Tuned Mass Damper or Viscoelastic Damper?

An analytical study on the effectivity of the Tuned Mass Damper and the Viscoelastic Damper in reducing the oscillations caused by an earthquake

Student:
Bradley van Hooff
5348803

Supervisor:
Valentina Vanisuhkina
TU Delft

4 July 2023

Shortened Summary

In this report, it will be researched which of the two dampers, a Tuned Mass Damper or a Viscoelastic Damper, is most effective in damping the vibrations of a building that occur during an earthquake. A Tuned Mass Damper is a damper which can be implemented in a building like a pendulum, and which can damp vibrations by swaying with a specified frequency. A Viscoelastic Damper damps an earthquake's effect by absorbing energy that comes from its movements. Both dampers will be added to models that describe the vibrations of a building, and simulations of their effects on vibrations will be generated. Based on these results, it will be concluded which damper is the most effective.

Summary

Within this report, an analytical research will be performed on the effectivity of Tuned Mass Dampers (TMDs) and Viscoelastic Dampers (VEDs), and a conclusion will be drawn as to which is most effective in damping the oscillations of an earthquake. First of all, a simple model of second-order differential equations describing the oscillations of each floor within a building of n floors is derived, in which no frictional forces are taken into account. The solution to this model is derived for a building of two floors, which demonstrates the characteristics of oscillations in this model and which shows how resonance is represented mathematically, and what patterns of oscillations look like.

Then, a TMD and VED are added to the model of one floor to expose what their working principles are. Subsequently, the methods of adding these dampers to the models for two, five and ten floors are given. For varying earthquake-frequencies, the maximum displacement within 5 seconds is determined, which shows the effect of TMDs and VEDs for different positions within the building. The conclusion from these results is that the effect of the TMDs is always that at the eigenfrequencies, the oscillations of the building are no longer resonating. However, the building will start resonating for other eigenfrequencies due to the TMD. The VED is more consistent, though its damping-effect on resonating frequencies is smaller than for the TMDs. The conclusion is that VEDs are most effective, though more dampers have to be added to improve its effectiveness.

After this, a new model is introduced, which does contain a friction force: air resistance. Based on the new model, it is concluded that even oscillations for resonating frequencies are eventually damped out, though they do still grow large. Again, TMDs and VEDs are added to the model, and similar conclusions about the effectivity of the dampers as for the first model were drawn.

Contents

List of variables

$x_i(t)$	the horizontal displacement of floor i with respect to its initial position (m)
$P(x_i(t))$	the momentum of the displacement $x_i(t)$
k	the stiffness of a floor (N/m)
m	the mass of one floor (kg)
F_0	the amplitude of the earthquake (m)
ω	the frequency of the earthquake (Hz)
k_T	the stiffness of the Tuned Mass Damper when attached to a floor (N/m)
m_T	the mass of a Tuned Mass Damper (kg)
c_V	the damping coefficient of a Viscoelastic Damper (Ns/m)
C_d	the drag coefficient of a floor
A	the cross-sectional area of a floor (m^2)
ρ	the density of air (kg/m^3)
$z_i(t)$	for $i = 1,2,...,n$, $z_i(t) = x_i'(t)$ and $z_{i+n}(t) = x_i(t)$
$y_i(t)$	for $i = 1,2,...,n$, $y_i(t) = x_{h,i}'(t)$ and $y_{i+n}(t) = x_{h,i}(t)$
K_n	the homegeneous form of both models for n floors can be described as $x_h''(t) = Kx_h(t)$
$\tilde{K}_n$	the matrix that is defined as $K = \begin{bmatrix} [0]_{n\times n} & K_n \\ I_n & [0]_{n\times n} \end{bmatrix}$
$f_{\text{Earthquake}}(t)$	the force of the earthquake, described by $F_0\omega^2 m\cos(\omega t)$
$f_{\text{Linear Restoring}}(t)$	the linear restoring force, described by $-k\Delta x(t)$
$f_{\text{Air Resistance}}(t)$	the air resistance force, described by $-\frac{1}{2}C_d A\rho(x_1')^2$
$\mathbf{d}_n$	the vector $[1, \quad 1, \quad ..., \quad 1]^T$ of n components
$\tilde{\mathbf{d}_n}$	the vector $[\mathbf{d}_n, \quad \mathbf{0}_n]^T$ of $2n$ components
$\mathbf{A}$	used in the particular solution of the model of two floors without air resistance, $\mathbf{x}_p(t) = \mathbf{A}\cos(\omega t)$.
L	the matrix defined as $L = (K_2 + \omega^2 I)$
$\mathbf{f}(t,\mathbf{z}(t))$	function used within the definition of the method RK4, such that $\mathbf{f}(t,\mathbf{z}(t)) = \mathbf{z}'(t)$

Matrices are denoted by captial letters (K), vectors by letters in bold ($\mathbf{A}$), and scalars by small letters (m).

Report Overview

First, in chapter 2 a simple model describing the movements of a building during an earthquake without friction forces will be introduced. The behaviour of the model will be analysed, and the working principles of dampers within the model will be examined. Subsequently, in chapter 4, the two types of dampers will, separately, be included in the model of two, five and ten floors, at several positions. It will be concluded which damper is most effective in the first model. Chapter 5 will expand the model by including air resisance, and the effect on the behaviour of the model will be considered. The dampers will be added to the new model in chapter 6, and it will be concluded whether the conclusions for the dampers in the previous model still stand.

1

Introduction

Resonance is a common physical phnomenon. For example, it occurs when one of two guitar strings of the same tuning is played: the other string will start oscillating without being plugged. The explanation for that, is that both strings have a common eigenfrequency: a frequency that is "natural" to the string in its current state. When the sound waves travels from the first towards the second string, a frequency in the sound wave corresponding to an eigenfrequency of the string will cause the string to oscillate.

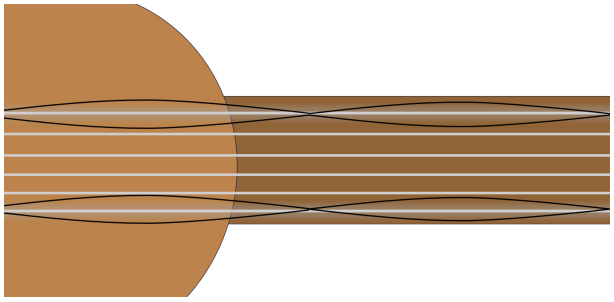


Figure 1.1: The strings have a common eigenfrequency, so that resonance occurs.

Unfortunately, buildings also have their own eigenfrequencies. When forced by gusts of wind, or by an earthquake, the buildings will start moving, while having a certain tendency to return to their original state: a restoring force. This restoring force causes the building to start oscillating with its own eigenfrequencies, and just like the two strings, when the earthquake or wind pushes the building with a frequency that is very close to one of these eigenfrequencies, the building has the natural tendency to start oscillating. However, during this case of resonance, the building will keep oscillating with an evergrowing amplitude, until either the earthquake stops, or the building collapses.

This is a situation that we want to prevent from happening when building a new building in an earthquake-prone area. Fortunately, there are methods to prevent this from happening: installing dampers. There is a great variety of dampers, each with their own specific characteristics, and therefore it can be a hard decision deciding which damper fits the requirements of a building best.

Therefore, in this report, the effect of two very different dampers on the oscillations of a building during an earthquake will be analysed. The two dampers that will be considered are the Tuned Mass Damper (TMD) and the Viscoelastic Damper (VED). The TMD is a device that can be attached to a building, and can be tuned in such a way that it reduces the oscillations of certain frequencies. In contrast, the VED scales down oscillations of every frequency by turning mechanical energy into heat.

It is of great importance that the dampers make sure the amplitude of the oscillations of the building does

Figure 1.2: A building that collapsed from the effects of an earthquake [13].

not grow too large. Therefore, their assessment will be based on how effectively they reduce oscillations of all frequencies: resonating and non-resonating. Furthermore, worst-case scenarios will be considered, i.e. the amplitude of the earthquake will be as large as it gets, and the building's material's properties will be chosen in such a way that they are both realistic and worst-case.

2

Literature Review

On the topics of modelling a building during an earthquake, adding Tuned Mass Dampers to a model, and adding Viscoelastic Dampers to a model, extensive research has already been done. In this chapter, a brief review will be given of important findings and conclusions on these issues.

2.1. Modelling a Building During an Earthquake

Constructions are often mathematically modelled a system of differential equations [14]. Such equations are also often used for modelling a building during an earthquake, by describing the displacements of each floor [26] [2] [12] [16]. Within these models, the differential equations are based on Newton's second law, where the external forces are taken to be the force of the earthquake and a linear restoring force. In some models, a damping force is already included. The buildings in these models all have natural frequencies, for which resonance occurs. The amount of natural frequencies of a building is equal to the amount of floors [12]. According to this model, for earthquake-frequencies near these frequencies, the maximum displacement within a few seconds is significantly larger than for frequencies that are not near a natural frequency [2].

2.2. Modelling a Tuned Mass Damper on a Building

Adding a Tuned Mass Damper (TMD) to a floor in the model of a building can be done by including the displacement of the TMD as a new differential equation. The Tuned Mass damper is modelled as a mass that is attached to the floor by a spring [11]. In some cases, a damping force is considered to be in the spring, while in other cases, the only force that acts between the floor and the damper is the linear restoring force [11]. The properties (stiffness, mass and damping) of a TMD are chosen in such a way that the effect of an earthquake of a certain frequency is minimized. For an earthquake with frequency ω, the optimal decisions for the stiffness k_T, mass m_T and (if included in the model) damping c_T are: $\omega^2 = \frac{k_T}{m_T}$ and $c_T = \sqrt{\frac{3\left(\frac{m_T}{m}\right)^3}{8\left(1+\frac{m_T}{m}\right)^3}}$, where m is the mass of a floor. [11]. Extensive research has already been executed on the performance of TMDs and their optimal positioning. If a single TMD is applied to a building, its optimal location is the highest floor of a building [9]. When multiple TMDs are included in a building, their optimal positions are determined by the structures and material properties of the building [9]. Performance of TMDs is usually based on their ability of reducing oscillations of a particular frequency [21] [17]. However, the TMD causes resonance to occur for different frequencies than the frequency it has been tuned to [11].

2.3. Modelling a Viscoelastic Damper on a Building

A damping force can be added to the model of differential equations, as was mentioned before [16], and a Viscoelastic Damper can be included as a damping force. Its damping coefficient is dependent on the properties of the material is it made of, but also on the amplitude and frequency of the earthquake, and on the temperature. Furthermore, adding larger or more VEDs to a building will lead to stronger damping properties [20] However, much more research needs to be done on the dependence on amplitude and frequency, though some models are already able to capture frequency-dependence [20]. The optimal placement of a VED within a building is not clear [10], though placing it on the first floor has a preference according to [4]

3

The Model without Air Resistance

In this chapter, the first simple model will be derived, and the values and ranges for the parameters within the model will be given and explained. The model will then be solved analytically, such that the behaviour of oscillations in the model becomes clear. Furthermore, a numerical method will be given, that will be used to describe the model for varying earthquake-frequencies. After the behaviour of the undamped model has been described, the properties of the Tuned Mass Damper and the Viscoelastic Damper will be discussed.

3.0.1. Derivation of the Model

The first model in this report, is based on the model introduced in [26]. The goal of the model is describing the oscillations of the floors in a building separately, by modelling the horizontal displacement of the center of gravity of each floor (also known as point masses). The displacement relative to the initial positions of the point masses will be denoted by $x_i(t)$ for the ith floor of the building.[1] A schematic view of this discretisation of the building is shown in figure 3.1.

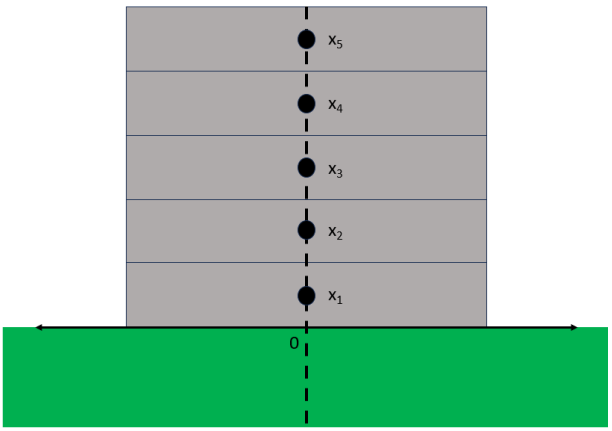


Figure 3.1: Schematic view of the discretisation by floor of a building, in which the point masses denote the horizontal displacement of each floor.

The movement of the floors is initially caused by the earthquake. Once the floors start shifting separately, due to the stiffness of the building, the floors have a natural tendency to return to their initial state. The force that embodies this tendency is the linear restoring force. **In this model, the linear restoring force and the force exerted by the earthquake are assumed to be the only force acting on the building, and will be modelled to**

[1] The ground floor is the the first floor.

operate on each floor (see figure 3.2). All other forces, such as damping forces and air resistance, are left out. The external forces and the movement of the point masses are related by Newton's second law[2], such that for the ith floor of the building,

$$\frac{d}{dt}P(x_i(t)) \quad = \quad f_{\text{Linear Restoring}}(t) \quad + \quad f_{\text{Earthquake}}(t). \tag{3.1}$$

In this equation, $P(x_i(t)) = mx_i'(t)$ denotes t he momentum of point mass $x_i(t)$, **with m the mass of a floor, which is assumed to be equal for all floors**.

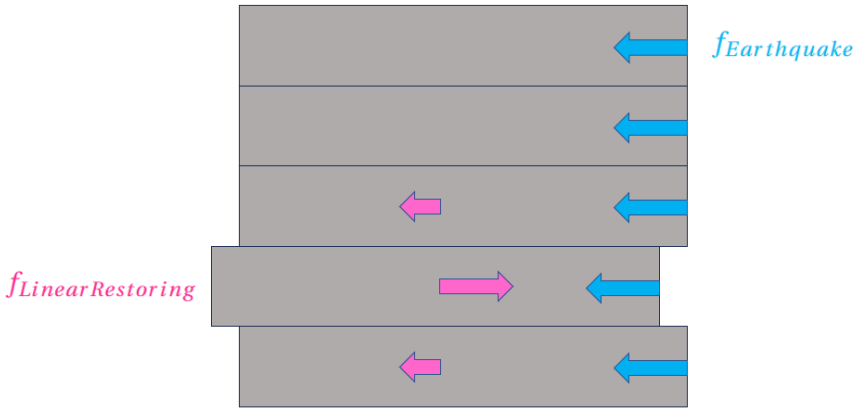


Figure 3.2: A scheme displaying the external forces acting on the building.

Furthermore, **it is assumed that the earthquake behaves as a sinusoidal function with frequency ω (Hz) and amplitude F_0 (m)**: $x_{\text{Earthquake}}(t) = F_0\cos(\omega t)$. The force of the earthquake on a point mass can be determined by multiplication of the mass m of a floor and the accelleration of the earthquake (i.e. the second derivative of $x_{\text{Earthquake}}(t)$):

$$f_{\text{Earthquake}}(t) = -F_0\omega^2 m\cos(\omega t). \tag{3.2}$$

The linear restoring force on each floor is determined by considering the building as a spring-mass system, as shown in figure 3.3. Each mass (i.e. floor) is attached to one or two springs, representing the stiffness between floors. Once a mass moves in any direction, the springs will exert a linear restoring force on any mass it is connected to. The linear restoring force is described by Hooke's law: $f_{\text{Linear Restoring}}(t) = -k\Delta x(t)$. As in figure 3.3, **each floor is assumed to have an identical stiffness** k. Note, furthermore, that only one spring is attached to the mass on the outer right (the highest floor), and that the spring attached to the left of the outer left mass (the ground floor) is attached to a fixed end (the ground). Based on equation (3.1), expression (3.2)

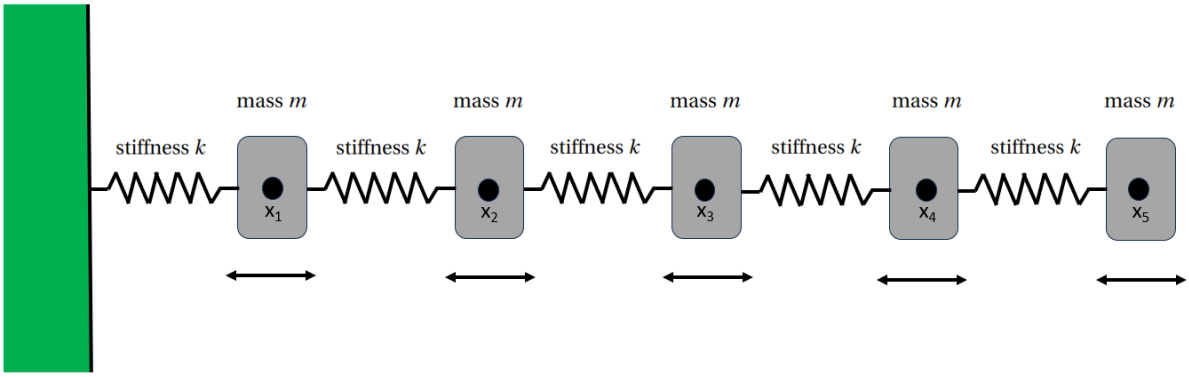


Figure 3.3: Configuration of the building as a sprng-mass system.

[2]Newton's second law states that, if forces are acting on a body, the time rate of change of the momentum of that body is equal to the total sum of the forces acting on it.

and the configuration in figure 3.3, the following model for a the displacements $x_i(t)$ of floor i in a building of n floors is established:

$$\begin{cases} mx_1'' = -kx_1 - k(x_1 - x_2) & - \quad F_0\omega^2 m\cos(\omega t), \\ mx_2'' = -k(x_2 - x_1) - k(x_2 - x_3) & - \quad F_0\omega^2 m\cos(\omega t), \\ mx_3'' = -k(x_3 - x_2) - k(x_3 - x_4) & - \quad F_0\omega^2 m\cos(\omega t), \\ \quad ... & \\ mx_{n-1}'' = -k(x_{n-1} - x_{n-2}) - k(x_{n-1} - x_n) & - \quad F_0\omega^2 m\cos(\omega t), \\ mx_n'' = -k(x_n - x_{n-1}) & - \quad F_0\omega^2 m\cos(\omega t). \end{cases} \tag{3.3}$$

Finally, **it is assumed that at the start of the earthquake, the building is in full rest (i.e. no displacement and the velocity equals zero**, such that the initial conditions are described by $x_i(0) = x_i'(0) = 0$ for $i = 1, 2, ..., n$.

3.0.2. Values and Ranges of the Parameter in the Model without Air Resistance

The model contains a number of parameters, which can either be fixed or varied. In this model, the values of the mass m (kg), the stiffness k (N/m) and the earthquake's amplitude F_0 (m) are kept constant, while the earthquake's frequency ω (Hz) is varied over a certain interval. This decision is based on the knowledge that the earthquake-frequency has significant influence on the oscillation-patterns of the building (e.g. resonance), and the fact that both the Tuned Mass Damper and the Viscoelastic Damper's performance is highly dependent on the frequency of the earthquake, as will be shown in sections 3.2 and 3.3. Furthermore, various building heights are considered when analysing the effectivity of the dampers, since the height of a building is proven to influence its eigenfrequencies, and therefore its oscillation-properties [2].

The precise values and range of the parameters are listed and argued below.

- The introduced model describes behaviours of buildings of n equal floors, and their height might influence the eigenfrequencies of the building. Therefore, when analysing the effectivity of dampers on a building, several heights of the construction are considered: a small building of two floors, a mid-rise building of five floors, and a high-rise building of ten floors, as depicted in figure 3.4

- Each floor in the buildings considered, are assumed to have equal mass m and stiffness k. The values that are chosen for these parameters are averaged building-properties, based on the masses and stiffnesses of floors within a building of nine floors as presented in [7]. The stiffness k is set to approximately $5{,}5 \cdot 10^9$ N/m, whereas the mass m is fixed as $9{,}0 \cdot 10^5$ kg.

- The amplitude of earthquakes can range from 10^{-10} to 0.1 m. In this model, the amplitude of the earthquake is set to $F_0 = 0.1$ m, as this is the scenario in which the earthquake has a maximized effect on the building, such that the effectivity of the dampers is modelled in a worst-case scenario. However, the amplitude of an earthquake will be of less influence on the size of displacements than the frequency of the earthquake.

- Earthquake-frequencies occur within a range from 0.001 to 100 Hz , which is the range that will be used for the values of ω within the model. The model will be analysed for each ω-step of 0.25 Hz.

3.1. The Behaviour of The Model

In this section, an analytical solution to the model (3.3) will be determined, which will expose the characteristics of the oscillations, including resonance.

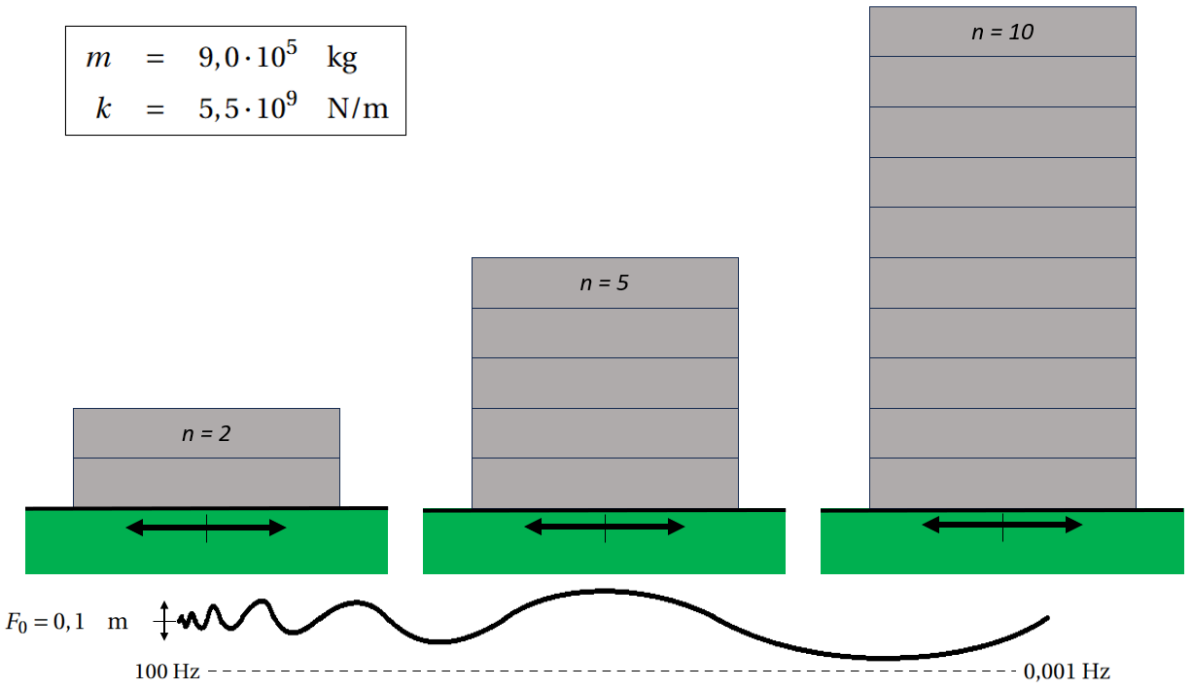


Figure 3.4: Summary of the values and range of parameters within the model.

3.1.1. The Analytical Solution Describing Oscillation-Behaviour

To demonstrate the theoretical characteristics that define the oscillations of a building in the model introduced in this chapter, the model will be solved analytically for a two-floor building ($n = 2$ in (3.3)):

$$\begin{cases} mx_1'' = & -kx_1 - k(x_1 - x_2) & -F_0\omega^2 m\cos(\omega t), \\ mx_2'' = & -k(x_2 - x_1) & -F_0\omega^2 m\cos(\omega t), \end{cases} \tag{3.4}$$

with initial conditions $x_i(0) = x_i'(0) = 0$ for $i = 1,2$. The model describes a nonhomogeneous system of ordinary differential equations. A method of determining the solution to such a system, is by finding a particular and a homogeneous solution, whereafter the general solution to the system can be found by addition of the two, and can be completed by substitution of the inital conditions [5].

To be able to apply the described method of solving the system of differential equations, it is beneficial to rewrite it in vector-matrix notation:

$$\mathbf{x}''(t) = K_2\mathbf{x}(t) - F_0\omega^2\cos(\omega t)\mathbf{d}, \tag{3.5}$$

where

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}, \qquad K_2 = \begin{bmatrix} -\frac{2k}{m} & \frac{k}{m} \\ \frac{k}{m} & -\frac{k}{m} \end{bmatrix}, \qquad \mathbf{d} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}. \tag{3.6}$$

The Particular Solution

First, a particular solution $\mathbf{x}_p(t)$ will be established.[3] A suggestion for its form is $\mathbf{x}_p(t) = \mathbf{A}\cos(\omega t)$. This form is a reasonable proposition, since its second derivative is a multiplication of itself, and a cos-function is already

[3] A particular solution is a solution that solves the system of differential equationn, but does not necessarily satisfy the initial conditions. Furthermore, there are no undetermined coefficients in the particular solution.

present within the equation. It is derived whether this form satisfies the differential equation and what the expression of **A** is by substitution into (3.5):

$$-\omega^2 \mathbf{A}\cos(\omega t) = K_2\mathbf{A}\cos(\omega t) - F_0\omega^2\cos(\omega t)\mathbf{d}. \tag{3.7}$$

To satisfy this equation for all values of t, it is required that $-\omega^2\mathbf{A} = K_2\mathbf{A} - F_0\omega^2\mathbf{d}$. In particular, **A** must satisfy

$$\left(K_2 + \omega^2 I\right)\mathbf{A} = F_0\omega^2\mathbf{d}. \tag{3.8}$$

In case the matrix $L = \left(K_2 + \omega^2 I\right)$ is invertible, equation (3.8) can be solved by computing the inverse of L, such that $\mathbf{A} = F_0\omega^2 L^{-1}\mathbf{d}$. To ascertain under which conditions the matrix is invertible, its determinant will be examined:

$$\begin{aligned}
\det(L) &= \begin{vmatrix} \omega^2 - \frac{2k}{m} & \frac{k}{m} \\ \frac{k}{m} & \omega^2 - \frac{k}{m} \end{vmatrix} \\
&= \left(\frac{k}{m}\right)^2 - 3\frac{k}{m}\omega^2 + \omega^4 \\
&= \left(\omega^2 - \frac{k}{2m}\left(3+\sqrt{5}\right)\right)\left(\omega^2 - \frac{k}{2m}\left(3-\sqrt{5}\right)\right)
\end{aligned} \quad . \tag{3.9}$$

From this expression, we derive that the determinant of L equals zero for $\omega^2 = \frac{k}{2m}\left(3 \pm \sqrt{5}\right)$, and therefore, for those values of ω, the inverse of L does not exist and no solution exists for equation (3.8). Therefore, it is assumed that L is invertible, i.e. $\omega^2 \neq \frac{k}{2m}\left(3 \pm \sqrt{5}\right)$, and its inverse equals

$$L^{-1} = \frac{1}{\left(\frac{k}{m}\right)^2 - 3\frac{k}{m}\omega^2 + \omega^4}\begin{bmatrix} \omega^2 - \frac{k}{m} & -\frac{k}{m} \\ -\frac{k}{m} & \omega^2 - \frac{k}{m} \end{bmatrix} \tag{3.10}$$

By means of this, the expression for **A** can be determined,

$$\mathbf{A} = F_0\omega^2 L^{-1}\mathbf{d} = \frac{F_0\omega^2}{\left(\frac{k}{m}\right)^2 - 3\frac{k}{m}\omega^2 + \omega^4}\begin{bmatrix} -\frac{2k}{m} + \omega^2 \\ -\frac{3k}{m} + \omega^2 \end{bmatrix}, \tag{3.11}$$

settling the particular solution $\mathbf{x}_p(t)$:

$$\mathbf{x}(t) = F_0\omega^2 L^{-1}\mathbf{d} = \frac{F_0\omega^2}{\left(\frac{k}{m}\right)^2 - 3\frac{k}{m}\omega^2 + \omega^4}\cos(\omega t)\begin{bmatrix} -\frac{2k}{m} + \omega^2 \\ -\frac{3k}{m} + \omega^2 \end{bmatrix}\cos(\omega t) \tag{3.12}$$

The Homogeneous Solution

Second, the a general solution $\mathbf{x}_h(t)$ to the homogeneous form of (3.5) will be determined.[4] The homogeneous equation has the following form:

$$\mathbf{x}_h''(t) = K_2\mathbf{x}_h(t). \tag{3.13}$$

An approach to find the general solution to this equation is by introducing a new set of variables, such that this system of second-order differential equations can be expressed as a system of first-order differential equations. Define $y_i(t) \equiv x_{h,i}'(t)$ and $y_{i+2}(t) \equiv x_{h,i}(t)$ for $i = 1, 2$. Then the homogeneous equation (3.13) converts to

$$\mathbf{y}'(t) = \tilde{K}_2\mathbf{y}(t), \tag{3.14}$$

[4] A homogeneous equation is an equation in which all terms that do not contain a term of **x** or its derivatives are set to **0**.

in which

$$\mathbf{y}(t) = \begin{bmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \\ y_4(t) \end{bmatrix}, \qquad \tilde{K_2} = \begin{bmatrix} 0 & 0 & -\frac{2k}{m} & \frac{k}{m} \\ 0 & 0 & \frac{k}{m} & -\frac{k}{m} \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{3.15}$$

From this system of differential equations, the general solution can be established by finding the eigenvalues and corresponding eigenvectors of the matrix $\tilde{K_2}$. First, its eigenvalues are determined:

$$\begin{aligned} \det\left(\tilde{K_2} - \lambda I\right) &= 0 \\ &= \lambda^4 + 3\frac{k}{m}\lambda^2 + \left(\frac{k}{m}\right)^2 = 0 \\ &= \left(\lambda^2 + \frac{k}{2m}\left(3+\sqrt{5}\right)\right)\left(\lambda^2 + \frac{k}{2m}\left(3+-\sqrt{5}\right)\right) = 0, \end{aligned} \tag{3.16}$$

And hence, $\lambda = \pm i \cdot \sqrt{\frac{k}{2m}\left(3 \pm \sqrt{5}\right)}$.

dAs a second step in determining the general solution to the homogeneous equation, the eigenvectors corresponding to the eigenvalues will be determined. These can be established by finding a solution $\mathbf{v}$ to the system $\left(\tilde{K_2} - \lambda I\right)\mathbf{v} = \mathbf{0}$ through elementary row operations:

$$\left[\begin{array}{cccc|c} -\lambda & 0 & -\frac{2k}{m} & \frac{k}{m} & 0 \\ 0 & -\lambda & \frac{k}{m} & -\frac{k}{m} & 0 \\ 1 & 0 & -\lambda & 0 & 0 \\ 0 & 1 & 0 & -\lambda & 0 \end{array}\right] \sim \left[\begin{array}{cccc|c} 0 & 0 & -\frac{2k}{m} - \lambda^2 & \frac{k}{m} & 0 \\ 0 & 0 & \frac{k}{m} & -\frac{k}{m} - \lambda^2 & 0 \\ 1 & 0 & -\lambda & 0 & 0 \\ 0 & 1 & 0 & -\lambda & 0 \end{array}\right] \tag{3.17}$$

$$\sim \left[\begin{array}{cccc|c} 0 & 0 & 0 & \frac{k}{m} - \frac{\left(\frac{k}{m}+\lambda^2\right)\left(\frac{k}{m}+\lambda^2\right)}{\frac{k}{m}} & 0 \\ 0 & 0 & \frac{k}{m} & -\frac{k}{m} - \lambda^2 & 0 \\ 1 & 0 & -\lambda & 0 & 0 \\ 0 & 1 & 0 & -\lambda & 0 \end{array}\right] \tag{3.18}$$

By substitution of λ (as found in (3.16)) into $\frac{k}{m} - \frac{\left(\frac{k}{m}+\lambda^2\right)\left(\frac{k}{m}+\lambda^2\right)}{\frac{k}{m}}$, we find that this expression equals zero.[5]
Therefore, the following relations for the compnents v_j of the eigenvectors $\mathbf{v}_i$ are established:

$$\begin{cases} v_1 - \lambda v_3 = 0, \\ v_2 - \lambda v_4 = 0, \\ \frac{k}{m} v_3 - \left(\frac{k}{m} + \lambda^2\right) v_4 = 0. \end{cases} \tag{3.19}$$

Choosing $v_4 = 1$ yields $v_2 = \lambda$, $v_3 = \frac{\left(\frac{k}{m}+\lambda^2\right)}{\frac{k}{m}}$ and $v_1 = \lambda\frac{\left(\frac{k}{m}+\lambda^2\right)}{\frac{k}{m}}$, implying the following form of the eigen-

[5] Since the established form of the matrix contains one row of zeros, one degree of freedom is acquired. In other words, the system has infinitely many solutions, and by choosing one of the components of $\mathbf{v}$ (freely), an eigenvector will be determined that satisfies $\left(\tilde{K_2} - \lambda I\right)\mathbf{v} = \mathbf{0}$. As a consequence of the degree of freedom, all scalar mulitples of that eigenvector $\mathbf{v}$ will also be an eigenvector of $\tilde{K_2}$.

vectors of the matrix $\tilde{K_2}$:

$$\mathbf{v} = \begin{bmatrix} \frac{\lambda\left(\frac{k}{m}+\lambda^2\right)}{\frac{k}{m}} \\ \lambda \\ \frac{\left(\frac{k}{m}+\lambda^2\right)}{\frac{k}{m}} \\ 1 \end{bmatrix}. \tag{3.20}$$

Now that both the eigenvalues and corresponding eigenvectors have been established, the general solution to the homogeneous system of differential equations can be constructed. Since the eigenvalues of $\tilde{K_2}$ are solely imaginary, the solution to the homogeneous differential system of differential equations in terms of $\mathbf{y}(t)$, (3.14), can be written as

$$\mathbf{y}(t) = \sum_{i=1}^{4} [a_i \cos(I_i t) + b_i \sin(I_i t)]\,\mathbf{v}_i, \tag{3.21}$$

in which I_i denotes the imaginary part of an eigenvalue λ_i of $\tilde{K_2}$, and $\mathbf{v}_i$ is the corresponding eigenvector.

