Project SRE: Part 1 - Seismic Input and Random Vibrations

Students' Names: Menno Marissen, Gabriele Mylonopoulos, Bart Slingerland and Jerin Thomas

Student ID number (choose any of your group members):

A	*B*	*C*	*D*	*E*	*F*	*G*
5	3	8	1	8	2	7

INTRODUCTION

In Unit 1 of the B2 Module and during the random vibrations lectures, you have seen how we can obtain the Power Spectral Density (PSD) of a random excitation, and the PSD of the response of a structure due to the random excitation. You have also seen how this can be used to assess limit states of a structure. During the SRE lectures, we analyzed seismic motions and we have seen how the ground motions are influenced by the presence of soft soil layers. We also discussed the resonances of a multi-layered soil on top of a bedrock and the one-dimensional (1D and 2D) SH-wave propagation in a horizontally stratified soil column.

In this assignment, you have the possibility to work on these concepts and to explain the various physical phenomena associated with these effects.

LEARNING OUTCOMES

The learning outcomes of this assignment are that you will be able to:

1. Derive dynamic amplification factors for a layered soil medium based on 1D SH-wave propagation;
2. Apply the theory of random vibrations in dynamic analysis;
3. Consider additional loads together with the seismic action.

INSTRUCTIONS & QUESTIONS TO BE ANSWERED

Consider the dynamics of the offshore wind turbine (OWT) shown in Figure 1 assuming that it vibrates in the plane of the figure. The OWT is founded upon a multi-layered soil medium and excited dynamically by seismic motion at the base of that medium and sea waves along its height. You may assume that the diameters of the circular cross section are constant over its height. You are asked to analyse the multi-layered soil medium and assess the OWT response using random vibrations theory.

The list of properties of the dynamical model under consideration are as follows (note that some of the variables given are based on your (student) ID number):

- $M = 8F0 \cdot 10^3$ kg
- $J = 300 \cdot 10^6$ kg m^2
- $\rho_t = 7850$ kg/m^3
- $E_t = 210$ GPa
- $D_{out} = 9.1$ m
- $D_{inner} = 8.96$ m
- $H_t = 140 + A$ m
- $H_w = 30 + F$ m

- $H_1 = 2 + B$ m
- $H_2 = 58 + G$ m
- $H_3 = 38 + F$ m
- $G_1 = 1F0$ MPa
- $\rho_1 = 1600$ kg/m^3
- $\eta_1 = 10C0$ Ns/m^2
- $G_2 = 1D0$ MPa
- $\rho_2 = 1900$ kg/m^3
- $\eta_2 = 10E0$ Ns/m^2
- $G_3 = 1E0$ MPa
- $\rho_3 = 2000$ kg/m^3
- $\eta_3 = 10D0$ Ns/m^2

