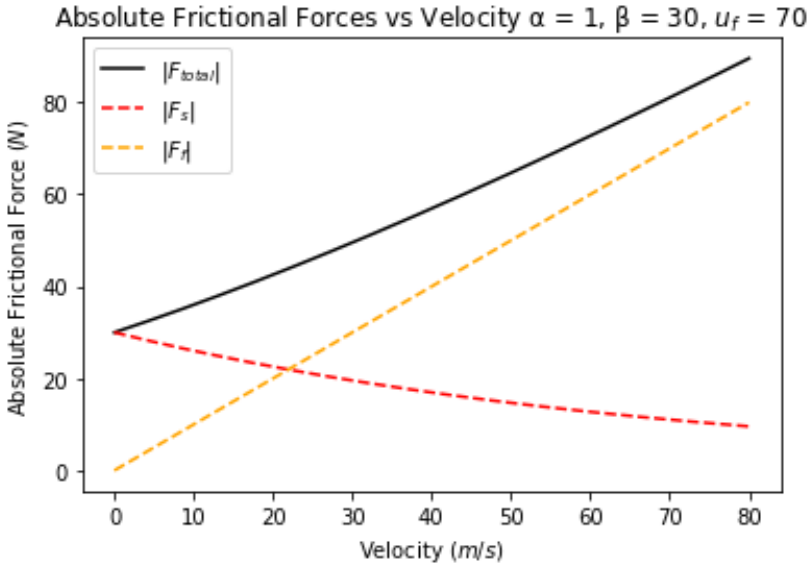


For an even larger scaling factor, we notice that the velocity value for which the static friction drops to $\frac{1}{e}$ of its original value does not shift, but the important observation is that the static friction at $u = 0\frac{m}{s}$ is much larger than before (and also equal to β in both cases). Thus, we can give the physical interpretation of β, as the value of the static friction when the object's velocity is 0, which also affects the overall strength of the static friction before it decays and becomes negligible (it makes the static friction stronger overall). The static friction also seems to dominate the TAFF for these small velocity values.

We now increase the value of u_f from 1, in order to observe the results:



For a small β value, the larger u_f seems to completely eliminate the small exponential properties of the static friction and the TAFF initially observed for a small β value, and turn the static friction and TAFF curves into linear curves. In order to further understand the meaning ofu_f, we increase the value of β while keeping the same u_f to observe the resulting curves:

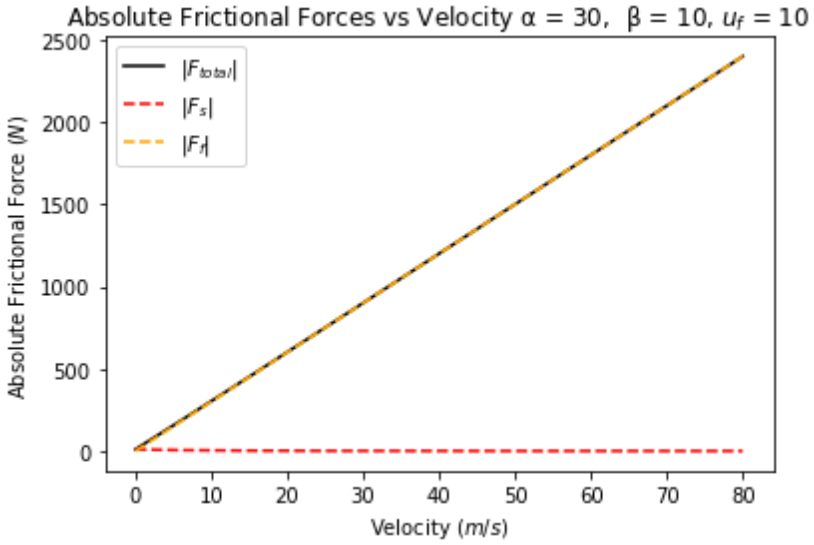


We again validate the role of β as the value of static friction for $u = 0\frac{m}{s}$, and we also notice that the difference between the two curves for $\beta = 30$ (with the second one having a larger u_f value), is that the latter appears to decay much slower (the object needs to move at much higher velocities in order for the static friction force to drop to $\frac{1}{e}$ of its original value). In the previous graph of small β but large u_f, the reason the pattern seemed linear is because the decay was much slower, while the initial value of the static friction was much smaller. Thus, the exponential decrease was so slow, it made the graph look linear (even though as is obvious from the above graph, it clearly still was an exponential decrease). The increased u_f also heavily affects the TAFF for larger velocities than before. We can physically interpret u_f as the threshold (velocity

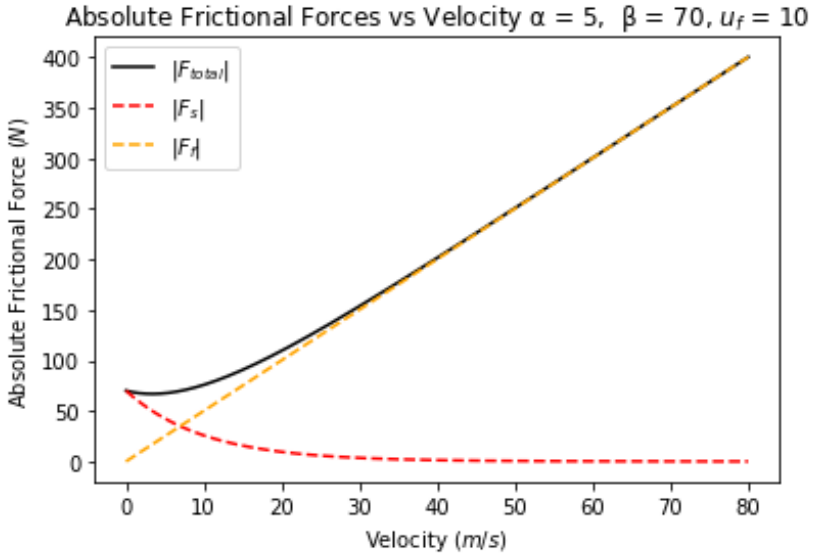
required) for the static friction to become negligible. The higher the $u_f$value, the larger the velocity required by the body in order for it to stop being affected by the static friction.