It is worth noting that for $I_{1,2} = \pm\sqrt{\frac{k}{2m}\left(3+\sqrt{5}\right)}$, the eigenvectors $\mathbf{v}_1$ and $\mathbf{v}_2$ coincide. Furthermore, the terms $a_1\cos(I_1 t)\mathbf{v}_1$ and $a_2\cos(I_2 t)\mathbf{v}_2$ can be expressed as a single term, for $\cos(I_1 t) = \cos(-I_2 t) = \cos(I_2 t)$. The same holds for the sin-terms of both eigenvalues, and similar arguments hold for the eigenvalues $I_{3,4} = \pm i\sqrt{\frac{k}{2m}\left(3-\sqrt{5}\right)}$. Hence, (3.21) can be rewritten as

$$\mathbf{y}(t) = \sum_{i=1}^{2} [a_i \cos(\omega_i t) + b_i \sin(\omega_i t)]\,\tilde{\mathbf{v}}_i, \tag{3.22}$$

where $\omega_{1,2} = \sqrt{\frac{k}{2m}\left(3\pm\sqrt{5}\right)}$ and $\tilde{\mathbf{v}}_i$ is the eigenvector corresponding to ω_i $(i = 1,2)$.

By definition of vector $\mathbf{y}(t)$, the general solution to the homogeneous system of differential equations in terms of $\mathbf{x}_h(t)$, (3.13), corresponds to the lower two components of the expression (3.22) of $\mathbf{y}(t)$. Hence, by writing $\mathbf{w}_i$ as the lower two components of $\tilde{\mathbf{v}}_i$, the general solution to the homogeneous equation is:

$$\mathbf{x}_h(t) = \sum_{i=1}^{2} [a_i \cos(\omega_i t) + b_i \sin(\omega_i t)]\,\mathbf{w}_i, \tag{3.23}$$

wherein

$$\mathbf{w}_i = \begin{bmatrix} \frac{\left(\frac{k}{m}+(i\omega_i)^2\right)}{\frac{k}{m}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\left(\frac{k}{m}-\omega_i^2\right)}{\frac{k}{m}} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{-1\mp\sqrt{5}}{2} \\ 1 \end{bmatrix} \tag{3.24}$$

The Complete Solution

The general solution to the nonhomogeneous system of differential equations (3.4) equals the sum of the homogeneous solution (3.23) and the particular solution $x\mathbf{x}_p(t) = \mathbf{A}\cos(\omega t)$, where the expression for $\mathbf{A}$ is given in (3.11). This leads to the following expression for $\mathbf{x}(t)$:

$$\mathbf{x}(t) = \sum_{i=1}^{2} [a_i \cos(\omega_i t) + b_i \sin(\omega_i t)]\,\mathbf{w}_i + \mathbf{A}\cos(\omega t) \tag{3.25}$$

By substitution the inital conditions $\mathbf{x}(0) = \mathbf{x}'(0) = \mathbf{0}$ into this expression, the values of the coefficients a_i and b_i can be determined. First, implementing the condition $\mathbf{x}'(0) = \mathbf{0}$ yields

$$\sum_{i=1}^{2} \omega_i b_i \mathbf{w}_i = \mathbf{0}. \tag{3.26}$$

Noting that the vectors $\mathbf{w}_i$ and $\mathbf{w}_2$ are linearly independent, it can be concluded that $b_1 = b_2 = 0$.

By substituting the condition $\mathbf{x}(0) = \mathbf{0}$ into equation (3.25), the following expression is derived:

$$\sum_{i=1}^{2} [a_i \mathbf{w}_i] + \mathbf{A} = \mathbf{0}. \tag{3.27}$$

This equation can be written as a vector-matrix equation $W\mathbf{a} = -\mathbf{A}$, in which $W = [\mathbf{w}_1 \quad \mathbf{w}_2]$ and $\mathbf{a} = [a_1 \quad a_2]^T$. Since the columns of the matrix W are linearly independent, this matrix is invertible [8], and therefore the coefficients a_1 and a_2 are determined by $\mathbf{a} = -W^{-1}\mathbf{A}$.

First, the determinant of W can be easily computed:

$$\det(W) = \frac{\left(\frac{k}{m} - \omega_1^2\right) - \left(\frac{k}{m} - \omega_2^2\right)}{\frac{k}{m}} = \frac{-\omega_1^2 + \omega_2^2}{\frac{k}{m}} = \frac{-\frac{k}{m}\sqrt{5}}{\frac{k}{m}} = -\sqrt{5}, \tag{3.28}$$

and therefore, the inverse of W is

$$W^{-1} = -\frac{1}{\sqrt{5}} \begin{bmatrix} 1 & \frac{1-\sqrt{5}}{2} \\ -1 & \frac{-1-\sqrt{5}}{2} \end{bmatrix}. \tag{3.29}$$

Hence, by using the definition of $\mathbf{A}$ as in equation (3.11), the coefficients a_1 and a_2 are given by

$$\begin{aligned} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} &= -\frac{F_0\omega^2}{\sqrt{5}\left(\left(\frac{k}{m}\right)^2 - 3\frac{k}{m}\omega^2 + \omega^4\right)} \begin{bmatrix} 1 & \frac{1-\sqrt{5}}{2} \\ -1 & \frac{-1-\sqrt{5}}{2} \end{bmatrix} \begin{bmatrix} -\frac{2k}{m} + \omega^2 \\ -\frac{3k}{m} + \omega^2 \end{bmatrix} \\ &= \frac{F_0\omega^2}{2\sqrt{5}\left(\left(\frac{k}{m}\right)^2 - 3\frac{k}{m}\omega^2 + \omega^4\right)} \begin{bmatrix} \frac{k}{m}(-7+3\sqrt{5}) + \omega^2(3-\sqrt{5}) \\ \frac{k}{m}(7+3\sqrt{5}) + \omega^2(-3-\sqrt{5}) \end{bmatrix} \end{aligned} \tag{3.30}$$

By combining these expression for the coefficients (and $b_1 = b_2 = 0$) with the general solution (3.25), the following expression for the complete solution to the system of differential equations (3.4) is derived:

$$\begin{aligned} \mathbf{x}(t) = &\; F\left(\omega, \frac{k}{m}\right) \cdot \frac{\frac{k}{m}\left(-7+3\sqrt{5}\right) + \omega^2\left(3-\sqrt{5}\right)}{2\sqrt{5}} \begin{bmatrix} \frac{-1\sqrt{5}}{2} \\ 1 \end{bmatrix} \cos\left(\sqrt{\frac{k}{2m}\left(3+\sqrt{5}\right)}t\right) \\ + &\; F\left(\omega, \frac{k}{m}\right) \cdot \frac{\frac{k}{m}\left(7+3\sqrt{5}\right) + \omega^2\left(-3-\sqrt{5}\right)}{2\sqrt{5}} \begin{bmatrix} \frac{-1+\sqrt{5}}{2} \\ 1 \end{bmatrix} \cos\left(\sqrt{\frac{k}{2m}\left(3-\sqrt{5}\right)}t\right) \\ + &\; F\left(\omega, \frac{k}{m}\right) \cdot \begin{bmatrix} -\frac{2k}{m} + \omega^2 \\ -\frac{3k}{m} + \omega^2 \end{bmatrix} \cos(\omega t), \end{aligned} \tag{3.31}$$

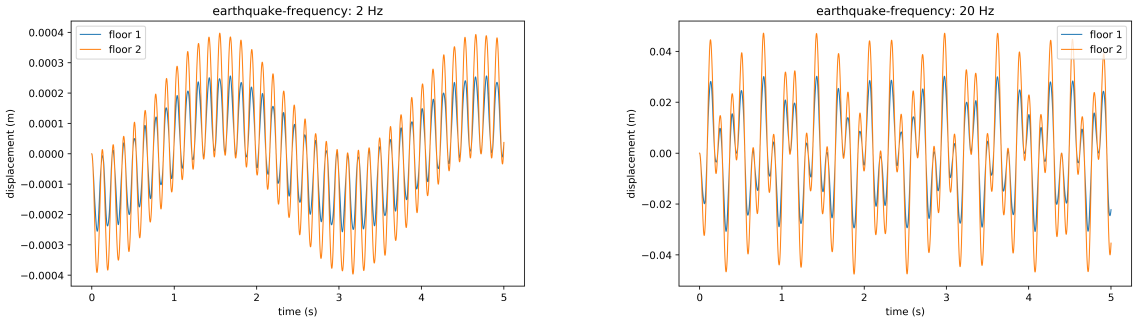
where

$$F\left(\omega, \frac{k}{m}\right) \equiv \frac{F_0\omega^2}{\left(\frac{k}{m}\right)^2 - 3\frac{k}{m}\omega^2 + \omega^4}. \tag{3.32}$$

Analysing the Analytical Solution

The solution (3.31) describes the oscillations within a two-floor building as described by the model introduced in section 3.0.1. From the solution, it can be seen that there are three frequencies that, with their

amplitudes, determine the oscillation-patterns of the floors. Two examples of the patterns of the oscillations of the floors is shown in figure 3.5.[6] It is worth noticing that only one of the frequencies depends on the earthquake's frequency ω, and the other two frequencies are merely dependent on the ratio $\frac{k}{m}$. These two frequencies are called the **eigenfrequencies** of the building, and are "natural" to the combination of dimensions and material properties of the building.



(a) Oscillations with a lower frquency.

(b) Oscillations with a higher frequency.

Figure 3.5: The different oscillation-patterns of the two floors caused by earthquakes whose frequencies are not near an eigenfrequency. The combination of eigenfrequencies and the earthquake-frequency cause the two figures to show different oscillation-patterns.

Another important notice about the solution, is that each (oscillating) term is multiplied by the coefficient $F\left(\omega, \frac{k}{m}\right)$. This term can grow considerably whenever its denominator gets closer to zero. When finding the particular solution, it was already established that the denominator of F has singularities for $\omega^2 = \frac{k}{2m}\left(3 \pm \sqrt{5}\right)$. Hence, if ω approaches $\sqrt{\frac{k}{2m}\left(3 \pm \sqrt{5}\right)}$, the amplitude of the oscillations will turn large. However, the fact that these values are equal to the eigenvalues of the system is essential. For example, by substituting $\omega \approx \omega_2 = \sqrt{\frac{k}{2m}\left(3 - \sqrt{5}\right)}$ into the solution, it is found that

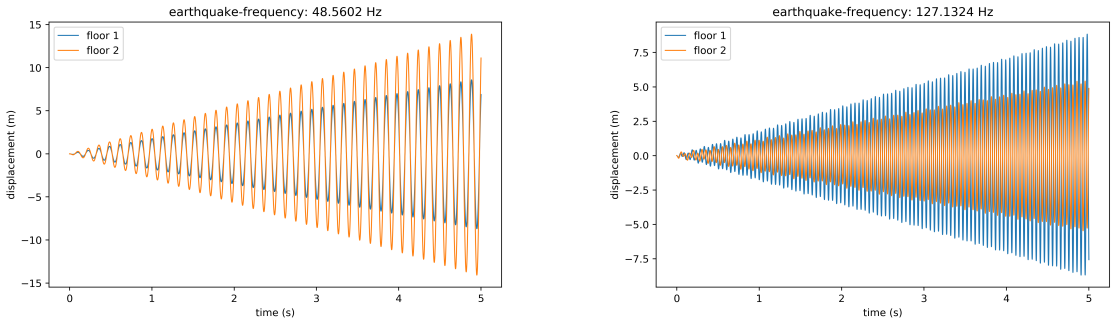
$$\begin{aligned}
\mathbf{x}(t) \approx \quad & F\left(\omega \approx \omega_2, \frac{k}{m}\right) \cdot \begin{bmatrix} 0 \\ 0 \end{bmatrix} \cos\left(\sqrt{\frac{k}{2m}\left(3+\sqrt{5}\right)}t\right) \\
+ \quad & F\left(\omega \approx \omega_2, \frac{k}{m}\right) \cdot \begin{bmatrix} \frac{k}{2m}\left(1+\sqrt{5}\right) \\ \frac{k}{2m}\left(-3+\sqrt{5}\right) \end{bmatrix} \cos\left(\sqrt{\frac{k}{2m}\left(3-\sqrt{5}\right)}t\right) \\
+ \quad & F\left(\omega \approx \omega_2, \frac{k}{m}\right) \cdot \begin{bmatrix} -\frac{k}{2m}\left(1+\sqrt{5}\right) \\ -\frac{k}{2m}\left(-3+\sqrt{5}\right) \end{bmatrix} \cos(\omega t) \\
\approx \quad & F\left(\omega \approx \omega_2, \frac{k}{m}\right) \cdot \begin{bmatrix} \frac{k}{2m}\left(1+\sqrt{5}\right) \\ \frac{k}{2m}\left(-3+\sqrt{5}\right) \end{bmatrix} (\cos(\omega_2 t) - \cos(\omega t))
\end{aligned} \tag{3.33}$$

The last rule of the above expression displays how resonance occurs mathematically. The amplitude of the oscillations is large due to the vast value of $F\left(\omega \approx \omega_2, \frac{k}{m}\right)$. However, the difference between the functions $\cos(\omega_2 t)$ and $\cos(\omega t)$ starts out small, but increases as t increases, and therefore the pattern of resonance shows limited oscillations at first, but will grow significantly with the passage of time.[7] This effect can be seen clearly in figure 3.6 below. Comparing these resonating patterns to the oscillation-patterns as shown in figure 3.5, it can be seen that within five seconds, the displacements during resonance grows up to 50 times the size of oscillations without resonance.[8]

[6] The Python-code for creating the plots can be found in Appendix B.1.1.

[7] This phenomenon can also been seen by considering the identity $\cos(\omega_2 t) - \cos(\omega t) = 2\sin(\frac{(\omega_2+\omega)t}{2})\sin(\frac{(\omega-\omega_2)t}{2})$. For small arguments, the sin-function can be approximated by its argument. Therefore, the expression for $\mathbf{x}(t)$ can be approximated by $\frac{F_0\omega^2}{(\omega^2-\omega_1^2)(\omega^2-\omega_2^2)}\frac{(\omega_2-\omega)t}{2}\sin(\frac{(\omega_2+\omega)t}{2}) = \frac{F_0\omega^2 t}{2(\omega_1^2-\omega^2)(\omega+\omega_2)}\sin(\frac{(\omega_2+\omega)t}{2})$, which causes the linear-like increase of the amplitude of the oscilaltions.

[8] The Python-code for creating the plots can be found in Appendix B.1.1.



(a) The earthquake's frequency is near the eigenfrequency $\omega_2 = \sqrt{\frac{k}{2m}(3-\sqrt{5})}$.

(b) The earthquake's frequency is near the eigenfrequency $\omega_1 = \sqrt{\frac{k}{2m}(3+\sqrt{5})}$.

Figure 3.6: The oscillation-patterns that occurs when the earthquake's frequency is close to an eigenfrequency (resonance).

3.1.2. The Numerical Solution Describing Behaviour for Varying Earthquake-Frequencies

To further illustrate the influence of the earthquake's frequency on the size of oscillations of a building, the maximum displacement within a certain time-range can be determined for varying frequencies. Not only does this portray the influence of the earthquake's frequencies, it will also be useful in demonstrating the effects of the dampers on the size of the oscillations. To do so, the oscillation-patterns of the model 3.3 have to be determined for a range of ω. In addition to that, as the amount of floors of a building increases, the computation time for determining the solution to the system of differential equations grows excessively. Therefore, it is of interest to consider a numerical integration method for approximating these displacements.

The Numerical Integration Method

In order to use a numerical integration method to approximate a solution to a system of second-order differential equations (3.3), it is required to rewrite the model using only first-order derivatives. To this end, first, the following vector-matrix notation is introduced to represent model (3.3) for n floors:

$$\mathbf{x}''(t) = K_n\mathbf{x} - F_0\omega^2 m\cos(\omega t)\mathbf{d}_n, \tag{3.34}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \dots \\ x_n(t) \end{bmatrix}, \quad K_n = \begin{bmatrix} -\frac{2k}{m} & \frac{k}{m} & 0 & 0 & \dots & 0 & 0 & 0 \\ \frac{k}{m} & -\frac{2k}{m} & \frac{k}{m} & 0 & \dots & 0 & 0 & 0 \\ 0 & \frac{k}{m} & -\frac{2k}{m} & \frac{k}{m} & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & \frac{k}{m} & -\frac{2k}{m} & \frac{k}{m} \\ 0 & 0 & 0 & 0 & \dots & 0 & \frac{k}{m} & -\frac{k}{m} \end{bmatrix}, \quad \mathbf{d}_n = \begin{bmatrix} 1 \\ 1 \\ \dots \\ 1 \end{bmatrix} \tag{3.35}$$

Second, the new variables $z_j(t)$ will be introduced, such that $z_i(t) \equiv x_i'(t)$ and $z_{i+n}(t) \equiv x_i(t)$ for $i = 1, 2, \dots, n$. This leads to the following system of first-order differential equations, which is equivalent to model (3.3):

$$\mathbf{z}'(t) = \tilde{K_n}\mathbf{z} - F_0\omega^2\cos(\omega t)\tilde{\mathbf{d}}_n, \tag{3.36}$$

where

$$\mathbf{z} = \begin{bmatrix} z_1(t) \\ z_2(t) \\ \dots \\ z_{2n}(t) \end{bmatrix}, \quad \tilde{K_n} = \begin{bmatrix} [0]_{n\times n} & K_n \\ I_n & [0]_{n\times n} \end{bmatrix}, \quad \tilde{\mathbf{d}}_n = \begin{bmatrix} \mathbf{d}_n \\ \mathbf{0}_n \end{bmatrix} \tag{3.37}$$

This system of first-order differential equations will be used to determine the numerical approximations of the solution to model (3.3). The numerical integration method that will be applied to this system is the method Runge-Kutta 4 (RK4). By defining

$$\mathbf{f}(t,\mathbf{z}(t)) \equiv \tilde{K_n}\mathbf{z}(t) - F_0\omega^2\cos(\omega t)\tilde{\mathbf{d}}_n, \tag{3.38}$$

RK4 can be described as follows [23]:

$$\begin{aligned}
\mathbf{k}_1 &= \Delta t\mathbf{f}(t_n,\mathbf{w}_n)\,,\\
\mathbf{k}_2 &= \Delta t\mathbf{f}\left(t_n+\frac{\Delta t}{2},\mathbf{w}_n+\frac{\mathbf{k}_1}{2}\right),\\
\mathbf{k}_3 &= \Delta t\mathbf{f}\left(t_n+\frac{\Delta t}{2},\mathbf{w}_n+\frac{\mathbf{k}_2}{2}\right),\\
\mathbf{k}_4 &= \Delta t\mathbf{f}(t_n+\Delta t,\mathbf{w}_n+\mathbf{k}_3)\,,\\
\mathbf{w}_{n+1} &= \mathbf{w}_n+\frac{1}{6}(\mathbf{k}_1+2\mathbf{k}_2+2\mathbf{k}_3+\mathbf{k}_4).
\end{aligned} \tag{3.39}$$

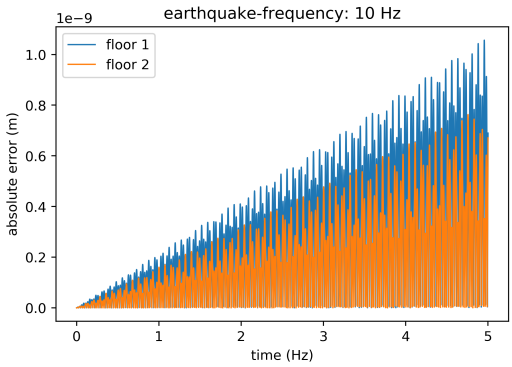
To analyse for which time-steps Δt RK4 converges, for all eigenvalues λ_i of matrix $\tilde{K_n}$, the stability bound on the amplification factor of the method must be analysed:

$$|Q(\lambda_i\Delta t)| = \left|1+\lambda_i\Delta t+\frac{(\lambda_i\Delta t)^2}{2}+\frac{(\lambda_i\Delta t)^3}{6}+\frac{(\lambda_i\Delta t)^4}{24}\right| \quad < \quad 1. \tag{3.40}$$

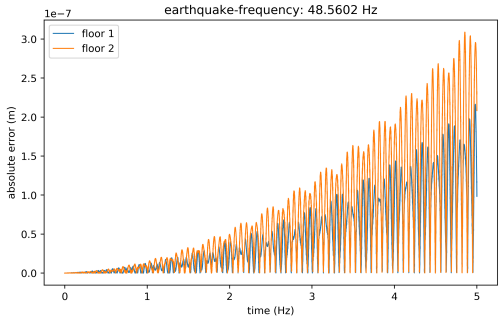
In Appendix A, an example of determining the bound on Δt will be worked out. In the Python-codes that use the numerical method, the stability bound will be verified for the exploited time-step.

As shown in figure 3.7, using the numerical time-integration method RK4 with time-step $\Delta t = 0,00025$ s, the oscillation patterns can be approximated accurately, with a relative error of at most 0.008 within five seconds, where the relative error is defined as the ratio of the absolute error and the analytical solution. However, the growing abolute erros, as shown in figures 3.7a and 3.7b, are growing larger as time passes.[9] This will not be a problem, as enough information concerning the behaviour of the oscillations can be derived from the numerical approximation within five seconds (see figures 3.5 and 3.6).

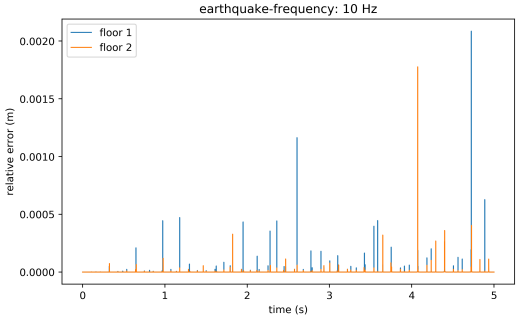
[9]The Python -code for creating the plots can be found in Appendix B.1.2.



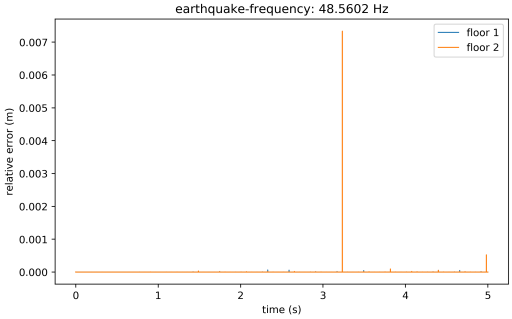
(a) The absolute error within 10 seconds, without resonance.



(b) The absolute error within 10 seconds, with resonance.



(c) The relative error within 10 seconds, without resonance.

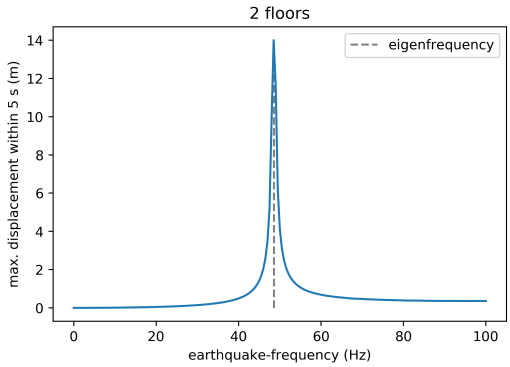


(d) The relative error within 10 seconds, with resonance.

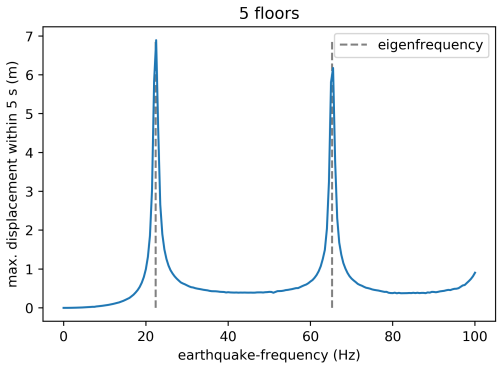
Figure 3.7: The absolute and relative error for approximating the oscillations of a building of two floors within 100 seconds, using time-step $\Delta t = 0,00025$ s, with and without resonance.

The Behaviour of the Building

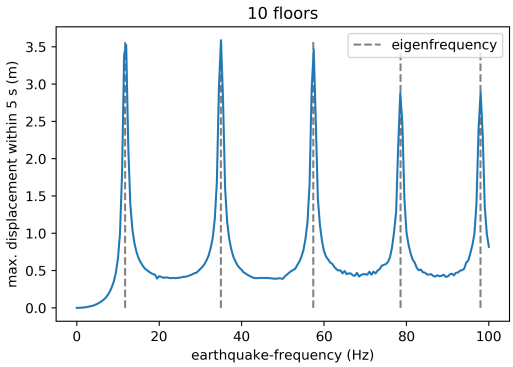
Based on the numerical approximations, the behaviour of the maximum displacements of the buildings described by model 3.3 for varying earthquake-frequencies can be determined. In figure 3.11, the relation between maximum displacement and earthquake-frequency ω is shown for buildings of two, five and ten floors. As expected, the displacement of the building is considerably larger around eigenfrequencies than for other earthquake-frequencies. Furthermore, it is worth noticing that larger buildings have more eigenfrequencies lying in the range of possible values for the earthquake's frequency. This is expected, as the amount of eigenfrequencies of a building is equal to the number of floors. Moreover, the tallest building of ten floors shows significantly larger displacements for earthuqake-frequencies around its eigenfrequencies.[10]



(a) A building of two floors.



(b) A building of five floors.

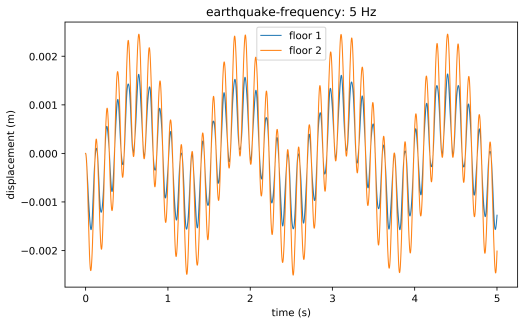


(c) A building of ten floors.

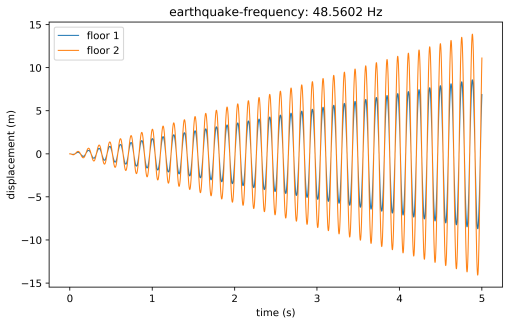
Figure 3.8: The maximum displacements of buildings of two, five and ten floors, for earthquake-frequencies varying from 0.001 Hz to 100 Hz.

In figure 3.9, it can be seen that for earthquake-frequencies that are not close an eigenfrequency, the numerical approximation of the model shows bounded oscillating behaviour. For frequencies close to eigenfrequencies, the oscillations, again, show a resonating pattern. These oscillations will grow unbounded, and therefore it is required that they be damped before causing irreversible damage.[11]

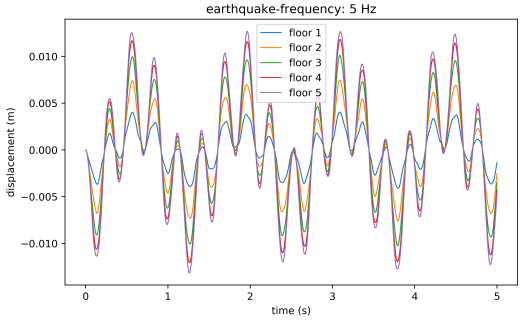
[10] The Python-code used for creating these figures can be found in Appendices B.1.3, B.1.4 and B.1.5
[11] The Python-code for creating these figures can be found in Appendices B.1.6, B.1.7 and B.1.8.



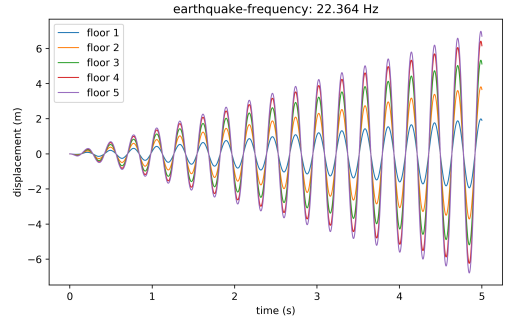
(a) An oscillation-pattern of a non-resonating frequency for a building of two floors.



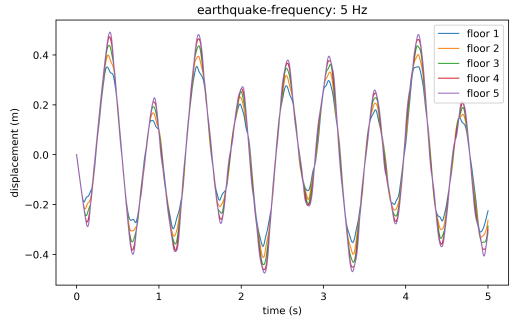
(b) An oscillation-pattern of a resonating frequency for a building of two floors.



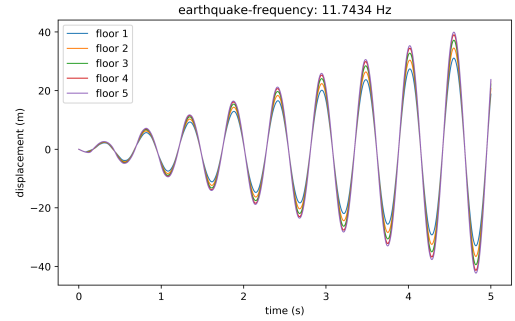
(c) An oscillation-pattern of a non-resonating frequency for a building of five floors.



(d) An oscillation-pattern of a resonating frequency for a building of five floors.



(e) An oscillation-pattern of a non-resonating frequency for a building of ten floors.



(f) An oscillation-pattern of a non-resonating frequency for a building of ten floors.

Figure 3.9: Oscillation-patterns for resonating and non-resonating earthquake-frequencies, for buildings of two, five and ten floors.

3.2. The Properties of the Tuned Mass Damper within the Model

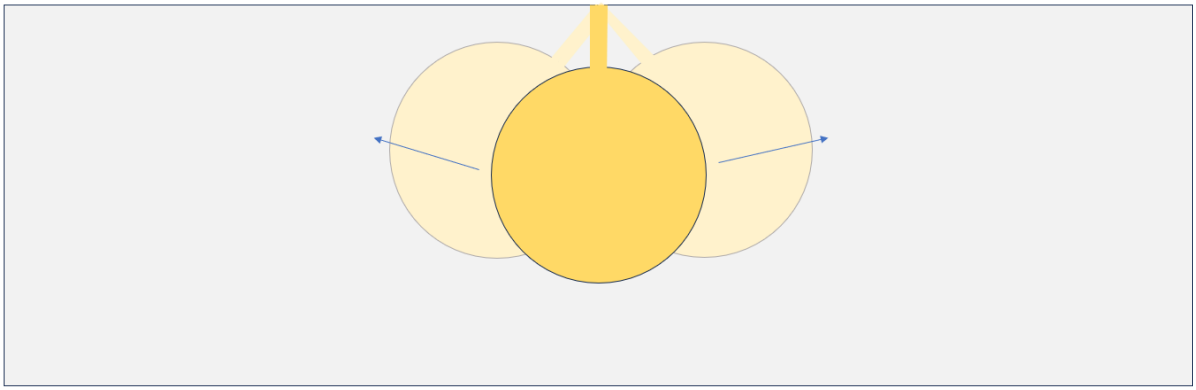
In this section, it will be described how a Tuned Mass Damper can be implemented in a building, and how it is able to damp the frequency it is tuned to.