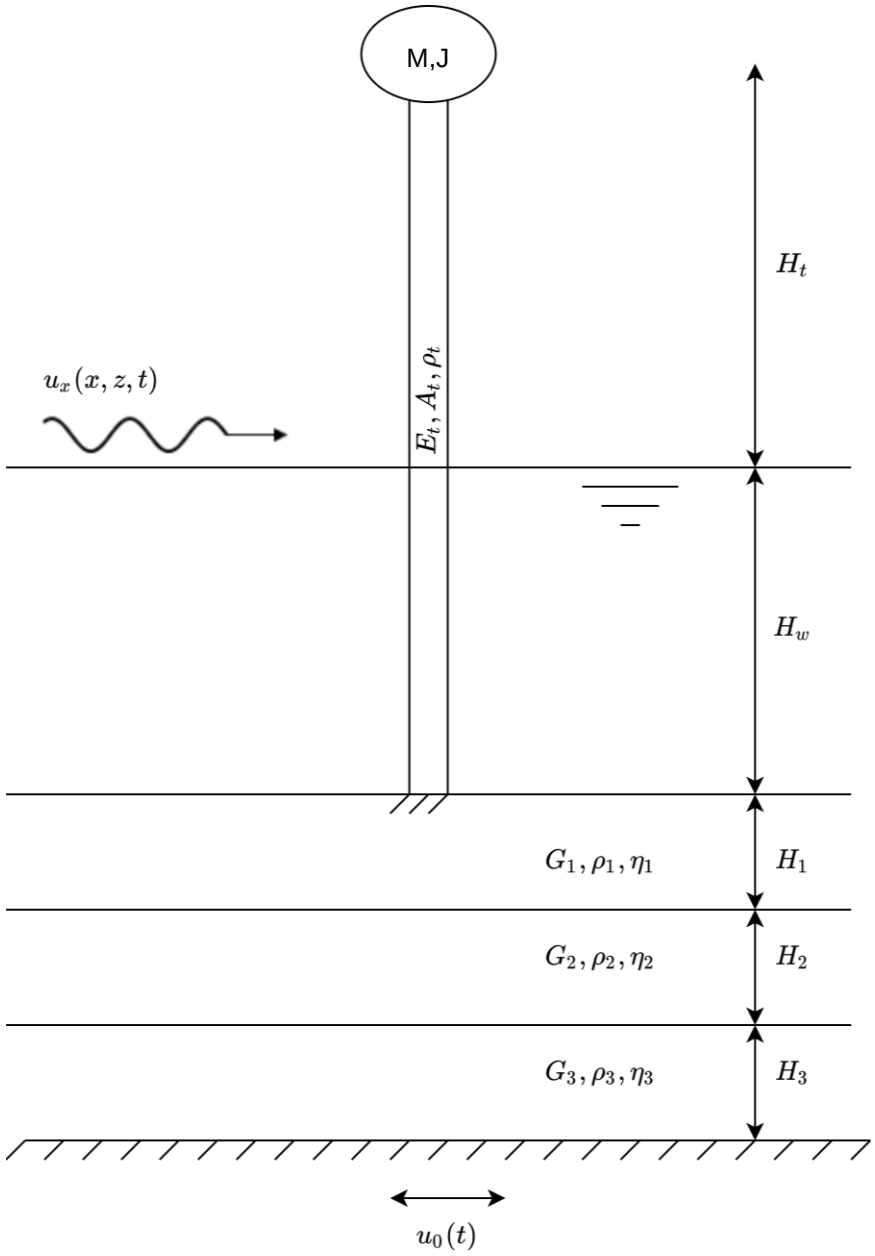


Figure 1: Simplified model of offshore wind turbine (OWT) at sea

Please answer the following questions:

1. Formulate the set of governing equations, i.e. equations of motion, boundary and interface conditions, for *the layered soil system* for a given input motion at the bedrock level assuming one-dimensional SH-wave propagation. You can ignore the presence of the OWT in your formulation.

2. Derive the natural frequencies and modes of vibration of the multi-layered soil medium of Figure 1. Plot the first 10 natural modes of vibration of the layered soil system. In your analysis, you may neglect material damping. In case you would like to include soil damping what effect will that have on the natural frequencies and the modes of vibration?

3. Derive the Frequency Response Functions $H(z,\omega)$ of the layered soil system. Plot the amplitude and the phase of the $H(z,\omega)$ at the top of layer 3 and the top of layer 1 and explain the differences you observe in the two cases. What is the physical meaning of the phase in the derived $H(z,\omega)$?

4. Formulate the set of governing equations, i.e. the equations of motion, boundary and interface conditions, of the OWT. In your formulation, please:
- Assume that the base of the wind turbine is rigidly connected to the top of the soil, as depicted in Figure 1.
- Neglect dynamic soil-structure interaction (SSI), i.e. assume that the motion at the surface of the ground derived by means of the analysis in question 3 is your kinematic excitation at the base of the OWT.
- Consider the wave forces acting on the turbine by using the Morrison's equation. You may neglect the drag force.