We are now interested in observing how the constants α, β, u_f affect the behavior of the TAFF when $u_{object} < u_f$. We graph the following two plots:

When $\beta < \alpha u_f$ (1):



When $\beta > \alpha u_f$ (2):

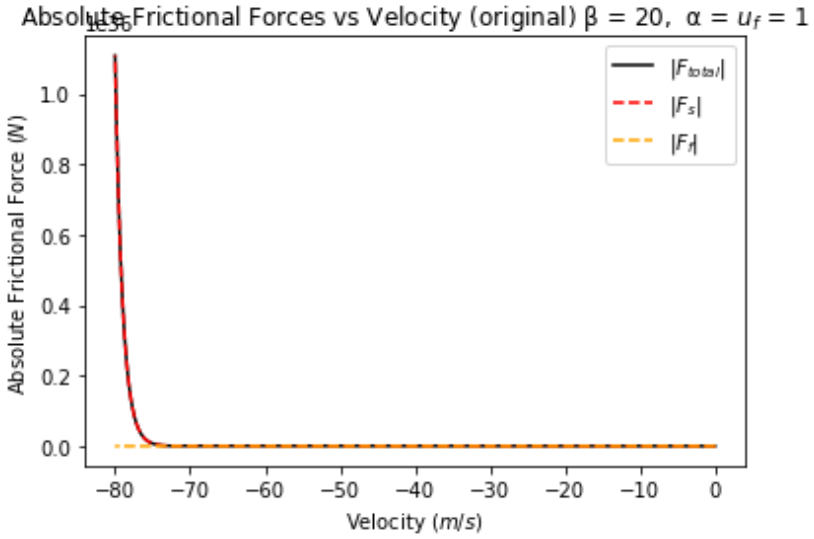


We clearly see that in case (1), the Stokes friction dominates the TAFF, by seemingly making the static friction negligible compared to the Stokes friction. That can be attributed to the scaling factor α of the stokes friction, combined with the static friction decay threshold u_f, being larger than β. In case (2) though, it seems that when the factor β is larger than the combined effect of u_f and a, the static friction (and also the TAFF) start to decay when the velocity of the object increases from $0\frac{m}{s}$, instead of the TAFF increasing linearly from $0\frac{m}{s}$. Thus, when we have a case where the static friction and Stokes friction are represented by the equations provided in the lab manual, if we want the TAFF to first decay when the object starts moving, (2) needs to hold. Otherwise, the TAFF just increases linearly from 0, as if there is no static friction. We

need this in our case, as we are trying to model how the mass' movement is affected by the transition from static friction to Stokes friction. Thus, if the Stokes friction overpowers the static friction, the static friction would be negligible, and our problem would be reduced to one where a mass moves linearly experiencing only the Stokes friction force at all times.

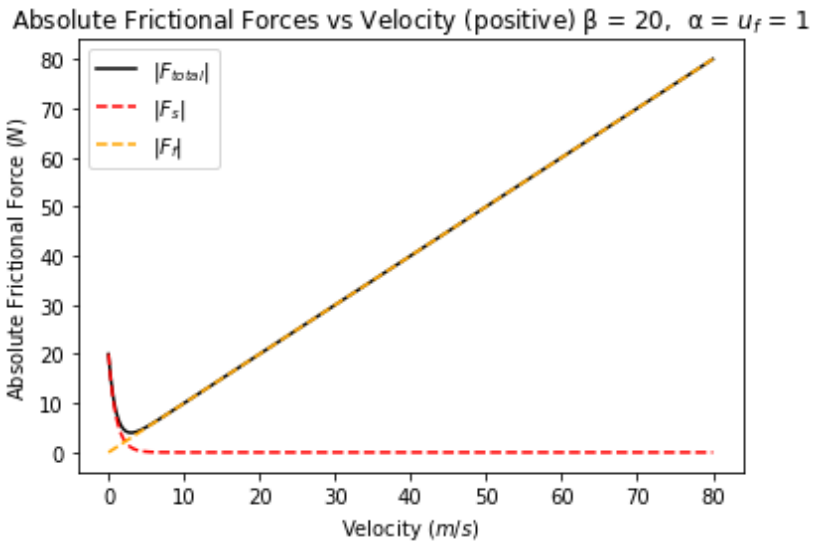
Q1. (a) (ii) - Nikolaos

Looking at eq.(16), it is clear that if the velocity vector points towards the negative direction ($u < 0$), the static friction force would behave in the opposite manner to what we want it to. That means that it would start from a miniscule, almost 0 value, and increase exponentially for larger and larger negative velocities. (shown in the graph below)

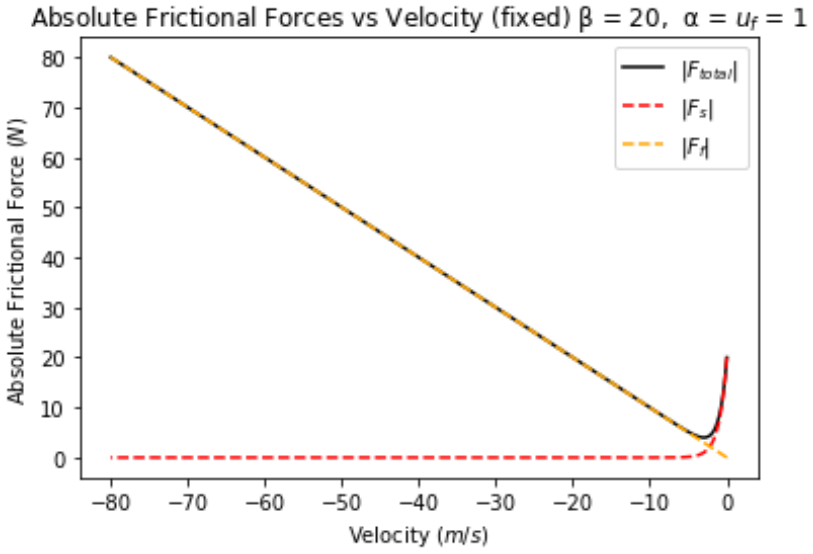


What we want is the graph to behave the same way as for positive velocities (1st graph below), but it should be symmetric about the Y axis. We can achieve that by evaluating the static friction of the object using eq.(16), but instead evaluating it using the absolute value of the velocity each time. As we can see from the second graph below, fixing our function with this method results in a graph symmetric with the one of positive velocities, about the Y axis.

