Journal of Engineering Science *Vol. XXXII, no. 3 (2025), pp. 7 - 19*
Fascicle *Industrial Engineering* *ISSN 2587-3474*
Topic *Mechanical Engineering and Technologies* *eISSN 2587-3482*

https://doi.org/10.52326/jes.utm.2025.32(3).01
UDC 534.1:621:519.6:004.8

VIBRATION ANALYSIS OF MECHANICAL SYSTEMS WITH MULTIPLE DEGREES OF FREEDOM USING PYTHON

Iulian Malcoci *, ORCID: 0000-0002-0263-5232

Technical University of Moldova, 168, Stefan cel Mare Blvd., Chisinau, Republic of Moldova
* Corresponding author: Iulian Malcoci, iulian.malcoci@bpm.utm.md

Received: 09. 01. 2025
Accepted: 09. 28. 2025

Abstract. This study investigates the dynamic behavior of a discrete multi-mass structure, which is highly relevant for engineers seeking to predict and control structural oscillations. The research starts from the hypothesis that numerical modeling and simulation can effectively describe coupled oscillations in systems with several masses and elastic connections. The main objective is to build and solve the equations of motion for a simple multi-degree-of-freedom model using an open-source programming approach. The theoretical background is explained concisely, while the equations are solved numerically with a Python implementation using standard scientific libraries. Modal analysis, eigenvalue computation and time-domain simulation are performed, and the results demonstrate the distribution of energy and the effects of damping. The combined response is visualized through both classic plots and an innovative 3D representation of the coupled displacements. This work confirms that flexible, free numerical tools can be applied successfully to dynamic studies and provides a clear example for engineering education and applied research.

Keywords: *damping, eigenvalues, modal analysis, numerical simulation, open-source tools.*

Rezumat. Studiul de față cercetează comportamentul dinamic al unei structuri mecanice discrete formată din mai multe mase, aspect extrem de relevant pentru inginerii care doresc să prezică și să controleze oscilațiile structurale. Cercetarea pornește de la ipoteza că modelarea și simularea numerică pot descrie eficient oscilațiile unor sisteme cu mai multe mase conectate elastice între ele. Obiectivul principal este de a construi și rezolva ecuațiile de mișcare pentru un model simplu cu mai multe grade de libertate folosind un limbaj de programare open-source. Bazele teoretice sunt explicate concis, în timp ce ecuațiile sunt rezolvate numeric cu Python folosind biblioteci științifice standard. Print intermediul analizei modale, s-au simulat și calculat frecvențele proprii în domeniul timp, iar rezultatele obținute demonstrează vizual distribuția energiei și efectele amortizării. Răspunsul combinat este vizualizat atât prin graficele clasice, cât și printr-o reprezentare 3D. Rezultatele obținute în această lucrare confirmă faptul că instrumentele numerice flexibile și gratuite pot fi aplicate cu succes în studii dinamice și oferă un exemplu de bună practică pentru educația inginerească și cercetarea aplicată.

Cuvinte cheie: *amortizare, valori proprii, analiză modală, simulare numerică, instrumente open-source.*

1. Introduction

Mechanical vibrations are present in almost every technical system, from microstructures to civil constructions and industrial equipment. Vibration analysis is vital to ensure the efficient and safe operation of installations and to avoid catastrophic failures caused by the resonance phenomenon [1].

The study of multi-degree-of-freedom (MDOF) systems is a mandatory step in the design of complex structures. In practice, engineers use both analytical and numerical methods to obtain the dynamic parameters [2].

The Python programming language [3], thanks to its open-source packages (NumPy, SciPy, MatPlotLib), is increasingly used as an alternative to commercial structural analysis programs, offering flexibility, transparency, and reproducibility [4].

2. Theoretical aspects

Systems that require two independent coordinates to describe their motion are called *two degree of freedom systems* (2DOF). One examples consider the system shown (Figure 1), show an un-dumped 2DOF system, with coordinates x_1 and x_2 measured from initial reference. Generally numerical results are easily obtained for 2DOF system. For system with MDOF matrix methods are essential. General equation of motion of an MDOF system with viscous damping and arbitrary excitation *F(t)* can be presented in following form Eq. (1):

$$\mathbf{M} \cdot \ddot{x}(t) + \boldsymbol{C} \cdot \dot{x}(t) + \boldsymbol{K} \cdot x(t) = \boldsymbol{F}(t) \tag{1}$$

where: $\mathbf{M} \in R^{n \times n}$ – mass matrix, $\mathbf{C} \in R^{n \times n}$ – damping matrix, $\mathbf{K} \in R^{n \times n}$ – stiffness matrix, x(t) – generalized displacement vector and **F**(t) – external force vector [5,6].

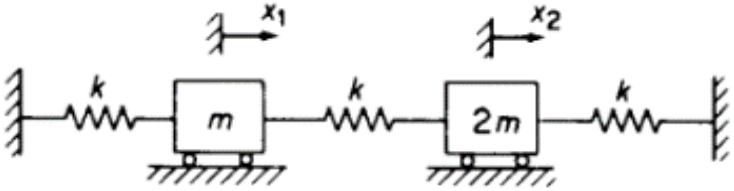

Figure 1. Two degree of freedom systems [6].