5. Using as input the Power Spectral Densities (PSDs) of the wave forces and the seismic excitation <u>at the bedrock level</u>[1] given below, and by further assuming that the PSDs of these two dynamic loads are uncorrelated, please answer the following questions:

5.1. Compute the variance and mean value of the acceleration at the top of the OWT and of the bending moment at the base of the OWT.

5.2. Derive the probability of exceedance of 80% of the yield limit stress (S335: $\sigma_y = 355\ \mathrm{N/mm^2}$) of the OWT at the base and the probability of exceedance of the acceleration value of 0.2[g] at the top of the OWT. You may assume that the Probability Density Function (PDF) of the structural response is the normal distribution.

5.3. Please explain whether you think that the assumption of stationarity and ergodicity in the case of the seismic motions in Groningen is appropriate. To answer this question you are free to analyse the set of recorded ground motions from seismic events in Groningen discussed in class.

5.4. If one would additionally need to consider the seismic excitation in the vertical direction acting at the base of the OWT, could one assume that the horizontal and vertical PSDs of the seismic motion are uncorrelated in the case of the Groningen earthquakes? Please substantiate your answer.

[1] The PSD of the ground motion was calculated based on the assumption of wide-sense stationarity (WSS) and ergodicity.

Input PSDs of the wave force and of the seismic motion:

- For the wave force please use the JONSWAP spectrum, as explained in unit 2 (see also formula sheet for the equations). For the parameters, use the ones provided to you in the assignment of unit 2 for load case 2.

- For the seismic motion at bedrock level, please use the PSD as given in notebook: 'PSD.ipynb'. Please use the principal horizontal component as seismic input.

RESOURCES

For this assignment you are free to use the *Maple* software which can be downloaded from the TU Delft software page. The Python package *PyDynSM* can also be used especially for derivation of the FRFs of the OWT or for the SH-wave propagation in the layered system. If you do so, make sure to install the latest version.

Tips for use of PyDynSM:

- the soil can be modelled as '*Shear Beam*'.
- The beam can be modelled as '*EulerBernoulli Beam foundation attachments*'

PRODUCTS

The final product of this assignment is a report which you submit before the deadline mentioned below. The report will count for **40%** of the final grade of this part of the module.

ASSESSMENT CRITERIA

The length of the report is irrelevant for your final mark; the grade is based solely on the content of the report. The weight of each part is specified below:

Question Nr.	**Points (0-5)**	**Percentile contribution to final score (%)**	**Final score per question**
1		15	
2		15	
3		15	
4		15	
5		40	
Sum			
Grade (1-10)			

SUPERVISION AND HELP

Feedback to the assignment will be given in two specific time moments:

- **Feedback on work in progress**: Every week, a student assistant will be available to provide feedback on the progress and help you with the assignment. Please check lecture slides for the days of the consultancy sessions.
- **Feedback on the final report**: You will receive feedback when this report will be graded on Brightspace. You will have the opportunity to discuss your mark and your questions after the announcement of the grade.

SUBMISSION FORMAT

Submission deadline: June 20, 2025 at 23:59 o'clock.

Your report should be submitted electronically as a single pdf-file and the file name should be of the following form:
"CIEM5220_SRE_PW1_2025_[Your Names+Your student ID numbers]".
For example:
CIEM5220_SRE_PW1_2025_*ATsouvalas1234567_PDitmar7654321.pdf*

Please submit this assignment specification form together with your final report as a single pdf-file **(please compile all documents into a single pdf-file)** directly on Brightspace under the correspondent folder allocated for assignments. Do not forget to fill in your names and Student ID numbers at the front page of this assignment specification form. Clearly indicate the ID number used for this assignment.

The solution must be either handwritten or typed (word, LateX, etc.) and then converted to pdf-format and submitted as defined above.