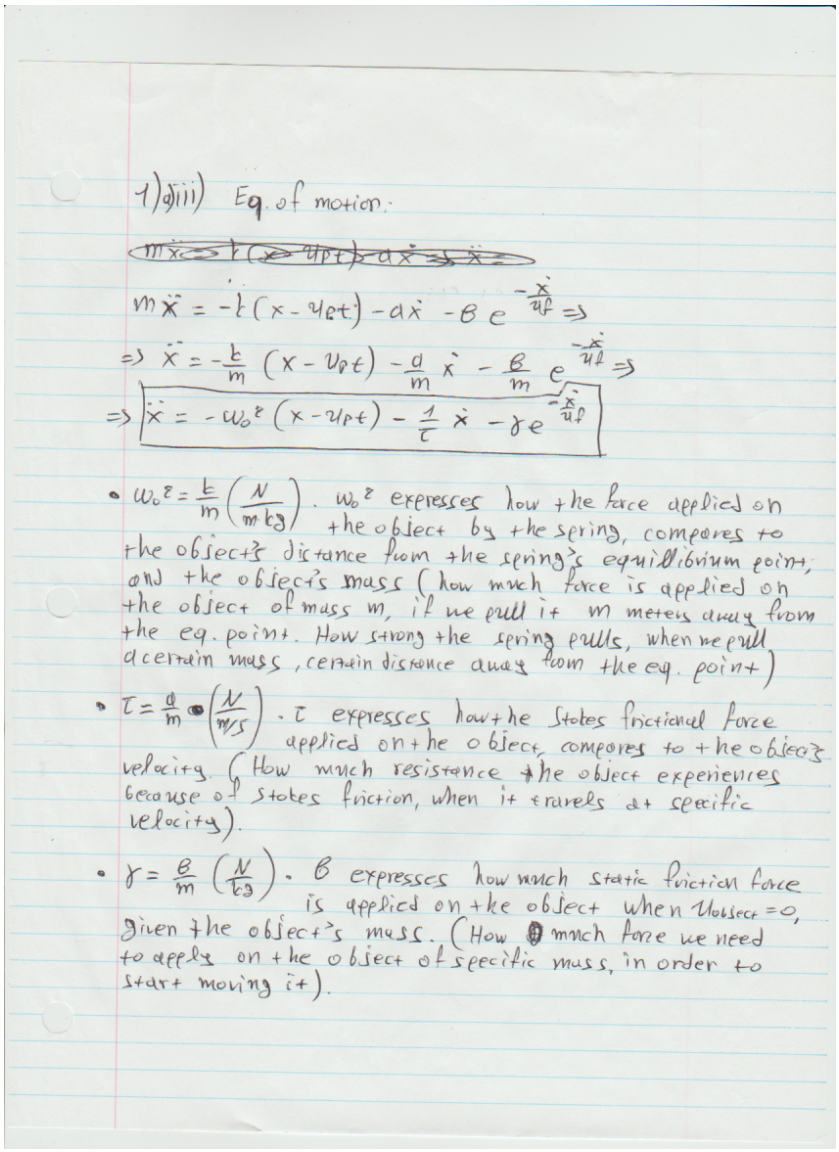
1st graph:



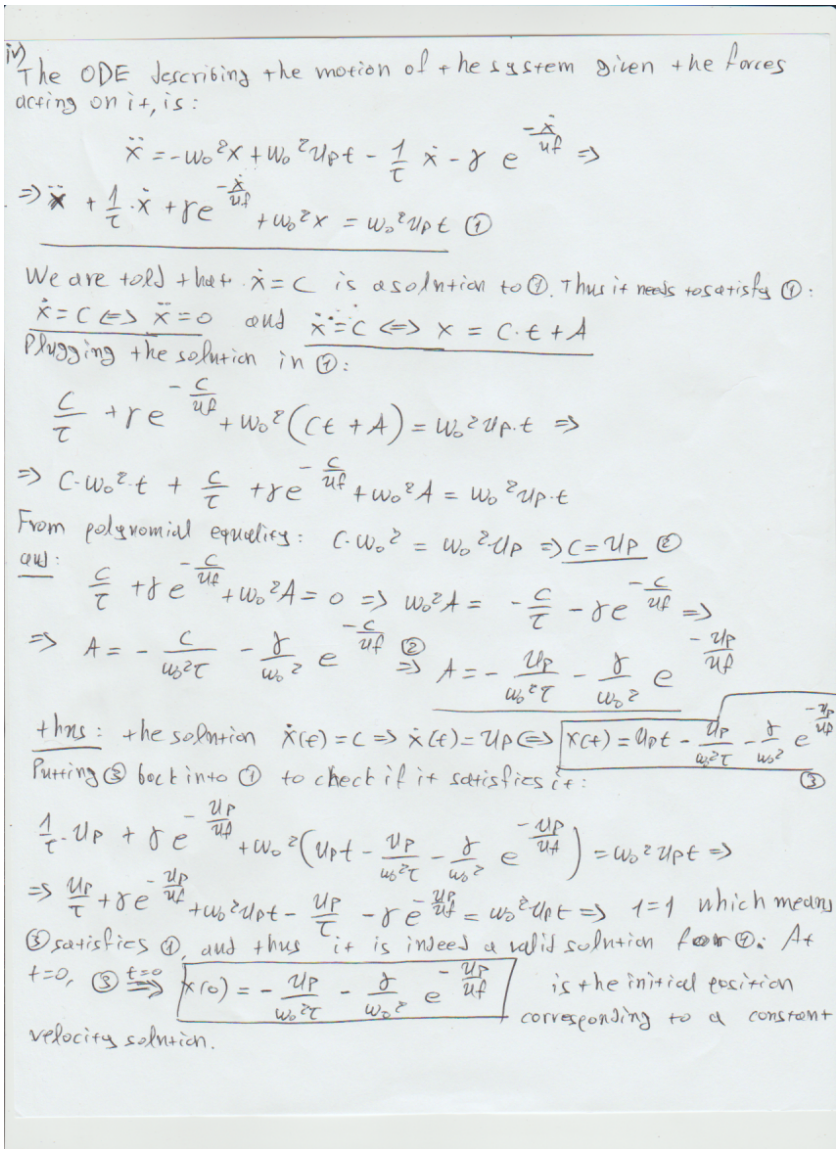
2nd graph:



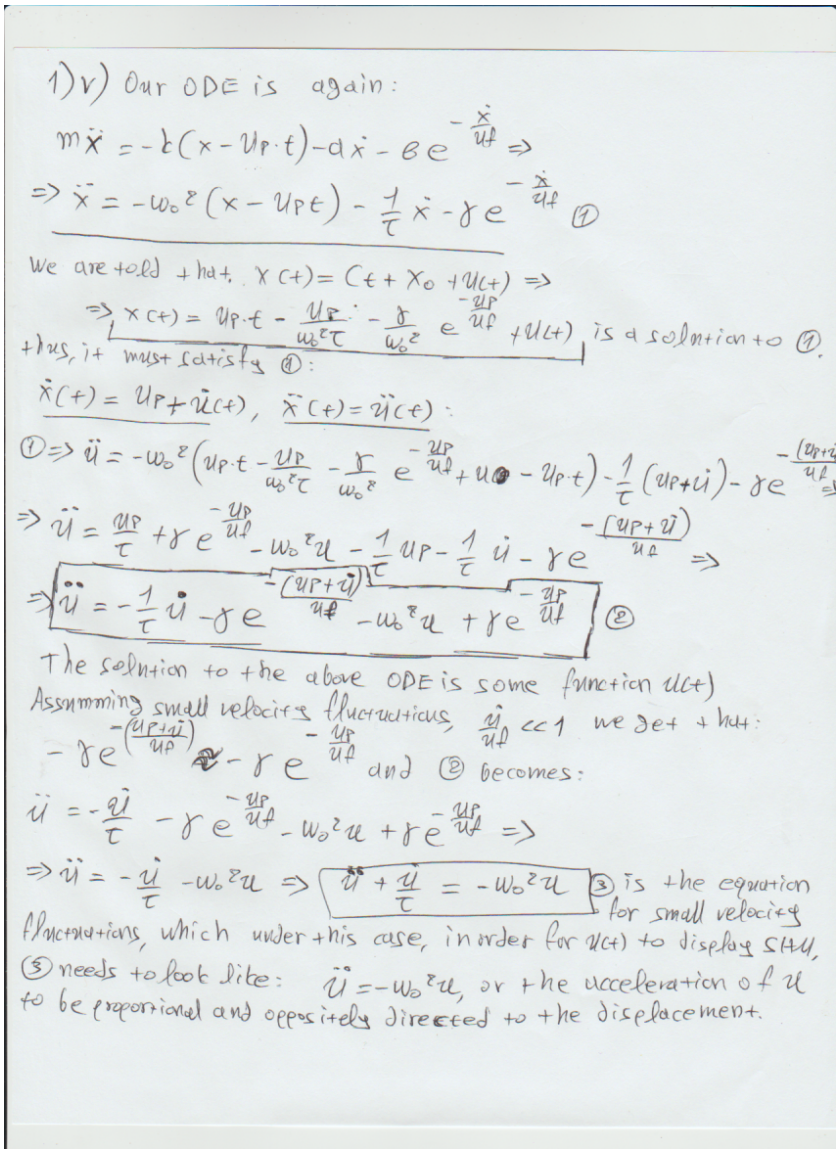
Q1. (a) (iii) - Nikolaos



Q1. (a) (iv) - Nikolaos



Q1. (a) (v) - Nikolaos

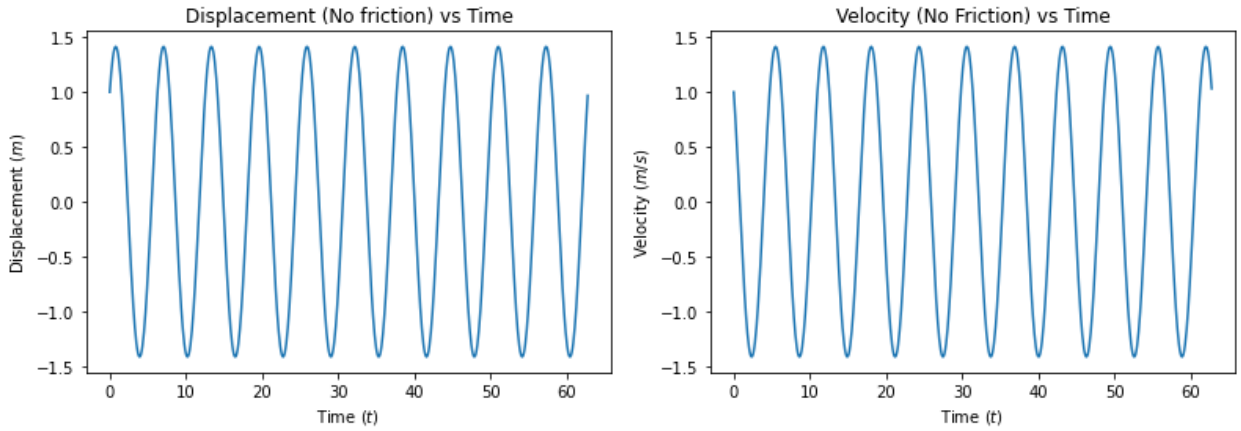


Q1. (b) (i) - Nikolaos

In this case, we are trying to solve a 2nd order equation of motion, for which when we convert it to a system of 2 1st order ODEs, the two resulting ODEs both depend on the same variable, u. The Verlet method only works in cases where the first of the two simpler ODEs depends on u but not x, and the second of the two simpler ODEs depends on x but not u. Here, both simpler ODEs depend on u, and thus the Verlet method does not apply in our case. (In this case we are restricted due to the velocity dependence of the frictional forces when applying Newton's second law in order to derive the ODE describing the motion of the system).

Q1. (b) (ii) - Nikolaos

For $\omega_0 = 1$, $u_p = 1$ and no frictional forces, the graphs of the estimates of the displacement and velocity of the object, evaluated at each of a range of 1000 equally spaced time values between $0s$and $10 \times 2\pi s$, are presented below:

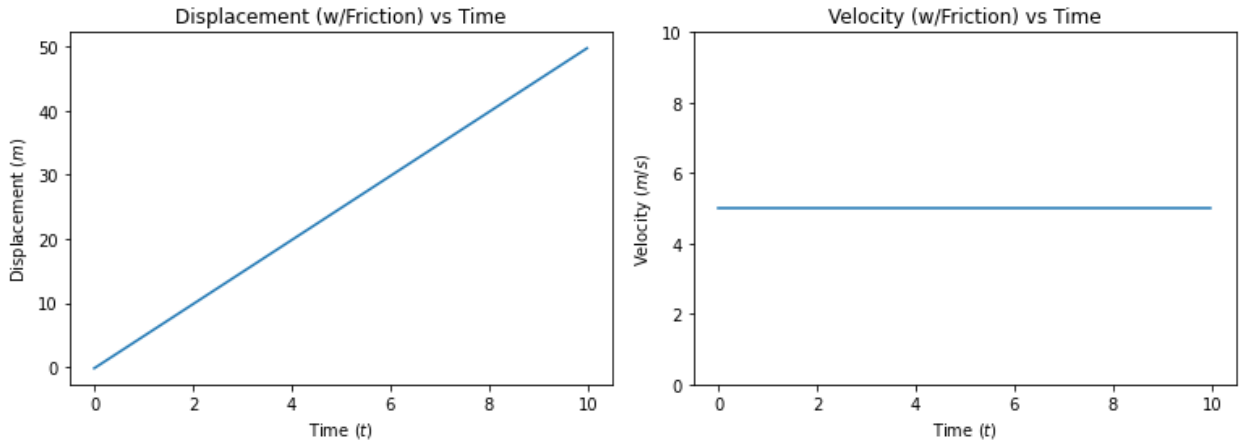


This are indeed the velocity and displacement plots corresponding to an object conducting Simple Harmonic Oscillations. The program correctly represents this expected result in this case.

