The eigenfrequencies of a system containing N masses interconnected with springs on a one dimensional frictionless surface

Thidius

June 2022

1 Introduction

The setup is as follows. Suppose we have N masses of mass m, all interconnected with springs whose spring constant is k. The masses are restricted to move on a horizontal and frictionless surface, and the most outer "blocks" are connected to walls through the springs. See Figure 1 for some reference.

The solution to this problem for $N = 1$ is trivial, and can be found in most classical mechanics books. Some might even have had a glimpse of the case $N = 2$. However, what if we extend the problem? How do the masses behave when we let $N \to \infty$, and does there exist a closed form solution for the frequencies (*eigenfrequencies*) of the individual masses? These are just some of the questions that'll be encountered in this paper.

In order to land at the equations of motion, we'll utilize the power of the Lagrangian and whose equations (*Euler - Lagrange's equations*) are derived from variational calculus. Through this, we'll hopefully be able to derive the analytical solutions that'll help us simulate the how the motion of this system develops over time from some set of initial values.

1.1 Setting up the Lagrangian and deriving the matrix equation

In order to use the Lagrangian formalism, we first have to introduce some set of generalized coordinates $q_1, ..., q_n$.

The purpose of these generalized coordinates is that they've to be able to determine the configuration of this system at any moment of time. Too many generalized coordinates are redundant, and with too few of them, we can't fixate the configuration.

In this case, we have N masses whose motion are restricted to what we can label as our x - direction. It therefore becomes trivial that we let $x_1, ..., x_n$ be our set of generalized coordinates, such that x_i is the coordinate which originates from the center of mass of block i.

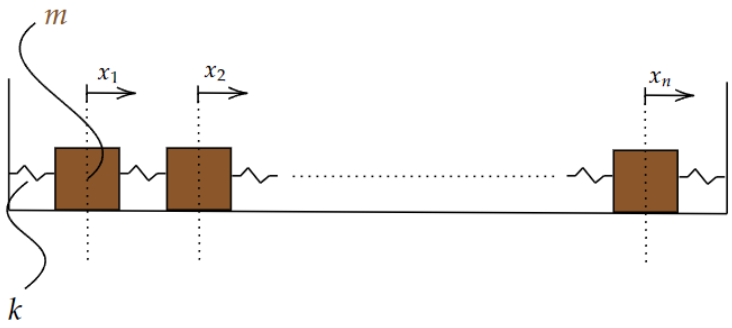

Figure 1: *A visual diagram over the arrangement of blocks and springs for the given setup with the already introduced generalized coordinates* $x_1, ..., x_n$.

After the generalized coordinates have been introduced, we have to find the kinetic energy T and the potential energy V of the system in order to form the Lagrangian.

The kinetic energy is just the sum of the individual kinetic energies of each and every block. Meaning:

$$T = \frac{m}{2} \sum_{i=1}^{n} \dot{x}_i^2$$

and the potential energy is just the sum of the potential energy of each spring. Notice that the contraction of each spring is the displacement of the block in front of it, minus the displacement of the block behind it. Hence:

$$V = \frac{k}{2}\sum_{i=1}^{n+1}(x_i - x_{i-1})^2$$

where we let $x_0 = x_{n+1} = 0$ in order for the potential energies of the springs that're connected to the wall to satisfy their correct values. Now, we are ready to form the Lagrangian:

$$\mathcal{L} = T - V = \frac{m}{2}\sum_{i=1}^{n}\dot{x}_i^2 - \frac{k}{2}\sum_{i=1}^{n+1}(x_i - x_{i-1})^2$$

We apply Euler - Lagrange's equation for some arbitrary generalized coordinate x_q from our previously defined set of generalized coordinates. Since our system is conservative, we get:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{x}_q} - \frac{\partial \mathcal{L}}{\partial x_q} = 0$$