3.2.1. Implementation of the Tuned Mass Damper in a Building

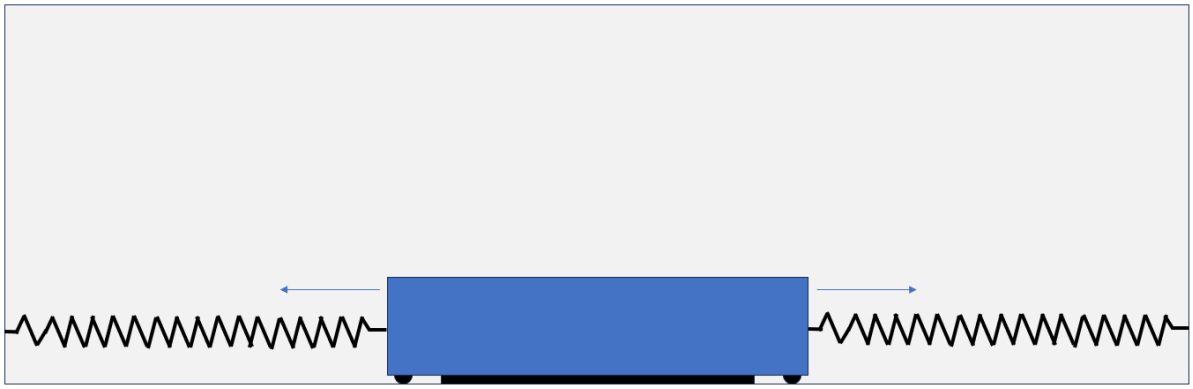
A Tuned Mass Damper (TMD) is, as the name suggests, a device that is tuned to reduce oscillations of a specific frequency. It is a large mass, attached to a floor in a building, either as a pendulum (figure 3.10a), or as an actual spring-mass system (figure 3.10b). [12] Therefore, the configuration of a TMD attached to a floor resembles the spring-mass system as demonstrated in figure 3.10c, a TMD may be attached by an extra

[12] The restoring force of the pendulum can also be described by Hooke's law, $f_{\text{Linear Restoring}}(t) = k\Delta x$.

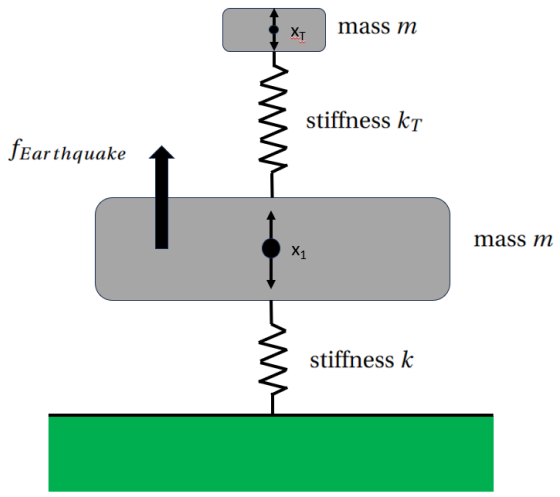
damper. Both systems can be modeled as a spring-mass system, as shown in figure 3.10c. However, the TMD depicted in figure 3.10d is also attached to the building from below, by means of a viscous damper. Since the focus of the research is centered on determining whether the TMD or the Viscoelastic Damper is the most effective damper, such combinations of a TMD and a vicous damper will not be considered. In other words, the TMD will be modeled as a mass with a linear restoring force.



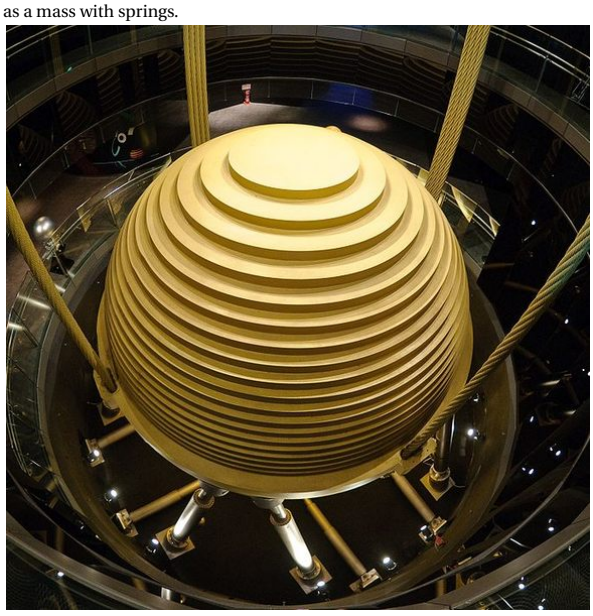
(a) A TMD implemented as a pendulum.



(b) A TMD implemented as a mass with springs.



(c) The configuration of a TMD attached to a floor.



(d) The TMD in Taipei 101, Taiwan, which is implemented as a pendulum, but also is also attached from below by viscous dampers [1].

Figure 3.10: Pictures indicating how a TMD is executed in a building.

3.2.2. Explanation of the Working Principle of TMD

The working principle of a TMD will be described by adding a TMD to the model of a building of one floor ($n = 1$ in (3.3)):

$$x_1'' = -\frac{k}{m}x_1 - F_0\omega^2\cos(\omega t). \tag{3.41}$$

The analytical solution describing the oscillations of the floor in this model can be determined by finding a particular and a homogeneous solution, as has been done in subsection 3.1.1 for the model of two floors. Executing this method yields:

$$x_1(t) = \frac{F_0\omega^2}{\omega^2 - \frac{k}{m}}\left(\cos\left(\sqrt{\frac{k}{m}}t\right) - \cos(\omega t)\right). \tag{3.42}$$

It can be derived that the eigenfrequency of the floor in this model is equal to $\sqrt{\frac{k}{m}}$

By adding a TMD to the model, a new variable denoting the displacement of the damper is required. Denote $x_T(t)$ as the displacement of the TMD. Then, since the damper is modelled as a mass attached to the floor by a spring (figure 3.10c), the model of one floor including a TMD converts to:

$$\begin{cases} mx_1'' = -kx_1 - k_T(x_1 - x_T) - F_0\omega^2 m\cos(\omega t), \\ mx_T'' = -k_T(x_T - x_1). \end{cases} \tag{3.43}$$

It is noteworthy that this system is very similar to the model of two floors, as in (3.4). To derive the solution to this model, the method that was worked out in subsection 3.1.1 can be applied once more, and a similar computation follows. The solution thence derived can be expressed as

$$\begin{aligned} \begin{bmatrix} x_1(t) \\ x_T(t) \end{bmatrix} = \quad & F_T\left(\omega, \frac{k}{m}, \frac{k_T}{m_T}\right) \cdot \frac{\frac{k_T}{m_T}(\omega^2 - \omega_2^2)}{\omega_1^2 - \omega_2^2} \begin{bmatrix} \frac{k_T}{m_T} - \omega_1^2 \\ \frac{k_T}{m_T} \\ 1 \end{bmatrix} \cos(\omega_1 t) \\ + \quad & F_T\left(\omega, \frac{k}{m}, \frac{k_T}{m_T}\right) \cdot \frac{\frac{k_T}{m_T}(-\omega^2 + \omega_1^2)}{\omega_1^2 - \omega_2^2} \begin{bmatrix} \frac{k_T}{m_T} - \omega_2^2 \\ \frac{k_T}{m_T} \\ 1 \end{bmatrix} \cos(\omega_2 t)' \\ + \quad & F_T\left(\omega, \frac{k}{m}, \frac{k_T}{m_T}\right) \cdot \begin{bmatrix} \omega^2 - \frac{k_T}{m_T} \\ -\frac{k_T}{m_T} \end{bmatrix} \cos(\omega t) \end{aligned} \tag{3.44}$$

where:

$$F_T\left(\omega, \frac{k}{m}, \frac{k_T}{m_T}\right) \equiv \frac{F_0\omega^2}{\frac{k}{m}\left(\frac{k_T}{m_T} - 2\omega^2\right) - \omega^2\frac{k_T}{m_T} + \omega^4}, \tag{3.45}$$

and ω_1 and ω_2 are the eigenfrequencies of the system including a TMD:

$$\omega_{1,2} = \sqrt{\frac{1}{2}\left(2\frac{k}{m} + \frac{k_T}{m_T} \pm \sqrt{4\left(\frac{k}{m}\right)^2 + \left(\frac{k_t}{m_T}\right)^2}\right)}. \tag{3.46}$$

From this solution, the working principle of a TMD can be seen. Assuming the frequency of the earthquake acting on the building is known, the values of the parameters m_T (mass) and k_T (stiffness) can be chosen such that their ratio $\frac{k_T}{m_T}$ equals the square of the earthquake's frequency, ω^2. Substituting this into (3.45) yields

$$F_T\left(\omega, \frac{k}{m}, \omega^2\right) = -\frac{F_0}{\frac{k}{m}}. \tag{3.47}$$

which is a bounded factor, independent of the value of ω. With this and further substitution of $\frac{k_t}{m_T} = \omega^2$, the solution (3.44) converts to

$$\begin{bmatrix} x_1(t) \\ x_T(t) \end{bmatrix} = \quad -\frac{F_0}{\frac{k}{m}} \cdot \left(\begin{bmatrix} \frac{(\omega^2 - \omega_1^2)(\omega^2 - \omega_2^2)}{\omega_1^2 - \omega_2^2} \\ \omega^2 \end{bmatrix} (\cos(\omega_1 t) - \cos(\omega_2 t)) + \begin{bmatrix} 0 \\ \omega^2 \end{bmatrix} \cos(\omega t) \right). \tag{3.48}$$

Based on this solution, it can be concluded that by tuning a TMD such that the ratio $\frac{k_T}{m_T}$ equals ω^2, the effect of the earthquake's frequency is completely damped out (figure 3.11). Therefore, the floor will only oscillate with the (new) eigenfrequencies of the system, ω_1 and ω_2, and resonance cannot occur. However, this only occurs when the frequency of an earthquake acting on the building can be predicted before installing the TMD. Since this is not the case, and (up until now) the configuration of TMDs cannot be adapted to the frequency of an earthquake during an earthquake, the tuning should be done beforehand.

The tuning of a TMD is mostly done by setting the ratio $\frac{k_T}{m_T}$ equal to the square of an eigenfrequency of the building when no TMD is included [22], since that is the frequency around which resonance occurs. Hence, the TMD will be tuned such that $\frac{k_T}{m_T} = \frac{k}{m}$. To demonstrate the effect of this decision, the tuning will be substituted into the factor (3.45) and the solution (3.44) once again, yielding

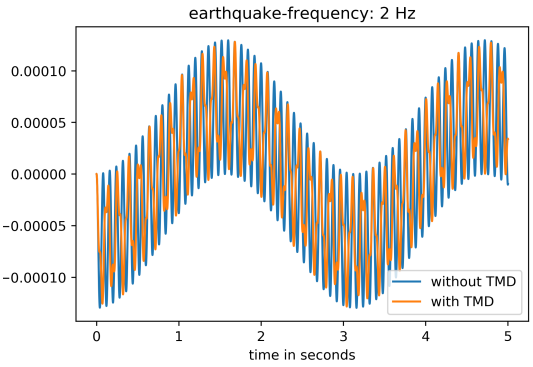
$$F_T\left(\omega, \frac{k}{m}, \frac{k}{m}\right) = \frac{F_0\omega^2}{\left(\frac{k}{m}\right)^2 - 3\omega^2\frac{k}{m} + \omega^4}, \tag{3.49}$$

and

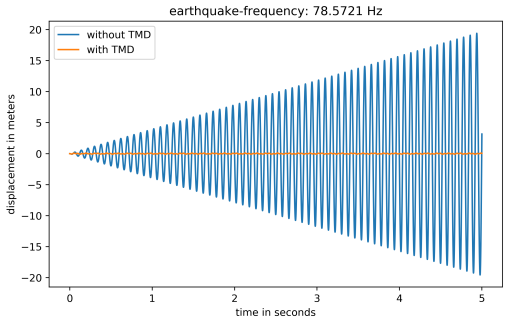
$$\begin{aligned}\begin{bmatrix} x_1(t) \\ x_T(t) \end{bmatrix} = &\quad F_T\left(\omega, \frac{k}{m}, \frac{k}{m}\right) \cdot \frac{(\omega^2 - \omega_2^2)}{\omega_1^2 - \omega_2^2} \begin{bmatrix} \frac{k}{m} - \omega_1^2 \\ \frac{k}{m} \end{bmatrix} \cos(\omega_1 t) \\ &+ \quad F_T\left(\omega, \frac{k}{m}, \frac{k}{m}\right) \cdot \frac{(-\omega^2 + \omega_1^2)}{\omega_1^2 - \omega_2^2} \begin{bmatrix} \frac{k}{m} - \omega_2^2 \\ \frac{k}{m} \end{bmatrix} \cos(\omega_2 t) \\ &+ \quad F_T\left(\omega, \frac{k}{m}, \frac{k}{m}\right) \cdot \begin{bmatrix} -\frac{k}{m} + \omega^2 \\ -\frac{k}{m} \end{bmatrix} \cos(\omega t).\end{aligned} \tag{3.50}$$

Substitution of $\omega \approx \sqrt{\frac{k}{m}}$ yields an expression similar to (3.48), in which the influence of the earthquake's frequency is (nearly) entirely damped out. However, the solution does show a recurring factor F_T which can grow significantly large whenever its denominator approaches zero. It can be shown that this occurs for earthquake-frequencies near the eigenfrequencies ω_1 and ω_2 of the system. As demonstrated in figure 3.17c, resonance occurs for values close to these new eigenfrequencies.[13]

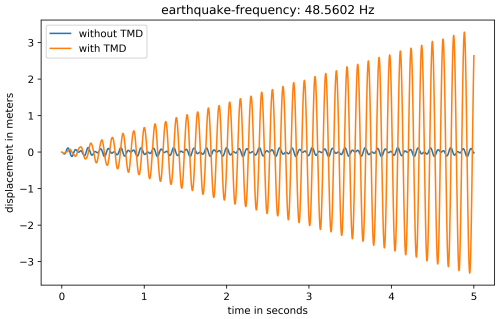
[13] The Python-code for creating these figures can be found in Appendix B.1.9.



(a) The influence of a TMD on the oscillation-pattern for an earthquake-frequency neither near the eigenfrequency of the building without damping, nor near a "new" eigenfrequency of the building including a TMD.



(b) The influence of a TMD on the oscillation-pattern for an earthquake-frequency near the eigenfrequency of the building without damping (the tuned frequency).



(c) The influence of a TMD on the oscillation-pattern for an earthquake-frequency near a "new" eigenfrequency of the building including a TMD.

Figure 3.11: The influence of a TMD on the building for several earthquake-frequencies.

3.3. The Properties of the Viscoelastic Damper within the Model

In this section, the implementation of the Viscoelastic Damper is described, followed by a subsection in which a function will be fitted to the viscoelastic damping constant (which is dependent on the earthquake's frequency), and the mathematical principle which describes the influence of the damper will be revealed.

3.3.1. Implementation of the Viscoelastic Damper in a Building

A Viscoelastic Damper (VED) is an oscillation-reducement device that contains a material possessing both viscous (for fluid-like materials) and elastic (for solids) properties. It can be implemented in a building by attaching it to a construction of braces (examples are shown in fgure 3.12).

The damping of a VED is caused by the conversion of mechanical energy (from movement of the building) into heat, due to friction occuring when the material inside the VED deforms (viscocity). Furthermore, the material inside the VED will deform back, by its elastic properties. The damping force exerted by a damping mechanism can be modeled by the equation $F_{\text{Damping}} = c\Delta v$ [24], where c is a damping coefficient, and Δv is the difference in velocity between the floor and the floor below it. A configuration for modelling a damper on a floor is shown in figure 3.13, in which C_V is denoted as the viscoelastic damping coefficient.

In other words, when including a VED in the building, the floor to which the damper is attached will be of the following form:

$$\frac{d}{dt}P(x_i(t)) = f_{\text{Linear Restoring}}(t) + f_{\text{Earthquake}}(t) + f_{\text{Damping}}, \tag{3.51}$$

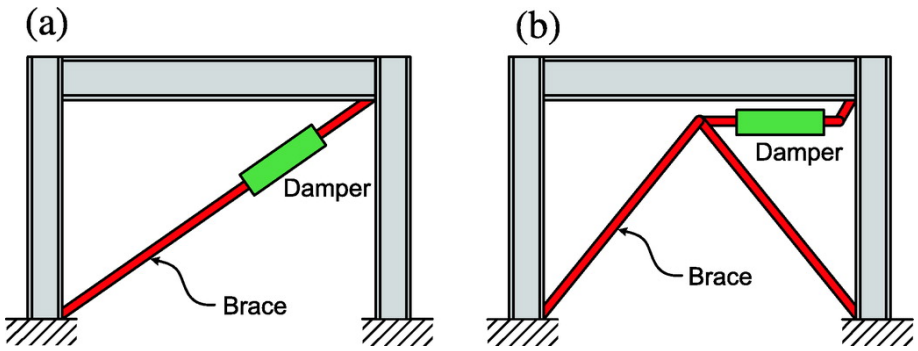


Figure 3.12: Possible installation methods for VEDs[6].

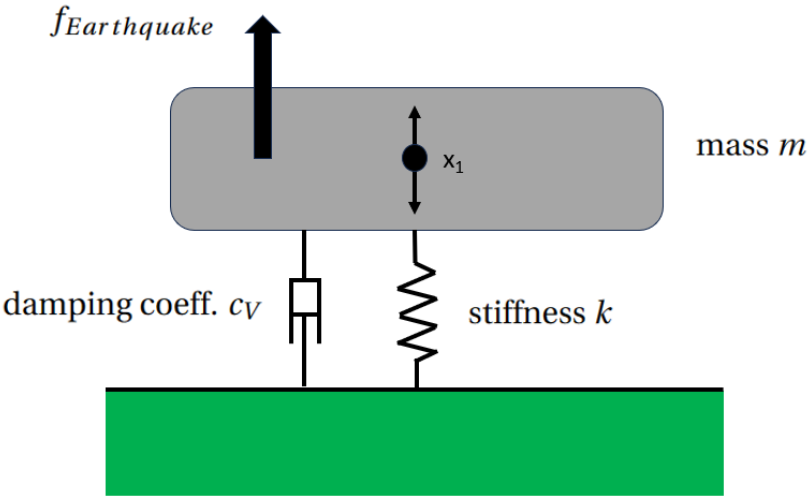


Figure 3.13: The configuration for adding a VED to a the model for one floor.

where $F_{\text{Damping}} = c_V(x'_i - x'_{i-1})$. Due to the elastic and viscous properties of a VED (the damper requires time to return to its original state after deformation), the coefficient c_V is dependent on the frequency of the earthquake. Since limited data is available for the value of the viscoelastic damping coefficient, the next subsection will concern finding a fitting function for the values of $c_V(\omega)$.

3.3.2. Determining the Viscoelastic Damping Coefficient

The data that is considered for the values of c_V is based on [19], and concerns various types fof viscoelastic dampers. Their data points can be found in table **??**. First of all, for each of the types of VEDs, a fitting function will be determined. Afterwards, a decision for the type of VED that will be modelled will be made.

Type of VED	Nominal load (kN)	5 Hz	10 Hz	15 Hz	20 Hz	25 Hz	30 Hz	35 Hz
VM010551E	5	11,6	8,7	7,3	6,5	5,9	5,5	5,2
VM020551E	10	24,07	18,1	15,3	13,6	12,4	11,5	10,8
VM030551E	15	41,0	30,8	26,1	23,2	21,1	19,6,	18,4
VM055551E	25	100,0	75,3	63,7	56,6	51,7	47,9	45,0
VM100551E	40	217,3	163,5	138,5	123,1	112,3	104,2	97,8
VM175551E	50	438,3	329,9	279,4	248,3	226,6	210,2	197,4

Table 3.1: The values of the viscoelastic damping coefficient c_V for various frequencies [19].

For determining the function $c_V(\omega)$ that fits the data points best, it is important to take two facts into con-

sideration. First of all, the function must fit the data points. Second, the function should incorporate the expected behaviour for the value of c_V. By examining the data points as plotted in figure 3.14, we see that the values of c_V show a descending behaviour. First, the Lagrange Interpolation Polynomial will be considered.

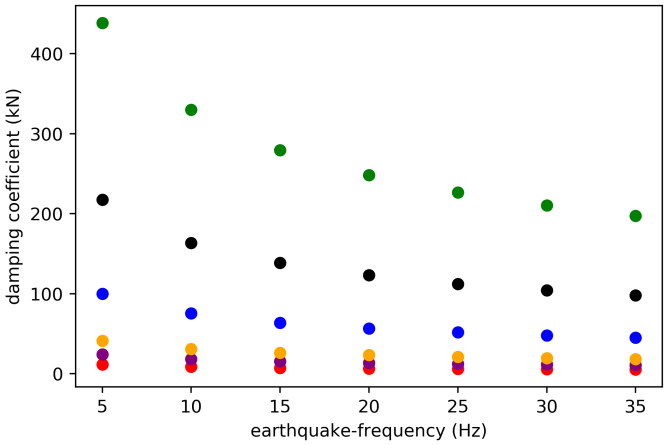


Figure 3.14: The given data poins from table **??** for the viscoelastic damping coefficient c_V.

The Sixth Degree Lagrange Interpolation Polynomial

Lagrange Interpolation is a method to determine a polynomial that fits a set of data points. Since for every type of VED, seven data points are available of the form (frequency, coefficient), the sixth degree Lagrange Interpolation Polynomial can be determined. The Lagrange Polynomials are of the form [23]

$$c_V(\omega) = a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 + a_4\omega^4 + a_5\omega^5 + a_6\omega^6. \tag{3.52}$$

The coefficients a_j can be determined by solving the system that follows by substituting the seven data points into this polynomial. Thence, the Lagrange Polynomial for each type of VED is determined. From figure 3.15 below, it is clear that the Lagrange Polynomial fits the data points for $\omega \leq 35$ Hz. However, for larger earthquake-frequencies, the polynomial diverges, which is behaviour that is not expected from c_V, based on the given data points. Therefore, a different approach for determining the behaviour of c_V for larger values of ω is required.

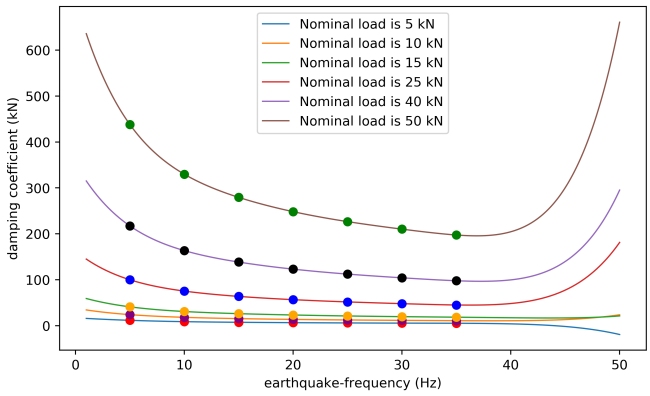


Figure 3.15: The sixth degree Lagrange Interpolation Polynomial, fitted to the given data points.

The Function $\mathbf{c_V}(\omega) = \dfrac{\mathbf{a}}{\omega^{\mathbf{b}}}$

As ω decreases towards 0.001, the data points show a growth with increasing slope. Therefore, a function of the form $c_V(\omega) = \frac{a}{\omega^b}$ fits the data points. This function includes two parameters, a and b, that need to be determined. To do so, two data points must be substituted in the equation, and the two outer points are used for this. Denoting these by (ω_1, c_1) and (ω_2, c_2), a system of equations is derived:

$$\begin{cases} c_1 = \dfrac{a}{\omega_1^b}, \\ c_1 = \dfrac{a}{\omega_1^b}. \end{cases} \tag{3.53}$$

Solving for a and b yields:

$$a = \exp\left(\frac{\ln\omega_1 \ln c_2 - \ln\omega_2 \ln c_1}{\ln\omega_1 - \ln\omega_2}\right), \tag{3.54}$$

$$b = \frac{\ln c_2 - \ln c_1}{\ln\omega_1 - \ln\omega_2}. \tag{3.55}$$

For every type of VED, the plot of the function $c_V(\omega) = \frac{a}{\omega^b}$ is displayed in figure 3.16. As can be seen, the function seems to fit the data points very well, and follows the behaviour that is expected form c_V based on the data points, for ω greater than 35.

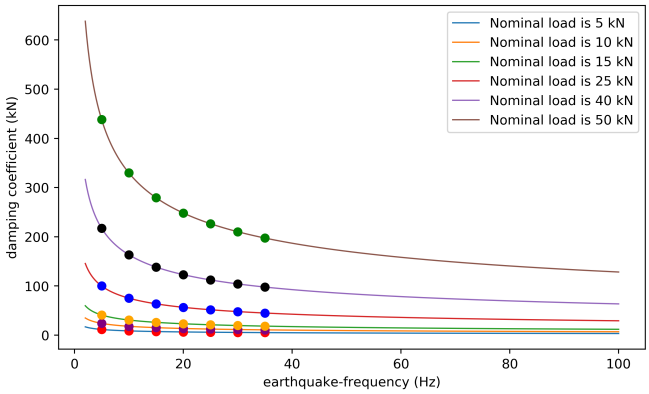


Figure 3.16: The function $c_V = \frac{a}{\omega^b}$, fitted to the given data points.

Hence, by assuming the trend of the viscoelastic damping coefficient follows a trend that is monotonely decreasing as ω increases, it is concluded that the function $c_V(\omega) = \frac{a}{\omega^b}$ can be well used to describe the behaviour of the viscoelastic damping coefficient, based on the data points. The values of the parameters a and b are given in (3.54) and (3.55), and are dependent on the type of VED which has been chosen.

3.3.3. Explanation of the Working Principle of the VED

Implementing eqaution (3.51) in the model of one floor yields:

$$mx_1'' = -kx_1 - c_V x_1' - F_0\omega^2 m\cos(\omega t), \tag{3.56}$$

where $c_V = c_V(\omega)$. Applying the standard method of determining a particular and a homogeneous solution, and substituting the initial conditions, this second-order ordinary differential equation can be solved, which yields the following expressions for the solution:

- if $c_V^2 < 4km$:

$$\begin{aligned} x_1(t) = \; & e^{-\frac{c_V}{2m}t}\left[-A\cos\left(\sqrt{D}t\right) + \frac{1}{\sqrt{D}}\left(-\frac{c_V}{2m}A - \omega B\right)\sin\left(\sqrt{D}t\right)\right] \\ + \; & A\cos(\omega t) + B\sin(\omega t); \end{aligned} \tag{3.57}$$

- if $c_V^2 = 4km$:

$$x_1(t) = \quad e^{-\frac{c_V}{2m}t}\left[-A + \left(-\frac{c_V}{2m}A - \omega B\right)t\right] + \quad A\cos(\omega t) + B\sin(\omega t); \tag{3.58}$$

- if $c_V^2 > 4km$:

$$\begin{aligned} x_1(t) = \quad & -\frac{1}{2\sqrt{-D}}\left\{\left(\frac{c_V}{2m} + \sqrt{-D}\right)A + \omega B\right\} e^{\left(-\frac{c_V}{2m} + \sqrt{-D}\right)t} \\ & -\frac{1}{2\sqrt{-D}}\left\{\left(\frac{c_V}{2m} + \sqrt{-D}\right)A - \omega B\right\} e^{\left(-\frac{c_V}{2m} - \sqrt{-D}\right)t} \\ & + \quad A\cos(\omega t) + B\sin(\omega t), \end{aligned} \tag{3.59}$$

where:

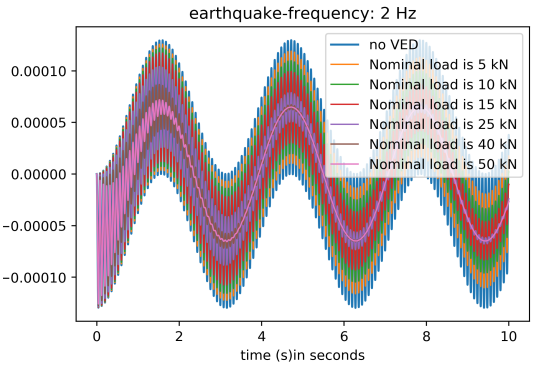
$$A = \frac{F_0\omega^2\left(\frac{k}{m} - \omega^2\right)}{\left(\frac{c_V\omega}{m}\right)^2 + \left(\frac{k}{m} - \omega^2\right)^2}, \tag{3.60}$$

$$B = -\frac{F_0\omega^2\frac{c_V\omega}{m}}{\left(\frac{c_V\omega}{m}\right)^2 + \left(\frac{k}{m} - \omega^2\right)^2}, \tag{3.61}$$

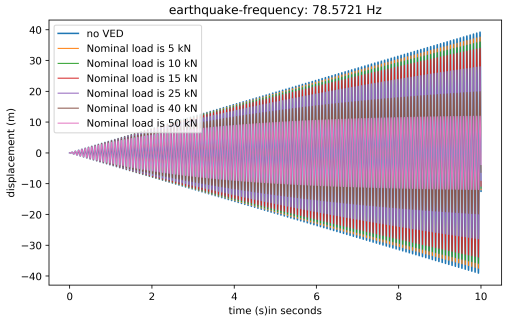
$$D = \frac{k}{m} - \left(\frac{c_V\omega}{2m}\right)^2. \tag{3.62}$$

All three expressions for the solution to the model (3.56) consist of two parts: first, a term that is multiplied by an exponantial function with a negative, time-dependent exponent, and second, the terms $A\cos(\omega t) + B\sin(\omega t)$. The first part of each solution, which includes the exponent, will be damped out as time advances,[14] and therefore each solution converges towards an oscillations described by $A\cos(\omega t) + B\sin(\omega t)$. Note that A and B, (3.61) and (3.62), respectively, are both bounded amplitudes, as their denominator will not approach zero (unless ω approaches zero and $\frac{k}{m}$ is close to zero, which is not the case). The rate of convergence is determined by the magnitude of the value of c_V, as shown in figure 3.17 (note that a larger nominal load implies that the damping coefficient is larger, see figure 3.16).

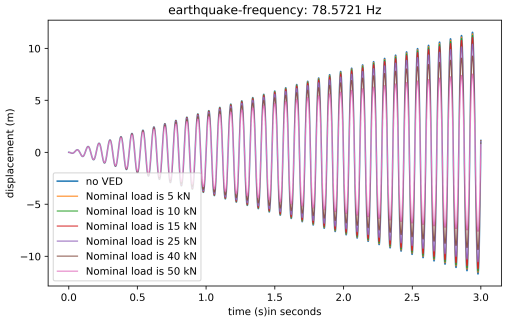
[14] Note that, in the solution for $c_V^2 > 4km$, the exponent $-\frac{c_V}{2m} + \sqrt{-D}$ too, is negative for any values of ω and $c_V > 0$: $-\frac{c_V}{2m} + \sqrt{-D} = -\frac{c_V}{2m} + \sqrt{\left(\frac{c_V\omega}{2m}\right)^2 - \frac{k}{m}} < -\frac{c_V}{2m} + \frac{c_V}{2m} = 0$.



(a) The influence of a VED on the oscillation-pattern for an earthquake-frequency that is not near an eigenfrequency of the building.



(b) The influence of a VED on the oscillation-pattern for an earthquake-frequency that is near an eigenfrequency of the building.



(c) The influence of a VED on the oscillation-pattern for an earthquake-frequency that is near an eigenfrequency of the building, zoomed in on the first three seconds.

Figure 3.17: The effect of a VED of several types on the oscillations of a building of one floor.

To conclude which type of VED is to be chosen for analysing the effectivity of VEDs, it will be determined which VED-type is most effective in damping oscillations. Figure 3.17 shows the effects of different types of VEDs when two of them are implemented in the model of a one-floor building. As can be seen, both for resonating and for non-resonating oscillation-patterns, the VEDs with a nominal load of 50 kN damp oscillations best. Therefore, this will be the type of VED which will be modelled when assessing the effectivity of VEDs.