In order to observe which time step causes the least decay, we ran the simulation for several values of time step h, for each of which we plotted the corresponding graph of the factor with which the total energy of the system is proportional to (if the non-constant proportionality factor decays, then the energy itself decays. Since we want to see how the energy estimates behave for different time steps, we are not interested in its true value, but how its estimated values vary for different time step values. Thus, plotting the graph of the equation to which the total energy is proportional to gives us all the information we need in order to estimate an adequate time step value). In this case, we had to make a compromise between small error, and time required to run the program. We ended up choosing a time step value of $h = 0.01$, for which the difference between the energy estimate at t_{start} , and the energy estimate at t_{end} was about $1.74 \times 10^{-5} J$, while the time required for the program to run remained low.

Q1. (b) (iii) - Nikolaos

For the given constants of: $\tau = 1\frac{N}{m/s}$,$\gamma 0.5\frac{N}{kg}$, $u_f = 0.1\frac{m}{s}$, passing the following initial conditions (at $t = 0s$) to our system, ICs: $u_0 = u_p$ and $x_0 = -\frac{u_p}{\omega_0^2 \tau} - \frac{\gamma}{\omega_0^2} e^{-\frac{u_p}{u_f}}$, we expect to have a constant velocity solution (as these ICs were evaluated by assuming a solution of the form $u = C$, or a constant velocity solution to the equation describing the motion of the system). In order to validate if our above initial conditions do indeed correspond to a constant velocity solution, we plot the displacement and velocity graphs of our complete system, for 1000 equally spaced time values between $0s$ and $10s$, and the same step size of $h = 0.01$ as determined previously (with all the frictional forces taken into consideration):

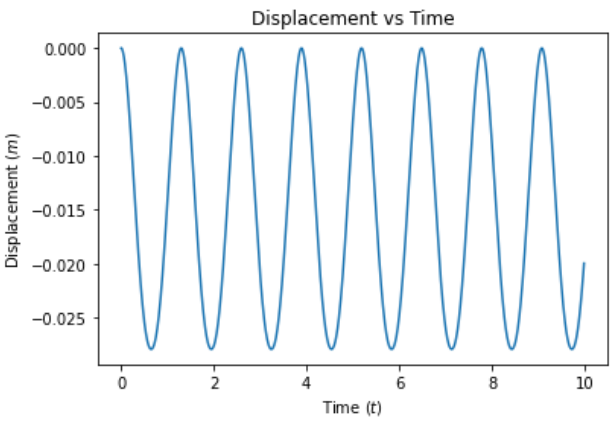


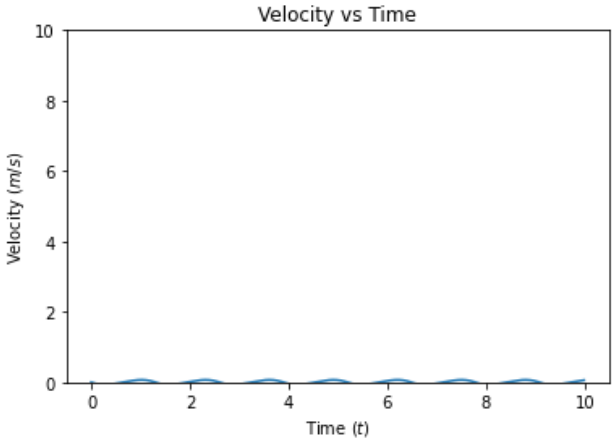
The difference between the maximum and minimum estimates of the velocity (graph right above), was determined to be about $2.99e. - 13m/s$. This means that our system when supplied the outlined initial conditions, estimated a constant velocity solution. Thus, our initial conditions did indeed correspond to a constant velocity solution (also obvious by the linear, increasing displacement graph, which also corresponds to constant velocity equal to its constant slope at any time t).

Q1. (c) - Nikolaos

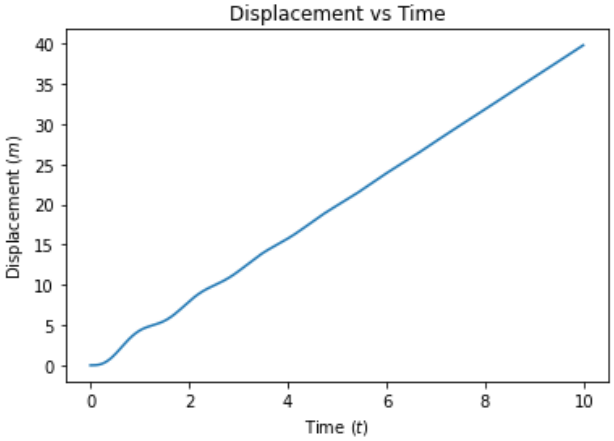
There were two notable results. The first was for the lower bound of the provided u_p values. For that, the displacement was sinusoidal, and around the same point for all the time values (i.e., the object was moving just back and forth). For this case, its velocity would go from zero, to a specific value and then back to zero, for the duration of the simulation. For the rest of the u_p values in the outlined interval, the velocity of the object would rise to a value higher than u_p initially, then would drop down below than u_p, then rise above it again and would display that behavior for about 6 to 7 times, until its velocity became almost the same as u_p. This increase and decrease pattern looked like a sine wave oscillating around a specific center, whose amplitude would decrease with each oscillation, with the graph finally stabilizing at the center of the decaying oscillation. For different u_p values, the same behavior for the velocity was observed, with the only difference being where the velocity would finally settle (as the value of u_p was increasing in each case, the center around which the decaying sine wave would oscillate would also move up as u_p increased). In these cases, the displacement of the object displayed an increasing pattern with time, but it seemed that the slope in the beginning of the simulation was not constant. It fluctuated between a flat slope (almost parallel to the x axis) and would increase, then would decrease back to an almost flat slope (although less flat than the first time) and then would increase again, until for larger time values, this fluctuation became negligible and the displacement graph approached that of the function $y = ax$, which corresponds to the behavior of the velocity for the same larger time values (the slope a corresponds to the almost constant velocity for larger time values described above). The graphs for the displacement and velocity in first case (back and forth motion given the lower bound of the u_p values provided) and in the second case (given the upper bound of the u_p values provided) are presented below for clarification:

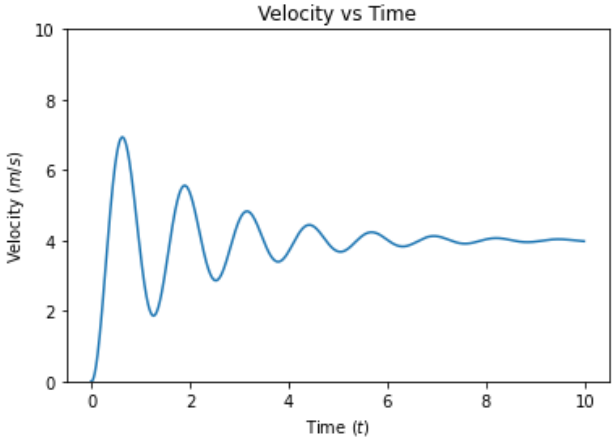
1st case:


