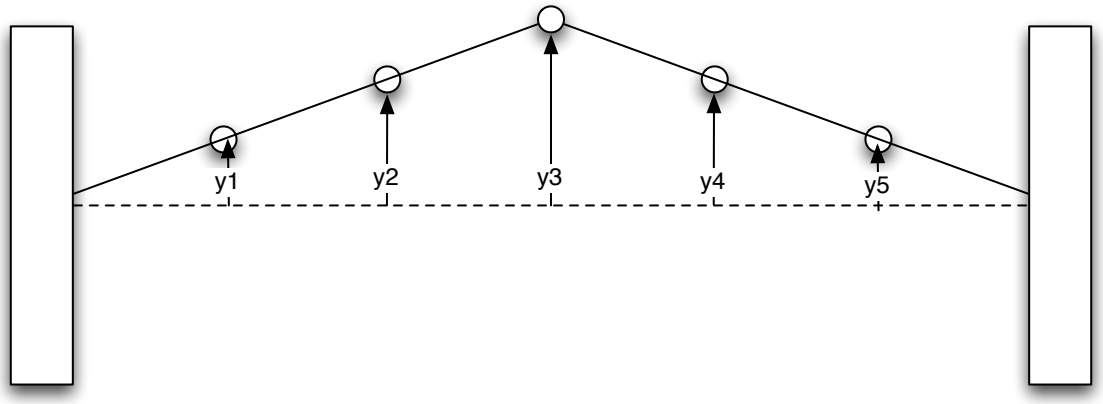

Figure 1: A set of 5 beads connected by elastic springs

Consider the simple model of an elastic string shown above. Here the string is modeled as n distinct point masses linked together by springs. The ends of the string are fixed and have zero displacement. We can model the vertical forces acting on each mass element as follows:

$$m\ddot{y}_i = k(y_{i-1} - y_i) + k(y_{i+1} - y_i) - c\dot{y}_i$$
$$\Rightarrow \ddot{y}_i = \frac{k}{m}(y_{i-1} - 2y_i + y_{i+1}) - \frac{c}{m}\dot{y}_i$$

Here the k captures the spring constant of the elastic links and the c captures the viscous drag/friction that each of the elements contends with.

We can aggregate the equations for all of the nodes into matrix form as shown below:

$$\frac{d}{dt}\begin{pmatrix}\dot{\mathbf{y}}\\ \mathbf{y}\end{pmatrix} = \begin{pmatrix}\frac{-c}{m}I & \frac{k}{m}T\\ I & 0\end{pmatrix}\begin{pmatrix}\dot{\mathbf{y}}\\ \mathbf{y}\end{pmatrix} \tag{1}$$

Where $\mathbf{y} = \begin{pmatrix}y_1 & y_2 & \cdots & y_n\end{pmatrix}^T$ and T is a tridiagonal matrix:

$$T = \begin{pmatrix}-2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & & & & & \\ & & & & 1 & -2 & 1 \\ & & & & & 1 & -2\end{pmatrix}$$

Note that you saw this tridiagonal matrix before when we were discussing sparse linear systems.

In your experiments you should simulate a string with $n = 21$ point masses, the spring constant $k = 5$ the damping coefficient $c = 0.01$ and the mass of each element $m = 1$.

Recall that the solution to a linear differential equation of the form $\frac{d}{dt}\mathbf{z}(t) = A\mathbf{z}(t)$ is given by. $\mathbf{z}(t) = e^{At}\mathbf{z}(0)$. This expression can be used to predict how the system will evolve over time.

Part 1

Your first task is to write a matlab script that can be used to visualize how the string will evolve if we pluck it in the center and let it evolve. You should create a vector, $\mathbf{z}(0)$ representing

the initial state where the velocity of all n beads is 0 and the position is set so that the string is in a triangular shape as shown in the figure. The maximum displacement should be 1. Your simulation should then compute what the state of the string will be at each subsequent timestep and plot what it will look like using the equation $\mathbf{z}(t) = e^{At}\mathbf{z}(0)$. You can look at the matlab script entitled plotExample which shows how you can go about producing a plot that changes over time. Your simulation should cover at least 10 seconds and the timestep between each simulation step should be 0.1 seconds.