4

Dampers on the Model without Air Resistance

In this chapter, the methods will be given, according to which the dampers will be included in the model. Furthermore, the results that follow from this will be described, with the conclusion on which damper is the most effective. Finally, a shortcoming of the model that is used for creating the results is mentioned.

4.1. Methods of Adding Dampers to the Model

All results will be created by using the numerical method RK4, as described in sebsection 3.1.2, with time-step $\Delta t = 0{,}00025$ s. The codes that are used to create the reults can be found in Appendix B.1.

Adding TMDs

TMDs will be added to the models of two, five and ten floors according to the method described below. All TMDs will have equal mass, set to 40% of the mass of a floor [3]. TMDs will be included in the models following the mathematical expressions described in subsection 3.2.2.

1. A single TMD will be implemented in the building, which will be tuned to the eigenfrequency that causes the largest maximum displacement. According to figure 3.11, for all buildings, the TMD will be tuned to damp the lowest eigenfrequency of the building. The TMD's position will be varied over every floor. Based on the results, it will be determined on which floor the TMD has the most effect.
2. For the buildings of five and ten floors, the TMD will be positioned on the floor it is most effective on. Furthermore, a second TMD will be implemented, tuned to the frequency which causes the largest oscillations when the first TMD is added to the model. Its position will be varied over the floors, to determine on which floor the second TMD has most effect.
3. For the building of two floors, the TMD will be positioned on the floor it is most effective on. Since the TMDs are expected to cause resonance at different frequencies than the eigenfrequencies of the model without damping, two new TMDs will be added to the model. Their tuning is set to the new resonating frequencies, and their positions will be varied over the two floors.

Adding VEDs

Similar to the TMDs, VEDs will be included in the models of two, five and ten floors. The dampers will always be included in the building in sets of ten. All VEDs are of the type that have a nominal load of 50 kN (see

subsection 3.3.2). They will be added to the models of two, five and ten floors, according to the following method.

1. A set of ten of VEDs will be implemented in the building, and their position will be varied over each floor. Based on this, the position will be chosen where the VEDs are most effective.
2. The first set of VEDs will be included in the model, and positioned on the floor where the dampers are most effective. A second set of ten VEDs is added, whose position will be varied.

Assessment

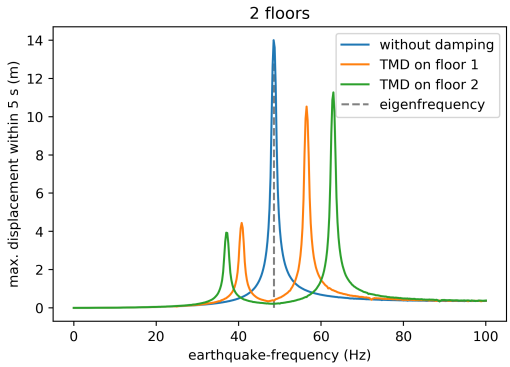
The effectivity of both dampers will be assessed based on their ability to reduce resonating oscillations as well as non-resonating oscillations of a building.

4.2. Results of Dampers on the Model

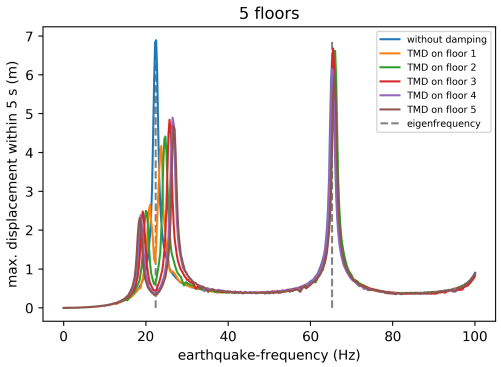
4.2.1. Results of adding TMDs

Figure 4.1 shows the effect of adding one TMD to the models for buildings of two, five and ten floors. As expected, the oscillations for the lowest eigenfrequency are damped significantly in each building, and no resonance occurs according to figure 4.2b. However, for certain frequencies around it, resonance does occur, as can be seen in figure 4.2a. Though the new peaks that can be seen in figure 4.1 are largest when the TMD is added to the highest floor of each building, the oscillations for the eigenfrequency are damped most, and therefore the position of the highest floor is decided to be the location where the TMD is most effective.

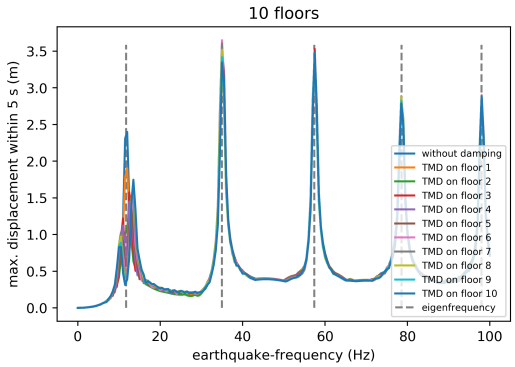
Placing a TMD on the highest floor of each building, figure 4.3 shows the effect of a second TMD - tuned to the second-lowest eigenfrequency - on the maximum oscillations within five seconds, for buildings of five and ten floors. Once again, the TMDs damp the oscillations for the tuned eigenfrequency very well, though resonance occurs for a different frequency. However, the model shows a promising result, as can be seen in figure 4.4. It can be seen that if a pair of TMDs - tuned to the new eigenfrequencies of the model of two floors with a damper on the highest floor - is included on the highest floor, the then created resonating frequencies are "pushed" outside of the range of the earthquake's frequency. However, it is not clear whether this is constructially possible for buildings of five and ten floors, as many TMDs have to be included to damp both all five resonating frequencies within the range of the earthquake's frequency, and damp the new eigenfrequencies that come with those.



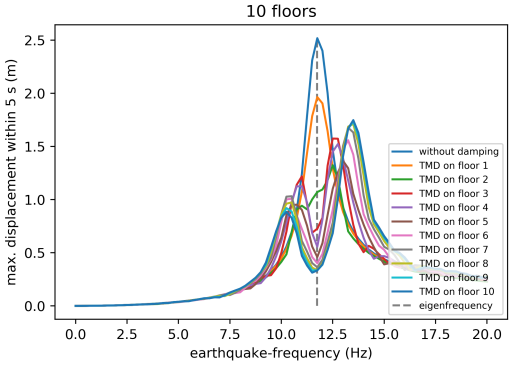
(a) The maximum displacement of a building of two floors, with a single TMD tuned to the first eigenfrequency.



(b) The maximum displacement of a building of five floors, with a single TMD tuned to the first eigenfrequency.

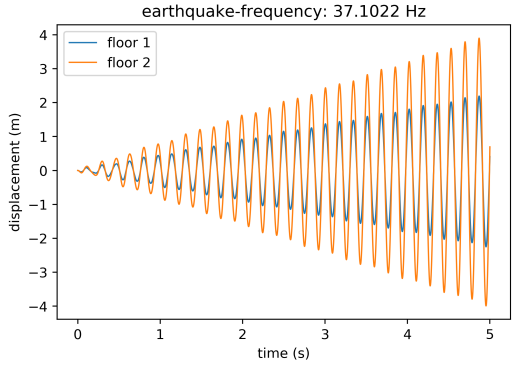


(c) The maximum displacement of a building of ten floors, with a single TMD tuned to the first eigenfrequency.

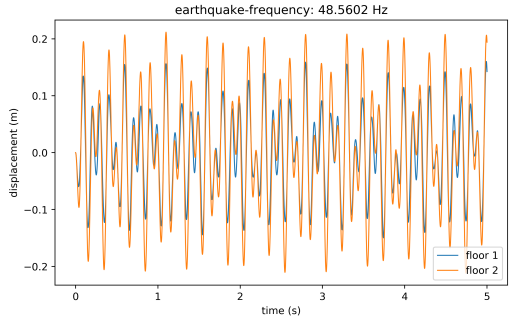


(d) The maximum displacement of a building of ten floors, with a single TMD tuned to the first eigenfrequency, zoomed in on the eigenfrequency which is being damped.

Figure 4.1: The maximum displacements of buildings of two, five and ten floors within five seconds, when a single TMD - tuned to the lowest eigenfrequency - is placed in the building. The position of the TMD within the building is varied.

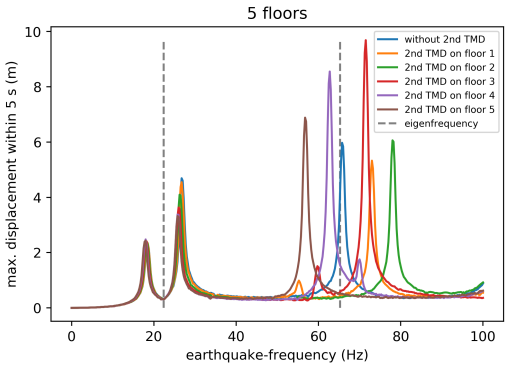


(a) For a new eigenfrequency.

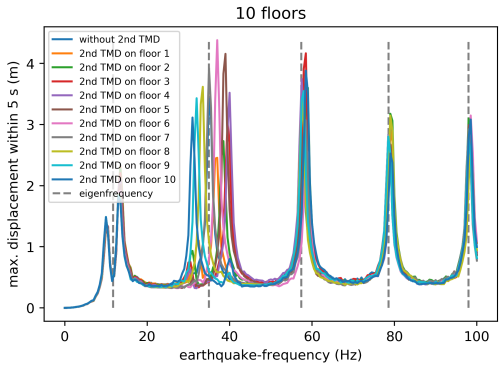


(b) For the old eigenfrequency.

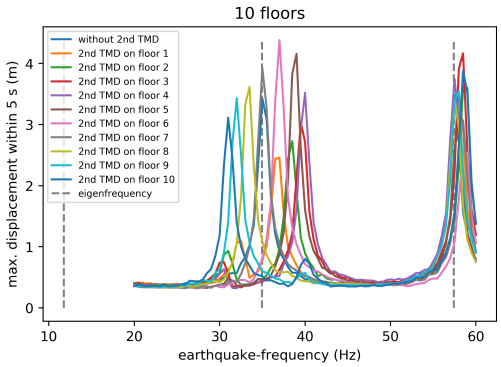
Figure 4.2: The oscillations for a building of two floors including a single TMD on the highest floor, for an eigenfrequency (which is now damped) and a new resonating frequency due to the TMD.



(a) The maximum displacement of a building of five floors.



(b) The maximum displacement of a building of ten floors.



(c) The maximum displacement of a building of ten floors, with a single TMD tuned to the first eigenfrequency, zoomed in on the eigenfrequency which is being damped.

Figure 4.3: The maximum discplacements within five seconds of buildings of five and ten floors, where a TMD (tuned to the loswest eigenfrequency) is added to the highest floor, and a second TMD (tuned to the second-highest floor) is varied over the building's floors.

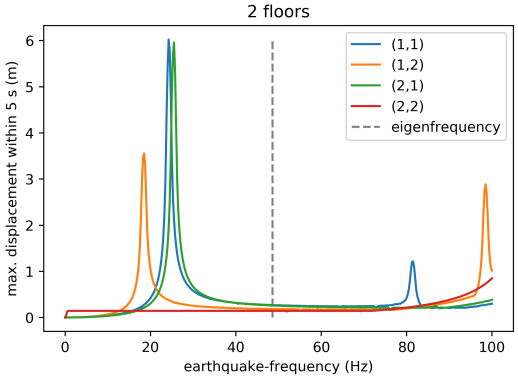


Figure 4.4: The maximum discplacements within five seconds of a building of two floors, in which a TMD is implemented on the highest floor, and two TMDs are added to damp the new eigenfrequencies due to the first TMD. Their positions are varied. The floors of the two extra TMDs are given by (floor TMD 1, floor TMD 2) in the legend, where TMD 1 damps the lowest new eigenfrequency, and TMD 2 the higst new eigenfrequency (see figure 4.1a).

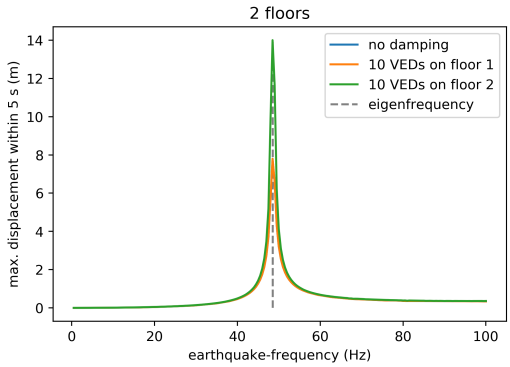
4.2.2. Results of adding VEDs

Figure 4.5 displays the effects of adding a set of ten VEDs to buildings of two, five and ten floors. It can be seen that in each case, the VEDs damp the oscillations for resonance, but significantly. As can be seen in figure 4.6b, the reonance-patterns are damped, but still the displacement grows quite large. Also, for non-

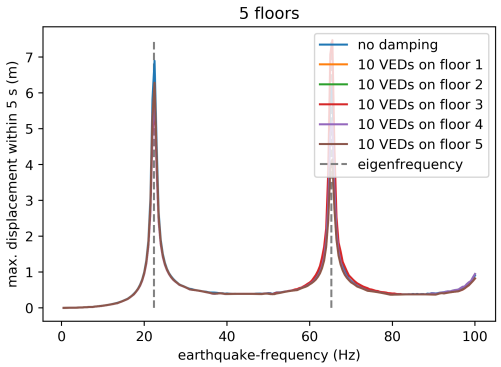
resonating frequencies, the effect of the dampers can be seen in figure 4.6a: the oscillations will grow smaller as time increases (as predicted in subsection 3.3.3).

From figure 4.5, it appears that for most locations, the VEDs have (almost) the same effect in buildings of five and ten floor. Therefore, one of the optimal positions for VEDs for both the buildings of five and ten floors is the highest floor, which will be the location for the first set of VEDs when adding a second set. Also, it can be seen that the optimal position for the VEDs in the building of two floors is at the first floor. The results of adding a second set of ten VEDs to each building is shown in figure 4.7. The damping effect of the VEDs for resonating frequencies is increased, though the oscillations are still large.

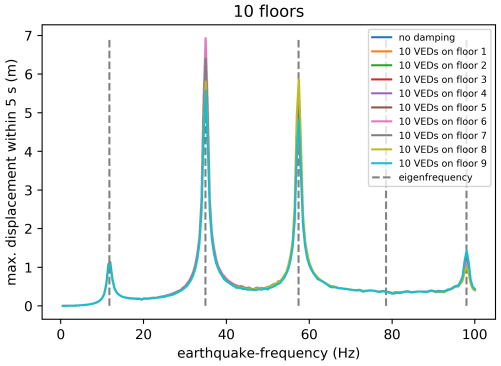
It is woth noting that it might be useful to include both VEDs and TMDs in a building. The TMDs can be used to damp the eigenfrequencies of the building, and since the new resonating frequencies cause smaller (unbounded) oscillations than without a TMD, these frequencies can be well damped by VEDs.



(a) The maximum displacement of a building of two floors, with a single set of VEDs.

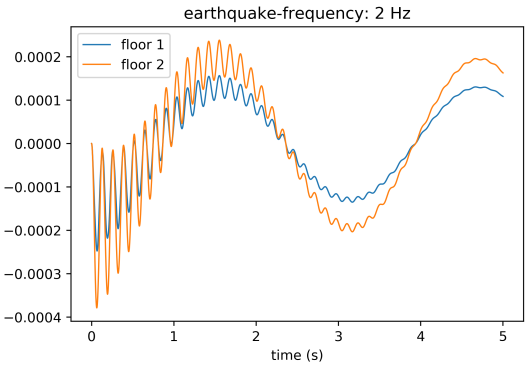


(b) The maximum displacement of a building of five floors, with a single set of VEDs.

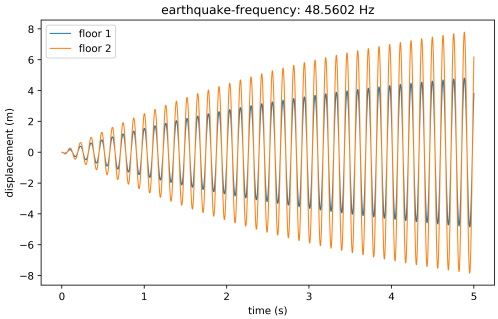


(c) The maximum displacement of a building of ten floors, with a single set of VEDs.

Figure 4.5: The maximum displacements of buildings of two, five and ten floors within five seconds, when a set of ten VEDs is placed in the building. The position of the VED within the building is varied.

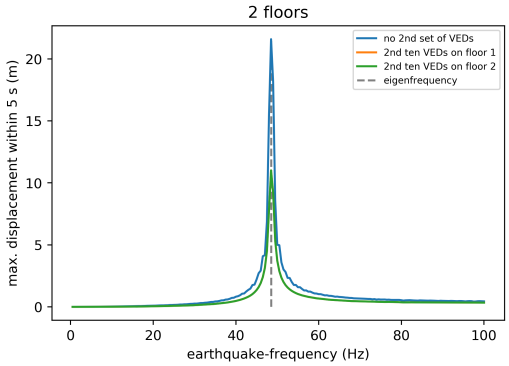


(a) For the non-resonating frequency

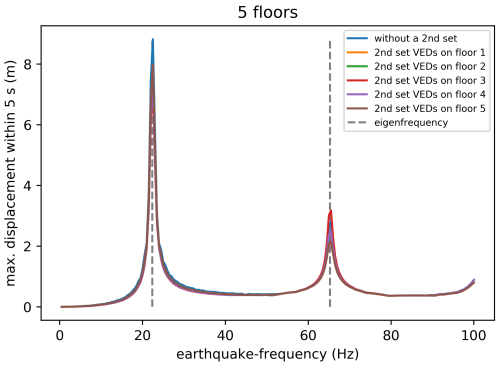


(b) For the resonating frequency.

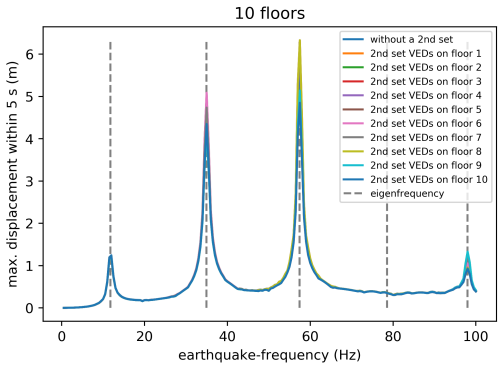
Figure 4.6: The oscillations of a two-floor building damped by a set of ten VEDs, for a resonating and a non-resonating frequency.



(a) The maximum displacement of a building of two floors with two sets of VEDs.



(b) The maximum displacement of a building of five floors with two sets of VEDs.



(c) The maximum displacement of a building of ten floors, with two sets of VEDs.

Figure 4.7: The maximum displacements of buildings of two, five and ten floors within five seconds, when a set of VEDs is placed in the building (at its moptimal position), and a second set of ten VEDs is added. The position of the new set of VEDs is varied.

4.2.3. Comparing the Results

From the results, it can be concluded that the application of TMD is very effective for damping resonating frequencies. However, a downside is that new eigenfrequencies occur for which resonance finds place. Even though these new eigenfrequencies seem to be damped by adding more TMDs in the building of two floors, many more TMDs need to be added to the buildings of five and ten floors to cause the same result, and even then it is not clear whether no resonance will occur. Furthermore, the TMDs do not damp the oscillations for

other frequencies.

The VEDs are more consistent in their damping, though for resonating frequencies, the damping effect is still small and oscillations remain large. By adding more VEDs, though, the damping effect increases. Furthermore, the VEDs also damp oscillations from frequencies without resonance.

Since the VEDs are more consistent in their damping, and do not cause resonance to occur for other frequencies, they are considered to be the most effective dampers according to this model. It should be noted, however, that either many VEDs should be included in the model, or mthe viscoelastic damping coefficient should be increased, to make sure damping oscillations resonating frequencies are also damped enough.

4.3. Limitations of the Model

As has been seen in the behaviour of the model without damping (subsection 3.1.2), the oscillations of a building may grow unboundedly, even including dampers (see subsection 4.2.1), which will undoubtedly cause irreparable damage and collapsement of buildings. However, according to measurements of oscillations during earthquakes, even when resonance occurs, the displacements of buildings will not grow unboundedly [15]. This might have several causes, and one of those might be that there is a frictional force acting on the building, which is not taken into account in the model of tha last two chapters. Therefore, to further examine the effectivity of the TMDs and VEDs on the oscillations of a building, the next chapter will introduce an expanded model in which a frictional force (air resistance) is included.

5

The Model including Air Resistance

In this chapter, the previous model is expanded by including air resistance. First, the new model is derived, and the parameters values that are added will be given. Then, the behaviour of the model will be exposed, and a comparison is made with the previous model.

5.1. Derivation of the Model

The movement of any body (except for when it is located in vacuum) is opposed by a friction force, which can be due to air, water, or any other kind of medium that is occupying the space surrounding the body. A building has to endure such friction forces, whenever it is moving (for example, during an earthquake), due to friction caused by air. Therefore, **the new model that will be introduced and applied in this chapter, will contain air resistance as an external force**, in addition to the earthuqake's force and the linear restoring force (see subsection 3).

Including the air resistance in the expression relating the external forces working on a floor to the movement of that floor (see (3.1)), yields:

$$\frac{d}{dt}P(x_i(t)) = f_{\text{Linear Restoring}}(t) + f_{\text{Earthquake}}(t) + f_{\text{Air Resistance}}(t). \tag{5.1}$$

A schematic view of the three external forces acting upon the building is given in figure 5.1.

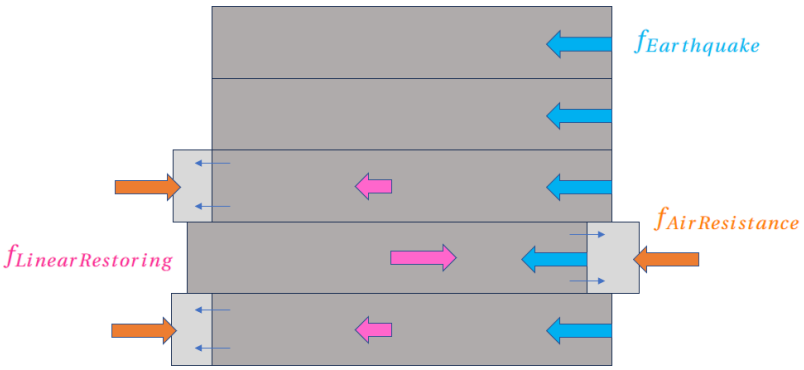


Figure 5.1: A scheme displaying the external forces acting on the building in the new model.

The expression that describes the force on a moving body due to air resistance is the following [25]:

$$f_{\text{Air Resistance}}(t) = -\frac{1}{2} C_d A \rho v^2 \cdot \text{sign}(v), \tag{5.2}$$

where C_d denotes the drag coefficient of the body, A (m^2) its cross-sectional area, v (m/s) its velocity, and ρ the density of air. Furthermore, the function $\text{sign}(v)$ is defined as follows:

$$\text{sign}(v) = \begin{cases} 1 & \text{if} \quad v > 0, \\ 0 & \text{if} \quad v = 0, \\ -1 & \text{if} \quad v < 0. \end{cases} \tag{5.3}$$

This function ascertains that the air resistance acts opposite to the movement of a body. By combining the formula for the force due to air resistance with expression (5.1), the following model (which is an expansion of (3.3)) is derived:

$$\begin{cases} mx_1'' = -kx_1 - k(x_1 - x_2) & - \quad F_0\omega^2 m\cos(\omega t) & -\frac{1}{2} C_d A \rho (x_1')^2 \cdot \text{sign}(x_1'), \\ mx_2'' = -k(x_2 - x_1) - k(x_2 - x_3) & - \quad F_0\omega^2 m\cos(\omega t) & -\frac{1}{2} C_d A \rho (x_2')^2 \cdot \text{sign}(x_2'), \\ mx_3'' = -k(x_3 - x_2) - k(x_3 - x_4) & - \quad F_0\omega^2 m\cos(\omega t) & -\frac{1}{2} C_d A \rho (x_3')^2 \cdot \text{sign}(x_3'), \\ \quad ... & & \\ mx_{n-1}'' = -k(x_{n-1} - x_{n-2}) - k(x_{n-1} - x_n) & - \quad F_0\omega^2 m\cos(\omega t) & -\frac{1}{2} C_d A \rho (x_{n-1}')^2 \cdot \text{sign}(x_{n-1}'), \\ mx_n'' = -k(x_n - x_{n-1}) & - \quad F_0\omega^2 m\cos(\omega t) & -\frac{1}{2} C_d A \rho (x_n')^2 \cdot \text{sign}(x_n'). \end{cases} \tag{5.4}$$

Furthermore, it is still assumed that the initial conditions of the point masses of the building are $x_i(0) = x_i'(0) = 0$ for $i = 1, 2, ..., n$.

5.2. Values of the Parameters in the Model Including Air Friction

By expanding the model from the previous two chapters, new parameters have been introduced. These parameters are: the drag coefficient C_d of a floor, the cross-sectional area A (m^2) of a floor, and the density ρ of air. Their values will be taken constant in the new model, and are listed below.

- The value of the drag coefficient C_d is a constant that is dependent on the shape of the moving body it belongs to. For a floor within a building, the shape is considered to be a rectangle, as shown in figure 5.2. The value of the drag coefficient C_d that belongs to this shape is 2, which is the value that will be used within this model.
- The crossectional area of a floor in the building that is modelled is based on the dimensions of the building from which the values of the stiffness k and the mass m were taken [7]. Each floor in that building has an area of 900 m^2 in the shape of a square, with a height of 3 m (see figure 5.2). Therefore, the cross-sectional area of a floor within the building is $\sqrt{900} \cdot 3 = 90$ m^2.
- The density ρ of air is based on [18], and is equal to 1.293 kg/m^3 for dry, cold air.

All other parameters, that have already been introduced in subsection 3.0.2, will retain the same value, or will remain varying within the same interval.

5.3. The Behaviour of the Model

The new model (5.4) is no longer a system of ordinary differential equations, due to the new terms $-\frac{1}{2} C_d A \rho (x_i')^2 \cdot \text{sign}(x_i')$. SInce the behaviour of this model is expected to be similar to the previous model, but However, its

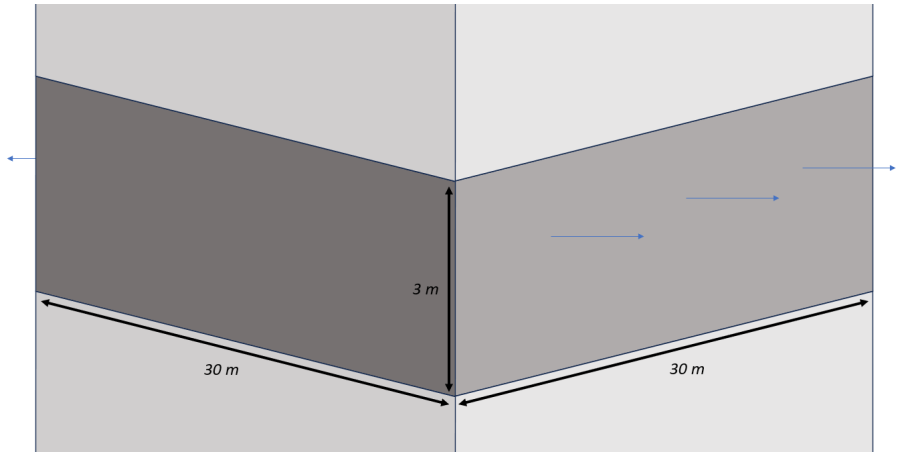


Figure 5.2: The dimensions of the building. The shape of the building is a rectangle on its side, which corresponds to a drag coefficient of $C_d = 2$.

behaviour can still be determined by applying a numerical integration method. Before doing so, the velocity of the point masses, $x_i'(t)$, has to be approximated. This can be done by using the backward difference:

$$x_i'(t) \approx \frac{x_i(t) - x_i(t-\Delta t)}{\Delta t}, \tag{5.5}$$

for $i = 1, 2, ..., n$. No forward or central difference can be used, as when implementing a numerical integration method, the next time-step $x_i(t+\Delta t)$ is not known yet. The truncation error of the backward difference is $\mathcal{O}(\Delta t)$.

The numerical method that will be used to determine the behaviour of the new model is, as for the previous model, the RK4-method (3.39). To be able to apply the method, it is again necessary to rewrite the system in terms of first-order differential equations. First, the following vector-matrix notation is used for the model:

$$\mathbf{x}''(t) = K_n\mathbf{x} - F_0\omega^2 m\cos(\omega t)\mathbf{d}_n - \frac{1}{2}C_d A\rho(x_1')^2 \cdot \text{sign}(x_1')\mathbf{d}_n, \tag{5.6}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1(t) \\ x_2(t) \\ ... \\ x_n(t) \end{bmatrix}, \quad K_n = \begin{bmatrix} -\frac{2k}{m} & \frac{k}{m} & 0 & 0 & ... & 0 & 0 & 0 \\ \frac{k}{m} & -\frac{2k}{m} & \frac{k}{m} & 0 & ... & 0 & 0 & 0 \\ 0 & \frac{k}{m} & -\frac{2k}{m} & \frac{k}{m} & ... & 0 & 0 & 0 \\ ... & ... & ... & ... & ... & ... & ... & ... \\ 0 & 0 & 0 & 0 & ... & \frac{k}{m} & -\frac{2k}{m} & \frac{k}{m} \\ 0 & 0 & 0 & 0 & ... & 0 & \frac{k}{m} & -\frac{k}{m} \end{bmatrix}, \quad \mathbf{d}_n = \begin{bmatrix} 1 \\ 1 \\ ... \\ 1 \end{bmatrix} \tag{5.7}$$

By introducing the new variables $z_j(t)$ (as in section 3.1.2), such that $z_i(t) \equiv x_i'(t)$ and $z_{i+n}(t) \equiv x_i(t)$ for $i = 1, 2, ..., n$, the following system of first-order differential equations is derived for the new model (5.4):

$$\mathbf{z}'(t) = \tilde{K_n}\mathbf{z} - F_0\omega^2\cos(\omega t)\tilde{\mathbf{d}}_n - \frac{1}{2}C_d A\rho(x_1')^2 \cdot \text{sign}(x_1')\tilde{\mathbf{d}}_n., \tag{5.8}$$

where

$$\mathbf{z} = \begin{bmatrix} z_1(t) \\ z_2(t) \\ ... \\ z_{2n}(t) \end{bmatrix}, \quad \tilde{K_n} = \begin{bmatrix} [0]_{n\times n} & K_n \\ I_n & [0]_{n\times n} \end{bmatrix}, \quad \mathbf{d}_n = \begin{bmatrix} \tilde{\mathbf{d}}_n \\ \mathbf{0}_n \end{bmatrix} \tag{5.9}$$

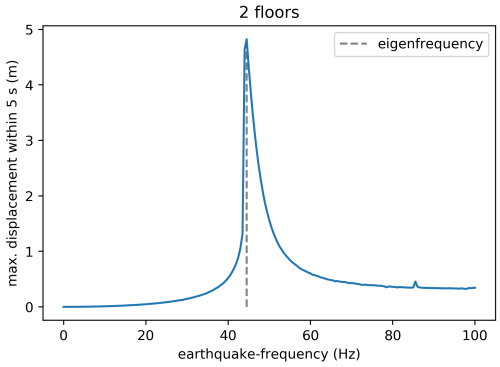
Incorporating the backward difference into this notation, the following expression for the new function $\mathbf{f} = [f_1, \quad f_2, \quad ..., f_{2n}]^T$ is derived:

$$f_i(t, \mathbf{z}(t)) \equiv [\tilde{K_n}\mathbf{z}(t)]_i - F_0\omega^2\cos(\omega t) - \frac{1}{2}C_d A\rho\left(\frac{z_{i+n}(t) - z_{i+n}(t-\Delta t)}{\Delta t}\right)^2 \cdot \text{sign}\left(\frac{z_{i+n}(t) - z_{i+n}(t-\Delta t)}{\Delta t}\right), \tag{5.10}$$

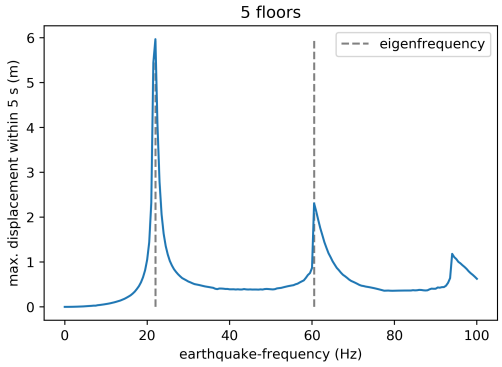
for $i = 1, 2..., n$, and

$$f_i(t, \mathbf{z}(t)) \equiv [\tilde{K}_n \mathbf{z}(t)]_i, \tag{5.11}$$

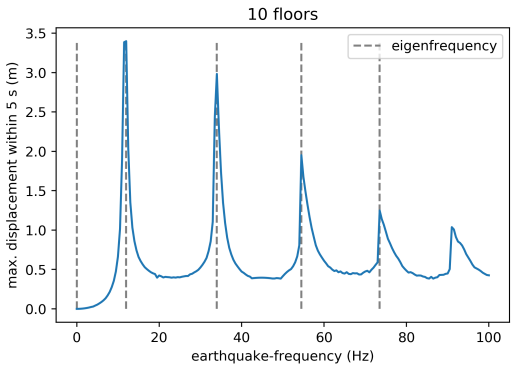
for $i = n+1, n+2, ..., 2n$. Using the updated definition and implementing the Runge-Kutta fourth-order (RK4) method with a time-step of $\Delta t = 0.00025$ s, the obtained results are presented in Figure 5.3. These figures depict the behavior of the model under varying earthquake frequencies, now accounting for air resistance. It is clear that for the two- and five-floor buildings, the primary difference lies in the reduced maximum displacement within five seconds during resonance, as compared to the previous model. Additionally, the eigenfrequencies have shifted to the left. Figure 5.4 demonstrates that resonance occurs at these new eigenfrequencies, although it is not as unbounded as observed in the previous model.