Solving it involves determining the eigenvalues and eigenvectors, which define the frequencies and eigenmodes of vibration. In this paper we will consider a system with 3 interconnected masses (Figure 2), to obtain an MDOF, thus Eq. 1 can be written in another way as Eq. 2:

$$\underbrace{\begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix}}_{\mathbf{M}} \ddot{\mathbf{x}} + \underbrace{\begin{bmatrix} c_1 + c_2 & -c_2 & 0 \\ -c_2 & c_2 + c_3 & -c_3 \\ 0 & -c_3 & c_3 \end{bmatrix}}_{\mathbf{C}} \dot{\mathbf{x}} + \underbrace{\begin{bmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix}}_{\mathbf{K}} \mathbf{x} = \mathbf{F}(t) \tag{2}$$

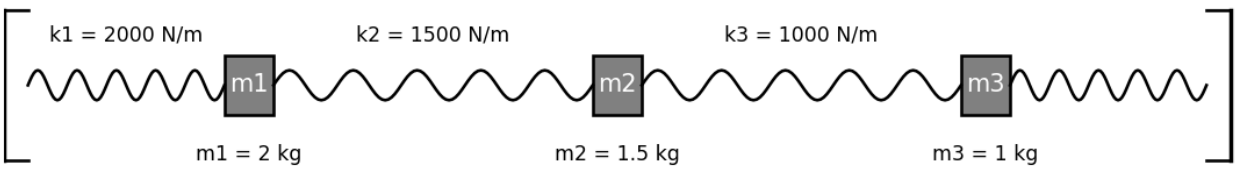

Figure 2. Scheme for MDOF systems used for numerical calculation.

The analyzed model consists of 3 masses connected to each other by springs and shock absorbers arranged linearly (Figure 2). This type of model can represent a discretized beam, a simplified vehicle suspension or an elementary kinematic chain.

In our case the input data used for numerical calculation is presented in Table 1.

Table 1

Parameters used for numerical calculation

Masses, kg	Stiffness, N/m	Damping, Ns/m
$m_1 = 2.0$	$k_1 = 2000$	$c_1 = 10$
$m_2 = 1.5$	$k_2 = 1500$	$c_2 = 8$
$m_3 = 1.0$	$k_3 = 1000$	$c_3 = 5$

Modal analysis is a fundamental method in vibration theory that allows engineers to understand and predict how a mechanical system behaves dynamically. A multi-degree-of-freedom (MDOF) system, such as a chain of connected masses and springs, can vibrate in several natural modes. Each mode is characterized by a specific natural frequency and a corresponding mode shape. The starting point is the general equation of motion for an undamped, free vibration system, thus Eq. 1 became:

$$\mathbf{M} \cdot \ddot{x}(t) + \boldsymbol{K} \cdot x(t) = \boldsymbol{F}(t) \tag{3}$$

The solution is assumed to be harmonic, like in Eq. 4:

$$x(t) = \boldsymbol{\Phi} \cdot \sin(\omega t) \tag{4}$$

Substituting solution Eq. 4 into the equation of motion Eq. 3 yields the **eigenvalue problem**

$$(\boldsymbol{K} - \omega^2 \cdot \boldsymbol{M}) \cdot \boldsymbol{\Phi} = 0 \tag{5}$$

This compact matrix equation means that only certain values of $\boldsymbol{\omega}$ (natural frequencies) and $\boldsymbol{\Phi}$ (mode shapes) satisfy the condition for free vibration.

Solving this generalized eigenvalue problem produces:

- Natural frequencies (ω) in radians per second, which tell us how fast each mode vibrates.

- Mode shapes (Φ), which describe how all masses move relative to each other in each mode.

Each mode is independent in a linear system, so the total response can be constructed as a superposition of these modes. This diagonalization (decoupling) simplifies the system of equations into a set of independent single-degree-of-freedom oscillators, which can then be analyzed or controlled separately. When damping is included, the modal analysis becomes more complex but still follows the same principle, assuming proportional or Rayleigh damping to preserve decoupling.

Modal analysis is critical in engineering because it helps prevent resonance by designing structures that do not operate near their natural frequencies, and it allows engineers to predict dynamic loads, fatigue, and noise [6,7].

3. Numerical calculation and visualization of results in Python (no dumping case)

First of all we need to install Python to PC [8], we will import in our code three library (Figure 3) ***NumPy***, ***SciPy*** and ***MatPlotLib***.

NumPy (*Numerical Python*) is the **fundamental library for numerical operations in Python**. In this project, it is used to:

- Define matrices and vectors (mass matrix **M**, stiffness matrix **K**);

- Perform **basic linear algebra operations** like matrix multiplication and solving systems;

- Handle arrays efficiently for numerical calculations [9,10].

SciPy (*Scientific Python*) builds on NumPy and provides **advanced scientific functions**. In this code, SciPy is used to:

- Solve the eigenvalue problem (scipy.linalg.eigh), which finds the natural frequencies and mode shapes of the system;

- Integrate ordinary differential equations (scipy.integrate.solve_ivp), which computes the time response of the coupled mass-spring system [11,12].

Matplotlib is Python's standard library for **plotting and data visualization**. In this code, Matplotlib is used to:

- Create **plots of the displacements** of each mass versus time;

- Display **clear, labeled graphs** to interpret the system's dynamic behavior;

- Enhance presentation of results with legends, titles, and grid lines.

```
1 import numpy as np
2 from scipy.linalg import eigh
3 from scipy.integrate import solve_ivp
4 import matplotlib.pyplot as plt
```

Figure 3. Coding start – import NumPy, SciPy and MatPlotLib library.