Project SRE: Part 1 - Seismic Input and Random Vibrations

- Names: Menno Marissen, Gabriele Mylonopoulos, Bart Slingerland and Jerin Thomas
- Student numbers: 5381827, 4807421, 5309913 and 6276423

- Date: 23 June 2025
- Course: Applied Dynamics of Structures - Structural Response to Earthquakes
- Course code: CIEM5220-U1

Exercise 1: Governing equations of the layered soil system

 No description has been provided for this image

The soil system consists of three layers with indices 1, 2 and 3 in downward direction. The bottom of the third layer is excited by an earthquake that occurs in the bedrock level below. The presence of the offshore wind turbine is disregarded in this question.

Equations of motion

The equations of motion for all three layers with $m \in (1, 2, 3)$ are:

$$\rho_m \frac{\partial^2 u_m(z_m, t)}{\partial t^2} - G_m \frac{\partial^2 u_m(z_m, t)}{\partial z^2} - \eta_m \frac{\partial^3 u_m(z_m, t)}{\partial z^2 \partial t} = 0$$

in which ρ_m is the density of the layer [kg/m^3], $u_m(z_m, t)$ is the horizontal displacement [m] in the local coordinate system $0 < z_m < H_m$ [m], t is the time [s], G_m is the shear modulus of the layer [N/m^2], and η_m is the damping coefficient [Ns/m^2].

The shear stress is given by $\tau_m(z_m, t) = G_m \gamma_m + \eta_m \frac{\partial \gamma_m}{\partial t}$ with $\gamma_m = \frac{\partial u_m}{\partial z}$.

Boundary conditions

The equations of motion are second-order in space, which means that there should be one boundary condition at the soil surface and one at the base of the multi-layered soil medium.

At the soil surface ($z_1 = 0$), the shear stress is zero:

$$\tau_1(z_1 = 0, t) = G_1 \frac{\partial u_1}{\partial z} + \eta_1 \frac{\partial^2 u_1}{\partial z \partial t} = 0$$

At the base, the ground displacement should match the motion of the base:

$$u_3(z_3 = H_3, t) = u_0(t)$$

Interface conditions

Following from the fact that the equations of motion are second-order in space, there should be two interface conditions at each interface. One relates the displacements to each other, while the second one takes care of shear stress equilibrium. Both conditions hold for $m \in (1, 2)$.

$$u_m(z_m = H_m, t) = u_{m+1}(z_{m+1} = 0, t)$$
$$\tau_m(z_m = H_m, t) = \tau_{m+1}(z_{m+1} = 0, t)$$

Exercise 2: Natural frequencies and modes of vibration

For the system described in Exercise 1, natural frequencies and modes of vibration can be found.

Converting to the frequency domain

Material damping may be neglected, so $\eta_m = 0; m \in (1, 2, 3)$. The equations of motion become (using the previously mentioned values for the index m):

$$\rho_m \frac{\partial^2 u_m(z_m, t)}{\partial t^2} - G_m \frac{\partial^2 u_m(z_m, t)}{\partial z^2} = 0$$

with $\tau_m(z_m, t) = G_m \frac{\partial u_m}{\partial z}$. A general solution is expected to be in the form of:

$$u_m(z_m, t) = \tilde{U}_m(z_m, \omega) e^{i\omega t}$$

Substitution of this into the equation of motion converts the problem to the frequency domain and yields:

$$\left(\omega^2 \rho_m \tilde{U}_m(z_m, \omega) + G_m \frac{d^2 \tilde{U}_m(z_m, \omega)}{dz^2} \right) e^{i\omega t} = 0$$

$$\frac{d^2 \tilde{U}_m(z_m, \omega)}{dz^2} + \frac{\omega^2 \rho_m}{G_m} \tilde{U}_m(z_m, \omega) = 0$$

$$\frac{d^2 \tilde{U}_m(z_m, \omega)}{dz^2} + k_m^2 \tilde{U}_m(z_m, \omega) = 0$$

with $k_m^2 = \frac{\omega^2 \rho_m}{G_m} = \frac{\omega^2}{c_m^2}$ with $c_m = \sqrt{\frac{G_m}{\rho_m}}$ and thus $k_m = \pm \frac{\omega}{c_m}$.