(a) A building of two floors.

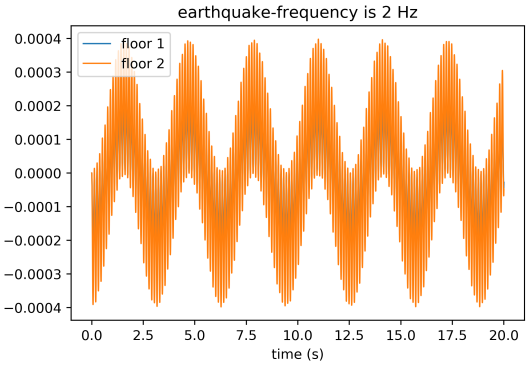


(b) A building of five floors.

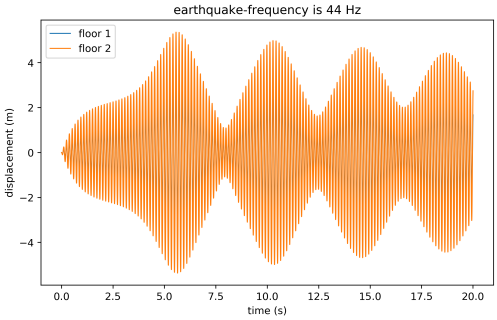


(c) A building of ten floors.

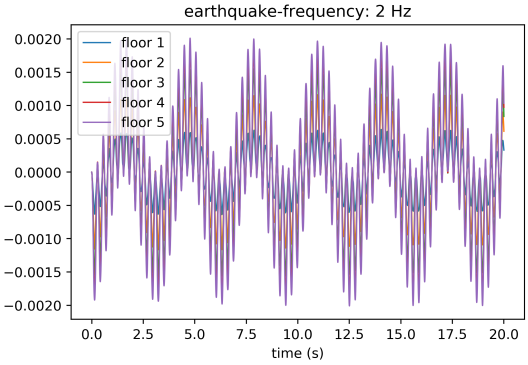
Figure 5.3: The maximum displacements of buildings of two, five and ten floors, for earthquake-frequencies varying from 0.001 Hz to 100 Hz.



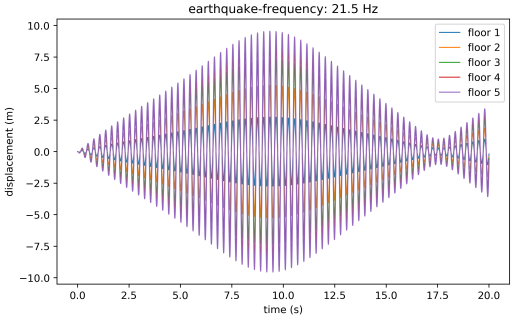
(a) An oscillation-pattern of a non-resonating frequency for a building of two floors.



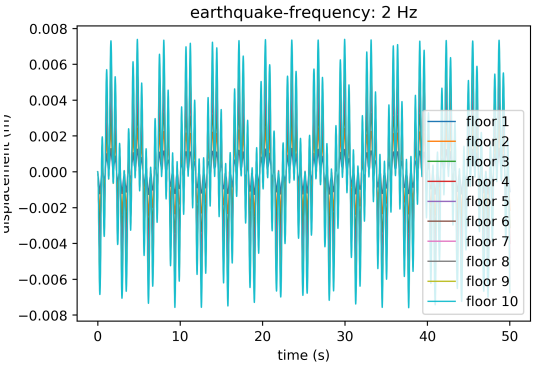
(b) An oscillation-pattern of a resonating frequency for a building of two floors.



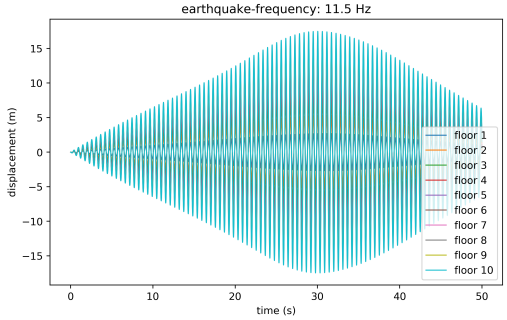
(c) An oscillation-pattern of a non-resonating frequency for a building of five floors.



(d) An oscillation-pattern of a resonating frequency for a building of five floors.



(e) An oscillation-pattern of a non-resonating frequency for a building of ten floors.



(f) An oscillation-pattern of a non-resonating frequency for a building of ten floors.

Figure 5.4: Oscillation-patterns for resonating and non-resonating earthquake-frequencies, for buildings of two, five and ten floors. The oscillations show a resonating pattern, but will eventually be damped due to the air resistance.

6

Dampers on the Model Including Air Resistance

In this chapter, the methods of adding dampers to the new model will be described, followed by the results that are created based on these methods. Based on that, a final conclusion will be drawn concerning which damper is most effective in damping the oscillation-effects of an earthquake.

6.1. Methods of Adding Dampers to the Model

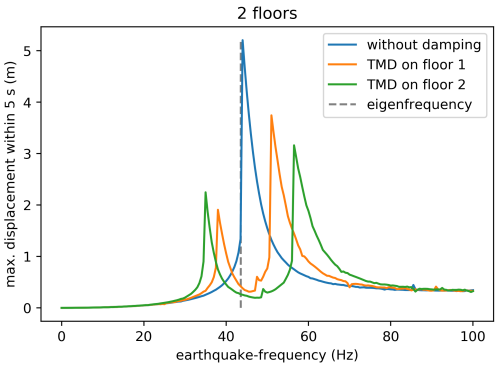
The methods of adding dampers to the new model are listed below. All results will be created by using the numerical method RK4, as described in sebsection 5.3, with time-step $\Delta t = 0{,}00025$ s. The Python-codes that are used to dcreate the results can be found in Appendix B.2.

1. For buildings of two, five and ten floors, a single TMD, tuned to the eigenfrequency with the largest displacement, will be added to the model to determine whether the model gives more realistic oscillations, and whether the dampers perform differently. The position of the TMD will be varied.
2. For the same three buildings, a set of ten VEDs will be added to the new model, and their positions will be varied.

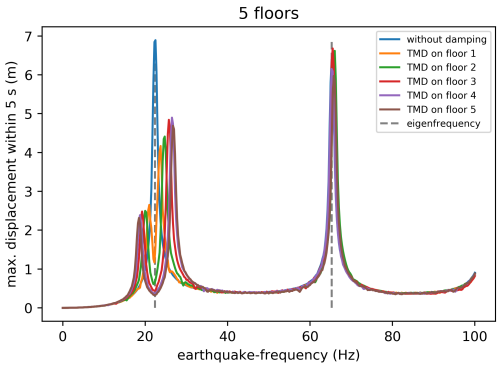
6.2. Results of Dampers on the Model

6.2.1. Results of Adding a TMD to the Model Including Air Resistance

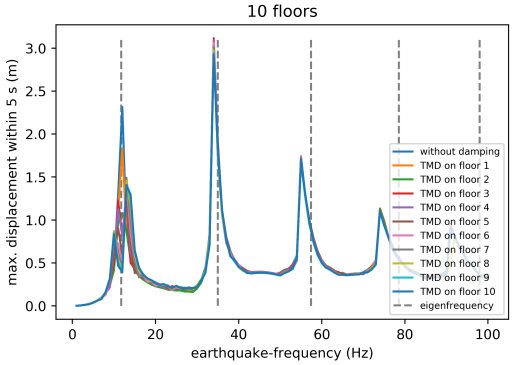
In figure 6.1, the effect of the addition of a single TMD on the models of two, five and ten floors is shown. It can be seen that the effects are very similar to the effects on the previous model.Again, as is shown in figure 6.2 for the building of two floors, no resonance occurs for the eigenfrequencies, but for a new eigenfrequency due to the TMD, resonance does occur. Even though this resonating pattern is damped, its oscillations can still grow quite large.



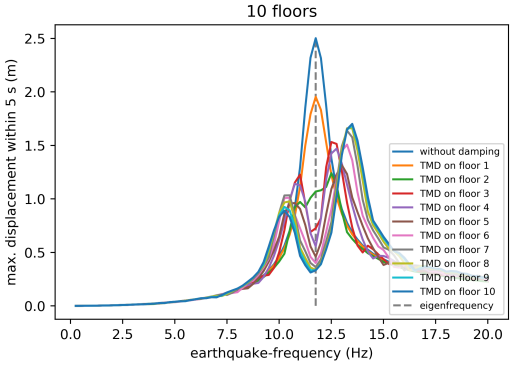
(a) A building of two floors, with a single TMD tuned to the first eigenfrequency.



(b) A building of five floors, with a single TMD tuned to the first eigenfrequency.

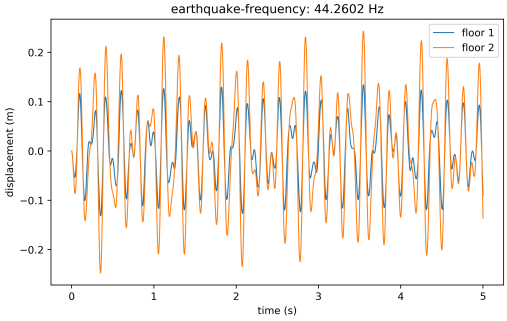


(c) A building of ten floors, with a single TMD tuned to the first eigenfrequency.

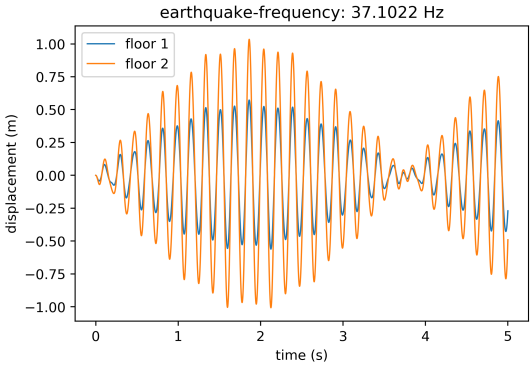


(d) A building of ten floors, with a single TMD tuned to the first eigenfrequency, zoomed on the first eigenfrequency.

Figure 6.1: The maximum displacements of buildings of two, five and ten floors within five seconds, when a single TMD - tuned to the lowest eigenfrequency - is placed in the building. The position of the TMD within the building is varied.



(a) For the old, damped eigenfrequency.



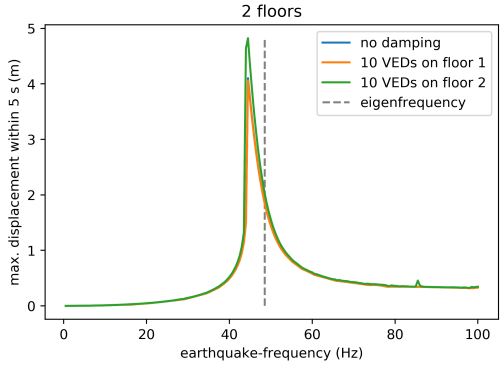
(b) The new, resonating eigenfrequency.

Figure 6.2: The oscillations of a two-floor building, damped by a single TMD on the highest floor, for an eigenfrequency (which is damped), and a new eigenfrequency due to a TMD.

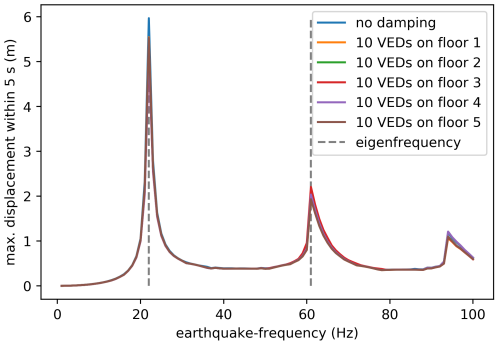
6.2.2. Results of Adding VEDs to the Model Including Air Resistance

In figure 6.3, the effects of adding a single set of VEDs is shown. Like for the TMDs, the result is very similar to the results from the previous model. Both the resonating patterns and the non-resonating patterns, as shown in figure 6.4, are damped, but for the resonating frequencies, the damping is still small. As this result is very similar to the results from the previous model, it is concluded that adding more VEDs will improve the

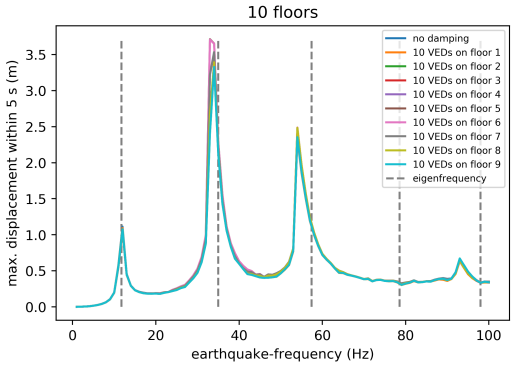
damping in such a way that even for resonating frequencies, the oscillations can be kept small.



(a) A building of two floors, with a single TMD tuned to the first eigenfrequency.

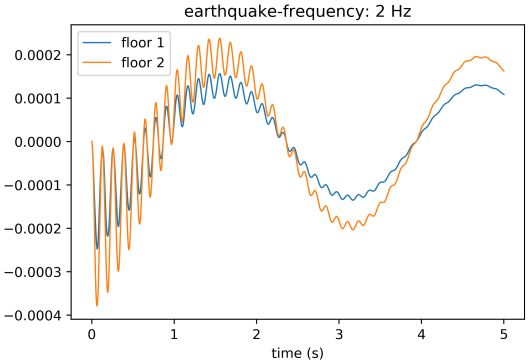


(b) A building of five floors, with a single TMD tuned to the first eigenfrequency.

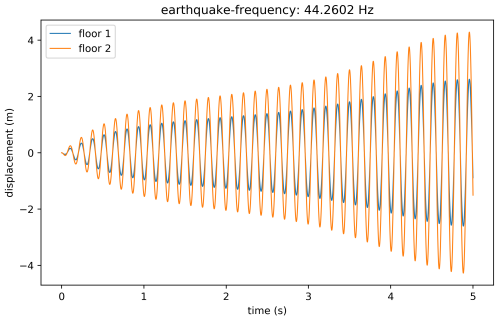


(c) A building of ten floors, with a single TMD tuned to the first eigenfrequency.

Figure 6.3: The maximum displacements of buildings of two, five and ten floors within five seconds, when a single TMD - tuned to the lowest eigenfrequency - is placed in the building. The position of the TMD within the building is varied.



(a) For the non-resonating frequency.



(b) For the resonating frequency.

Figure 6.4: The oscillations of a two-floor building, damped by a set of VEDs on the lowest floor, for a resonating and a non-resonating frequency.

6.2.3. Comparing the Results of Adding TMDs and Adding VEDs to the Model Including Air Resistance

Since the results from this model are very similar to the results from the previous model, the same conclusion can be drawn. Even though the oscillations due to resonance caused by a TMDs will not grow unboundedly due to the air resistance, the TMDs are not able to damp the new resonating oscillations. Adding more VEDs to a building, however, does improve the damping. Therefore, the VEDs are still considered as more effective dampers for earthquake-oscillations of a building.

7

Conclusion

In this research, Tuned Mass Dampers (TMDs) and Viscoelastic Dampers (VEDs) were added to buildings of two, five and ten floors, and their damping performance was assessed, based on their ability to reduce oscillations for both resonating and non-resonating frequencies. The dampers have been added to two models, one without and one including air resistance.

First, in the model without air resistance, a single TMD was added to the buildings, tuned to the lowest eigenfrequency. Its position in the building was varied over all floors. The TMD effectively reduced the oscillations for earthquake-frequencies close to the eigenfrequency, and no resonance occured for those frquencies. However, adding the TMD caused resonance to occur for two new eigenfrequencies. Furthermore, for other frequencies, the TMD has no effect in damping at all.

For all three buildings, it was concluded that the damping of the oscillations for the lowest eigenfrequency was most effective for a TMD on the highest floor. Therefore, for the buildings of five and ten floors, a TMD was placed on the upper floor, and a second TMD was added, tuned to the second-lowest eigenfrequency of the building. A similar effect was observed as for the first TMD: resonance was damped for the eigenfrequency, but new resonating eigenfrequencies appeared. Furthermore, the dampers still had no effect on other frequencies.

In pursuing to damp the new peaks in maximum displacement that occur due to resonance caused by a TMD, two extra TMDs were added to the building of two floors, when a TMD was already on the highest floor. These new TMDs were tuned to the new eigenfrequencies caused by the first TMD. It appeared that, if the two TMDs were btoth implemented on the highest floor, the new resonating frequencies were shifted out of the range of the earthquake-frequency. However, if such a shifting of resonating frequencies is even possible for larger buildings, a large amount of TMDs need to be implemented, which is unpractical, since these dampers take up a significant amount of space.

After this, the VEDs were implemented. For all three buildings, a set of ten VEDs was added, and the position was varied over all floors. Though the most optimal position of the VEDs did not become clear, the VEDs proved more consistent in reducing the oscillations than the TMDs. For non-resonating frequencies, the VEDs damped the oscillations effectively, though for resonating frequencies, the (once) unbounded growth was damped, yet still large. Therefore, a second set of ten VEDs was included in each building, which proved useful, since the oscillations both during and without resonance were reduced further. An important notice is that during resonance, the oscillations do not grow unbounded when VEDs are included. However, the oscillations still remained vast, and more VEDs need to be added to further reduce oscillations, or the viscoelastic damping coefficient needs to be increased (if possible).

Since the VED is more consistent then the TMD in damping oscillation in the model without air resistance, and reduces the resonating oscillations, it is considered to be the most effective damper of the two. It should

be noted, however, that many VEDs should be included in the building so that resonating oscillations are reduced to vibrations that are small enough to prevent damage to the building, or the viscoelastic coefficient should be increased (if possible).

When adding TMDs and VEDs to the model including air resistance, similar results occur as for the previous model, though eigenfrequencies are shifted, and the resonating oscillations do not grow unbounded. Since resonance is no longer unbounded in this model, a damper has to reduce the oscillations only for a finite amount of time. However, the new resonating frequencies due to a TMD on a building still cause resonance to occur, with ocscillations that are vast and will cause damage to a building. These oscillations can, however, still be reduced by VEDs, and therefore it is again concluded that the VED is a more effectie damper of earthquake-induced oscillations.

It should, furthermore, be noted that it might be useful to include both VEDs and TMDs in a building. The TMDs can be used to damp the eigenfrequencies of the building, and since the new resonating frequencies cause smaller oscillations than without a TMD, these frequencies can be well damped by VEDs.

8

Discussion

The choices for models, methods and parameter values and -ranges within this research were based on the objective of reaching results that would give information on which damper, the TMD or the VED, is most effective in damping oscillations. Further variation of parameters and expansion of the model can be done to either support or counter the results from this research. Below, a number of possibilities is given for expansion and modification of the choices made within this research, while it will also be argued whether this would influence the results of the research.

8.1. Model Simplification

In both models within this research, all floors were assumed to have equal dimensions and equal properties (stiffness k, mass m). However, such simplification implies that the results might only be valid for buildings that satisfy these properties. Though further research upon this subject can and should be performed, it is expected that the differentiation between floors solely influences the value of the eigenfrequencies of the buildings. Therefore, the only effect that this is expected to have on the performance of the TMD and the VED, is that the TMD should be tuned to a different frequency, and a VED might have less effect on resonance due to higher eigenfrequencies (for which the viscoelastic damping coefficient c_V is smaller). However, both these topics have been discussed within the research, since the eigenfrequencies of buildings of different height differed from one another, and the TMD's and VED's effect on those different buildings has been modelled.

Further model simplification has been done by considering only the oscillations of the building in one direction, due to the earthquake acting only in one dimension. However, in reality, the direction of oscilations is unpredictable, and due to construction-properties of a building, factors such as the stiffness, cross-sectional area and the drag coefficient of a floor might differ from when the direction is purely aimed one way. Furthermore, the model does not take into account any kind of rotational and bending movements of the building. These movements, however, might influence the performance of the dampers. Therefore, further research is required for considering the impact of those factors upon the effectivity of the dampers.

Moreover, the models assume that the force of the eartquake is acting upon every floor. This is questionable, as this force comes from the ground, and is therefore only directly connected to the lowest floor. Therefore, the model can be made more realistic, for example by applying the earthquake‘s force only on the lowest floor, or by considering the ground as a function with a displacement, which moves according to the movements of the earthquake.

8.2. Methods

The method of modelling an earthquake has been done by considering an earthquake of maximum amplitude ($F_0 = 0,1$ m), and by considering every possibility of large oscillation due to resonance through variation over the earthquake's frequency. Based on all worst-case scenarios, the dampers have been assessed. However, when constructing a new building, not all worst-case scenarios might apply, and the risk of having earthquakes with certain amplitudes or frequencies might be significantly low, so that these scenarios can be left out when considering the effect of a certain damper on the new building. In that case, a thorough research should be done on the risks of the occurence of certain earthquakes, and on the risks that are acceptable to be taken. Nevertheles, since the research in this report is based on worst-case scenarios, the results might be useful in determining these risks.

Furthermore, the methods that have been used in this research for adding the dampers to the floors are limited. Even by considering the effects of dampers on various floors, there are many more combinations in which dampers could have been added to the models. Yet, the possibilities of adding dampers to a model are also limited. For example, as TMDs require a lot of space, it might not be practically possible to have one on multiple floors in a building (specially in a building of two floors), and a smilar statement applies to VEDs. Therefore, the methods used within this research already give an extensive view on the effectivity of the placement of dampers on various locations within a building.

Moreover, the method of assessing the effects of dampers on the building is by considering the displacement of the floors. However, not only the displacement is of interest for possible damage to the building. Properties of movements such as velocity and accelleration of the floors can both influence the damage the building takes during an earthquake. Further research upon modelling these factors while adding dampers to the building can be done.

Lastly, the assessment of effectivity of dampers has been done by only considering their effect on the oscillations of a floor. However, it had not been taken into consideration what the financial costs of adding these types of dampers are.

A

Determining an Upper Bound for the Time-Step for Stability using RK4 on the Model of Two Floors

In this appendix, the upper bound for the time-step for RK4 on the simple model (without air resistance) with two floors is determined.

To determine for which time-steps Δt the RK4-method converges, we analyse its amplification factor [23]:

$$Q(\lambda_i \Delta t) = 1 + \lambda_i \Delta t + \frac{(\lambda_i \Delta t)^2}{2} + \frac{(\lambda_i \Delta t)^3}{6} + \frac{(\lambda_i \Delta t)^4}{24}. \tag{A.1}$$

For the method to be absolutely stable, it is required that $|Q(\lambda_i \Delta t)| < 1$ for all eigenvalues λ_i of the matrix K. It is known that these eigenvalues are fully imaginary. Therefore, in this subsection, we will only consider λ_i to be fully imaginary, i.e. $\lambda_i = I_i i$. We write $\lambda_i = I_i i$, such that for absolute stability:

$$\begin{aligned} |Q(\lambda_i \Delta t)| &= |Q(i \cdot I_i \Delta t)| \\ &= |1 - \frac{(I_i \Delta t)^2}{2} + \frac{(I_i \Delta t)^4}{24} + i\left(I_i \Delta t - \frac{(I_i \Delta t)^3}{6}\right) \\ &= \sqrt{\left(1 - \frac{(I_i \Delta t)^2}{2} + \frac{(I_i \Delta t)^4}{24}\right)^2 + \left(I_i \Delta t - \frac{(I_i \Delta t)^3}{6}\right)^2} \\ &= \frac{1}{24}\sqrt{(I_i \Delta t)^8 - 8(I_i \Delta t)^6 + 576} < 1. \end{aligned} \tag{A.2}$$

To derive the bound on Δt explicitly, consider the function $g(s) = s^8 - 8s^6$. Note that, to satisfy (A.2), we require $g(I_i \Delta t) + 576 < 24^2 = 576$, or equivalently, $g(I_i \Delta t) < 0$. It can be deduced immediately that $s = 0$ is a root of $g(s)$ with mulitplicity 6. Furthermore, $g(s) = 0$ if $s^2 - 8 = 0$, implying $s = \pm 2\sqrt{2}$ are two other distinct roots of $g(s)$. To find where $g(s) < 0$, consider the derivative of g: $g'(s) = 8s^7 - 48a^5$. For the roots of $g(s)$, the derivatives of $g(s)$ are:

$$g'(-2\sqrt{2}) = -2048\sqrt{2} < 0, \tag{A.3}$$

$$g'(0) = 0, \tag{A.4}$$

$$g'(2\sqrt{2}) = 2048\sqrt{2} > 0, \tag{A.5}$$

which, in combination with $g(-2\sqrt{2}) = g(0) = g(2\sqrt{2}) = 0$, can only be satisfied if $s = 0$ is a local maximum, and $g(s) < 0$ if $-2\sqrt{2} < s < 0$ or $0 < s < 2\sqrt{2}$, i.e. $0 < |s| < 2\sqrt{2}$ (as can also be seen in Figure A.1). Hence, (A.2)

is satisfied if and only if $0 < |I_i \Delta t| < 2\sqrt{2}$, which implies that the bound for absolute stability of the numerical integration method RK4 is as follows[1]:

$$\Delta t < \frac{2\sqrt{2}}{|\lambda_i|}. \tag{A.6}$$

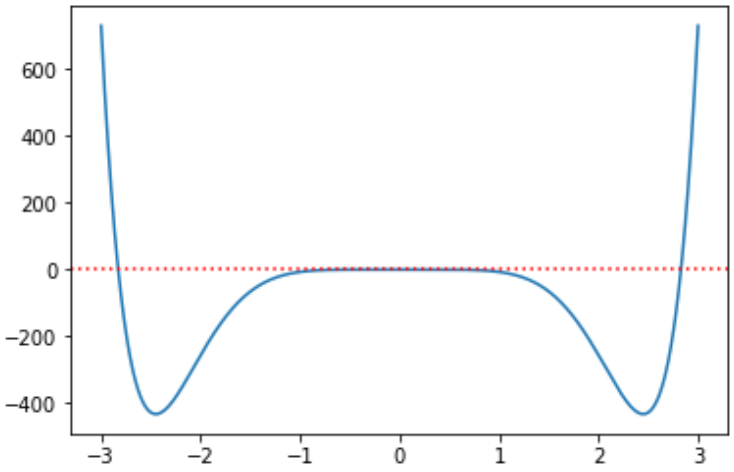


Figure A.1: This figure displays the graph of $g(s)$, where the value of g is denoted on the vertical axis, for every value of s on the horizontal axis. Note that equations (A.3), (A.4) and (A.5), together with the knowledge that $g(-2\sqrt{2}) = g(0) = g(2\sqrt{2}) = 0$, imply that $g(s) < 0$ if $0 < |s| < 2\sqrt{2}$.

[1] Note that $\Delta t > 0$ is always satisfied, and that $\lambda = 0$ is not an eigenvalue of K, so that $|I_i| = |\lambda_i| > 0$ for all $i \in \{1,2,3,...,10\}$.