Next we enter in the code system parameters from Table 1 and define mass matrix and stiffness matrix (Figure 4) in steps # 1, # 2 and # 3.

```
5 # 1. Define system parameters
6 m1, m2, m3 = 2.0, 1.5, 1.0  # masses [kg]
7 k1, k2, k3 = 2000, 1500, 1000  # stiffness coefficients [N/m]
8 # 2. Build the mass matrix (diagonal)
9 M = np.diag([m1, m2, m3])
10 # 3. Build the stiffness matrix
11 K = np.array([
12     [k1 + k2, -k2, 0],
13     [-k2, k2 + k3, -k3],
14     [0, -k3, k3]])
```

Figure 4. Initial data for masses and stiffness. Matrix for mass and stiffness from Eq. 2.

In step # 4 we will solve eigenvalue problem that will help to find out natural frequencies (Figure 5).

```
15 # 4. Solve the eigenvalue problem
16 # (natural frequencies and mode shapes)
17 eigvals, eigvecs = eigh(K, M)
18 frequencies = np.sqrt(eigvals)
19 print("Natural frequencies [rad/s]:", frequencies)
```

Figure 5. Eigenvalue problem solution.

In step # 5 we will define motion equation for the simplified example without dumping (Figure 6).

```
# 5. Define the equation of motion
#(no damping in this simple example)
def motion(t, y):
    x = y[:3]  # displacements
    v = y[3:]  # velocities
    dxdt = v
    dvdt = np.linalg.solve(M, -K @ x)  # M * a = -K * x
    return np.hstack((dxdt, dvdt))
```

Figure 6. Simplified equation of motion (no dumping).

In the next two steps # 6 and # 7 we will define initial condition for displacement and will set time span for integration (Figure 7).

```
# 6. Initial conditions:
# small displacement for mass 1, rest zero
y0 = [0.1, 0.0, 0.0,   # initial displacements
      0.0, 0.0, 0.0]   # initial velocities
# 7. Time span and integration
t_span = (0, 10)
t_eval = np.linspace(0, 10, 1000)
solution = solve_ivp(motion, t_span, y0, t_eval=t_eval)
```

Figure 7. Displacement initial condition. Time span for integration.

In the last step # 8 we will write the code for data visualization to obtain plot result for masses displacements (Figure 8).

```
# 8. Plot results: displacements for each mass
x1, x2, x3 = solution.y[0], solution.y[1], solution.y[2]
plt.figure(figsize=(10, 5))
plt.plot(solution.t, x1, label='Mass 1')
plt.plot(solution.t, x2, label='Mass 2')
plt.plot(solution.t, x3, label='Mass 3')
plt.xlabel('Time [s]')
plt.ylabel('Displacement [m]')
plt.title('Time response of each mass')
plt.legend()
plt.grid(True)
plt.show()
```

Figure 8. Plot results for masses displacement.

Now we can save the code and run it. First we will obtain text response for natural frequencies in [rad/s] (Figure 9).

```
Natural frequencies [rad/s]: [16.15371556 37.16535635 52.67314712]
```

Figure 9. Numerical solution for masses natural frequencies.

After natural frequencies data we will obtain data visualization for tine response of each mass, otherwise displacement [m] of each mass related to time [s] (Figure 10). But this kind of response (visualization) is too hard to understand and distinguish the displacement for each mass, although each mass is drawn in a different color.

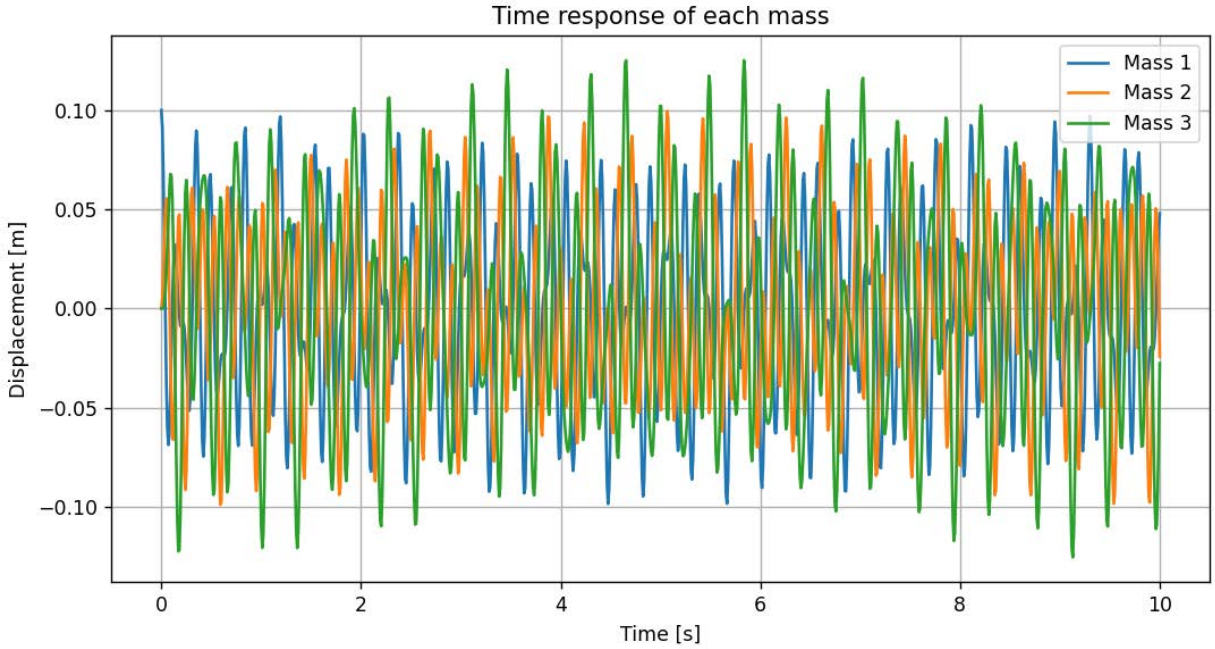

Figure 10. Data visualization – time response for Mass 1, Mass 2 and Mass 3.