Plugging in the Lagrangian yields us:

$$\begin{aligned}
&\frac{d}{dt}(m\dot{x}_q) + k(x_q - x_{q-1}) - k(x_{q+1} - x_q) = 0 \\
&\Leftrightarrow m\ddot{x}_q - k(x_{q-1} - 2x_q + x_{q+1}) = 0 \\
&\Leftrightarrow \ddot{x}_q + \frac{k}{m}(-x_{q-1} + 2x_q - x_{q+1}) = 0
\end{aligned}$$

Since we have derived this for a general q, we can now form the equivalent system of 2nd order ODE's that have to be solved for this system. We present it on its' corresponding matrix form:

$$\underbrace{\begin{pmatrix} \ddot{x}_1 \\ \ddot{x}_2 \\ \vdots \\ \ddot{x}_n \end{pmatrix}}_{:=\ddot{X}} + \frac{k}{m}\underbrace{\begin{pmatrix} 2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & -1 & 2 \end{pmatrix}}_{:=M_n}\underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}}_{:=X} = 0$$

Notice the beautiful symmetry of the matrix above. For convenience and future reference, we'll label the $n \times n$ matrix as M_n.

In the next section, the solution ansatz will be presented, and we'll discover how the matrix equation presented above yields us an eigenvalue problem.

2 The solution ansatz and the eigenvalue problem

Just as the case for solving second order differential equations of the form $x'' + k_1 x' + k_2 x = 0$ for some real constants k_1, k_2, we invent a guess for our solutions. We make a so called ansatz, and depending on the constans above, the solutions can take on various forms.

It can be shown that the most general ansatz is of the form $x(t) = ae^{i\omega t}$ for some constant a, and this translates into $X(t) = Ae^{i\omega t}$ for the case of our system of differential equations, where A is now a matrix containing the amplitudes of our solutions. For the case of the matrix equation derived in the previous section, we get:

$$Ae^{i\omega t}(-\omega^2 I_{n\times n} + M_n) = 0$$

which can then be reduced to the following equation:

$$A(-\omega^2 I_{n\times n} + M_n) = 0$$

since $e^{i\omega t} \neq 0 \forall t \in \mathbb{R}$. Notice that this is an eigenvalue problem, with eigenvalues ω^2 and eigenvector A. Each and every eigenvalue with its' corresponding eigenvector forms a basis whose span. By linearity and the principle by superposition, the sum of each solution with some constant c_i for every solution i then forms the whole solution of the system.

3 A tridiagonal Toeplitz matrix and reccurence relations

The previously defined matrix M_n is a tridiagonal matrix (since it has three diagonals). However, this is actually a special case of a tridiagonal matrix, it's a Toeplitz matrix, since the elements in each of the three diagonals represent the same value.

As we saw from the previous section, we want to find the eigenvalues of M_n. If we forget about the constant term k/m we have:

$$\begin{vmatrix} 2-\omega^2 & -1 & 0 & 0 & 0 & 0 \\ -1 & 2-\omega^2 & -1 & 0 & 0 & 0 \\ 0 & -1 & 2-\omega^2 & -1 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & -1 & 2-\omega^2 \end{vmatrix} := D_n$$

which we define as D_n for determinant of the $n \times n$ matrix M_n. Notice, that if we expand along the first row and then along the first column, we yield:

$$(2-\omega^2)\begin{vmatrix} 2-\omega^2 & -1 & 0 & 0 & 0 \\ -1 & 2-\omega^2 & -1 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & -1 & 2-\omega^2 \end{vmatrix} + \begin{vmatrix} -1 & -1 & 0 & 0 & 0 \\ 0 & 2-\omega^2 & -1 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & -1 & 2-\omega^2 \end{vmatrix} =$$

$$= (2-\omega^2)\begin{vmatrix} 2-\omega^2 & -1 & 0 & 0 & 0 \\ -1 & 2-\omega^2 & -1 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & -1 & 2-\omega^2 \end{vmatrix} - \begin{vmatrix} 2-\omega^2 & -1 & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & -1 & 2-\omega^2 \end{vmatrix} =$$

$$= (2-\omega^2)D_{n-1} - D_{n-2}$$

Hence, the tridiagonal Teoplitz matrix gives us a recurrence relation of the form:

$$\begin{cases} D_n = (2-\omega^2)D_{n-1} - D_{n-2} \\ D_0 = 1 \\ D_1 = (2-\omega^2) \end{cases}$$

We want to find the general solution for ω by $D_n = 0$, to thus find the eigenfrequencies of the system. The solution to this problem will be presented in the next section.