In order to tackle this part you are required to write the following Matlab functions:

StringMatrix : 15 pts

You will start by writing a matlab function that will generate the matrix shown in Equation 1.

`function A = StringMatrix (n, k, c, m).`

This function should produce a $2n \times 2n$ matrix based on the parameters, n, k, c, m.

ComputeState : 15 pts

You will write a function that you can use to predict the evolution of a system governed by a differential equation. More specifically you will implement a function that implements the equation described above $\mathbf{z}(t) = e^{At}\mathbf{z}(0)$.

`function zt = ComputeState (A, t, z0)`

In this function zt represents the state of the system at time t, A represents the relevant matrix, t represents the amount of time elapsed and $z0$ is a vector representing the initial state of the system. Again you can use the `expm` function which computes the exponential of a matrix.

Part 2 : 10 pts

The eigenvalues and eigenvectors of the matrix $A = \begin{pmatrix} \frac{-c}{m}I & \frac{k}{m}T \\ I & 0 \end{pmatrix}$ reveal the modes of vibration and the associated *resonant frequencies*, the complex parts of the eigenvalues.

You will write the following function:

`function resonances = ComputeResonantFrequencies (A)`

Which takes as input a square matrix A governing the linear differential equation, finds all of the eigenvalues using the `eig` function, computes the imaginary parts of those eigenvalues using the `imag` function then returns a list of these resonant frequencies sorted in increasing order using the `sort` function. Remember that the eigenvalues will occur in *complex conjugate pairs* so if $a + ib$ is an eigenvalue $a - ib$ will be as well. You can disregard negative frequency values or just consider the absolute values.

Forced System : 20 pts

In this part of the project you will simulate and study what happens if we hook the first element in the string to an oscillator which bounces up and down. Here we end up solving a differential equation of the following form:

$$\dot{\mathbf{z}}(t) - A\mathbf{z}(t) = (\sin \omega t)\mathbf{b} \tag{2}$$

In this case $\mathbf{b} \in \mathbb{R}^{2N}$, $\mathbf{b} = \frac{k}{m}\begin{pmatrix}1\\0\\\vdots\\0\end{pmatrix}$. And the term $(\sin \omega t)\mathbf{b}$ models the forcing function.

The solution to this differential equation is:

$$\mathbf{z}(t) = e^{At}(\mathbf{z}(0) - \mathbf{c}_2) + ((\sin \omega t)\mathbf{c}_1 + (\cos \omega t)\mathbf{c}_2) \tag{3}$$

Where $(\omega I + \frac{1}{\omega}A^2)\mathbf{c}_2 = -\mathbf{b}$ and $\mathbf{c}_1 = \frac{1}{\omega}A\mathbf{c}_2$

For this part you must write the following function:

```
function zt = ComputeForcedState (A, t, b, w)
```

In this function zt represents the state of the system at time t, A represents the relevant matrix, t represents the amount of time elapsed, the vector b and the scalar w capture the forcing function shown in equation 2. Your function should implement the solution given in 3.

You will use this function to write a script to simulate the behavior of the system. The timestep of your simulation should be 0.1 seconds and your simulation should cover at least 100 seconds. You should simulate for various values of ω including 0.5, 1.5 and 5. You should also run the simulation when you set ω to be the first second and third resonant frequencies. Do you notice anything interesting when you drive the system with these resonant frequencies?

Extra Credit : 20 pts

Note that we could extend what we have done here with strings to deal with two dimensional membranes like the surface of a trampoline. You can look at Matlab's `vibes` demo for inspiration. Here we just need to modify our equation to reflect the fact that each point mass is connected to four of its neighbors, north, south, east and west. Your job is to write a script similar to the one that you wrote in the first part that will simulate a trampoline composed of a 20×20 array of point masses with the same values of k, c and m as before. The initial state should be a plucked membrane where the center of the trampoline is indented, the 2D equivalent of the plucked string. The edges of the membrane are all fixed to zero as is the case with the string in Part 1. You can use the `mesh` function to plot the 2D membrane. Your script should also compute and print out the resonant frequencies of this system.