Thus, Figure 10 illustrates the **time response** of a simple **3-degree-of-freedom mass–spring system** with the given masses and stiffness's. The plot shows how the **displacement** of each mass (x_1, x_2, x_3) varies with time when the system is **initially displaced** and then released.

This system has **three natural frequencies**, each producing an oscillatory mode. Because the masses are coupled through springs, their motions **interact** — they don't oscillate independently. So, the plot shows **how energy shifts between masses** in the coupled system.

Why Figure 10 is hard to read?

- The **amplitude is small**, but the oscillations are rapid — so lines overlap.
- The plot shows **all three masses** in one chart → lines cross each other.
- There is **no damping in this example**, so the oscillations do not decay — they stay at the same level forever, making the signal busy.

What Figure 10 tell us?

- The system vibrates with **multiple frequencies simultaneously**.
- The coupled behavior is realistic for structures with multiple parts.
- If we will apply damping (**C** matrix), the oscillations would **gradually decay**, and the plot would be clearer.
- The energy shifts from mass 1 → mass 2 → mass 3 and back.

Thus, Figure 10 presents the coupled dynamic response of all three masses in a free, undamped vibration scenario. The visible signal is a superposition of the three natural modes interacting with each other. The overlapping, non-decaying oscillations reflect the system's inherent modal behavior. In practice, engineers add damping to observe realistic decay and avoid infinite resonance. The figure illustrates how multi-degree-of-freedom systems redistribute vibrational energy among connected masses over time.

What we can do to make Figure 10 more readable (clearer?). First solution is to change the code to Show a **zoomed window**: e.g. first 2 seconds only. The new data visualization is shown in Figure 11.

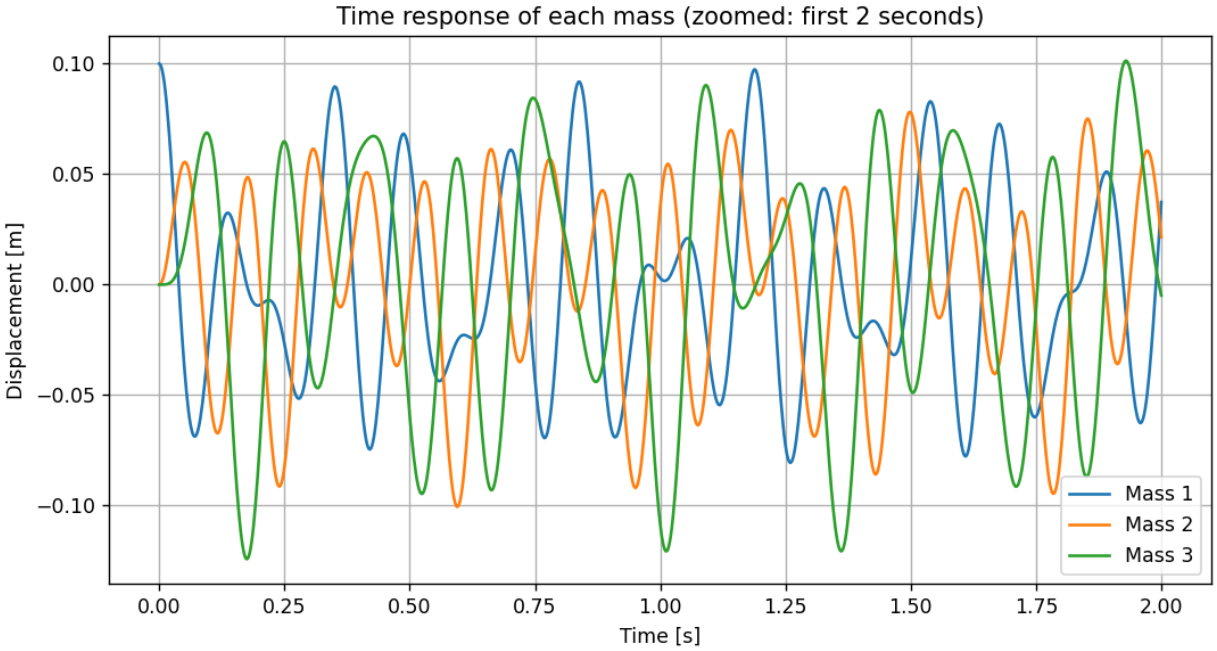

Figure 11. Data visualization – time response for each mass (zoomed: first 2 seconds).