A general solution of this new equation is expected to be in the form of:

$$\tilde{U}_m(z_m, \omega) = \tilde{A}_{m,1} e^{i k_m z_m} + \tilde{A}_{m,2} e^{-i k_m z_m}$$

with $k_m = \frac{\omega}{c_m}$, so the negative solution is written as $-k_m$. Only $\tilde{A}_{m,1}$ and $\tilde{A}_{m,2}$ remain unknown for this second-order problem, which can thus be solved for using the two boundary and/or interface conditions per layer.

Applying boundary conditions

The upper boundary condition:

$$\tau_1(z_1 = 0, t) = G_1 \frac{\partial u_1}{\partial z} = 0$$

$$G_1 i k_1 (\tilde{A}_{1,1} e^{i k_1 0} - \tilde{A}_{1,2} e^{-i k_1 0}) = 0$$

$$G_1 i k_1 (\tilde{A}_{1,1} - \tilde{A}_{1,2}) = 0$$

$$\tilde{A}_{1,1} = \tilde{A}_{1,2}$$

The lower boundary condition:

$$u_3(z_3 = H_3, t) = u_0(t)$$

$$\left(\tilde{A}_{3,1} e^{i k_3 H_3} + \tilde{A}_{3,2} e^{-i k_3 H_3}\right) e^{i\omega t} = u_0(t)$$

Applying interface conditions

Equilibrium of displacements:

$$u_m(z_m = H_m, t) = u_{m+1}(z_{m+1} = 0, t)$$

$$\tilde{U}_m(H_m, \omega) e^{i\omega t} = \tilde{U}_{m+1}(0, \omega) e^{i\omega t}$$

$$\tilde{U}_m(H_m, \omega) = \tilde{U}_{m+1}(0, \omega)$$

$$\tilde{A}_{m,1} e^{i k_m H_m} + \tilde{A}_{m,2} e^{-i k_m H_m} = \tilde{A}_{m+1,1} e^{i k_{m+1} 0} + \tilde{A}_{m+1,2} e^{-i k_{m+1} 0}$$

$$\tilde{A}_{m,1} e^{i k_m H_m} + \tilde{A}_{m,2} e^{-i k_m H_m} = \tilde{A}_{m+1,1} + \tilde{A}_{m+1,2}$$

Equilibrium of shear stresses:

$$\tau_m(z_m = H_m, t) = \tau_{m+1}(z_{m+1} = 0, t)$$

$$G_m \frac{\partial u_m}{\partial z}\Big|_{z_m = H_m} = G_{m+1} \frac{\partial u_{m+1}}{\partial z}\Big|_{z_{m+1} = 0}$$

$$G_m \frac{d\tilde{U}_m(z_m, \omega)}{dz} e^{i\omega t}\Big|_{z_m = H_m} = G_{m+1} \frac{d\tilde{U}_{m+1}(z_{m+1}, \omega)}{dz} e^{i\omega t}\Big|_{z_{m+1} = 0}$$

$$G_m i k_m \left(\tilde{A}_{m,1} e^{i k_m H_m} - \tilde{A}_{m,2} e^{-i k_m H_m}\right) = G_{m+1} i k_{m+1} \left(\tilde{A}_{m+1,1} e^{i k_{m+1} 0} - \tilde{A}_{m+1,2} e^{-i k_{m+1} 0}\right)$$

$$G_m i k_m \left(\tilde{A}_{m,1} e^{i k_m H_m} - \tilde{A}_{m,2} e^{-i k_m H_m}\right) = G_{m+1} i k_{m+1} \left(\tilde{A}_{m+1,1} - \tilde{A}_{m+1,2}\right)$$

$$\tilde{A}_{m+1,1} - \tilde{A}_{m+1,2} = a_m \left(\tilde{A}_{m,1} e^{i k_m H_m} - \tilde{A}_{m,2} e^{-i k_m H_m}\right)$$

with $a_m = \frac{G_m k_m}{G_{m+1} k_{m+1}}$.