4 A general solution for the eigenfrequencies

For our given reccurence relation, let us introduce a substitution of variables. For instance, we let $2-\omega^2 := 2x$, hence we get:

$$\begin{cases} D_n = 2xD_{n-1} - D_{n-2} \\ D_0 = 1 \\ D_1 = 2x \end{cases}$$

Observe, this matches the exact recurrence relation for the Chebyshev polynomials of the second kind. It can further be shown that, for $|x| \leq 1$, meaning $2|x| \leq 2$ (which is true for our case)that:

$$D_n(x) = \frac{\sin\left((n+1)\arccos x\right)}{\sin\left(\arccos x\right)}$$

Setting this equal to 0, means that $x = \cos\left(\frac{k\pi}{n+1}\right) \Leftrightarrow 2x = 2\cos\left(\frac{k\pi}{n+1}\right)$ but $2x = 2-\omega^2$ and hence:

$$\omega_k = \sqrt{2 - 2\cos\left(\frac{k\pi}{n+1}\right)}\sqrt{\frac{k}{m}}$$

for $k = 1, 2, ..., n$. Notice that we omitted the constant term from M_n before, so we made sure to add it back in the step above. Hence, we have found the eigenfrequencies of the system. The last step is to find the eigenvectors so that we can express the full general solution of or system.

We won't show this in detail, but it can be shown that the eigenvectors corresponding to the k:th eigenfrequency is on the form:

$$\vec{v}_k = \begin{pmatrix} \sin(\frac{k\pi}{n+1}) \\ \sin(\frac{2k\pi}{n+1}) \\ \vdots \\ \sin(\frac{nk\pi}{n+1}) \end{pmatrix}$$

therefore, the solution of $X(t)$ can be expressed as:

$$\boxed{X(t) = \Re\left(\sum_{k=1}^{n} a_k \vec{v}_k e^{i\omega_k t}\right)}$$

for some constants a_k who are purely determined from the initial conditions of the system.

5 Python simulation

Implementing a Python code, we can easily find the eigenvalues and eigenvectors numerically. We can even confirm the argumentation held in the previous paragraph using some developed Python scrips. These, along with the python simulation using PyGame, will be presented in the next section:

```
# libraries
import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt
import pygame
from sys import exit

# boundary conditions
k = 4 # stiffness of spring
m = 1 # mass of block
boundaries = np.array([0.2,-0.2,0,0.2])
k_m = np.sqrt(k/m) #angular frequency

def matrix_oscillation(n):

    matrix = np.zeros((n,n))

    for i in range(0, n):
        for j in range(0, n):
            if i == j:
                if i == 0:
                    matrix[i,j] = 2
                    matrix[i,j+1] = -1
                elif i == n-1:
                    matrix[i,j-1] = -1
                    matrix[i,j] = 2
                else:
                    matrix[i, j - 1] = -1
                    matrix[i, j] = 2
                    matrix[i, j + 1] = -1

    return matrix
```

```
def eigenvaluefinder(matrix):

    w, v = LA.eig(matrix)
    return w, v


matrix = matrix_oscillation(len(boundaries))
eigenvals, eigenvector = eigenvaluefinder(matrix)
```