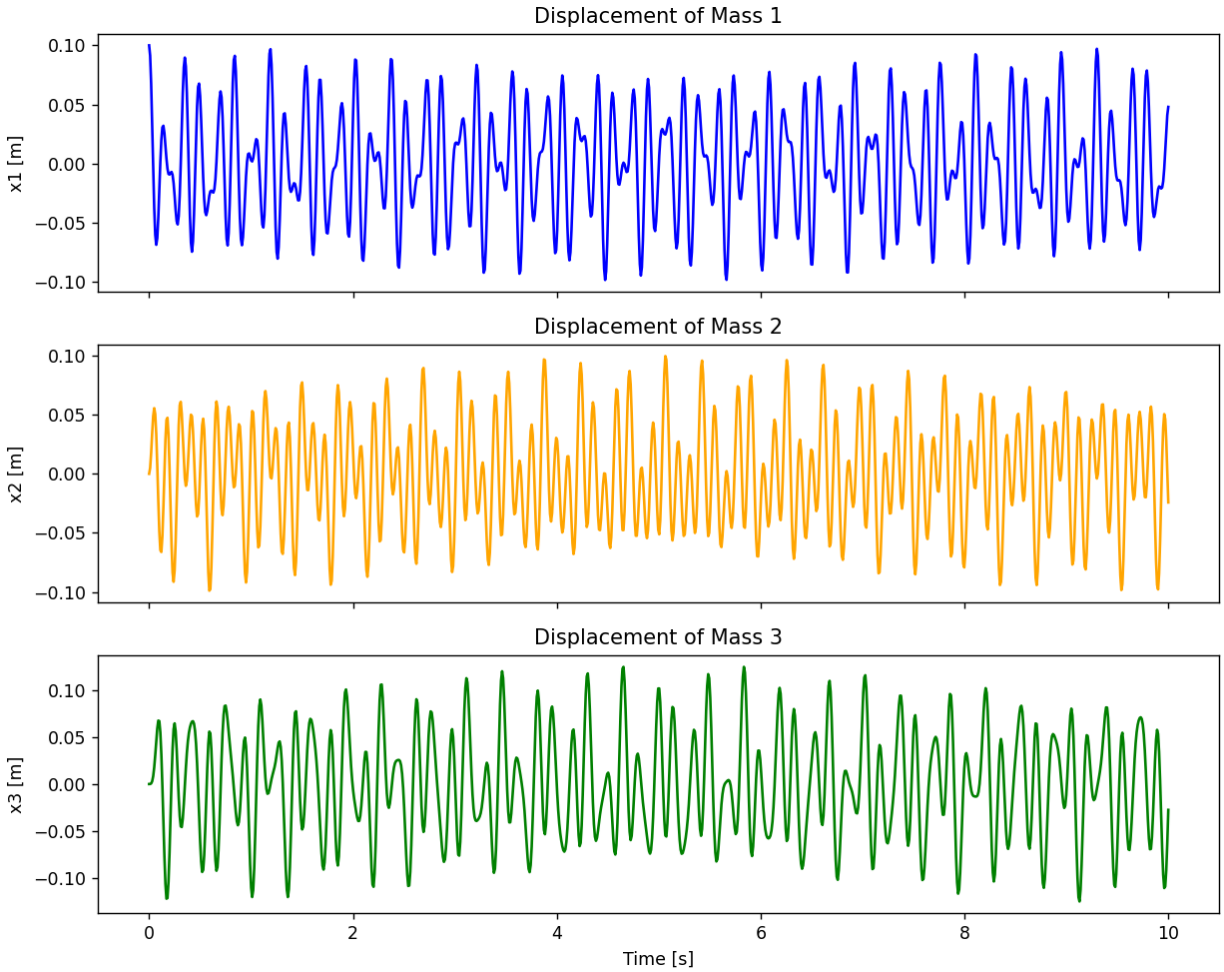

Figure 12. Separate subplots for each mass displacements.

Other solution to improve Figure 10 is to plot each displacement **in a separate subplot**. This solution is shown in Figure 12. We may encounter an inconvenience in this case (too large a figure takes up a lot of space).

The last solution to improve Figure 10 is Use a **Fourier transform (FTT)** to show which frequencies dominate. Thus, in Figure 13 it is show FTT for mass 3.

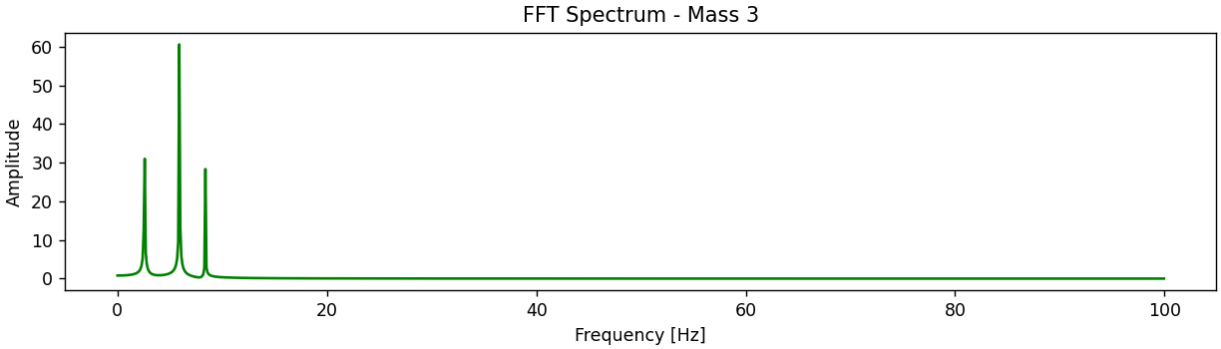

Figure 13. FTT Spectrum for mass 3.

Solution presented in Figure 13 dynamically solves the MDOF system for 10 seconds with higher resolution (2000 points), calculate the FTT for each displacement signal, Displays the amplitude spectrum for each mass (in our case mass 3), on the X axis you see the frequency in [Hz], on the Y axis the amplitude of the spectral mode in [m], the peaks correspond directly to the natural frequencies that we calculated previously by solving the eigenvalue problem.

4. Numerical calculation and visualization of results in Python (dumping case)

In this part we will take into account the damping effect. So we will modify a little the code shown in paragraph. First of all, we will add input values for damping coefficients and damping matrix into the code (Figure 14).

```
10 c1, c2, c3 = 10, 8, 5  # damping coefficients [Ns/m]
22 # 4. Damping matrix
23 C = np.array([
24     [c1 + c2, -c2, 0],
25     [-c2, c2 + c3, -c3],
26     [0, -c3, c3]])
```

Figure 14. Modified cod with damping coefficients values and damping matrix.