Combining everything to obtain the solution

Solving the two final equations from the interface conditions for the unknown deeper-layer coefficients $\tilde{A}_{m+1,1}$ and $\tilde{A}_{m+1,2}$ yields:

$$\tilde{A}_{m+1,1} = \frac{1}{2}\tilde{A}_{m,1}(1+a_m)e^{ik_mH_m} + \frac{1}{2}\tilde{A}_{m,2}(1-a_m)e^{-ik_mH_m}$$

$$\tilde{A}_{m+1,2} = \frac{1}{2}\tilde{A}_{m,1}(1-a_m)e^{ik_mH_m} + \frac{1}{2}\tilde{A}_{m,2}(1+a_m)e^{-ik_mH_m}$$

which can be combined with the two previously found boundary conditions:

$$\tilde{A}_{1,1} = \tilde{A}_{1,2}$$

$$\left(\tilde{A}_{3,1}e^{ik_3H_3} + \tilde{A}_{3,2}e^{-ik_3H_3}\right)e^{i\omega t} = u_0(t)$$

and solved for, to finally be substituted to obtain the final solution:

$$u_m(z_m,t) = \tilde{U}_m(z_m,\omega)e^{i\omega t} = \left(\tilde{A}_{m,1}e^{ik_mz_m} + \tilde{A}_{m,2}e^{-ik_mz_m}\right)e^{i\omega t}$$

Implementation in PyDynSM

The layered soil structure can also be modeled using the PyDynSM Python package. This is done below, using shear beams as the element type of the soil layers.

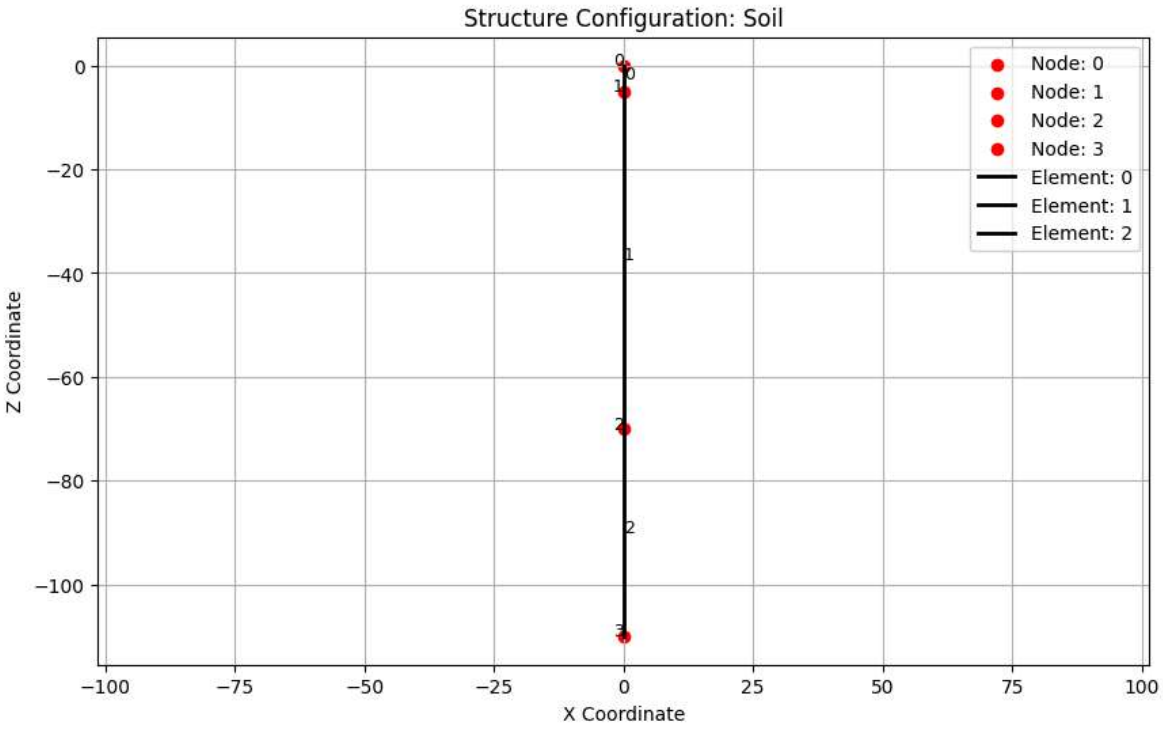
In [1]:
```
#Importing packages

import matplotlib.pyplot as plt
import numpy as np
import pydynsm as PDM
import scipy.optimize as opt
from scipy.fft import rfft, rfftfreq
from scipy.optimize import fsolve
from scipy.signal import butter, correlate, filtfilt
from scipy.stats import norm
```