Now follows the second part of the code, which constitutes the simulation of the system.

```
def equation_system(eigenvals, eigenvector,boundaries): #solving the matrix

#equation Ac = b where b is our boundary conditions

    inv_eigen = np.linalg.inv(eigenvector)
    constants = inv_eigen.dot(boundaries)

    return constants

constants = equation_system(eigenvals, eigenvector, boundaries)

def graph(eigenvals, eigenvector, constants):

    dt = 0.001
    ts = np.arange(0, 10, dt)
    ys = []

    # creating the arrays of displacement for each block
    for i in range(0, len(boundaries)):
        arr = []
        for m in range(0, len(ts)):
            sum = 0
            t = ts[m]
            for k in range(0,len(boundaries)):
                sum += constants[k]*eigenvector[i,k]*
                np.cos(np.sqrt(eigenvals[k])*k_m*t)
            arr.append(sum)
        ys.append(arr)
```

```
    # plotting the displacement of each block wrt time
    for i in range(0, len(ys)):
        plt.plot(ts, ys[i], label = "Mass number {}".format(i+1))
    plt.xlabel("Time")
    plt.ylabel("Displacement")
    plt.legend()
    plt.show()

    return ys, len(ts)

ys, lt = graph(eigenvals, eigenvector, constants)

#pygame code for simulation

#prestuff
pygame.init()
screen = pygame.display.set_mode((800+50*len(boundaries)
+50*max(ys[len(boundaries)-1]),400))
pygame.display.set_caption("Mass-spring system")
clock = pygame.time.Clock()

# surfaces
test_surface = pygame.Surface((50,50))
surfaces = [test_surface for i in range(0, len(boundaries))]
test_surface.fill('brown')

#pictures and fonts
picture = pygame.image.load('floor.jpg')
picture = pygame.transform.scale(picture, (800+50*len(boundaries)
+50*max(ys[len(boundaries)-1]), 400))
font = pygame.font.SysFont('arial', 40)
largeText = font.render("COUPLED MASS-SPRING SYSTEM", True, (0,0,0))

# simulation
while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            exit()
```

```
for j in range(0, lt):
    screen.blit(picture, (0, 0))
    screen.blit(largeText,((800+50*len(boundaries)
    +50*max(ys[len(boundaries)-1]))/2-300,100))
    for i in range(0,len(boundaries)):
        screen.blit(surfaces[i], (50 + (800+50*len(boundaries))/
        (len(boundaries))*i +
                                  50*ys[i][j], 175))
        pygame.draw.line(screen, (0, 0, 0), (0, 200),
                         (50 + 50 * ys[0][j], 200))
        pygame.draw.line(screen, (0, 0, 0), (
        50 + (800+50*len(boundaries))
        /(len(boundaries)) *
        (len(boundaries) - 1) + 50 * ys[len(boundaries) - 1][j]+50, 200),
                         (800+50*len(boundaries)
                         +50*max(ys[len(boundaries)-1]), 200))
        if 0< i:
            pygame.draw.line(screen, (0,0,0),
                             ((800+50*len(boundaries))/
                             (len(boundaries))*(i-1) +
                             50*ys[i-1][j]+100, 200),
                             ((800+50*len(boundaries))/
                             (len(boundaries))*i + 50*ys[i][j]+50, 200))
    pygame.display.update()
    screen.fill((225, 225, 225))

clock.tick(60)
```

6 Discussion

Examining the expression for ω_k, we clearly see that $\omega_k \in [0, 2] \forall k = 1, 2, ..., n$. Whether this has an intuitive explanation or not is unclear from this stand of point point. Furthermore, we realize that $\omega_n \to 2$ as $n \to \infty$.

If we go ahead and examine the expression for our eigenvector $\vec{v}_k$, we notice some interesting patterns. For instance, if we have an odd number of masses, some components of certain eigenvectors will be able to become 0 since $n+1$ becomes even for odd n's. This is in some sense trivial as it corresponds to some masses oscillating at a frequency in such a way that it leaves the rest of the system untouched. This also

depends on the initial conditions of the system too.

For instance, if we have three masses and the masses to the left and right of the center mass move in an anti fashion, we get a cancellation of the motion of the middle block. But for even number of masses, this won't be possible, since we won't get the symmetry that allows some masses to not move at all. Let's try and confirm this using our Python simulation above.