Also we have to modify motion equation to consider damping effect (Figure 15).

```
33 # 6. Equation of motion with damping
34 def motion(t, y):
35     x = y[:3]
36     v = y[3:]
37     dxdt = v
38     dvdt = np.linalg.solve(M, -C @ v - K @ x)
39     return np.hstack((dxdt, dvdt))
```

Figure 15. Equation of motion that take into account damping effect.

For visualization we also will modify the code to obtain individual displacement and combined response (Figure 16).

```
# 10. Plots: Individual displacements and stackplot
fig, axs = plt.subplots(2, 1, figsize=(10, 8))
# a) Individual displacements
axs[0].plot(sol.t, x1, label='Mass 1')
axs[0].plot(sol.t, x2, label='Mass 2')
axs[0].plot(sol.t, x3, label='Mass 3')
axs[0].set_title('Individual displacements')
axs[0].set_xlabel('Time [s]')
axs[0].set_ylabel('Displacement [m]')
axs[0].legend()
# b) Stackplot + combined response
axs[1].stackplot(sol.t, x1, x2, x3,\
                 labels=['Mass 1', 'Mass 2', 'Mass 3'])
axs[1].plot(sol.t, x_combined, 'k--', label='Combined response')
axs[1].set_title('Stackplot and combined response')
axs[1].set_xlabel('Time [s]')
axs[1].set_ylabel('Cumulative displacement [m]')
axs[1].legend()

plt.tight_layout()
plt.show()
```

Figure 16. Equation of motion that take into account damping effect.

The last modification into the cod is to add displacements to the space which will provide 3D trajectory (Figure 17).

```
# 11. 3D trajectory: Displacements in space
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.plot(x1, x2, x3, lw=1.5)
ax.set_title('3D trajectory in displacement space')
ax.set_xlabel('x1 [m]')
ax.set_ylabel('x2 [m]')
ax.set_zlabel('x3 [m]')
plt.show()
```

Figure 17. Code for 3D plot.

After these modifications we can run the code and will obtain first two plot individual displacement and individual response (Figure 18).

Figure 18 illustrates the **time-domain response** of a three-degree-of-freedom (MDOF) mass–spring–damper system subjected to an initial displacement (in our case 3DOF). The system includes damping, which causes the oscillations to gradually decay.

The top subplot shows the displacement response of each mass (Mass 1, Mass 2, and Mass 3) separately over the full time range of 10 seconds:

- at **time zero**, only Mass 1 is given an initial displacement of 0.1 m, while Mass 2 and Mass 3 start from rest;

- due to the **mechanical coupling** (springs) between the masses, the initial energy of Mass 1 is transferred to Mass 2 and Mass 3;

- All three masses oscillate at the system's **natural modes**, and the displacements clearly **exchange energy**;

- The damping matrix causes the amplitudes to **decay exponentially** over time. You can see that after about 6–8 seconds the system reaches near rest;

- The curves show typical **phase differences** due to coupling: the masses do not oscillate in perfect synchronization.

The bottom subplot provides a different perspective:

The **stackplot** displays how the individual displacements of Mass 1, Mass 2, and Mass 3 **contribute cumulatively** to the total system response. Each colored area represents the absolute displacement for that mass at each instant.

- The **black dashed line** represents the **combined response**, calculated as the sum of the three displacements. This total response shows how the **global motion evolves** as energy shifts through the structure.

Thus, Figure 18 provides a clear, visual confirmation that the three-mass system responds dynamically with coupled oscillations that gradually die out due to damping. The individual plots reveal how each mass oscillates in time, while the stack plot illustrates how the energy is shared and absorbed within the system as a whole. This is a realistic, practical example of modal interaction in multi-degree-of-freedom mechanical systems. This behavior is typical for real mechanical structures: when one part vibrates, neighboring parts are also affected and engineers must understand this to design systems that **avoid resonance** and minimize unwanted vibrations.