Displacement vs Time
Displacement (m)
Time (t)


Velocity vs Time
Velocity (m/s)
Time (t)

2nd case:


Displacement vs Time
Displacement (m)
Time (t)


Velocity vs Time
Velocity (m/s)
Time (t)

The first case where the object just moved back and forth corresponds to the real-world example of a squeaky door being moved back and forth slowly. In that case, one can feel that the door sort of gets stuck and unstuck as it moves. There is no constant opposing force. The opposing force seems to increase and decrease (which coincides with a distinct squeak). The same thing happens when a violin string is touched slowly, which results in more of a squeak than a musical sound. (the same feeling exists regarding the opposing forces). The second case in our simulations where the velocity would initially oscillate a bit until it stabilized, corresponds to the squeaky door being opened or closed faster, which results in an initial squeak which then stops, or the violin string being touched faster, which causes an initial squeak for a very brief time period, which then goes away and instead the sound of a proper note replaces it. These are examples of how static friction (Which depends on the object's velocity's relation to a specific velocity value) is interchanged with the Stokes friction (which is proportional to the object's velocity). For small velocities, the Static friction (stronger force) interchanges constantly with the Stokes friction and this change between the two causes the constant squeak heard when opening the door with a small velocity. For the higher velocities, the static friction ultimately gives way to the Stokes friction, but this transition is not like flipping a switch, as initially the two friction forces interchange, but with each interchange the static friction affects the object less and less, until it becomes negligible. That is why for higher initial velocities, the squeak is initially there, it sounds less and less loud, until it disappears.

Q2. (a) - Brendan

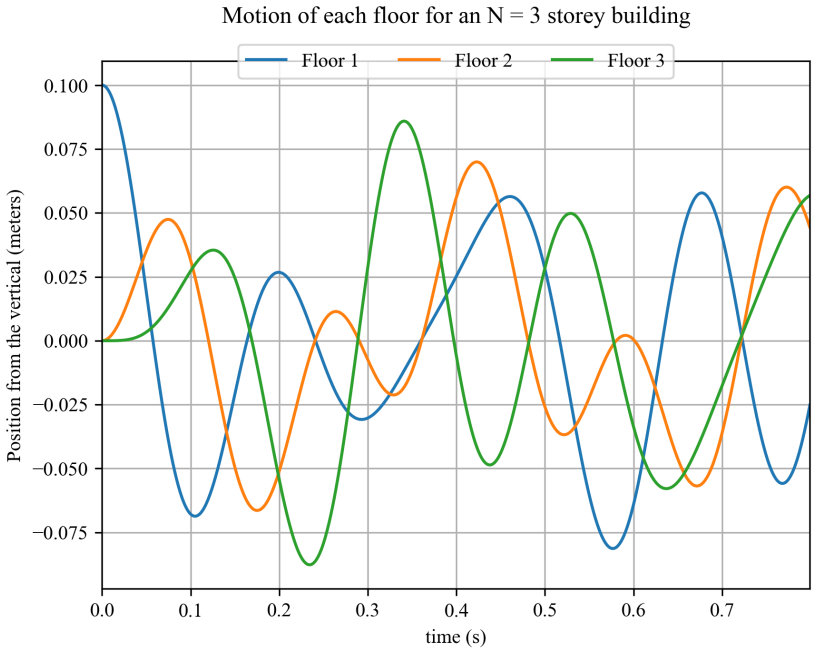
The stiffness matrix was generated directly and used to initialize the first velocity vector $\vec{v}$ and update the vector $\vec{k}$ for each iteration of the program. The following function uses the A.dot() to perform regular matrix vector multiplication as you would by hand:

```
def verlet(v_0, r_0, A, time, dt):
    """
    This function is an algorithm for the
    verlet method. This function takes in
    two initial vectors for position and velocity.
    It then outputs a list of vectors of r and
    v for each timestep.
    """
    # Empty lists to store updated vectors for each time step for
    V = [] # Velocity
    R = [] # and Position.


    # initialize velocity vector. Here we multiply r_0 by
    # a vector valued function or in our case, the N x N stiffness matrix.
    v = v_0 + 0.5 * dt * A.dot(r_0)

    V.append(v) # append velocity vector to V
    r = r_0 # initialize position vector
    R.append(r) # append position vector to R

    for i in range(len(time)-1):
        r = r + dt * v # new position vector
        k = dt * A.dot(r) # new k vector
        v = v + k # new velocity vector
        # Remember that each element in r and v
        # correspond to the position and velocity
        # of a specific floor...
        R.append(r)
        V.append(v)
        # ...whereas each array in R and V correspond to
        # the entire motion of the building for a specified
        # time.

    return R, V # Each array in R and V are vecotrs for each time step
```

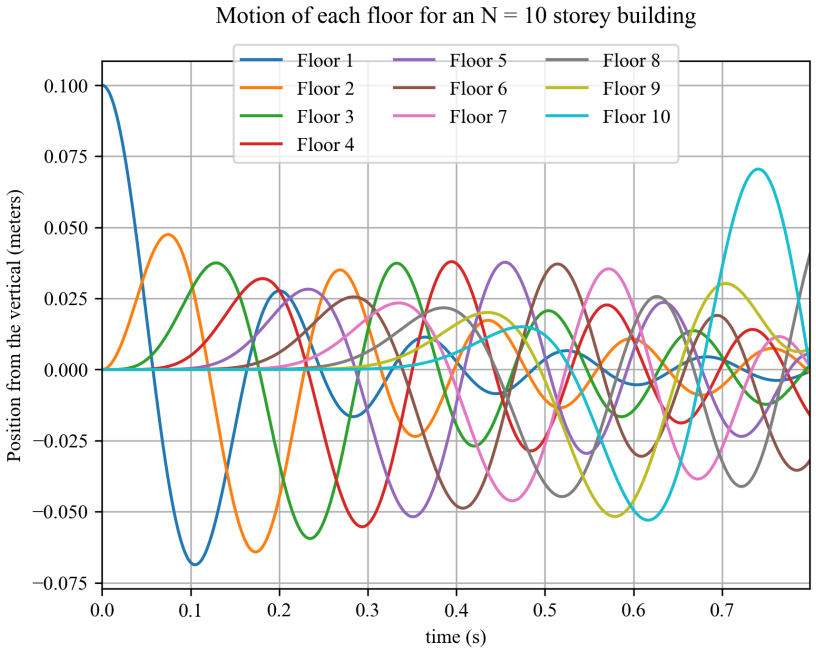
Essentially, the matrix A is the vector valued function $\vec{f}$. When A is dotted with the position vector $\vec{r}(t)$ it takes on the same value as $\vec{f}(\vec{r}(t), t)$ for some particular t. To see how the matrix A was generated for any

N and $\frac{k}{m}$, view the code for the function $A(\frac{k}{m}, N)$ included with the report.