B

Python Codes

B.1. Codes for the Model without Air Resistance

B.1.1. The Analytical Solution for Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Fri Jun 16 14:39:24 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
```

```
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

def F(b,a):
    return F0*b/(a**2-3*a*b+b**2)

a = k/m
omegalist = [2,10,20,80,fr(1)-0.00001,fr(2)-0.00001]
for omega in omegalist:
    b = omega**2

    c1 = (a*(7+3*math.sqrt(5))+b*(-3-math.sqrt(5)))/(2*math.sqrt(5))
    c2 = (a*(-7+3*math.sqrt(5))+b*(3-math.sqrt(5)))/(2*math.sqrt(5))
    w1 = np.zeros((2,1))
    w2 = np.zeros((2,1))
    w1[0,0] = (-1+math.sqrt(5))/2
    w1[1,0] = 1
    w2[0,0] = (-1-math.sqrt(5))/2
    w2[1,0] = 1
    Av = np.zeros((2,1))
    Av[0,0] = -2*a+b
    Av[1,0] = -3*a+b
    ef1 = math.sqrt((a/2)*(3-math.sqrt(5)))
    ef2 = math.sqrt((a/2)*(3+math.sqrt(5)))
    def x(t):
        term1 = c1*w1*math.cos(ef1*t)
        term2 = c2*w2*math.cos(ef2*t)
        term3 = Av*math.cos(omega*t)
        return F(b,a)*(term1+term2+term3)

    xlist = [[],[]]
    xrellist = [[],[]]
    for i in range(0,2):
        for t in tlist:
            xival = x(t)[i,0]
            if i == 0:
                xibelow = 0
            else:
                xibelow = x(t)[i-1,0]
            xrellist[i].append(xival-xibelow)
            xlist[i].append(xival)
    for i in range(0,2):
        plt.plot(tlist,xrellist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('relative displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 2:
        plt.savefig('M1F2 om=2 rel.png',dpi=400)
    if omega == 10:
```

```
        plt.savefig('M1F2 om=10 rel.png',dpi=400)
    if omega == 20:
        plt.savefig('M1F2 om=20 rel.png',dpi=400)
    elif omega == 80:
        plt.savefig('M1F2 om=80 rel.png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F2 om=48.5602 rel.png',dpi=400)
    else:
        plt.savefig('M1F2 om=127.1324 rel.png',dpi=400)
    plt.figure(figsize=(8,4.8))
    for i in range(0,2):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 2:
        plt.savefig('M1F2 om=2.png',dpi=400)
    if omega == 10:
        plt.savefig('M1F2 om=10.png',dpi=400)
    if omega == 20:
        plt.savefig('M1F2 om=20.png',dpi=400)
    elif omega == 80:
        plt.savefig('M1F2 om=80.png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F2 om=48.5602.png',dpi=400)
    else:
        plt.savefig('M1F2 om=127.1324.png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.1.2. The Errors in the Numerical Approximation for Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Fri Jun 16 14:39:24 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)
```

```
m = 9E5
k = 5556222222
F0 = 0.1
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

def F(b,a):
    return F0*b/(a**2-3*a*b+b**2)

a = k/m
omegalist = [2,10,20,80,fr(1)-0.00001,fr(2)-0.00001]
for omega in omegalist:
    b = omega**2

    c1 = (a*(7+3*math.sqrt(5))+b*(-3-math.sqrt(5)))/(2*math.sqrt(5))
    c2 = (a*(-7+3*math.sqrt(5))+b*(3-math.sqrt(5)))/(2*math.sqrt(5))
    w1 = np.zeros((2,1))
    w2 = np.zeros((2,1))
    w1[0,0] = (-1+math.sqrt(5))/2
    w1[1,0] = 1
    w2[0,0] = (-1-math.sqrt(5))/2
    w2[1,0] = 1
    Av = np.zeros((2,1))
    Av[0,0] = -2*a+b
    Av[1,0] = -3*a+b
    ef1 = math.sqrt((a/2)*(3-math.sqrt(5)))
    ef2 = math.sqrt((a/2)*(3+math.sqrt(5)))
    def x(t):
        term1 = c1*w1*math.cos(ef1*t)
        term2 = c2*w2*math.cos(ef2*t)
        term3 = Av*math.cos(omega*t)
        return F(b,a)*(term1+term2+term3)

    xlist = [[],[]]
    xrellist = [[],[]]
    for i in range(0,2):
        for t in tlist:
            xival = x(t)[i,0]
            if i == 0:
                xibelow = 0
            else:
                xibelow = x(t)[i-1,0]
            xrellist[i].append(xival-xibelow)
```

```
            xlist[i].append(xival)
    for i in range(0,2):
        plt.plot(tlist,xrellist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('relative displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 2:
        plt.savefig('M1F2 om=2 rel.png',dpi=400)
    if omega == 10:
        plt.savefig('M1F2 om=10 rel.png',dpi=400)
    if omega == 20:
        plt.savefig('M1F2 om=20 rel.png',dpi=400)
    elif omega == 80:
        plt.savefig('M1F2 om=80 rel.png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F2 om=48.5602 rel.png',dpi=400)
    else:
        plt.savefig('M1F2 om=127.1324 rel.png',dpi=400)
    plt.figure(figsize=(8,4.8))
    for i in range(0,2):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 2:
        plt.savefig('M1F2 om=2.png',dpi=400)
    if omega == 10:
        plt.savefig('M1F2 om=10.png',dpi=400)
    if omega == 20:
        plt.savefig('M1F2 om=20.png',dpi=400)
    elif omega == 80:
        plt.savefig('M1F2 om=80.png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F2 om=48.5602.png',dpi=400)
    else:
        plt.savefig('M1F2 om=127.1324.png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.1.3. The Maximum Displacement: Two floors

```
    # -*- coding: utf-8 -*-
"""
Created on Fri Jun 16 14:39:24 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))
```

```
tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

def F(b,a):
    return F0*b/(a**2-3*a*b+b**2)

a = k/m
omegalist = [2,10,20,80,fr(1)-0.00001,fr(2)-0.00001]
for omega in omegalist:
    b = omega**2

    c1 = (a*(7+3*math.sqrt(5))+b*(-3-math.sqrt(5)))/(2*math.sqrt(5))
    c2 = (a*(-7+3*math.sqrt(5))+b*(3-math.sqrt(5)))/(2*math.sqrt(5))
    w1 = np.zeros((2,1))
    w2 = np.zeros((2,1))
    w1[0,0] = (-1+math.sqrt(5))/2
    w1[1,0] = 1
    w2[0,0] = (-1-math.sqrt(5))/2
    w2[1,0] = 1
    Av = np.zeros((2,1))
    Av[0,0] = -2*a+b
    Av[1,0] = -3*a+b
    ef1 = math.sqrt((a/2)*(3-math.sqrt(5)))
    ef2 = math.sqrt((a/2)*(3+math.sqrt(5)))
    def x(t):
        term1 = c1*w1*math.cos(ef1*t)
        term2 = c2*w2*math.cos(ef2*t)
        term3 = Av*math.cos(omega*t)
        return F(b,a)*(term1+term2+term3)
```

```
    xlist = [[],[]]
    xrellist = [[],[]]
    for i in range(0,2):
        for t in tlist:
            xival = x(t)[i,0]
            if i == 0:
                xibelow = 0
            else:
                xibelow = x(t)[i-1,0]
            xrellist[i].append(xival-xibelow)
            xlist[i].append(xival)
    for i in range(0,2):
        plt.plot(tlist,xrellist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('relative displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 2:
        plt.savefig('M1F2 om=2 rel.png',dpi=400)
    if omega == 10:
        plt.savefig('M1F2 om=10 rel.png',dpi=400)
    if omega == 20:
        plt.savefig('M1F2 om=20 rel.png',dpi=400)
    elif omega == 80:
        plt.savefig('M1F2 om=80 rel.png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F2 om=48.5602 rel.png',dpi=400)
    else:
        plt.savefig('M1F2 om=127.1324 rel.png',dpi=400)
    plt.figure(figsize=(8,4.8))
    for i in range(0,2):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 2:
        plt.savefig('M1F2 om=2.png',dpi=400)
    if omega == 10:
        plt.savefig('M1F2 om=10.png',dpi=400)
    if omega == 20:
        plt.savefig('M1F2 om=20.png',dpi=400)
    elif omega == 80:
        plt.savefig('M1F2 om=80.png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F2 om=48.5602.png',dpi=400)
    else:
        plt.savefig('M1F2 om=127.1324.png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.1.4. The Maximum Displacement: Five floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sat Jul  1 16:51:28 2023
```

```
@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_5 = np.zeros((10,10))
for i in range(0,4):
    Kt_5[i,i+5] = -2*k/m
    Kt_5[i,i+6] = k/m
    Kt_5[i+1,i+5] = k/m
Kt_5[4,9] = -k/m
for i in range(0,5):
    Kt_5[i+5,i] = 1
valsKt_5 = np.linalg.eigvals(Kt_5)
Ilist = []
for i in range(0,5):
    Ilist.append(valsKt_5[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

dt_5 = np.zeros((10,1))
for i in range(0,5):
    dt_5[i,0] = 1
def f(t,y, omega):
    term1 = np.dot(Kt_5,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_5
    return term1 + term2

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(0,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = []
xrelabsmaxlist = []
```

```
for omega in omegalist:
    # Runge-Kutta 4
    y0 = np.zeros((10,1))
    wlist = [y0]
    for i in range(0,steps+1):
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,omega)
        k2 = dt*f(tn+dt/2,wn+k1/2,omega)
        k3 = dt*f(tn+dt/2,wn+k2/2,omega)
        k4 = dt*f(tn+dt,wn+k3,omega)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)
    xiabslist = [[],[],[],[],[]]
    xireallist = [[],[],[],[],[]]
    xirelabslist = [[],[],[],[],[]]
    xiabsmaxlist = []
    xirelabsmaxlist = []
    for i in range(0,5):
        for j in range(0,steps+1):
            xival = wlist[j][i+5,0]
            if i == 0:
                xibelow = 0
            else:
                xibelow = xireallist[i-1][j]
            xiabslist[i].append(abs(xival))
            xireallist[i].append(xival)
            xirelabslist[i].append(abs(xival-xibelow))
        xiabsmaxlist.append(max(xiabslist[i]))
        xirelabsmaxlist.append(max(xirelabslist[i]))

    xabsmaxlist.append(max(xiabsmaxlist))
    xrelabsmaxlist.append(max(xirelabsmaxlist))
    print('finished with omega =', omega)

topreal = max(xabsmaxlist)
toprel = max(xrelabsmaxlist)
plt.plot(omegalist,xabsmaxlist)
for n in range(1,6):
    if fr(n) <= omegamax:
        if n == 1:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey',label='eigenfrequency')
        else:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey')
plt.title('5 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F5 varying omega real.png',dpi=400)
plt.figure(figsize=(8,4.8))

plt.plot(omegalist,xrelabsmaxlist)
for n in range(1,6):
    if fr(n) <= omegamax:
        if n == 1:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey',label='eigenfrequency')
```

```
        else:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey')
plt.title('5 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. relative displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F5 varying omega relative.png',dpi=400)
plt.figure(figsize=(8,4.8))

plt.plot(omegalist,xabsmaxlist,label='real displacement')
plt.plot(omegalist,xrelabsmaxlist,label='relative displacement')
for n in range(1,6):
    if fr(n) <= omegamax:
        if n == 1:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey',label='eigenfreque
        else:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey')
plt.title('5 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. (relative) displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F5 varying omega real and relative.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.5. The Maximum Displacement: Ten floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sat Jul  1 17:39:03 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_10 = np.zeros((20,20))
for i in range(0,9):
    Kt_10[i,i+10] = -2*k/m
```

```
    Kt_10[i,i+11] = k/m
    Kt_10[i+1,i+10] = k/m
Kt_10[9,19] = -k/m
for i in range(0,10):
    Kt_10[i+10,i] = 1
valsKt_10 = np.linalg.eigvals(Kt_10)
Ilist = []
for i in range(0,10):
    Ilist.append(valsKt_10[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

dt_10 = np.zeros((20,1))
for i in range(0,10):
    dt_10[i,0] = 1
def f(t,y, omega):
    term1 = np.dot(Kt_10,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_10
    return term1 + term2

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(0,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = []
xrelabsmaxlist = []
for omega in omegalist:
    # Runge-Kutta 4
    y0 = np.zeros((20,1))
    wlist = [y0]
    for i in range(0,steps+1):
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,omega)
        k2 = dt*f(tn+dt/2,wn+k1/2,omega)
        k3 = dt*f(tn+dt/2,wn+k2/2,omega)
        k4 = dt*f(tn+dt,wn+k3,omega)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)
    xiabslist = [[],[],[],[],[],[],[],[],[],[]]
    xireallist = [[],[],[],[],[],[],[],[],[],[]]
    xirelabslist = [[],[],[],[],[],[],[],[],[],[]]
    xiabsmaxlist = []
    xirelabsmaxlist = []
    for i in range(0,10):
        for j in range(0,steps+1):
            xival = wlist[j][i+10,0]
            if i == 0:
                xibelow = 0
            else:
                xibelow = xireallist[i-1][j]
            xiabslist[i].append(abs(xival))
```

```
            xireallist[i].append(xival)
            xirelabslist[i].append(abs(xival-xibelow))
        xiabsmaxlist.append(max(xiabslist[i]))
        xirelabsmaxlist.append(max(xirelabslist[i]))

    xabsmaxlist.append(max(xiabsmaxlist))
    xrelabsmaxlist.append(max(xirelabsmaxlist))
    print('finished with omega =', omega)

topreal = max(xabsmaxlist)
toprel = max(xrelabsmaxlist)
plt.plot(omegalist,xabsmaxlist)
for n in range(1,11):
    if fr(n) <= omegamax:
        if n == 1:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey',label='eigenfreque
        else:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey')
plt.title('10 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F10 varying omega real.png',dpi=400)
plt.figure(figsize=(8,4.8))

plt.plot(omegalist,xrelabsmaxlist)
for n in range(1,11):
    if fr(n) <= omegamax:
        if n == 1:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey',label='eigenfreque
        else:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey')
plt.title('10 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. relative displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F10 varying omega relative.png',dpi=400)
plt.figure(figsize=(8,4.8))

plt.plot(omegalist,xabsmaxlist,label='real displacement')
plt.plot(omegalist,xrelabsmaxlist,label='relative displacement')
for n in range(1,11):
    if fr(n) <= omegamax:
        if n == 1:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey',label='eigenfreque
        else:
            plt.vlines(x=fr(n),ymin=0,ymax=topreal,linestyle='--',color='grey')
plt.title('10 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. (relative) displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F10 varying omega real and relative.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.6. Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sat Jun 17 13:09:44 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

dt_2 = np.zeros((4,1))
for i in range(0,2):
    dt_2[i,0] = 1
def f(t,y, omega):
    term1 = np.dot(Kt_2,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2
    return term1 + term2

omegalist = [5,fr(1)-0.00001]
for omega in omegalist:
    # Runge-Kutta 4
    y0 = np.zeros((4,1))
```

```
    wlist = [y0]
    for i in range(0,steps+1):
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,omega)
        k2 = dt*f(tn+dt/2,wn+k1/2,omega)
        k3 = dt*f(tn+dt/2,wn+k2/2,omega)
        k4 = dt*f(tn+dt,wn+k3,omega)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)

    xlist = [[],[]]
    xrellist = [[],[]]
    for i in range(0,2):
        for j in range(0,steps+1):
            xival = wlist[j][i+2,0]
            if i == 0:
                xibelow = 0
            else:
                xibelow = xlist[i-1][j]
            xrellist[i].append(xival-xibelow)
            xlist[i].append(xival)
    for i in range(0,2):
        plt.plot(tlist,xrellist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('relative displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 5:
        plt.savefig('M1F2 om=5 rel (num).png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F2 om=48.5602 rel (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))
    for i in range(0,2):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 5:
        plt.savefig('M1F2 om=5 (num).png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F2 om=48.5602 (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))

######################## NUMERICAL STABILITY OF RUNGE KUTTA 4
def Q(l,dt):
    I = l*dt
    terms = I**8-8*I**6+576
    return math.sqrt(terms)/24
for l in valsKt_2.imag:
    if abs(Q(l,dt))>1:
        print("The timestep is too large.")
        break
    else:
        if l == valsKt_2.imag[-1]:
```

```
            print("The timestep is small enough.")
print("Your timestep is", dt)
dtmaxlist = []
for l in valsKt_2.imag:
    dtmax = abs(2*math.sqrt(2)/l)
    dtmaxlist.append(dtmax)
print("The maximum time-step for which RK4 converges is", round(min(dtmaxlist),10))
```

B.1.7. Five Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sat Jul  1 23:09:48 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_5 = np.zeros((10,10))
for i in range(0,4):
    Kt_5[i,i+5] = -2*k/m
    Kt_5[i,i+6] = k/m
    Kt_5[i+1,i+5] = k/m
Kt_5[4,9] = -k/m
for i in range(0,5):
    Kt_5[i+5,i] = 1
valsKt_5 = np.linalg.eigvals(Kt_5)
Ilist = []
for i in range(0,5):
    Ilist.append(valsKt_5[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

dt_5 = np.zeros((10,1))
for i in range(0,5):
```

```
    dt_5[i,0] = 1
def f(t,y, omega):
    term1 = np.dot(Kt_5,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_5
    return term1 + term2

omegalist = [5,fr(1)-0.00001]
for omega in omegalist:
    # Runge-Kutta 4
    y0 = np.zeros((10,1))
    wlist = [y0]
    for i in range(0,steps+1):
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,omega)
        k2 = dt*f(tn+dt/2,wn+k1/2,omega)
        k3 = dt*f(tn+dt/2,wn+k2/2,omega)
        k4 = dt*f(tn+dt,wn+k3,omega)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)

    xlist = [[],[],[],[],[]]
    xrellist = [[],[],[],[],[]]
    for i in range(0,5):
        for j in range(0,steps+1):
            xival = wlist[j][i+5,0]
            if i == 0:
                xibelow = 0
            else:
                xibelow = xlist[i-1][j]
            xrellist[i].append(xival-xibelow)
            xlist[i].append(xival)
    for i in range(0,5):
        plt.plot(tlist,xrellist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('relative displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 5:
        plt.savefig('M1F5 om=5 rel (num).png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F5 om=22.364 rel (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))
    for i in range(0,5):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 5:
        plt.savefig('M1F5 om=5 (num).png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F5 om=22.364 (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))

######################## NUMERICAL STABILITY OF RUNGE KUTTA 4
```

```
def Q(l,dt):
    I = l*dt
    terms = I**8-8*I**6+576
    return math.sqrt(terms)/24
for l in valsKt_5.imag:
    if abs(Q(l,dt))>1:
        print("The timestep is too large.")
        break
    else:
        if l == valsKt_5.imag[-1]:
            print("The timestep is small enough.")
print("Your timestep is", dt)
dtmaxlist = []
for l in valsKt_5.imag:
    dtmax = abs(2*math.sqrt(2)/l)
    dtmaxlist.append(dtmax)
print("The maximum time-step for which RK4 converges is", round(min(dtmaxlist),10))
```

B.1.8. Ten Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sat Jul  1 23:14:49 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_10 = np.zeros((20,20))
for i in range(0,9):
    Kt_10[i,i+10] = -2*k/m
    Kt_10[i,i+11] = k/m
    Kt_10[i+1,i+10] = k/m
Kt_10[9,19] = -k/m
for i in range(0,10):
    Kt_10[i+10,i] = 1
valsKt_10 = np.linalg.eigvals(Kt_10)
```

```
Ilist = []
for i in range(0,10):
    Ilist.append(valsKt_10[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

dt_10 = np.zeros((20,1))
for i in range(0,10):
    dt_10[i,0] = 1
def f(t,y, omega):
    term1 = np.dot(Kt_10,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_10
    return term1 + term2

omegalist = [5,fr(1)-0.00001]
for omega in omegalist:
    # Runge-Kutta 4
    y0 = np.zeros((20,1))
    wlist = [y0]
    for i in range(0,steps+1):
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,omega)
        k2 = dt*f(tn+dt/2,wn+k1/2,omega)
        k3 = dt*f(tn+dt/2,wn+k2/2,omega)
        k4 = dt*f(tn+dt,wn+k3,omega)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)

    xlist = [[],[],[],[],[]]
    xrellist = [[],[],[],[],[]]
    for i in range(0,5):
        for j in range(0,steps+1):
            xival = wlist[j][i+5,0]
            if i == 0:
                xibelow = 0
            else:
                xibelow = xlist[i-1][j]
            xrellist[i].append(xival-xibelow)
            xlist[i].append(xival)
    for i in range(0,5):
        plt.plot(tlist,xrellist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('relative displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 5:
        plt.savefig('M1F10 om=5 rel (num).png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F10 om=11.7434 rel (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))
    for i in range(0,5):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
```

```
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 5:
        plt.savefig('M1F10 om=5 (num).png',dpi=400)
    elif omega == fr(1)-0.00001:
        plt.savefig('M1F10 om=11.7434 (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))

######################## NUMERICAL STABILITY OF RUNGE KUTTA 4
def Q(l,dt):
    I = l*dt
    terms = I**8-8*I**6+576
    return math.sqrt(terms)/24
for l in valsKt_10.imag:
    if abs(Q(l,dt))>1:
        print("The timestep is too large.")
        break
    else:
        if l == valsKt_10.imag[-1]:
            print("The timestep is small enough.")
print("Your timestep is", dt)
dtmaxlist = []
for l in valsKt_10.imag:
    dtmax = abs(2*math.sqrt(2)/l)
    dtmaxlist.append(dtmax)
print("The maximum time-step for which RK4 converges is", round(min(dtmaxlist),10))
```

B.1.9. The TMD on the Model of One Floor

-*- coding: utf-8 -*- """ Created on Wed Jun 21 22:06:05 2023

@author: bradl """

import os import math import numpy as np import matplotlib.pyplot as plt dir$_name$ = "$C:/Users/bradl/BEP-Figures/$"$plt.rcParams$["$savefig.directory$"] = $os.chdir(os.path.dirname(dir_name)$)

tbegin = 0 tend = 5 steps = 4000*(tend-tbegin) dt = (tend-tbegin)/steps tlist = [tbegin] for i in range(1,steps+1): t = tbegin+dt*i tlist.append(t)

m = 9E5 k = 5556222222 F0 = 0.1 mT = m*0.4 kT = math.sqrt(k/m)**2*mT

a = k/m c = kT/mT

def x(t,omega): factor = (F0*omega**2)/(omega**2-k/m) tpart = math.cos(omega*t)-math.cos(math.sqrt(k/m)*t) return factor*tpart def xTMD(t,Av,a1,a2,v1,v2,om1,om2,omega): term1 = factor*a1*v1*math.cos(om1*t) term2 = factor*a2*v2*math.cos(om2*t) term3 = factor*Av*math.cos(omega*t) return term1 + term2 + term3

omegalist = [2,math.sqrt(k/m)-0.00001,math.sqrt(k/(2*m)*(3-math.sqrt(5)))-0.00001] for omega in omegalist: b = omega**2

xlist = [] for t in tlist: xval = x(t,omega) xlist.append(xval)

om1 = math.sqrt((2*a+c+math.sqrt(4*a**2+c**2))/2) om2 = math.sqrt((2*a+c-math.sqrt(4*a**2+c**2))/2) factor = F0*omega**2/(a*c-2*a*b-b*c+b**2) a1 = c*(b-om2**2)/(om1**2-om2**2) a2 = c*(-b+om1**2)/(om1**2-om2**2) v1 = np.zeros((2,1)) v1[0,0] = (c-om1**2)/c v1[1,0] = 1 v2 = np.zeros((2,1)) v2[0,0] = (c-om2**2)/c

v2[1,0] = 1 Av = np.zeros((2,1)) Av[0,0] = -c+b Av[1,0] = -c

xTMDlist = [] for t in tlist: xTMDval = xTMD(t,Av,a1,a2,v1,v2,om1,om2,omega)[0,0] xTMDlist.append(xTMDval)

plt.plot(tlist, xlist, label='without TMD') plt.plot(tlist, xTMDlist, label='with TMD') plt.xlabel('time in seconds') plt.ylabel('displacement in meters') plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz') plt.legend() if omega == 2: plt.savefig('M1F1 TMD om=2.png',dpi=400) elif omega == math.sqrt(k/m)-0.00001: plt.savefig('M1F1 TMD om=78.5721.png',dpi=400) elif omega == math.sqrt(k/(2*m)*(3-math.sqrt(5)))-0.00001: plt.savefig('M1F1 TMD om=48.5602.png',dpi=400) plt.figure(figsize=(8,4.8))

B.1.10. The VED on the Model of One Floor

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 02:04:04 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 20
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1

FREQS = [5,10,15,20,25,30,35]
NLOADS = [5,10,15,25,40,50]
DATA = np.array([[11.6,8.7,7.3,6.5,5.9,5.5,5.2],
                 [24.0,18.1,15.3,13.6,12.4,11.5,10.8],
                 [41.0,30.8,26.1,23.2,21.1,19.6,18.4],
                 [100.0,75.3,63.7,56.6,51.7,47.9,45.0],
                 [217.3,163.5,138.5,123.1,112.3,104.2,97.8],
                 [438.3,329.9,279.4,248.3,226.6,210.2,197.4]])

def c(omega,index):
    c1 = DATA[index,0]
    o1 = FREQS[0]
    c2 = DATA[index,-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
```

```
    return -4*1000*a/(omega**b)

omegalist = [2,math.sqrt(k/m)-0.00001]
for omega in omegalist:
    for indexp1 in range(0,7):
        if indexp1 == 0:
            cV = 0
        else:
            index = indexp1-1
            Nload = NLOADS[index]
            cV = c(omega,index)
        F = F0*omega**2/((cV**2*omega**2/m**2) + (k/m - omega**2)**2)
        A = -F*(k/m - omega**2)
        B = F*(cV*omega/m)
        D = k/m - cV**2/(4*m**2)
        D1 = -D
        if cV**2 == 4*k*m:
            def xVED(t):
                term1 = -A*math.exp(cV/(2*m)*t)
                term2 = (cV/(2*m)*A - omega*B)*t*math.exp(cV/(2*m)*t)
                term3 = A*math.cos(omega*t)
                term4 = B*math.sin(omega*t)
                return term1 + term2 + term3 + term4
        elif cV**2 < 4*k*m:
            def xVED(t):
                factor = math.exp(cV/(2*m)*t)
                term1 = -A*math.cos(math.sqrt(D)*t)
                term2 = (cV/(2*m)*A - omega*B)*math.sin(math.sqrt(D)*t)/math.sqrt(D)
                term3 = A*math.cos(omega*t)
                term4 = B*math.sin(omega*t)
                return factor*(term1 + term2) + term3 + term4
        else:
            def xVED(t):
                a1 = ((cV/(2*m) - math.sqrt(D1))*A - omega*B)/(2*math.sqrt(D1))
                a2 = -((cV/(2*m) + math.sqrt(D1))*A - omega*B)/(2*math.sqrt(D1))
                term1 = a1*math.exp((cV/(2*m)+math.sqrt(D1))*t)
                term2 = a2*math.exp((cV/(2*m)-math.sqrt(D1))*t)
                term3 = A*math.cos(omega*t)
                term4 = B*math.sin(omega*t)
                return term1 + term2 + term3 + term4
        xVEDlist = []
        for t in tlist:
            xVEDlist.append(xVED(t))
        if indexp1 == 0:
            plt.plot(tlist,xVEDlist,label='no VED')
        else:
            plt.plot(tlist,xVEDlist,label='Nominal load is '+str(Nload)+' kN',linewidth=1)
    plt.legend()
    plt.xlabel('time (s)in seconds')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    if omega == 2:
        plt.savefig('M1F1 VED om=2.png',dpi=400)
    elif omega == math.sqrt(k/m)-0.00001:
        plt.savefig('M1F1 VED om=78.5721.png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.1.11. Adding 1 TMD to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Tue Jul  4 01:09:59 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT = fr(1)**2*mT
K_2T = np.zeros((3,3))
K_2T[0,0] = -2*k/m
K_2T[0,1] = k/m
K_2T[1,0] = k/m
K_2T[1,1] = -k/m
def Kt_2T(n):
    T2_n = np.zeros((3,3))
```

```
    if n != 0:
        i = n-1
        T2_n[i,i] = -kT/m
        T2_n[i,2] = kT/m
        T2_n[2,i] = kT/mT
        T2_n[2,2] = -kT/mT
    upperright = K_2T + T2_n
    matrix = np.zeros((6,6))
    for j1 in range(0,3):
        matrix[j1+3,j1] = 1
        for j2 in range(0,3):
            matrix[j1,j2+3] = upperright[j1,j2]
    return matrix
neigvals = np.linalg.eigvals(Kt_2T(2))
nIlist = []
nIlist = []
for i in range(0,3):
    nIlist.append(neigvals[2*i].imag)
nIlist.sort()
def nfr(n):
    return nIlist[n-1]
print(nIlist)

dt_2T = np.zeros((6,1))
for i in range(0,2):
    dt_2T[i,0] = 1
def f(t,y, omega):
    term1 = np.dot(Kt_2T(2),y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2T
    return term1 + term2

omegalist = [nfr(1)-0.00001,fr(1)-0.00001]
for omega in omegalist:
    # Runge-Kutta 4
    y0 = np.zeros((6,1))
    wlist = [y0]
    for i in range(0,steps+1):
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,omega)
        k2 = dt*f(tn+dt/2,wn+k1/2,omega)
        k3 = dt*f(tn+dt/2,wn+k2/2,omega)
        k4 = dt*f(tn+dt,wn+k3,omega)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)

    xlist = [[],[]]
    xrellist = [[],[]]
    for i in range(0,2):
        for j in range(0,steps+1):
            xival = wlist[j][i+3,0]
            if i == 0:
                xibelow = 0
            else:
                xibelow = xlist[i-1][j]
            xrellist[i].append(xival-xibelow)
```

```
            xlist[i].append(xival)
    # for i in range(0,2):
    #     plt.plot(tlist,xrellist[i],label='floor ' + str(i+1),linewidth=1)
    # plt.xlabel('time (s)')
    # plt.ylabel('relative displacement (m)')
    # plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    # plt.legend()
    # if omega == omegalist[0]:
    #     plt.savefig('M1F2 om=nfr1 rel (num).png',dpi=400)
    # elif omega == omegalist[1]:
    #     plt.savefig('M1F2 om=nfr1 rel (num).png',dpi=400)
    # plt.figure(figsize=(8,4.8))
    for i in range(0,2):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == omegalist[0]:
        plt.savefig('M1F2 + TMD om=nfr1 (num).png',dpi=400)
    elif omega == omegalist[1]:
        plt.savefig('M1F2 + TMD om=fr1 (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.1.12. Adding 1 VED to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Tue Jul  4 01:26:06 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
A = 36
```

```
rho = 1.293
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

def C_2(omega,n):
    matrix = np.zeros((2,2))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_2V(omega,n):
    matrix = np.zeros((4,4))
    for i in range(0,4):
        for j in range(0,4):
            matrix[i,j] = Kt_2[i,j]
    for i in range(0,2):
        for j in range(0,2):
            matrix[i,j] = C_2(omega,n)[i,j]
    return matrix
dt_2 = np.zeros((4,1))
for i in range(0,2):
    dt_2[i,0] = 1
def f(t,y,ydiff, omega,matrix):
    term1 = np.dot(Kt_2V(omega,1),y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2
    term3 = np.zeros((4,1))
    Favec = np.ones((4,1))
    for i in range(0,2):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
```

```
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist = [2,fr(1)-4.3]#0.00001]
for omega in omegalist:
    matrix = Kt_2V(omega,1)
    # Runge-Kutta 4
    y0 = np.zeros((4,1))
    wlist = [y0]
    for i in range(0,steps+1):
        if i > 0:
            wnold = wlist[-2]
        elif i == 0:
            wnold = y0
        tn = tlist[i]
        wn = wlist[-1]
        wndiff = wn - wnold
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,wndiff,omega,matrix)
        k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega,matrix)
        k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega,matrix)
        k4 = dt*f(tn+dt,wn+k3,wndiff,omega,matrix)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)

    xlist = [[],[]]
    for i in range(0,2):
        for j in range(0,steps+1):
            xival = wlist[j][i+2,0]
            xlist[i].append(xival)
    for i in range(0,2):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == omegalist[0]:
        plt.savefig('M2F2 + VED om=2 (num).png',dpi=400)
    elif omega == omegalist[1]:
        plt.savefig('M2F2 + VED om=fr1 (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.1.13. Maximum Displacement: Adding 1 TMD to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 11:37:43 2023

@author: bradl
"""
```

```
import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT = fr(1)**2*mT
K_2T = np.zeros((3,3))
K_2T[0,0] = -2*k/m
K_2T[0,1] = k/m
K_2T[1,0] = k/m
K_2T[1,1] = -k/m
def Kt_2T(n):
    T2_n = np.zeros((3,3))
    if n != 0:
        i = n-1
        T2_n[i,i] = -kT/m
        T2_n[i,2] = kT/m
        T2_n[2,i] = kT/mT
        T2_n[2,2] = -kT/mT
    upperright = K_2T + T2_n
    matrix = np.zeros((6,6))
    for j1 in range(0,3):
        matrix[j1+3,j1] = 1
        for j2 in range(0,3):
```

```
            matrix[j1,j2+3] = upperright[j1,j2]
    return matrix
neigvals = np.linalg.eigvals(Kt_2T(2))
nIlist = []
nIlist = []
for i in range(0,3):
    nIlist.append(neigvals[2*i].imag)
nIlist.sort()
def nfr(n):
    return nIlist[n-1]
print(nIlist)

dt_2T = np.zeros((6,1))
for i in range(0,2):
    dt_2T[i,0] = 1
def f(t,y, omega,matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2T
    return term1 + term2

omegalist=[]
omegasteps = 400
omegamax = 100
for val in range(0,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[]]
for n in range(0,3):
    matrix = Kt_2T(n)
    for omega in omegalist:
        y0 = np.zeros((6,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[]],[[],[]],[[],[]]]
        xiabsmaxlist = [[],[],[]]
        for i in range(0,2):
            for j in range(0,steps+1):
                xival = wlist[j][i+3,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        print('finished with omega =', omega, ', n =', n)

top = max(max(xabsmaxlist[0]),max(xabsmaxlist[1]),max(xabsmaxlist[2]))
for n in range(0,3):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without damping')
    else:
```

```
        plt.plot(omegalist,xabsmaxlist[n],label='TMD on floor '+ str(n))
for m in range(1,3):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency')
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('2 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F2 + 1 TMD varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.14. Maximum Displacement: Adding extra TMDs to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 16:19:42 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
```

```
    return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT = fr(1)**2*mT
K_2T = np.zeros((3,3))
K_2T[0,0] = -2*k/m
K_2T[0,1] = k/m
K_2T[1,0] = k/m
K_2T[1,1] = -k/m
T2_2 = np.zeros((3,3))
T2_2[1,1] = -kT/m
T2_2[1,2] = kT/m
T2_2[2,1] = kT/mT
T2_2[2,2] = -kT/mT
upperright = K_2T + T2_2
matrix = np.zeros((6,6))
for j1 in range(0,3):
    matrix[j1+3,j1] = 1
    for j2 in range(0,3):
        matrix[j1,j2+3] = upperright[j1,j2]
neigvals = np.linalg.eigvals(matrix)
nIlist = []
for i in range(0,3):
    nIlist.append(neigvals[2*i].imag)
nIlist.sort()
def nfr(n):
    return nIlist[n-1]
print(nIlist)

kT1 = nfr(1)**2*mT
kT2 = nfr(2)**2*mT
K_2TTT = np.zeros((5,5))
K_2TTT[0,0] = -2*k/m
K_2T[0,1] = k/m
K_2T[1,0] = k/m
K_2T[1,1] = -k/m
TT1T2_2 = np.zeros((5,5))
for i in range(0,3):
    for j in range(0,3):
        TT1T2_2[i,j] = T2_2[i,j]
def Kt_2TTT(n1,n2):
    TT2T2_n1 = np.zeros((5,5))
    TT3T2_n2 = np.zeros((5,5))
    i = n1-1
    TT2T2_n1[i,i] = -kT1/m
    TT2T2_n1[i,2] = kT1/m
    TT2T2_n1[2,i] = kT1/mT
    TT2T2_n1[2,2] = -kT1/mT
    j = n2-1
    TT3T2_n2[j,j] = -kT2/m
    TT3T2_n2[j,2] = kT2/m
    TT3T2_n2[2,j] = kT2/mT
    TT3T2_n2[2,2] = -kT2/mT

    upperright = K_2TTT + TT1T2_2 + TT2T2_n1 + TT3T2_n2
```

```
    matrix = np.zeros((10,10))
    for j1 in range(0,5):
        matrix[j1+5,j1] = 1
        for j2 in range(0,5):
            matrix[j1,j2+5] = upperright[j1,j2]
    return matrix
print(np.linalg.eigvals(Kt_2TTT(2,2)).imag)

dt_2T = np.zeros((10,1))
for i in range(0,2):
    dt_2T[i,0] = 1
def f(t,y, omega,matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2T
    return term1 + term2

omegalist=[]
omegasteps = 400
omegamax = 100
for val in range(0,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[[],[]],[[],[]]]
for n1 in range(1,3):
    for n2 in range(1,3):
        matrix = Kt_2TTT(n1,n2)
        for omega in omegalist:
            y0 = np.zeros((10,1))
            wlist = [y0]
            for i in range(0,steps+1):
                tn = tlist[i]
                wn = wlist[-1]
                k1 = dt*f(tn,wn,omega,matrix)
                k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
                k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
                k4 = dt*f(tn+dt,wn+k3,omega,matrix)
                wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
                wlist.append(wnew)
            xiabslist = [[[[],[]],[[],[]]],[[[],[]],[[],[]]]]
            xiabsmaxlist = [[[],[]],[[],[]]]
            for i in range(0,2):
                for j in range(0,steps+1):
                    xival = wlist[j][i+5,0]
                    xiabslist[n1-1][n2-1][i].append(abs(xival))
                xiabsmaxlist[n1-1][n2-1].append(max(xiabslist[n1-1][n2-1][i]))
            xabsmaxlist[n1-1][n2-1].append(max(xiabsmaxlist[n1-1][n2-1]))
            print('finished with omega =', omega, ', n1 =', n1, ', n2 =', n2)

toplist = []
for i in range(0,2):
    for j in range(0,2):
        toplist.append(max(xabsmaxlist[i][j]))
top = max(toplist)
for n1 in range(1,3):
    for n2 in range(1,3):
        plt.plot(omegalist,xabsmaxlist[n1-1][n2-1],label='('+str(n1)+','+str(n2)+')')
```