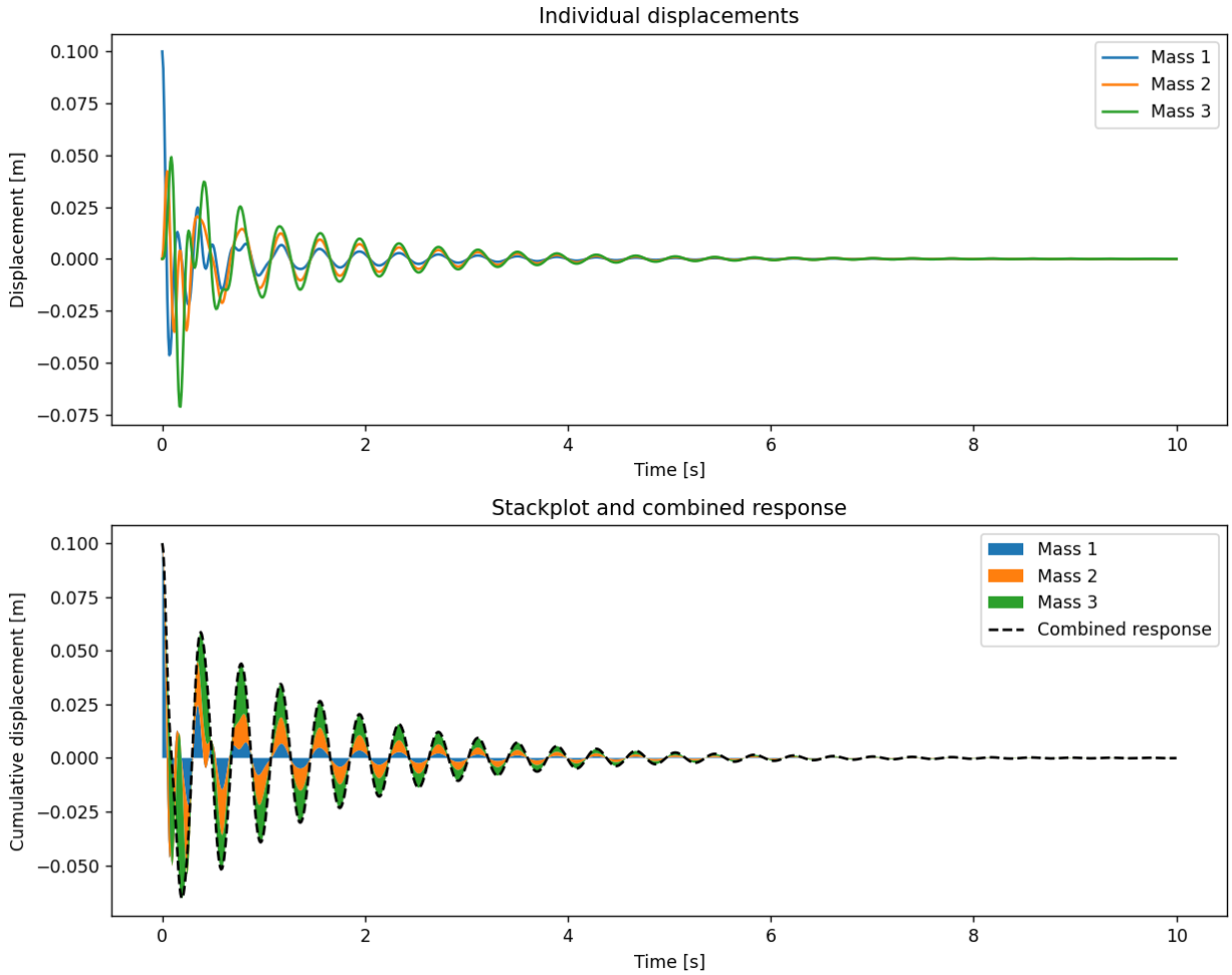

Figure 18. Visualization of individual displacements and combined response.

The last plot shows us 3D displacements in space (Figure 19) [15].

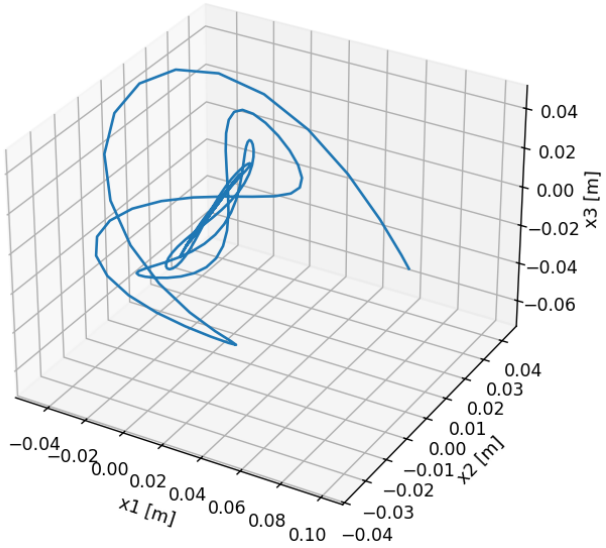

Figure 19. 3D trajectory visualization of displacements in space.

The 3D trajectory plot (Figure 19) provides a compact, clear visualization of how all three masses in the multi-degree-of-freedom system interact dynamically and gradually come to rest due to damping. It visually confirms the modal behavior, phase coupling, and energy dissipation of the structure in a single figure.

5. Discussion

The results of this study confirm that the numerical approach used for solving (MDOF) vibration problems is effective and reliable. The time-domain responses and the frequency domain analyses demonstrate that the system's behavior matches the theoretical modal predictions, with energy being transferred between connected masses through the elastic and damping elements.

These findings support the working hypothesis that open-source numerical tools such as Python can accurately replicate classical analytical solutions for vibration problems. Compared to previous studies and textbook examples, the results align well with established vibration theory and modal analysis principles.

Moreover, the practical implementation of diagonalization, eigenvalue computation, time integration, and visualization show that students and engineers can easily adapt these techniques for more complex cases. An important implication is that the same numerical methods form the basis of the **Finite Element Method (FEM)**, as detailed by **Zienkiewicz & Taylor [16].** The MDOF approach can therefore be expanded to discretize beams, plates, and 3D structures, allowing the study of realistic engineering problems with many degrees of freedom.

However, this study is limited to a simple linear model with lumped masses, linear springs, and viscous damping. Real systems may include nonlinearities, more complex boundary conditions, or external excitations that require additional modeling techniques. Future research could extend this work by implementing forced vibrations, variable damping, or modal testing on physical prototypes. Extending the model to full FEM frameworks would also allow the analysis of large-scale structures and detailed stress distributions.

Recent studies [17,18], focus on solving multi-degree-of-freedom vibration problems using commercial tools like MATLAB and Simulink, emphasizing user-friendly interfaces but limited flexibility in algorithm customization. Unlike these approaches, the present study utilizes Python programming, which allows direct access to algorithm development, transparent matrix manipulation, and customized numerical integration. Furthermore, Python offers the advantage of being open-source, highly adaptable, and supported by an extensive community, making it ideal for both academic and industrial applications without licensing constraints.