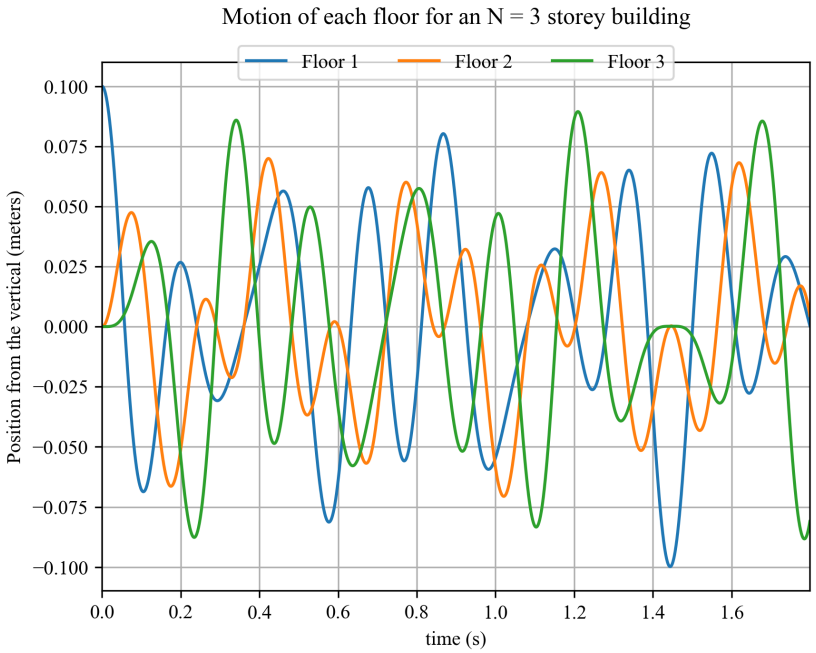
The following are graphs for the motion of a 3 storey and a 10 storey building given a slight perturbation to the first floor only:



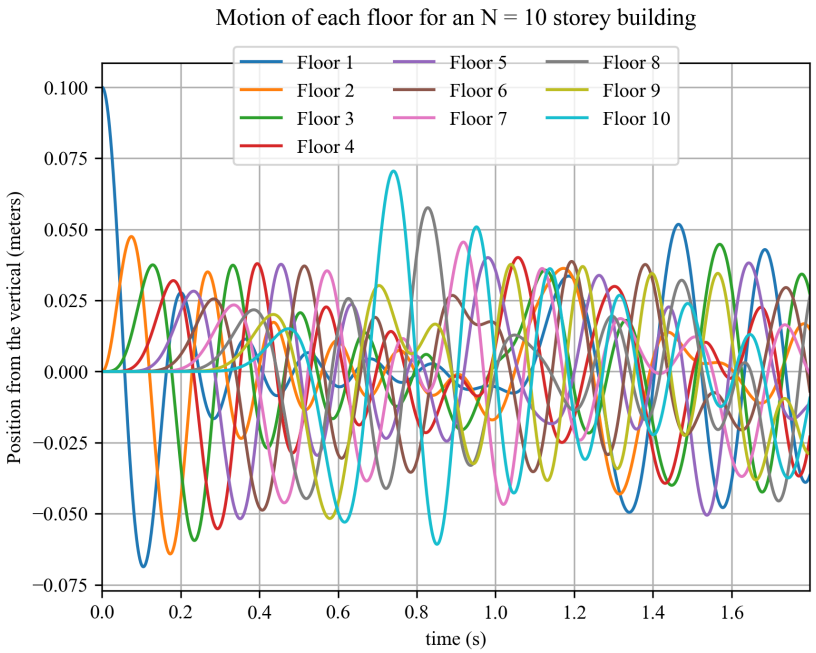
For the first graph, we see that the initial displacement of the first floor from the vertical sets the other floors in motion in order of increasing floor number.



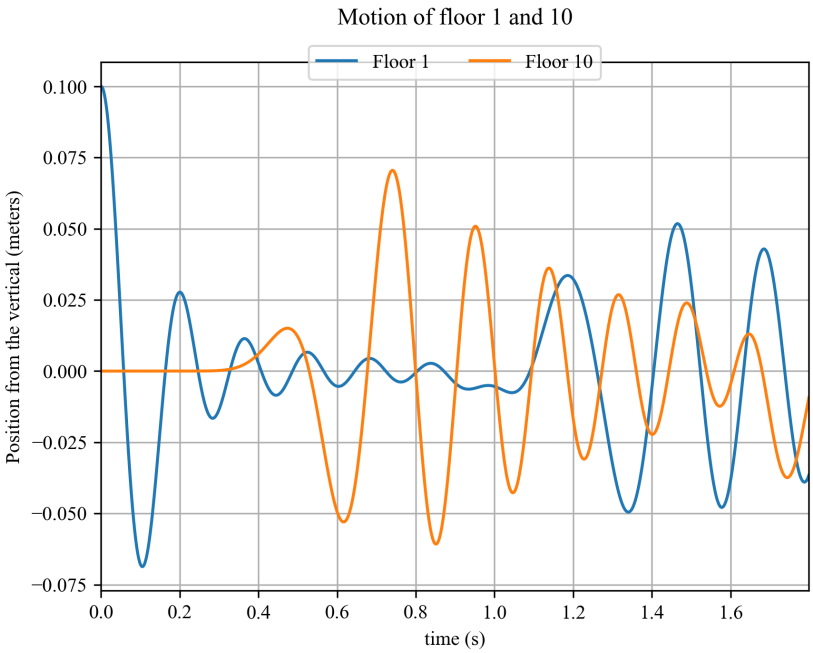
In the above graph, we see the same phenomenon as before where each floor only begins to move when the floor below it is already in motion. This shows us a wave propagating through the building. Now increasing the time scale, we have these graphs:



As the first floor grows in amplitude, the third floor dies off. As the third floor grows in amplitude, the first floor dies off. This is evidence of wave propagation in the building.