```
for z in range(1,3):
    if fr(z) <= omegamax:
        if z == 1:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency'
        else:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey')
    # if nfr(z) <= omegamax:
    #     if z == 1:
    #         plt.vlines(x=nfr(z),ymin=0,ymax=top,linestyle='--',color='moccasin',label='new eige
    #     else:
    #         plt.vlines(x=nfr(z),ymin=0,ymax=top,linestyle='--',color='moccasin')
plt.title('2 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F2 + extra TMDs varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.15. Maximum Displacement: Adding 1 TMD to Five Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 00:00:38 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_5 = np.zeros((10,10))
for i in range(0,4):
    Kt_5[i,i+5] = -2*k/m
    Kt_5[i,i+6] = k/m
    Kt_5[i+1,i+5] = k/m
Kt_5[4,9] = -k/m
for i in range(0,5):
    Kt_5[i+5,i] = 1
valsKt_5 = np.linalg.eigvals(Kt_5)
```

```
Ilist = []
for i in range(0,5):
    Ilist.append(valsKt_5[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT = fr(1)**2*mT
K_5T = np.zeros((6,6))
K_5T[0,0] = -2*k/m
for i in range(0,4):
    K_5T[i,i+1] = k/m
    K_5T[i+1,i] = k/m
for i in range(0,3):
    K_5T[i+1,i+1] = -2*k/m
K_5T[4,4] = -k/m
def Kt_5T(n):
    T1T5_n = np.zeros((6,6))
    if n != 0:
        i = n-1
        T1T5_n[i,i] = -kT/m
        T1T5_n[i,5] = kT/m
        T1T5_n[5,i] = kT/mT
        T1T5_n[5,5] = -kT/mT
    upperright = K_5T + T1T5_n
    matrix = np.zeros((12,12))
    for j1 in range(0,6):
        matrix[j1+6,j1] = 1
        for j2 in range(0,6):
            matrix[j1,j2+6] = upperright[j1,j2]
    return matrix

dt_5TT = np.zeros((12,1))
for i in range(0,5):
    dt_5TT[i,0] = 1
def f(t,y, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_5TT
    return term1 + term2

omegalist=[]
omegasteps = 400
omegamax = 100
for val in range(0,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[]]
for n in range(0,6):
    matrix = Kt_5T(n)
    for omega in omegalist:
        y0 = np.zeros((12,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
```

```
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]]]
        xiabsmaxlist = [[],[],[],[],[],[]]
        for i in range(0,5):
            for j in range(0,steps+1):
                xival = wlist[j][i+6,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,6):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,6):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='TMD on floor '+ str(n))
for m in range(1,5):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency'
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('5 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7")
plt.savefig('M1F5 + 1 TMD varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.16. Maximum Displacement: Adding 2 TMDs to Five Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 15:06:57 2023

@author: bradl
"""

import os
import math
```

```
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_5 = np.zeros((10,10))
for i in range(0,4):
    Kt_5[i,i+5] = -2*k/m
    Kt_5[i,i+6] = k/m
    Kt_5[i+1,i+5] = k/m
Kt_5[4,9] = -k/m
for i in range(0,5):
    Kt_5[i+5,i] = 1
valsKt_5 = np.linalg.eigvals(Kt_5)
Ilist = []
for i in range(0,5):
    Ilist.append(valsKt_5[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT1 = fr(1)**2*mT
kT2 = fr(2)**2*mT
K_5TT = np.zeros((7,7))
K_5TT[0,0] = -2*k/m
for i in range(0,4):
    K_5TT[i,i+1] = k/m
    K_5TT[i+1,i] = k/m
for i in range(0,3):
    K_5TT[i+1,i+1] = -2*k/m
K_5TT[4,4] = -k/m
T1T5_5 = np.zeros((7,7))
T1T5_5[4,4] = -kT1/m
T1T5_5[4,5] = kT1/m
T1T5_5[5,4] = kT1/mT
T1T5_5[5,5] = -kT1/mT
def Kt_5TT(n):
    T2T5_n = np.zeros((7,7))
    if n != 0:
        i = n-1
        T2T5_n[i,i] = -kT2/m
        T2T5_n[i,6] = kT2/m
```

```
        T2T5_n[6,i] = kT2/mT
        T2T5_n[6,6] = -kT2/mT
    upperright = K_5TT + T1T5_5 + T2T5_n
    matrix = np.zeros((14,14))
    for j1 in range(0,7):
        matrix[j1+7,j1] = 1
        for j2 in range(0,7):
            matrix[j1,j2+7] = upperright[j1,j2]
    return matrix

dt_5TT = np.zeros((14,1))
for i in range(0,5):
    dt_5TT[i,0] = 1
def f(t,y, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_5TT
    return term1 + term2

omegalist=[]
omegasteps = 400
omegamax = 100
for val in range(0,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[]]
frlist = [[0],[0],[0],[0],[0]]
xfrlist = [[0],[0],[0],[0],[0]]
def bound(l):
    if l == 0:
        return 0
    elif l == 1:
        return 20
    elif l == 2:
        return 40
    elif l == 3:
        return 60
    elif l == 4:
        return 100
for n in range(0,6):
    matrix = Kt_5TT(n)
    for omega in omegalist:
        y0 = np.zeros((14,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
```

```
                    [[],[],[],[],[]],
                    [[],[],[],[],[]]]
        xiabsmaxlist = [[],[],[],[],[],[]]
        for i in range(0,5):
            for j in range(0,steps+1):
                xival = wlist[j][i+7,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        if n == 1:
            for l in range(0,4):
                if bound(l)< omega and omega < bound(l+1):
                    if xfrlist[l][-1]<xabsmaxlist[n][-1]:
                        frlist[l].append(omega)
                        xfrlist[l].append(xabsmaxlist[n][-1])
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,6):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,6):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without 2nd TMD')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='2nd TMD on floor '+ str(n))
for m in range(1,5):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency')
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
for n in range(1,6):
    print(frlist[n-1][-1])
plt.title('5 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7")
plt.savefig('M1F5 + 2 TMDs varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.17. Maximum Displacement: Adding 1 TMD to Ten Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 12:40:25 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
```

```
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_10 = np.zeros((20,20))
for i in range(0,9):
    Kt_10[i,i+10] = -2*k/m
    Kt_10[i,i+11] = k/m
    Kt_10[i+1,i+10] = k/m
Kt_10[9,19] = -k/m
for i in range(0,10):
    Kt_10[i+10,i] = 1
valsKt_10 = np.linalg.eigvals(Kt_10)
Ilist = []
for i in range(0,10):
    Ilist.append(valsKt_10[2*i].imag)
Ilist.sort()
def fr(n):
    if n == 0:
        return 0
    else:
        return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT = fr(1)**2*mT
K_10T = np.zeros((11,11))
K_10T[0,0] = -2*k/m
for i in range(0,9):
    K_10T[i,i+1] = k/m
    K_10T[i+1,i] = k/m
for i in range(0,8):
    K_10T[i+1,i+1] = -2*k/m
K_10T[9,9] = -k/m
def Kt_10T(n):
    T10_n = np.zeros((11,11))
    if n != 0:
        i = n-1
        T10_n[i,i] = -kT/m
        T10_n[i,10] = kT/m
        T10_n[10,i] = kT/mT
        T10_n[10,10] = -kT/mT
    upperright = K_10T + T10_n
    matrix = np.zeros((22,22))
    for j1 in range(0,11):
        matrix[j1+11,j1] = 1
```

```
        for j2 in range(0,11):
            matrix[j1,j2+11] = upperright[j1,j2]
    return matrix

dt_10T = np.zeros((22,1))
for i in range(0,10):
    dt_10T[i,0] = 1
def f(t,y, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_10T
    return term1 + term2

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(0,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[],[]]
for n in range(0,11):
    matrix = Kt_10T(n)
    for omega in omegalist:
        y0 = np.zeros((22,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]]]

        xiabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[]]
        for i in range(0,5):
            for j in range(0,steps+1):
                xival = wlist[j][i+11,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        print('finished with omega =', omega, ', n =', n)

top = max(max(xabsmaxlist[0]),max(xabsmaxlist[1]),max(xabsmaxlist[2]),max(xabsmaxlist[3]),max(xabsmaxli
for n in range(0,11):
```

```
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='TMD on floor '+ str(n))
for m in range(1,10):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency'
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('10 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7",loc="lower right")
plt.savefig('M1F10 + TMD varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.18. Maximum Displacement: Adding 2 TMDs to Ten Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 14:49:29 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_10 = np.zeros((20,20))
for i in range(0,9):
    Kt_10[i,i+10] = -2*k/m
    Kt_10[i,i+11] = k/m
    Kt_10[i+1,i+10] = k/m
Kt_10[9,19] = -k/m
for i in range(0,10):
    Kt_10[i+10,i] = 1
valsKt_10 = np.linalg.eigvals(Kt_10)
Ilist = []
```

```
for i in range(0,10):
    Ilist.append(valsKt_10[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT1 = fr(1)**2*mT
kT2 = fr(2)**2*mT
K_10TT = np.zeros((12,12))
K_10TT[0,0] = -2*k/m
for i in range(0,9):
    K_10TT[i,i+1] = k/m
    K_10TT[i+1,i] = k/m
for i in range(0,8):
    K_10TT[i+1,i+1] = -2*k/m
K_10TT[9,9] = -k/m
T1T10_10 = np.zeros((12,12))
T1T10_10[9,9] = -kT1/m
T1T10_10[9,10] = kT1/m
T1T10_10[10,9] = kT1/mT
T1T10_10[10,10] = -kT1/mT
def Kt_10TT(n):
    T2T10_n = np.zeros((12,12))
    if n != 0:
        i = n-1
        T2T10_n[i,i] = -kT2/m
        T2T10_n[i,11] = kT2/m
        T2T10_n[11,i] = kT2/mT
        T2T10_n[11,11] = -kT2/mT
    upperright = K_10TT + T1T10_10 + T2T10_n
    matrix = np.zeros((24,24))
    for j1 in range(0,12):
        matrix[j1+12,j1] = 1
        for j2 in range(0,12):
            matrix[j1,j2+12] = upperright[j1,j2]
    return matrix

dt_10T = np.zeros((24,1))
for i in range(0,10):
    dt_10T[i,0] = 1
def f(t,y, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_10T
    return term1 + term2

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(0,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[]]
for n in range(0,11):
    matrix = Kt_10TT(n)
```

```
    for omega in omegalist:
        y0 = np.zeros((24,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]]]

        xiabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[]]
        for i in range(0,10):
            for j in range(0,steps+1):
                xival = wlist[j][i+12,0]
                xiabslist[n-1][i].append(abs(xival))
            xiabsmaxlist[n-1].append(max(xiabslist[n-1][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n-1]))
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,10):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,11):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without 2nd TMD')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='2nd TMD on floor '+ str(n))
for m in range(1,11):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency'
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('10 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7")
plt.savefig('M1F10 + 2 TMDs varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.19. Maximum Displacement: Adding 1 VED to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 17:53:06 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

def C_2(omega,n):
```

```
    matrix = np.zeros((2,2))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_2V(omega,n):
    matrix = np.zeros((4,4))
    for i in range(0,4):
        for j in range(0,4):
            matrix[i,j] = Kt_2[i,j]
    for i in range(0,2):
        for j in range(0,2):
            matrix[i,j] = C_2(omega,n)[i,j]
    return matrix
dt_2 = np.zeros((4,1))
for i in range(0,2):
    dt_2[i,0] = 1
def f(t,y, omega,matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2
    return term1 + term2

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[]]
for n in range(0,3):
    for omega in omegalist:
        matrix = Kt_2V(omega,n)
        y0 = np.zeros((4,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[]],[[],[]],[[],[]]]
        xiabsmaxlist = [[],[],[]]
        for i in range(0,2):
            for j in range(0,steps+1):
                xival = wlist[j][i+2,0]
                xiabslist[n-1][i].append(abs(xival))
            xiabsmaxlist[n-1].append(max(xiabslist[n-1][i]))
        xabsmaxlist[n-1].append(max(xiabsmaxlist[n-1]))
        print('finished with omega =', omega, ', n =', n)

toplist = []
```

```
for i in range(0,3):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,3):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='no damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='10 VEDs on floor '+str(n))
for z in range(1,3):
    if fr(z) <= omegamax:
        if z == 1:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency')
        else:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('2 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F2 + 1 VED varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.20. Maximum Displacement: Adding 2 VEDs to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 23:38:42 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
```

```
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

x=1
def C_2V1_x(omega):
    matrix = np.zeros((2,2))
    matrix[x-1,x-1] = -cV(omega)/m
    matrix[x-1,x-2] = cV(omega)/m
    return matrix

def C_2V2(omega,n):
    matrix = np.zeros((2,2))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_2VV(omega,n):
    matrix = np.zeros((4,4))
    for i in range(0,4):
        for j in range(0,4):
            matrix[i,j] = Kt_2[i,j]
    upperleft = C_2V1_x(omega) + C_2V2(omega,n)
    for i in range(0,2):
        for j in range(0,2):
            matrix[i,j] = upperleft[i,j]
    return matrix
dt_2 = np.zeros((4,1))
for i in range(0,2):
    dt_2[i,0] = 1
def f(t,y, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2
    return term1 + term2

omegalist=[]
omegasteps = 200
```

```
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[]]
for n in range(0,3):
    for omega in omegalist:
        matrix = Kt_2VV(omega,n)
        y0 = np.zeros((4,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[]],[[],[]],[[],[]]]
        xiabsmaxlist = [[],[],[]]
        for i in range(0,2):
            for j in range(0,steps+1):
                xival = wlist[j][i+2,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,3):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,3):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without a 2nd set')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='2nd ten VEDs on floor '+str(n))
for z in range(1,3):
    if fr(z) <= omegamax:
        if z == 1:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency')
        else:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('2 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7")
plt.savefig('M1F2 + 2 VEDs varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.21. Maximum Displacement: Adding 1 VED to Five Floors

```
    # -*- coding: utf-8 -*-
"""
```

```
Created on Sun Jul  2 19:21:21 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_5 = np.zeros((10,10))
for i in range(0,4):
    Kt_5[i,i+5] = -2*k/m
    Kt_5[i,i+6] = k/m
    Kt_5[i+1,i+5] = k/m
Kt_5[4,9] = -k/m
for i in range(0,5):
    Kt_5[i+5,i] = 1
valsKt_5 = np.linalg.eigvals(Kt_5)
Ilist = []
for i in range(0,5):
    Ilist.append(valsKt_5[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

def C_5(omega,n):
    matrix = np.zeros((5,5))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
```

```
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_5V(omega,n):
    matrix = np.zeros((10,10))
    for i in range(0,10):
        for j in range(0,10):
            matrix[i,j] = Kt_5[i,j]
    upperleft = C_5(omega,n)
    for i in range(0,5):
        for j in range(0,5):
            matrix[i,j] = upperleft[i,j]
    return matrix
dt_5 = np.zeros((10,1))
for i in range(0,5):
    dt_5[i,0] = 1
def f(t,y, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_5
    return term1 + term2

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[]]
for n in range(0,6):
    for omega in omegalist:
        matrix = Kt_5V(omega,n)
        y0 = np.zeros((10,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]]]
        xiabsmaxlist = [[],[],[],[],[],[]]
        for i in range(0,5):
            for j in range(0,steps+1):
                xival = wlist[j][i+5,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
```

```
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,5):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,6):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='no damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='10 VEDs on floor '+str(n))
for z in range(1,3):
    if fr(z) <= omegamax:
        if z == 1:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency'
        else:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('5 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M1F5 + 1 VED varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.22. Maximum Displacement: Adding 2 VEDs to Five Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 23:45:18 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_5 = np.zeros((10,10))
for i in range(0,4):
    Kt_5[i,i+5] = -2*k/m
```

```
    Kt_5[i,i+6] = k/m
    Kt_5[i+1,i+5] = k/m
Kt_5[4,9] = -k/m
for i in range(0,5):
    Kt_5[i+5,i] = 1
valsKt_5 = np.linalg.eigvals(Kt_5)
Ilist = []
for i in range(0,5):
    Ilist.append(valsKt_5[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

x=1
def C_5V1_x(omega):
    matrix = np.zeros((5,5))
    matrix[x-1,x-1] = -cV(omega)/m
    matrix[x-1,x-2] = cV(omega)/m
    return matrix

def C_5V2(omega,n):
    matrix = np.zeros((5,5))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_5VV(omega,n):
    matrix = np.zeros((10,10))
    for i in range(0,10):
        for j in range(0,10):
            matrix[i,j] = Kt_5[i,j]
    upperleft = C_5V1_x(omega) + C_5V2(omega,n)
    for i in range(0,5):
        for j in range(0,5):
            matrix[i,j] = upperleft[i,j]
    return matrix
dt_5 = np.zeros((10,1))
for i in range(0,5):
    dt_5[i,0] = 1
def f(t,y, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_5
```

```
    return term1 + term2

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[]]
for n in range(0,6):
    for omega in omegalist:
        matrix = Kt_5VV(omega,n)
        y0 = np.zeros((10,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]]]
        xiabsmaxlist = [[],[],[],[],[],[]]
        for i in range(0,5):
            for j in range(0,steps+1):
                xival = wlist[j][i+5,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,6):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,6):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without a 2nd set')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='2nd set VEDs on floor '+str(n))
for z in range(1,6):
    if fr(z) <= omegamax:
        if z == 1:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency'
        else:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('5 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
```

```
plt.legend(fontsize="7")
plt.savefig('M1F5 + 2 VEDs varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.23. Maximum Displacement: Adding 1 VED to Ten Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Jul  2 17:46:38 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_10 = np.zeros((20,20))
for i in range(0,9):
    Kt_10[i,i+10] = -2*k/m
    Kt_10[i,i+11] = k/m
    Kt_10[i+1,i+10] = k/m
Kt_10[9,19] = -k/m
for i in range(0,10):
    Kt_10[i+10,i] = 1
valsKt_10 = np.linalg.eigvals(Kt_10)
Ilist = []
for i in range(0,10):
    Ilist.append(valsKt_10[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
```

```
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

def C_10(omega,n):
    matrix = np.zeros((10,10))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_10V(omega,n):
    matrix = np.zeros((20,20))
    for i in range(0,20):
        for j in range(0,20):
            matrix[i,j] = Kt_10[i,j]
    upperleft = C_10(omega,n)
    for i in range(0,10):
        for j in range(0,10):
            matrix[i,j] = upperleft[i,j]
    return matrix
dt_10 = np.zeros((20,1))
for i in range(0,5):
    dt_10[i,0] = 1
def f(t,y, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_10
    return term1 + term2

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[],[]]
for n in range(0,11):
    for omega in omegalist:
        matrix = Kt_10V(omega,n)
        y0 = np.zeros((20,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
```

```
                    [[],[],[],[],[],[],[],[],[],[]],
                    [[],[],[],[],[],[],[],[],[],[]],
                    [[],[],[],[],[],[],[],[],[],[]],
                    [[],[],[],[],[],[],[],[],[],[]],
                    [[],[],[],[],[],[],[],[],[],[]],
                    [[],[],[],[],[],[],[],[],[],[]],
                    [[],[],[],[],[],[],[],[],[],[]]]
        xiabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[]]
        for i in range(0,10):
            for j in range(0,steps+1):
                xival = wlist[j][i+10,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,10):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,10):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='no damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='10 VEDs on floor '+ str(n))
for m in range(1,10):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency')
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('10 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7")
plt.savefig('M1F10 + 1 VED varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.1.24. Maximum Displacement: Adding 2 VEDs to Ten Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 01:38:34 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
```

```
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Kt_10 = np.zeros((20,20))
for i in range(0,9):
    Kt_10[i,i+10] = -2*k/m
    Kt_10[i,i+11] = k/m
    Kt_10[i+1,i+10] = k/m
Kt_10[9,19] = -k/m
for i in range(0,10):
    Kt_10[i+10,i] = 1
valsKt_10 = np.linalg.eigvals(Kt_10)
Ilist = []
for i in range(0,10):
    Ilist.append(valsKt_10[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

x=1
def C_10V1_x(omega):
    matrix = np.zeros((10,10))
    matrix[x-1,x-1] = -cV(omega)/m
    matrix[x-1,x-2] = cV(omega)/m
    return matrix

def C_10V2(omega,n):
    matrix = np.zeros((10,10))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_10VV(omega,n):
    matrix = np.zeros((20,20))
```

```
    for i in range(0,20):
        for j in range(0,20):
            matrix[i,j] = Kt_10[i,j]
    upperleft = C_10V1_x(omega) + C_10V2(omega,n)
    for i in range(0,10):
        for j in range(0,10):
            matrix[i,j] = upperleft[i,j]
    return matrix
dt_10 = np.zeros((20,1))
for i in range(0,5):
    dt_10[i,0] = 1
def f(t,y, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_10
    return term1 + term2

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[],[]]
for n in range(0,11):
    for omega in omegalist:
        matrix = Kt_10VV(omega,n)
        y0 = np.zeros((20,1))
        wlist = [y0]
        for i in range(0,steps+1):
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[]]]
        xiabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[],[]]
        for i in range(0,10):
            for j in range(0,steps+1):
                xival = wlist[j][i+10,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        print('finished with omega =', omega, ', n =', n)
```

```
toplist = []
for i in range(0,11):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,11):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without a 2nd set')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='2nd set VEDs on floor '+str(n))
for m in range(1,10):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency'
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('10 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7")
plt.savefig('M1F10 + 2 VEDs varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.2. Codes for the Model including Air Resistance

B.2.1. Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 21:46:15 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 20
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
```

```
A = 36
rho = 1.293
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    if n == 1:
        return 0
    elif n == 2:
        return 0

dt_2 = np.zeros((4,1))
for i in range(0,2):
    dt_2[i,0] = 1
def f(t,y, ydiff, omega):
    term1 = np.dot(Kt_2,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2
    term3 = np.zeros((4,1))
    Favec = np.ones((4,1))
    for i in range(0,2):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist = [2,44]
for omega in omegalist:
    y0 = np.zeros((4,1))
    wlist = [y0]
    for i in range(0,steps+1):
        if i > 0:
            wnold = wlist[-2]
        elif i == 0:
            wnold = y0
        tn = tlist[i]
        wn = wlist[-1]
        wndiff = wn - wnold
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,wndiff,omega)
        k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega)
        k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega)
        k4 = dt*f(tn+dt,wn+k3,wndiff,omega)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
```

```
        wlist.append(wnew)
    xilist = [[],[]]
    for i in range(0,2):
        for j in range(0,steps+1):
            xival = wlist[j][i+2,0]
            xilist[i].append(xival)
    for i in range(0,2):
        plt.plot(tlist,xilist[i],label='floor '+str(i+1),linewidth=1)
    plt.title('earthquake-frequency is '+str(omega)+' Hz')
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.legend()
    if omega == 2:
        plt.savefig('M2F2 om=2.png',dpi=400)
    if omega == omegalist[1]:
        plt.savefig('M2F2 om=44.png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.2.2. Five Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 01:51:12 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 20
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
A = 36
rho = 1.293
Kt_5 = np.zeros((10,10))
for i in range(0,4):
    Kt_5[i,i+5] = -2*k/m
    Kt_5[i,i+6] = k/m
    Kt_5[i+1,i+5] = k/m
Kt_5[4,9] = -k/m
```

```
for i in range(0,5):
    Kt_5[i+5,i] = 1
valsKt_5 = np.linalg.eigvals(Kt_5)
Ilist = []
for i in range(0,5):
    Ilist.append(valsKt_5[2*i].imag)
Ilist.sort()

d = np.zeros((10,1))
for i in range(0,5):
    d[i,0] = 1
def f(t,y, ydiff, omega):
    term1 = np.dot(Kt_5,y)
    term2 = -F0*omega**2*math.cos(omega*t)*d
    term3 = np.zeros((10,1))
    Favec = np.ones((10,1))
    for i in range(0,5):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist = [2,21.5]
for omega in omegalist:
    # Runge-Kutta 4
    y0 = np.zeros((10,1))
    wlist = [y0]
    for i in range(0,steps+1):
        if i > 0:
            wnold = wlist[-2]
        elif i == 0:
            wnold = y0
        tn = tlist[i]
        wn = wlist[-1]
        wndiff = wn - wnold
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,wndiff,omega)
        k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega)
        k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega)
        k4 = dt*f(tn+dt,wn+k3,wndiff,omega)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)

    xlist = [[],[],[],[],[]]
    for i in range(0,5):
        for j in range(0,steps+1):
            xival = wlist[j][i+5,0]
            xlist[i].append(xival)
    for i in range(0,5):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
```

```
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 2:
        plt.savefig('M2F5 om=2 (num).png',dpi=400)
    elif omega == omegalist[1]:
        plt.savefig('M2F5 om=21.5 (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))

######################## NUMERICAL STABILITY OF RUNGE KUTTA 4
def Q(l,dt):
    I = l*dt
    terms = I**8-8*I**6+576
    return math.sqrt(terms)/24
for l in valsKt_5.imag:
    if abs(Q(l,dt))>1:
        print("The timestep is too large.")
        break
    else:
        if l == valsKt_5.imag[-1]:
            print("The timestep is small enough.")
print("Your timestep is", dt)
dtmaxlist = []
for l in valsKt_5.imag:
    dtmax = abs(2*math.sqrt(2)/l)
    dtmaxlist.append(dtmax)
print("The maximum time-step for which RK4 converges is", round(min(dtmaxlist),10))
```

B.2.3. Ten Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Tue Jul  4 00:33:12 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 50
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
```

```
Cd = 2
A = 36
rho = 1.293
Kt_10 = np.zeros((20,20))
for i in range(0,9):
    Kt_10[i,i+10] = -2*k/m
    Kt_10[i,i+11] = k/m
    Kt_10[i+1,i+10] = k/m
Kt_10[9,19] = -k/m
for i in range(0,10):
    Kt_10[i+10,i] = 1
valsKt_10 = np.linalg.eigvals(Kt_10)
Ilist = []
for i in range(0,10):
    Ilist.append(valsKt_10[2*i].imag)
Ilist.sort()

dt_10 = np.zeros((20,1))
for i in range(0,10):
    dt_10[i,0] = 1
def f(t,y, ydiff, omega):
    term1 = np.dot(Kt_10,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_10
    term3 = np.zeros((20,1))
    Favec = np.ones((20,1))
    for i in range(0,10):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist = [2,11.5]
for omega in omegalist:
    # Runge-Kutta 4
    y0 = np.zeros((20,1))
    wlist = [y0]
    for i in range(0,steps+1):
        if i > 0:
            wnold = wlist[-2]
        elif i == 0:
            wnold = y0
        tn = tlist[i]
        wn = wlist[-1]
        wndiff = wn - wnold
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,wndiff,omega)
        k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega)
        k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega)
        k4 = dt*f(tn+dt,wn+k3,wndiff,omega)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)
    xlist = [[],[],[],[],[],[],[],[],[],[]]
```

```
    for i in range(0,10):
        for j in range(0,steps+1):
            xival = wlist[j][i+10,0]
            xlist[i].append(xival)
    for i in range(0,10):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == 2:
        plt.savefig('M2F10 om=2 (num).png',dpi=400)
    elif omega == omegalist[1]:
        plt.savefig('M2F10 om=11.5 (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.2.4. Adding 1 TMD to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Tue Jul  4 01:45:46 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
A = 36
rho = 1.293
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
```

```
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT = fr(1)**2*mT
K_2T = np.zeros((3,3))
K_2T[0,0] = -2*k/m
K_2T[0,1] = k/m
K_2T[1,0] = k/m
K_2T[1,1] = -k/m
def Kt_2T(n):
    T2_n = np.zeros((3,3))
    if n != 0:
        i = n-1
        T2_n[i,i] = -kT/m
        T2_n[i,2] = kT/m
        T2_n[2,i] = kT/mT
        T2_n[2,2] = -kT/mT
    upperright = K_2T + T2_n
    matrix = np.zeros((6,6))
    for j1 in range(0,3):
        matrix[j1+3,j1] = 1
        for j2 in range(0,3):
            matrix[j1,j2+3] = upperright[j1,j2]
    return matrix
neigvals = np.linalg.eigvals(Kt_2T(2))
nIlist = []
nIlist = []
for i in range(0,3):
    nIlist.append(neigvals[2*i].imag)
nIlist.sort()
def nfr(n):
    return nIlist[n-1]
print(nIlist)
dt_2T = np.zeros((6,1))
for i in range(0,2):
    dt_2T[i,0] = 1
def f(t,y,ydiff, omega,matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2T
    term3 = np.zeros((6,1))
    Favec = np.ones((6,1))
    for i in range(0,5):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3
```

```
omegalist = [nfr(1)-0.00001,fr(1)-4.3]
for omega in omegalist:
    matrix = Kt_2T(2)
    # Runge-Kutta 4
    y0 = np.zeros((6,1))
    wlist = [y0]
    for i in range(0,steps+1):
        if i > 0:
            wnold = wlist[-2]
        elif i == 0:
            wnold = y0
        tn = tlist[i]
        wn = wlist[-1]
        wndiff = wn - wnold
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,wndiff,omega,matrix)
        k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega,matrix)
        k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega,matrix)
        k4 = dt*f(tn+dt,wn+k3,wndiff,omega,matrix)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)

    xlist = [[],[]]
    for i in range(0,2):
        for j in range(0,steps+1):
            xival = wlist[j][i+3,0]
            xlist[i].append(xival)
    for i in range(0,2):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == omegalist[0]:
        plt.savefig('M2F2 + TMD om=nfr1 (num).png',dpi=400)
    elif omega == omegalist[1]:
        plt.savefig('M2F2 + TMD om=fr1 (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.2.5. Adding 1 VED to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Tue Jul  4 01:26:06 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))
```

```
tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
A = 36
rho = 1.293
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

def C_2(omega,n):
    matrix = np.zeros((2,2))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_2V(omega,n):
    matrix = np.zeros((4,4))
    for i in range(0,4):
        for j in range(0,4):
            matrix[i,j] = Kt_2[i,j]
```