Table 1: *A table over the eigenfrequencies and eigenvectors of the mass spring system consisting of three masses. These results have been numerically derived from our Python script.*

k	ω_k	$\vec{v}_k$
1	1.84775906	$[-0.5, 0.7071, -0.5]$
2	1.41421356	$[-0.7071, 4.05 \cdot 10^{-16}, 0.7071]$
3	0.76536686	$[0.5, 0.7071, 0.5]$

Notice $\vec{v}_2 \cdot [0, 1, 0] \approx 0$, which corresponds to the motion where the mass to the left of the center mass and to the right move in an antifashion. We can also confirm that further, since $\vec{v}_2 \cdot [1, 0, 0]$ and $\vec{v}_2 \cdot [0, 0, 1]$ are equal in magnitude but opposite in direction. We try to represent the equations of motions of each mass graphically in the upcoming Figure, for which we find our argumentation above to be correct.

We can also bring more masses in, and we'll clearly see how the system spirals down to motions with large complexity.

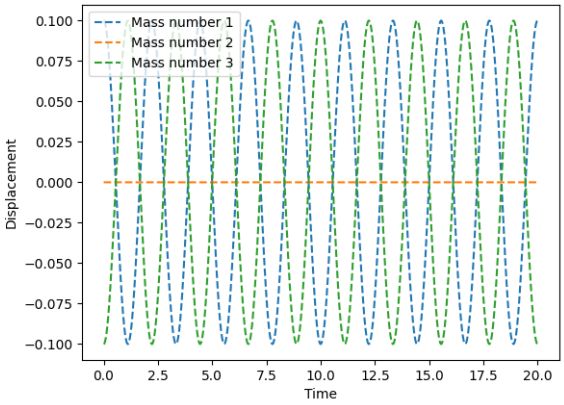

Figure 2: *The figure represents the displacements of mass* $1, 2$ *and* 3 *with respect to time. Here, mass 1 has initially been displaced* 0.10 *meters to the right, and mass 3 has been displaced* 0.10 *meters to the left. Hence, the middle block becomes stationary. Notice that the displacements correspond to the previously introduced generalized coordinates.*

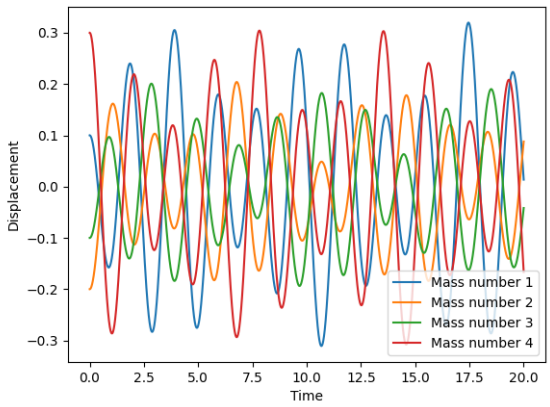

Figure 3: *The figure represents the displacements of mass* $1, 2, 3$ *and* 4 *with respect to time. The initial displacements are given by* $0.10, -0.20, -0.10, 0.30$ *meters correspondingly.*

7 Conclusion

From what we've found, there's no doubt that the interconnected mass spring system is rather complicated, and even further so when adding in more masses, as expected. Finding the general expression for the matrix M_n was relatively easy, but a lot of re-

search was needed in order to find the eigenvalues of this matrix. Hence, the problem was majorly a mathematical challenge.

Through the found expressions for the eigenfrequencies and eigenvectors, we found the general solution for the motion of our system. Finally, we developed a Python script that numerically could confirm our found results. It turned out to be very satisfying, and a challenge as well, since it was the first time testing out the PyGame module. All in all, we can conclude that the beauty of physics and its' connection to mathematics and possibility to be confirmed through numerical results in programming is a very satisfying connection between these STEM - fields.