For $N = 10$, the motion seems rather chaotic (I'm using this term in the colloquial sense). However, we can see that the tenth-floor peaks in amplitude as the first floor dies off. After some time, the tenth floor dies off and the wave seems to return to the first as it peeks. This behaviour can be seen more clearly if you extract only the first and the tenth floor:



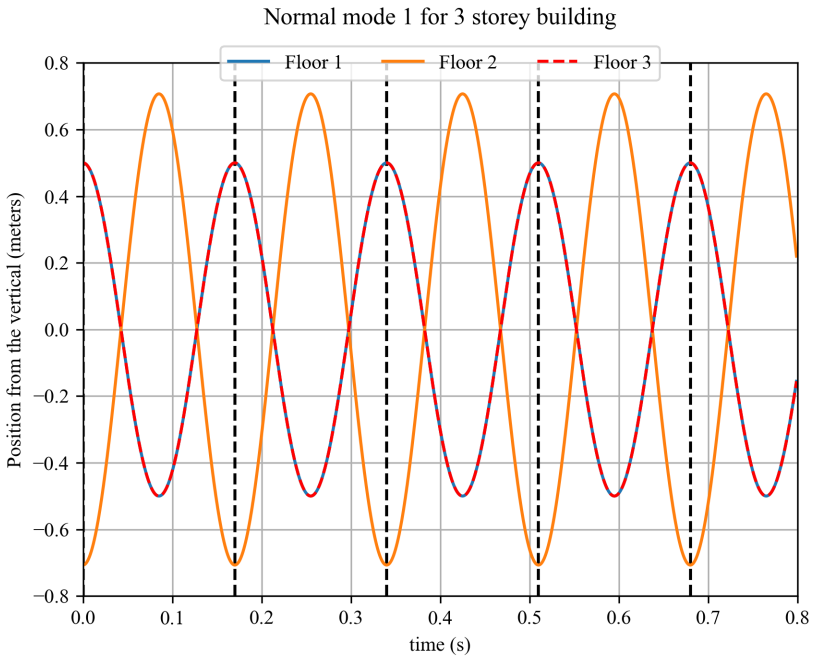
Q2. (b) - Brendan

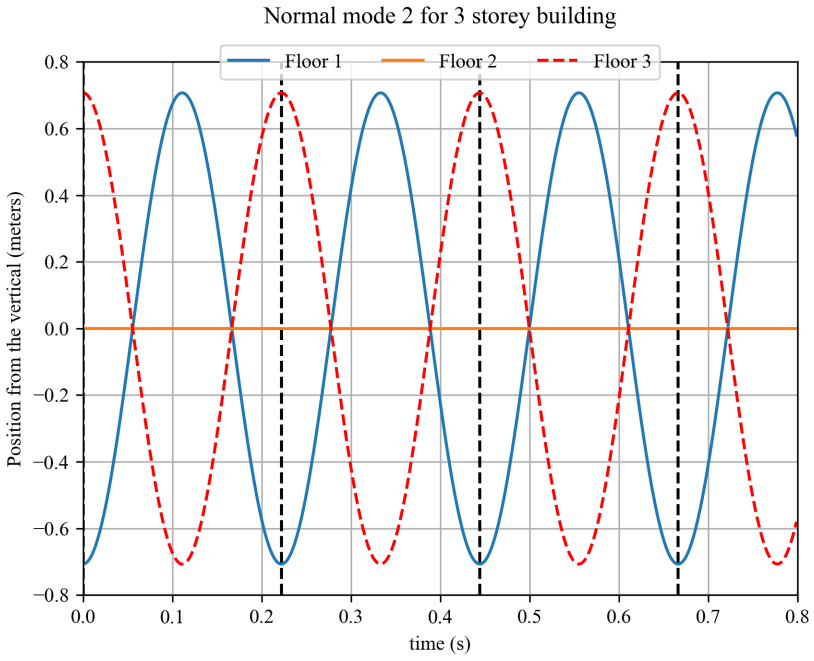
The program written to calculate the normal modes and normal frequencies using np.linalg.eigh(A) gives the following output.

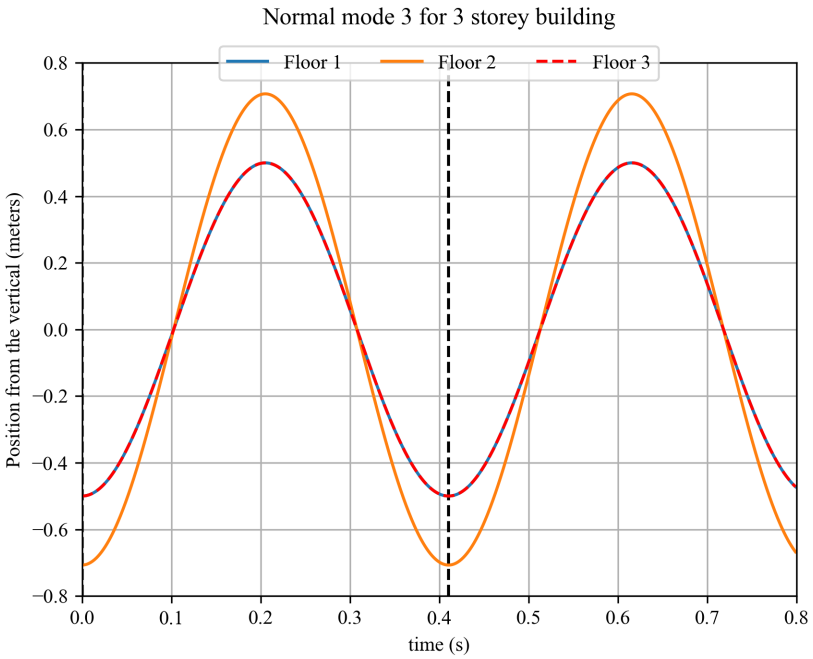
Table 1: Eigenvectors and normal frequencies corresponding to each normal mode

	1st Mode	**2nd Mode**	**3rd Mode**
Normal Frequency	5.88 Hz	4.50 Hz	2.44 Hz
Eigenvector $\hat{x}$	$(0.50, -0.71, 0.50)$	$(-0.71, 0.0, 0.71)$	$(-0.50, -0.71, -0.50)$

Now for each normal mode, the eigenvector was taken as the initial condition for the displacement in $\vec{x}_0$. These initial conditions were fed into the Verlet algorithm function and the program outputs the following graphs:


Normal mode 1 for 3 storey building
Floor 1
Floor 2
Floor 3
Position from the vertical (meters)
time (s)
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8


Normal mode 2 for 3 storey building
Floor 1
Floor 2
Floor 3
Position from the vertical (meters)
time (s)
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8



Where the vertical dotted lines are the periods associated with the normal frequencies for each mode. The periods line up with the periods of the sinusoids, thus the normal modes calculated using eigenvalue analysis are indeed those of the building.