```
    for i in range(0,2):
        for j in range(0,2):
            matrix[i,j] = C_2(omega,n)[i,j]
    return matrix
dt_2 = np.zeros((4,1))
for i in range(0,2):
    dt_2[i,0] = 1
def f(t,y,ydiff, omega,matrix):
    term1 = np.dot(Kt_2V(omega,1),y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2
    term3 = np.zeros((4,1))
    Favec = np.ones((4,1))
    for i in range(0,2):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist = [2,fr(1)-4.3]#0.00001]
for omega in omegalist:
    matrix = Kt_2V(omega,1)
    # Runge-Kutta 4
    y0 = np.zeros((4,1))
    wlist = [y0]
    for i in range(0,steps+1):
        if i > 0:
            wnold = wlist[-2]
        elif i == 0:
            wnold = y0
        tn = tlist[i]
        wn = wlist[-1]
        wndiff = wn - wnold
        tn = tlist[i]
        wn = wlist[-1]
        k1 = dt*f(tn,wn,wndiff,omega,matrix)
        k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega,matrix)
        k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega,matrix)
        k4 = dt*f(tn+dt,wn+k3,wndiff,omega,matrix)
        wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
        wlist.append(wnew)

    xlist = [[],[]]
    for i in range(0,2):
        for j in range(0,steps+1):
            xival = wlist[j][i+2,0]
            xlist[i].append(xival)
    for i in range(0,2):
        plt.plot(tlist,xlist[i],label='floor ' + str(i+1),linewidth=1)
    plt.xlabel('time (s)')
    plt.ylabel('displacement (m)')
    plt.title('earthquake-frequency: ' + str(round(omega,4)) + ' Hz')
    plt.legend()
    if omega == omegalist[0]:
```

```
        plt.savefig('M2F2 + VED om=2 (num).png',dpi=400)
    elif omega == omegalist[1]:
        plt.savefig('M2F2 + VED om=fr1 (num).png',dpi=400)
    plt.figure(figsize=(8,4.8))
```

B.2.6. Maximum Displacement: Adding 1 TMD to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 03:42:00 2023

@author: bradl
"""


import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
A = 36
rho = 1.293
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

mT = m*0.4
```

```
kT = fr(1)**2*mT
K_2T = np.zeros((3,3))
K_2T[0,0] = -2*k/m
K_2T[0,1] = k/m
K_2T[1,0] = k/m
K_2T[1,1] = -k/m
def Kt_2T(n):
    T2_n = np.zeros((3,3))
    if n != 0:
        i = n-1
        T2_n[i,i] = -kT/m
        T2_n[i,2] = kT/m
        T2_n[2,i] = kT/mT
        T2_n[2,2] = -kT/mT
    upperright = K_2T + T2_n
    matrix = np.zeros((6,6))
    for j1 in range(0,3):
        matrix[j1+3,j1] = 1
        for j2 in range(0,3):
            matrix[j1,j2+3] = upperright[j1,j2]
    return matrix
neigvals = np.linalg.eigvals(Kt_2T(2))
nIlist = []
nIlist = []
for i in range(0,3):
    nIlist.append(neigvals[2*i].imag)
nIlist.sort()
def nfr(n):
    return nIlist[n-1]
print(nIlist)

dt_2T = np.zeros((6,1))
for i in range(0,2):
    dt_2T[i,0] = 1
def f(t,y,ydiff, omega,matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2T
    term3 = np.zeros((6,1))
    Favec = np.ones((6,1))
    for i in range(0,5):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(0,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[]]
for n in range(0,3):
```

```
    matrix = Kt_2T(n)
    for omega in omegalist:
        y0 = np.zeros((6,1))
        wlist = [y0]
        for i in range(0,steps+1):
            if i > 0:
                wnold = wlist[-2]
            elif i == 0:
                wnold = y0
            tn = tlist[i]
            wn = wlist[-1]
            wndiff = wn - wnold
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,wndiff,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,wndiff,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[]],[[],[]],[[],[]]]
        xiabsmaxlist = [[],[],[]]
        for i in range(0,2):
            for j in range(0,steps+1):
                xival = wlist[j][i+3,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        print('finished with omega =', omega, ', n =', n)

top = max(max(xabsmaxlist[0]),max(xabsmaxlist[1]),max(xabsmaxlist[2]))
for n in range(0,3):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='TMD on floor '+ str(n))
for m in range(1,3):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m)-5,ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency')
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('2 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M2F2 + 1 TMD varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.2.7. Maximum Displacement: Adding 1 TMD to Five Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 17:56:36 2023
```

```
@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
A = 36
rho = 1.293
Kt_5 = np.zeros((10,10))
for i in range(0,4):
    Kt_5[i,i+5] = -2*k/m
    Kt_5[i,i+6] = k/m
    Kt_5[i+1,i+5] = k/m
Kt_5[4,9] = -k/m
for i in range(0,5):
    Kt_5[i+5,i] = 1
valsKt_5 = np.linalg.eigvals(Kt_5)
Ilist = []
for i in range(0,5):
    Ilist.append(valsKt_5[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT = fr(1)**2*mT
K_5T = np.zeros((6,6))
K_5T[0,0] = -2*k/m
for i in range(0,4):
    K_5T[i,i+1] = k/m
    K_5T[i+1,i] = k/m
for i in range(0,3):
    K_5T[i+1,i+1] = -2*k/m
K_5T[4,4] = -k/m
def Kt_5T(n):
    T1T5_n = np.zeros((6,6))
    if n != 0:
        i = n-1
```

```
        T1T5_n[i,i] = -kT/m
        T1T5_n[i,5] = kT/m
        T1T5_n[5,i] = kT/mT
        T1T5_n[5,5] = -kT/mT
    upperright = K_5T + T1T5_n
    matrix = np.zeros((12,12))
    for j1 in range(0,6):
        matrix[j1+6,j1] = 1
        for j2 in range(0,6):
            matrix[j1,j2+6] = upperright[j1,j2]
    return matrix

dt_5TT = np.zeros((12,1))
for i in range(0,5):
    dt_5TT[i,0] = 1
def f(t,y,ydiff, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_5TT
    term3 = np.zeros((12,1))
    Favec = np.ones((12,1))
    for i in range(0,5):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[]]
frlist = [[0],[0],[0]]
xfrlist = [[0],[0],[0]]
def bound(l):
    if l == 0:
        return 0
    elif l == 1:
        return 40
    elif l == 2:
        return 70
    elif l == 3:
        return 100
for n in range(0,6):
    matrix = Kt_5T(n)
    for omega in omegalist:
        y0 = np.zeros((12,1))
        wlist = [y0]
        for i in range(0,steps+1):
            if i > 0:
                wnold = wlist[-2]
            elif i == 0:
```

```
                wnold = y0
            tn = tlist[i]
            wn = wlist[-1]
            wndiff = wn - wnold
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,wndiff,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,wndiff,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[],[]],
                     [[],[],[],[],[],[]],
                     [[],[],[],[],[],[]],
                     [[],[],[],[],[],[]],
                     [[],[],[],[],[],[]],
                     [[],[],[],[],[],[]]]
        xiabsmaxlist = [[],[],[],[],[],[],[]]
        for i in range(0,5):
            for j in range(0,steps+1):
                xival = wlist[j][i+6,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        if n == 0:
            for l in range(0,3):
                if bound(l)< omega and omega < bound(l+1):
                    if xfrlist[l][-1]<xabsmaxlist[n][-1]:
                        frlist[l].append(omega)
                        xfrlist[l].append(xabsmaxlist[n][-1])
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,6):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,6):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='TMD on floor '+ str(n))
for n in range(1,3):
    if n == 1:
        plt.vlines(x=frlist[0][-1],ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfreque
        print(frlist[0][-1])
    else:
        plt.vlines(x=frlist[n-1][-1],ymin=0,ymax=top,linestyle='--',color='grey')
        print(frlist[n-1][-1])
plt.title('5 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7")
plt.savefig('M2F5 + 1 TMD varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
toplist = []
```

```
for i in range(0,6):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,6):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='TMD on floor '+ str(n))
for n in range(1,4):
    if n == 1:
        plt.vlines(x=frlist[0][-1],ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency')
        print(frlist[0][-1])
    else:
        plt.vlines(x=frlist[n-1][-1],ymin=0,ymax=top,linestyle='--',color='grey')
        print(frlist[n-1][-1])
plt.title('5 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7")
plt.savefig('M2F5 + 1 TMD varying omega (more eigfr).png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.2.8. Maximum Displacement: Adding 1 TMD to Ten Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 18:49:04 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
A = 36
rho = 1.293
Kt_10 = np.zeros((20,20))
for i in range(0,9):
```

```
    Kt_10[i,i+10] = -2*k/m
    Kt_10[i,i+11] = k/m
    Kt_10[i+1,i+10] = k/m
Kt_10[9,19] = -k/m
for i in range(0,10):
    Kt_10[i+10,i] = 1
valsKt_10 = np.linalg.eigvals(Kt_10)
Ilist = []
for i in range(0,10):
    Ilist.append(valsKt_10[2*i].imag)
Ilist.sort()
def fr(n):
    if n == 0:
        return 0
    else:
        return Ilist[n-1]
print(Ilist)

mT = m*0.4
kT = fr(1)**2*mT
K_10T = np.zeros((11,11))
K_10T[0,0] = -2*k/m
for i in range(0,9):
    K_10T[i,i+1] = k/m
    K_10T[i+1,i] = k/m
for i in range(0,8):
    K_10T[i+1,i+1] = -2*k/m
K_10T[9,9] = -k/m
def Kt_10T(n):
    T10_n = np.zeros((11,11))
    if n != 0:
        i = n-1
        T10_n[i,i] = -kT/m
        T10_n[i,10] = kT/m
        T10_n[10,i] = kT/mT
        T10_n[10,10] = -kT/mT
    upperright = K_10T + T10_n
    matrix = np.zeros((22,22))
    for j1 in range(0,11):
        matrix[j1+11,j1] = 1
        for j2 in range(0,11):
            matrix[j1,j2+11] = upperright[j1,j2]
    return matrix

dt_10T = np.zeros((22,1))
for i in range(0,10):
    dt_10T[i,0] = 1
def f(t,y,ydiff, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_10T
    term3 = np.zeros((22,1))
    Favec = np.ones((22,1))
    for i in range(0,10):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
```

```
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist=[]
omegasteps = 100
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[],[]]
# frlist = [[0],[0],[0],[0],[0]]
# xfrlist = [[0],[0],[0],[0],[0]]
# def bound(l):
#     if l == 0:
#         return 0
#     elif l == 1:
#         return 20
#     elif l == 2:
#         return 40
#     elif l == 3:
#         return 60
#     elif l == 4:
#         return 80
#     elif l == 5:
#         return 100
for n in range(0,11):
    matrix = Kt_10T(n)
    for omega in omegalist:
        y0 = np.zeros((22,1))
        wlist = [y0]
        for i in range(0,steps+1):
            if i > 0:
                wnold = wlist[-2]
            elif i == 0:
                wnold = y0
            tn = tlist[i]
            wn = wlist[-1]
            wndiff = wn - wnold
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,wndiff,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,wndiff,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
```

```
                    [[],[],[],[],[],[],[],[],[],[]],
                    [[],[],[],[],[],[],[],[],[],[]],
                    [[],[],[],[],[],[],[],[],[],[]]]

        xiabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[]]
        for i in range(0,5):
            for j in range(0,steps+1):
                xival = wlist[j][i+11,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        # if n == 0:
        #     for l in range(0,5):
        #         if bound(l)< omega and omega < bound(l+1):
        #             if xfrlist[l][-1]<xabsmaxlist[n][-1]:
        #                 frlist[l].append(omega)
        #                 xfrlist[l].append(xabsmaxlist[n][-1])
        print('finished with omega =', omega, ', n =', n)

top = max(max(xabsmaxlist[0]),max(xabsmaxlist[1]),max(xabsmaxlist[2]),max(xabsmaxlist[3]),max(xab
for n in range(0,11):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='without damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='TMD on floor '+ str(n))
for m in range(1,10):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency'
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('10 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7",loc="lower right")
plt.savefig('M2F10 + TMD varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.2.9. Maximum Displacement: Adding 1 VED to Two Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 16:25:10 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
```

```
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
A = 36
rho = 1.293
Kt_2 = np.zeros((4,4))
for i in range(0,2):
    Kt_2[i+2,i] = 1
Kt_2[0,2] = -2*k/m
Kt_2[0,3] = k/m
Kt_2[1,2] = k/m
Kt_2[1,3] = -k/m
valsKt_2 = np.linalg.eigvals(Kt_2)
Ilist = []
for i in range(0,2):
    Ilist.append(valsKt_2[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

def C_2(omega,n):
    matrix = np.zeros((2,2))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_2V(omega,n):
    matrix = np.zeros((4,4))
    for i in range(0,4):
        for j in range(0,4):
            matrix[i,j] = Kt_2[i,j]
    for i in range(0,2):
        for j in range(0,2):
```

```
            matrix[i,j] = C_2(omega,n)[i,j]
    return matrix
dt_2 = np.zeros((4,1))
for i in range(0,2):
    dt_2[i,0] = 1
def f(t,y,ydiff, omega,matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_2
    term3 = np.zeros((4,1))
    Favec = np.ones((4,1))
    for i in range(0,2):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist=[]
omegasteps = 200
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[]]
for n in range(0,3):
    for omega in omegalist:
        matrix = Kt_2V(omega,n)
        y0 = np.zeros((4,1))
        wlist = [y0]
        for i in range(0,steps+1):
            if i > 0:
                wnold = wlist[-2]
            elif i == 0:
                wnold = y0
            tn = tlist[i]
            wn = wlist[-1]
            wndiff = wn - wnold
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,wndiff,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,wndiff,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[]],[[],[]],[[],[]]]
        xiabsmaxlist = [[],[],[]]
        for i in range(0,2):
            for j in range(0,steps+1):
                xival = wlist[j][i+2,0]
                xiabslist[n-1][i].append(abs(xival))
            xiabsmaxlist[n-1].append(max(xiabslist[n-1][i]))
        xabsmaxlist[n-1].append(max(xiabsmaxlist[n-1]))
        print('finished with omega =', omega, ', n =', n)
```

```
toplist = []
for i in range(0,3):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,3):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='no damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='10 VEDs on floor '+str(n))
for z in range(1,3):
    if fr(z) <= omegamax:
        if z == 1:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency')
        else:
            plt.vlines(x=fr(z),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('2 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M2F2 + 1 VED varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.2.10. Maximum Displacement: Adding 1 VED to Five Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 21:19:29 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
F0 = 0.1
Cd = 2
A = 36
rho = 1.293
Kt_5 = np.zeros((10,10))
```

```
for i in range(0,4):
    Kt_5[i,i+5] = -2*k/m
    Kt_5[i,i+6] = k/m
    Kt_5[i+1,i+5] = k/m
Kt_5[4,9] = -k/m
for i in range(0,5):
    Kt_5[i+5,i] = 1
valsKt_5 = np.linalg.eigvals(Kt_5)
Ilist = []
for i in range(0,5):
    Ilist.append(valsKt_5[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

def C_5(omega,n):
    matrix = np.zeros((5,5))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_5V(omega,n):
    matrix = np.zeros((10,10))
    for i in range(0,10):
        for j in range(0,10):
            matrix[i,j] = Kt_5[i,j]
    upperleft = C_5(omega,n)
    for i in range(0,5):
        for j in range(0,5):
            matrix[i,j] = upperleft[i,j]
    return matrix
dt_5 = np.zeros((10,1))
for i in range(0,5):
    dt_5[i,0] = 1
def f(t,y,ydiff, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_5
    term3 = np.zeros((10,1))
    Favec = np.ones((10,1))
    for i in range(0,5):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
```

```
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist=[]
omegasteps = 100
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[]]
frlist = [[0],[0]]
xfrlist = [[0],[0]]
bound1 = 30
bound2 = 70
for n in range(0,6):
    for omega in omegalist:
        matrix = Kt_5V(omega,n)
        y0 = np.zeros((10,1))
        wlist = [y0]
        for i in range(0,steps+1):
            if i > 0:
                wnold = wlist[-2]
            elif i == 0:
                wnold = y0
            tn = tlist[i]
            wn = wlist[-1]
            wndiff = wn - wnold
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,wndiff,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,wndiff,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]],
                     [[],[],[],[],[]]]
        xiabsmaxlist = [[],[],[],[],[],[]]
        for i in range(0,5):
            for j in range(0,steps+1):
                xival = wlist[j][i+5,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        if n == 0:
            if omega<bound1:
                if xfrlist[0][-1]<xabsmaxlist[n][-1]:
                    frlist[0].append(omega)
                    xfrlist[0].append(xabsmaxlist[n][-1])
```

```
            elif omega<bound2:
                if xfrlist[1][-1]<xabsmaxlist[n][-1]:
                    frlist[1].append(omega)
                    xfrlist[1].append(xabsmaxlist[n][-1])
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,5):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,6):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='no damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='10 VEDs on floor '+str(n))
for n in range(1,3):
    if n == 1:
        plt.vlines(x=frlist[0][-1],ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfreque
        print(frlist[0][-1])
    else:
        plt.vlines(x=frlist[n-1][-1],ymin=0,ymax=top,linestyle='--',color='grey')
        print(frlist[n-1][-1])
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend()
plt.savefig('M2F5 + 1 VED varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.2.11. Maximum Displacement: Adding 1 VED to Ten Floors

```
    # -*- coding: utf-8 -*-
"""
Created on Mon Jul  3 20:09:52 2023

@author: bradl
"""

import os
import math
import numpy as np
import matplotlib.pyplot as plt
dir_name = "C:/Users/bradl/BEP - Figures/"
plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))

tbegin = 0
tend = 5
steps = 4000*(tend-tbegin)
dt = (tend-tbegin)/steps
tlist = [tbegin]
for i in range(1,steps+1):
    t = tbegin+dt*i
    tlist.append(t)

m = 9E5
k = 5556222222
```

```
F0 = 0.1
Cd = 2
A = 36
rho = 1.293
Kt_10 = np.zeros((20,20))
for i in range(0,9):
    Kt_10[i,i+10] = -2*k/m
    Kt_10[i,i+11] = k/m
    Kt_10[i+1,i+10] = k/m
Kt_10[9,19] = -k/m
for i in range(0,10):
    Kt_10[i+10,i] = 1
valsKt_10 = np.linalg.eigvals(Kt_10)
Ilist = []
for i in range(0,10):
    Ilist.append(valsKt_10[2*i].imag)
Ilist.sort()
def fr(n):
    return Ilist[n-1]
print(Ilist)

FREQS = [5,10,15,20,25,30,35]
DATA = np.array([438.3,329.9,279.4,248.3,226.6,210.2,197.4])
def cV(omega):
    c1 = DATA[0]
    o1 = FREQS[0]
    c2 = DATA[-1]
    o2 = FREQS[-1]
    b = math.log(c2/c1)/math.log(o1/o2)
    a = o2**b*c2
    return 10*1000*a/(omega**b)

def C_10(omega,n):
    matrix = np.zeros((10,10))
    if n == 1:
        matrix[0,0] = -cV(omega)/m
    elif n > 1:
        matrix[n-1,n-1] = -cV(omega)/m
        matrix[n-1,n-2] = cV(omega)/m
    return matrix
def Kt_10V(omega,n):
    matrix = np.zeros((20,20))
    for i in range(0,20):
        for j in range(0,20):
            matrix[i,j] = Kt_10[i,j]
    upperleft = C_10(omega,n)
    for i in range(0,10):
        for j in range(0,10):
            matrix[i,j] = upperleft[i,j]
    return matrix
dt_10 = np.zeros((20,1))
for i in range(0,5):
    dt_10[i,0] = 1
def f(t,y,ydiff, omega, matrix):
    term1 = np.dot(matrix,y)
    term2 = -F0*omega**2*math.cos(omega*t)*dt_10
```

```
    term3 = np.zeros((20,1))
    Favec = np.ones((20,1))
    for i in range(0,10):
        vi = ydiff[i,0]/dt
        if ydiff[i,0] > 0:
            Favec[i,0] = -1
        elif ydiff[i,0] == 0:
            Favec[i,0] = 0
        term3[i] = Favec[i,0]*Cd*A*rho*vi**2/(2*m)
    return term1 + term2 + term3

omegalist=[]
omegasteps = 100
omegamax = 100
for val in range(1,omegasteps+1):
    o = val/omegasteps*omegamax
    omegalist.append(o)
xabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[],[]]
for n in range(0,11):
    for omega in omegalist:
        matrix = Kt_10V(omega,n)
        y0 = np.zeros((20,1))
        wlist = [y0]
        for i in range(0,steps+1):
            if i > 0:
                wnold = wlist[-2]
            elif i == 0:
                wnold = y0
            tn = tlist[i]
            wn = wlist[-1]
            wndiff = wn - wnold
            tn = tlist[i]
            wn = wlist[-1]
            k1 = dt*f(tn,wn,wndiff,omega,matrix)
            k2 = dt*f(tn+dt/2,wn+k1/2,wndiff,omega,matrix)
            k3 = dt*f(tn+dt/2,wn+k2/2,wndiff,omega,matrix)
            k4 = dt*f(tn+dt,wn+k3,wndiff,omega,matrix)
            wnew = wn + (k1 + 2*k2 + 2*k3 + k4)/6
            wlist.append(wnew)
        xiabslist = [[[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]],
                     [[],[],[],[],[],[],[],[],[],[],[]]]
        xiabsmaxlist = [[],[],[],[],[],[],[],[],[],[],[]]
        for i in range(0,10):
            for j in range(0,steps+1):
                xival = wlist[j][i+10,0]
                xiabslist[n][i].append(abs(xival))
            xiabsmaxlist[n].append(max(xiabslist[n][i]))
```

```
        xabsmaxlist[n].append(max(xiabsmaxlist[n]))
        print('finished with omega =', omega, ', n =', n)

toplist = []
for i in range(0,10):
    toplist.append(max(xabsmaxlist[i]))
top = max(toplist)
for n in range(0,10):
    if n == 0:
        plt.plot(omegalist,xabsmaxlist[n],label='no damping')
    else:
        plt.plot(omegalist,xabsmaxlist[n],label='10 VEDs on floor '+ str(n))
for m in range(1,10):
    if fr(m) <= omegamax:
        if m == 1:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey',label='eigenfrequency')
        else:
            plt.vlines(x=fr(m),ymin=0,ymax=top,linestyle='--',color='grey')
plt.title('10 floors')
plt.xlabel('earthquake-frequency (Hz)')
plt.ylabel('max. displacement within ' + str(tend) + ' s (m)')
plt.legend(fontsize="7")
plt.savefig('M2F10 + 1 VED varying omega.png',dpi=400)
plt.figure(figsize=(8,4.8))
```

B.3. Other Codes

B.3.1. Fitting a Function to c_V

-*- coding: utf-8 -*- """ Created on Sun Jul 2 00:48:45 2023

@author: bradl """

import os import math import numpy as np import matplotlib.pyplot as plt dir_name = "$C:/Users/bradl/BEP-Figures/$" $plt.rcParams["savefig.directory"] = os.chdir(os.path.dirname(dir_name))$

FREQS = [5,10,15,20,25,30,35] NLOADS = [5,10,15,25,40,50] DATA = np.array([[11.6,8.7,7.3,6.5,5.9,5.5,5.2], [24.0,18.1,15.3,13.6,12.4,11
[41.0,30.8,26.1,23.2,21.1,19.6,18.4], [100.0,75.3,63.7,56.6,51.7,47.9,45.0], [217.3,163.5,138.5,123.1,112.3,104.2,97.8],
[438.3,329.9,279.4,248.3,226.6,210.2,197.4]])

Coeffslist = [] Matrix = np.zeros((7,7)) for power in range(0,7): for j in range(0,7): Matrix[j,power] = FREQS[j]**power
Matrixinv = np.linalg.inv(Matrix) for i in range(0,6): Vector = np.zeros((7,1)) for j in range(0,7): Vector[j,0] =
DATA[i,j] Coeffs = Matrixinv@Vector Coeffslist.append(Coeffs)

def c2(omega): terms = 0 for power in range(0,7): terms += Coeffslist[index][power]*omega**power return
terms

def c1(omega,index): c1 = DATA[index,0] o1 = FREQS[0] c2 = DATA[index,-1] o2 = FREQS[-1] b = math.log(c2/c1)/math.log(o1/o2)
a = o2**b*c2 return a/(omega**b)

omegalist1=[] omegalist2=[] omegasteps = 400 omegamax1 = 100 omegamax2 = 50 for val in range(8,omegasteps+1):
o = val/omegasteps*omegamax1 omegalist1.append(o) for val in range(8,omegasteps+1): o = val/omegasteps*omegamax2
omegalist2.append(o) c1list = [[],[],[],[],[],[]] c2list = [[],[],[],[],[],[]] for index in range(0,6): Nload = NLOADS[index]
for j in range(0,7): if index == 0: plt.plot(FREQS[j],DATA[index,j],marker='o',color='red') elif index == 1: plt.plot(FREQS[j],DATA[index
elif index == 2: plt.plot(FREQS[j],DATA[index,j],marker='o',color='orange') elif index == 3: plt.plot(FREQS[j],DATA[index,j],marker='

```
elif index == 4: plt.plot(FREQS[j],DATA[index,j],marker='o',color='black') elif index == 5: plt.plot(FREQS[j],DATA[index,j],mar
plt.xlabel('earthquake-frequency (Hz)') plt.ylabel('damping coefficient (kN)') plt.savefig('cV data points.png',dpi=400)
plt.figure(figsize=(8,4.8)) for index in range(0,6): Nload = NLOADS[index] for omega in omegalist1: c1list[index].append(c1(c
plt.plot(omegalist1,c1list[index],label = 'Nominal load is '+str(Nload)+' kN',linewidth=1) for j in range(0,7): if
index == 0: plt.plot(FREQS[j],DATA[index,j],marker='o',color='red') elif index == 1: plt.plot(FREQS[j],DATA[index,j],marker='
elif index == 2: plt.plot(FREQS[j],DATA[index,j],marker='o',color='orange') elif index == 3: plt.plot(FREQS[j],DATA[index,j],m
elif index == 4: plt.plot(FREQS[j],DATA[index,j],marker='o',color='black') elif index == 5: plt.plot(FREQS[j],DATA[index,j],mar
plt.xlabel('earthquake-frequency (Hz)') plt.ylabel('damping coefficient (kN)') plt.legend() plt.savefig('cV per-
fect.png',dpi=400) plt.figure(figsize=(8,4.8))

for index in range(0,6): Nload = NLOADS[index] for omega in omegalist2: c2list[index].append(c2(omega,index))
plt.plot(omegalist2,c2list[index],label = 'Nominal load is '+str(Nload)+' kN',linewidth=1) for j in range(0,7): if
index == 0: plt.plot(FREQS[j],DATA[index,j],marker='o',color='red') elif index == 1: plt.plot(FREQS[j],DATA[index,j],marker='
elif index == 2: plt.plot(FREQS[j],DATA[index,j],marker='o',color='orange') elif index == 3: plt.plot(FREQS[j],DATA[index,j],m
elif index == 4: plt.plot(FREQS[j],DATA[index,j],marker='o',color='black') elif index == 5: plt.plot(FREQS[j],DATA[index,j],mar
plt.xlabel('earthquake-frequency (Hz)') plt.ylabel('damping coefficient (kN)') plt.legend() plt.savefig('cV La-
grange.png',dpi=400) plt.figure(figsize=(8,4.8))
```

Bibliography

[1] Atlas Obscura. Tuned Mass Damper of Taipei 101. https://www.atlasobscura.com/places/tuned-mass-damper-of-taipei-101, n.y. Accessed: 15 June, 2023.

[2] Madaan P. Khajuria T. Bagai, S. A mathematical model for the effect of earthquake on high rise buildings of different shapes. *DU Journal of Undergraduate Research and Innovation*, 2:180–188, 2016.

[3] BEKDAŞ, G., NİGDELİ, S.M. Mass ratio factor on optimum TMD design in frequency domain . Research paper -, Istanbul University, Istanbul, 2018.

[4] Aleksandra Bogdanovic and Zoran Rakicevic. Optimal damper placement using combined fitness function. *Frontiers in Built Environment*, 5, 2019. ISSN 2297-3362. doi: 10.3389/fbuil.2019.00004. URL https://www.frontiersin.org/articles/10.3389/fbuil.2019.00004.

[5] Braun, M. *Differential Equations and Their Applications*. Springer, New York, 1993.

[6] Hyunhoon Choi and Jinkoo Kim. New installation scheme for viscoelastic dampers using cables. *Canadian Journal of Civil Engineering*, 37(9):1201–1211, 2010. doi: 10.1139/L10-068. URL https://doi.org/10.1139/L10-068.

[7] Nhan Dao and Chung Ai. A constant friction coefficient model for concave friction bearings. *Journal of Science and Technology in Civil Engineering (STCE) - NUCE*, 14:112–126, 01 2020. doi: 10.31814/stce.nuce2020-14(1)-10.

[8] EECS. Eecs, 16a designing information devices and systems i, spring 2019, lecture notes, note 6. Ebook, n.y. URL https://inst.eecs.berkeley.edu/~ee16a/sp19/lecture/Note6.pdf. 6.

[9] Said Elias and Vasant Matsagar. Research developments in vibration control of structures using passive tuned mass dampers. *Annual Reviews in Control*, 44:129–156, 2017. ISSN 1367-5788. doi: https://doi.org/10.1016/j.arcontrol.2017.09.015. URL https://www.sciencedirect.com/science/article/pii/S1367578817301372.

[10] Kohei Fujita, Abbas Moustafa, and Izuru Takewaki. Optimal placement of visco-elastic dampers and supporting members under variable critical excitations. *Earthquakes and Structures*, 1, 03 2010. doi: 10.12989/eas.2010.1.1.043.

[11] Gavin, H.P. Tuned Mass Damper. Technical report -, Duke University, Durham, 2016.

[12] Johnson, P. Building the Foundation of Earthquakes: An Exposition on Ordinary Differential Equations. Research paper -, West Chester University of Pennsylvania, West Chester, n.y.

[13] Karson, Y., Winson, M. Death toll rises to 9 in earthquake that toppled buildings in Taiwan. https://abcnews.go.com/International/rescuers-scramble-find-dozens-people-trapped-earthquake-kills/story?id=52901523, 2018. Accessed: 3-7-2023.

[14] Alexander N. Koshev and Valentina V. Kuzina. Mathematical models in tasks of construction. *Procedia Engineering*, 161:1874–1878, 2016. ISSN 1877-7058. doi: https://doi.org/10.1016/j.proeng.2016.08.726. URL https://www.sciencedirect.com/science/article/pii/S1877705816329551. World Multidisciplinary Civil Engineering-Architecture-Urban Planning Symposium 2016, WMCAUS 2016.

[15] Hao Li, Weiguo Yang, Pei Liu, and Meng Wang. Resonance measurement and vibration reduction analysis of an office building induced by nearby crane workshop vibration. *Journal of Building Engineering*, 58:105018, 2022. ISSN 2352-7102. doi: https://doi.org/10.1016/j.jobe.2022.105018. URL https://www.sciencedirect.com/science/article/pii/S2352710222010282.

[16] McDevitt T. Marchand, R. Learning differential equations by exploring earthquake induced structural vibrations: A case study. *Int. J. Engng*, 15:477–485, 1999.

[17] Mohtasham Mohebbi and A. Joghataie. Optimal tmds for improving the seismic performance of historical buildings. *Scientia Iranica*, 23:79–90, 01 2016. doi: 10.24200/sci.2016.2099.

[18] NASA. Air Mass/Density. https://www.earthdata.nasa.gov/topics/atmosphere/atmospheric-pressure/air-mass-density#:~:text=Pure%2C%20dry%20air%20has%20a,a%20pressure%20of%20101.325%20kPa, n.y. Accessed: 25 June, 2023.

[19] *Title of the Catalog*. Organization or Company, Location or City, Year.

[20] Zhan Shu, Ruokai You, and Ying Zhou. Viscoelastic materials for structural dampers: A review. *Construction and Building Materials*, 342:127955, 2022. ISSN 0950-0618. doi: https://doi.org/10.1016/j.conbuildmat.2022.127955. URL https://www.sciencedirect.com/science/article/pii/S0950061822016257.

[21] Lekshmi Suresh and K M Mini. Effect of multiple tuned mass dampers for vibration control in high-rise buildings. *Practice Periodical on Structural Design and Construction*, 24:04019031, 11 2019. doi: 10.1061/(ASCE)SC.1943-5576.0000453.

[22] Tenitskaya, T. . http://tm.spbstu.ru/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B3%D0%B0%D1%81%D0%B8%D1%82%D0%B5%D0%BB%D1%8C, n.y. Accessed: 15 June, 2023.

[23] Vuik, C., Vermolen, F.J., Van Gijzen, M.B., Vuik, M.J. *Numerical Methods for Ordinary Differential Equations*. TU Delft Open, Delft, 2015.

[24] Wikipedia. Damping. https://en.wikipedia.org/wiki/Damping, 2023. Accessed: 17 June, 2023.

[25] Wikipedia. Drag (physics). https://en.wikipedia.org/wiki/Drag_(physics), 2023. Accessed: 20 June, 2023.

[26] Wilkins, C. Earthquake Modelling with Differential Equations. Research report -, University of Newcastle, Newcastle, 2018.