The scientific novelty of this work lies in combining classical vibration theory with an open programming approach in Python, enhanced by clear visualization methods, including 3D trajectory representation and frequency spectrum analysis.

Compared to [17], who mainly address linear system responses in MATLAB, this research provides an innovative perspective by demonstrating how Python can serve as a versatile tool for vibration analysis and educational purposes, enabling engineers to expand models easily towards complex structures and larger systems.

6. Conclusions

This research demonstrates MDOF vibration problem can be solved efficiently using Python's scientific computing libraries. The numerical solution accurately matches theoretical expectations, showing clear energy transfer between masses, modal interaction, and damping effects. The visualization tools, including time plots, frequency spectra, and 3D trajectory diagrams, provide clear insights into the dynamic response of the system. Importantly, the study highlights that the same numerical principles used here form the core of the FEM, which enables engineers to analyze complex structures in practical applications. Overall, the approach presented is flexible, accessible, and highly suitable for both educational and research purposes in the field of mechanical vibrations. The scientific contribution of this work consists in integrating open-source programming into the classical analysis of MDOF vibration systems. By leveraging Python's scientific libraries, the study demonstrates that engineers can bypass commercial software limitations and directly implement advanced analysis methods, including modal decomposition, eigenvalue problem solving, and 3D visualization, making the research both innovative and educationally significant.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Malcoci, Iu.; Ciobanu, O.; Ciobanu. R. *Coding in Python for Mathematics, Science and Engineering. Acta Technica Napocensis Journal*: *Applied Mathematics, Mechanics and Engineering* 2022, 65(4S), pp. 1215-1220. https://atna-mam.utcluj.ro/index.php/Acta/article/view/2050
2. Inman, D. J. *Engineering Vibration*. Pearson, Boston, USA, pp. 220-250.
3. Van Rossum, G.; Drake Jr., F.L. *Python 3 Reference Manual*. CreateSpace Independent Publishing Platform, Charleston, USA, 2009, 130 p.
4. Negru, I. *Kinematic analysis of the rod-crank mechanism. classical and modern calculation methods (numerical calculation in the python programming language)*. in: technical scientific conference of undergraduate, master and phd students, 27-29 march 2024, Technical University of Moldova, Chisinau, Republic of Moldova, 2024, 2, pp. 1268-1273 [in romanian].
5. Rao, S.S. *Mechanical vibrations*. Pearson Prentice Hall, New Jersey, USA, 2010, pp. 230-300.
6. Thomson, W.H.; Dahlen, M.D.; Padmanabhan, C. *Theory of VIBRATION with Application*. Perason, New Delhi, India, 2008, pp. 133-170.
7. Clough, R.W.; Penzien, J. *Dynamics of Structures*. Computers & Structures, Inc., Berkeley, California, USA, 2003, 730 p.
8. *Python Installation*. Available online: https://www.python.org/downloads/ (accessed on 13 February 2025).

9. *NumPy Library Documetnation*. Available online: https://numpy.org/doc/ (accessed on 01 July 2025).
10. Harris, C.R.; Millman, K.J.; van der Walt, S.J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N.J.; Kern, R.; Picus, M.; Hoyer, S.; van Kerkwijk, M. H.; Brett, M.; Haldane, A.; Fernández del Río, J.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; Oliphant, T.E. *Array programming with NumPy*. *Nature* 2020, 585, pp. 357–362.
11. *SciPy Library Documentation*. Available online: https://docs.scipy.org/doc/scipy/ (accessed on 03 July 2025).
12. Virtanen, P.; Gommers, R.; Oliphant, T.E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A.R.J.; Jones, E.; Kern, R.; Larson, E.; Carey, C.J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E.A.; Harris, C. R.; Archibald, A.M.; Ribeiro, A.H.; Pedregosa, F.; van Mulbregt, P. *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*. Nature Methods 2020, 17, pp. 261–272.
13. *MatPlotLib Tutorial*. Available online: https://matplotlib.org/stable/tutorials/index.html (accessed on 03 July 2025).
14. Hunter, J. D. *Matplotlib: A 2D Graphics Environment*. *Computing in Science & Engineering* 2007, 9(3), pp. 90–95.
15. *MatPlotLib 3D Plotting*. Available online: https://matplotlib.org/stable/gallery/mplot3d/lines3d.html#sphx-glr-gallery-mplot3d-lines3d-py (accessed on 03 July 2025).
16. Zienkiewicz, O.C.; Taylor, R.L.; Zhu, J.Z. *The Finite Element Method: Its Basis and Fundamentals*. Elsevier Ltd., Oxford, UK, 2013, 756 p.
17. Öksüz, M. *Vibration Analysis: Validation of the Mathematical Model and the Physical Simulink Multibody Model*. Journal of Mathematics, Engineering, Natural & Medical Sciences, EUROASIA 2025, 12(1), pp. 15-26.
18. Edris, Z.F.; Elswie. H.I. Optimized MDOF Vibration Isolation System Design by MATLAB. Sebha University Conference Proceedings 2025, 4(1), pp. 18–22.

Citation: Malcoci, Iu. Vibration analysis of a mechanical system with multiple degrees of freedom using Python. *Journal of Engineering Science*. 2025, XXXII (3), pp. 7-19. https://doi.org/10.52326/jes.utm.2025.32(3).01.

Submission of manuscripts: jes@meridian.utm.md