In [2]:
```
#Defining parameters

student_number = 5381827
student_number = list(str(student_number))
for i in range(len(student_number)):
    student_number[i] = int(student_number[i])
A, B, C, D, E, F, G = student_number

M = (800 + F * 10) * 1E3 #kg
J = 300E6 #kgm²
rho_t = 7850 #kg/m³
E_t = 210E9 #N/m²
D_out = 9.1 #m
D_inner = 8.96 #m
H_t = 140 + A #m
H_w = 30 + F #m
H_1 = 2 + B #m
H_2 = 58 + G #m
H_3 = 38 + F #m
G_1 = (100 + F * 10) * 1E6 #N/m²
rho_1 = 1600 #kg/m³
```

```
eta_1 = 1000 + C * 10 #Ns/m²
G_2 = (100 + D * 10) * 1E6 #N/m²
rho_2 = 1900 #kg/m³
eta_2 = 1000 + E * 10 #Ns/m²
G_3 = (100 + E * 10) * 1E6 #N/m²
rho_3 = 2000 #kg/m³
eta_3 = 1000 + D * 10 #Ns/m²

f_max1 = 200 #Hz
omega_range1 = np.linspace(1, f_max1 * 2 * np.pi, 1000)
```

In [3]:
```
#Creating the layered soil system

#Instantiating class
s1 = PDM.Assembler('Soil')

#Creating nodes
nodes1 = []
H_list1 = [0, -H_1, -(H_1 + H_2), -(H_1 + H_2 + H_3)]

for H in H_list1:
    node = s1.CreateNode(0, H, dof_config=['x'])
    nodes1.append(node)

nodes1[-1].fix_node('x') #x is fixed, because it should be unforced (so no earth

#Creating elements
elems1 = []

rho = [rho_1, rho_2, rho_3]
A_s = 1 #A is divided out of the EoM
G_list = [G_1, G_2, G_3]

for i in range(len(nodes1) - 1):
    elem = s1.CreateElement([nodes1[i], nodes1[i + 1]])
    soil_section_no_damping = {'rho': rho[i], 'A': A_s, 'G': G_list[i]} #no damp
    elem.SetSection('Shear Beam', soil_section_no_damping)
    elems1.append(elem)

s1.PlotStructure(plot_elements=True)
```

```
Assembler 'Soil' successfully initialised
Successfully added element of type: Shear Beam to Element 0
Successfully added element of type: Shear Beam to Element 1
Successfully added element of type: Shear Beam to Element 2
```

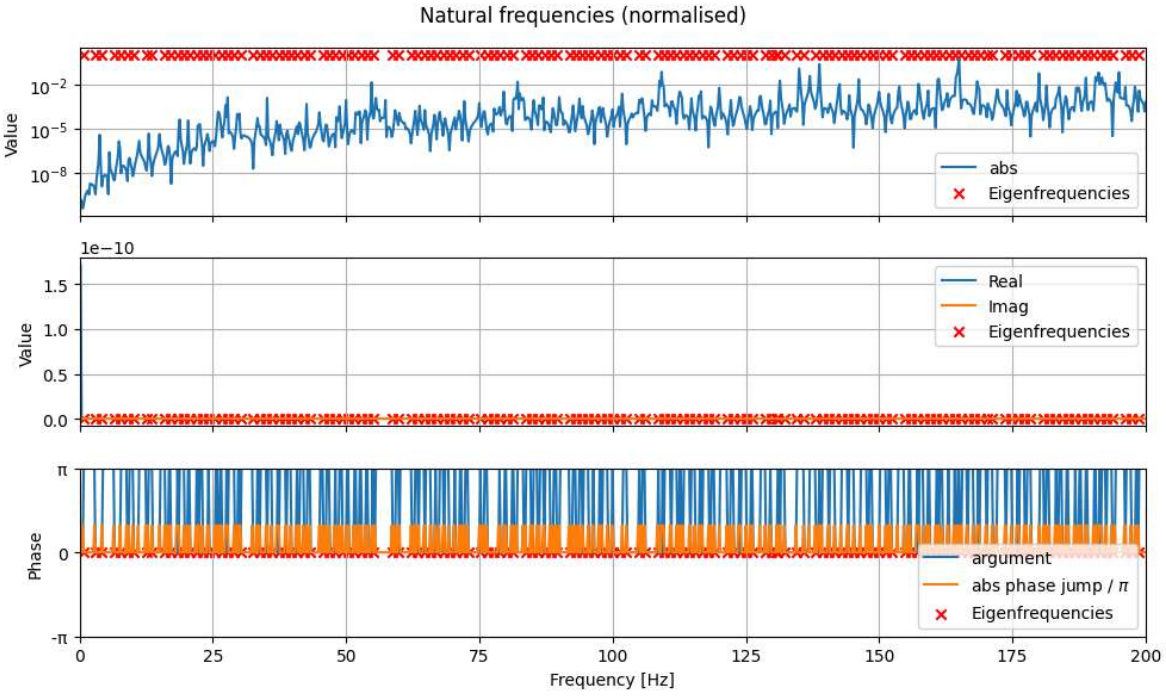


In [4]:
```python
#Some functions needed to find the natural frequencies

def BVP(ww, s):
    return s.GlobalConstrainedStiffness(ww)

def det_func(ww, s):
  return np.linalg.det(BVP(ww, s) / 1e10)

def find_eigen_frequencies(omega, s):
    Det_M = np.array([det_func(ww, s) for ww in omega])
    omega_initial = omega[np.where(np.isclose(abs(np.diff(np.angle(Det_M)))/np.p
    omega_m = []
    for ww in omega_initial:
        omega_m.append(opt.newton(det_func,ww,args=(s,),maxiter=2000).real)
    return np.unique(omega_m)

def remove_close_roots(roots, tol=1e-6):
    roots = np.sort(roots)
    filtered_roots = [roots[0]]

    for i in range(1, len(roots)):
        if not np.isclose(roots[i], filtered_roots[-1], atol=tol):
            filtered_roots.append(roots[i])

    return np.array(filtered_roots)
```

In [5]:
```python
#Finding the natural frequencies and visualising that process and the results

s1.run_connectivity()

omega_m1 = np.array(find_eigen_frequencies(omega_range1, s1))
omega_m1 = remove_close_roots(omega_m1)
f_m1 = omega_m1 / (2 * np.pi)

Det_M1 = np.array([det_func(omega, s1) for omega in omega_range1])
arg_dets1 = np.angle(Det_M1)
```

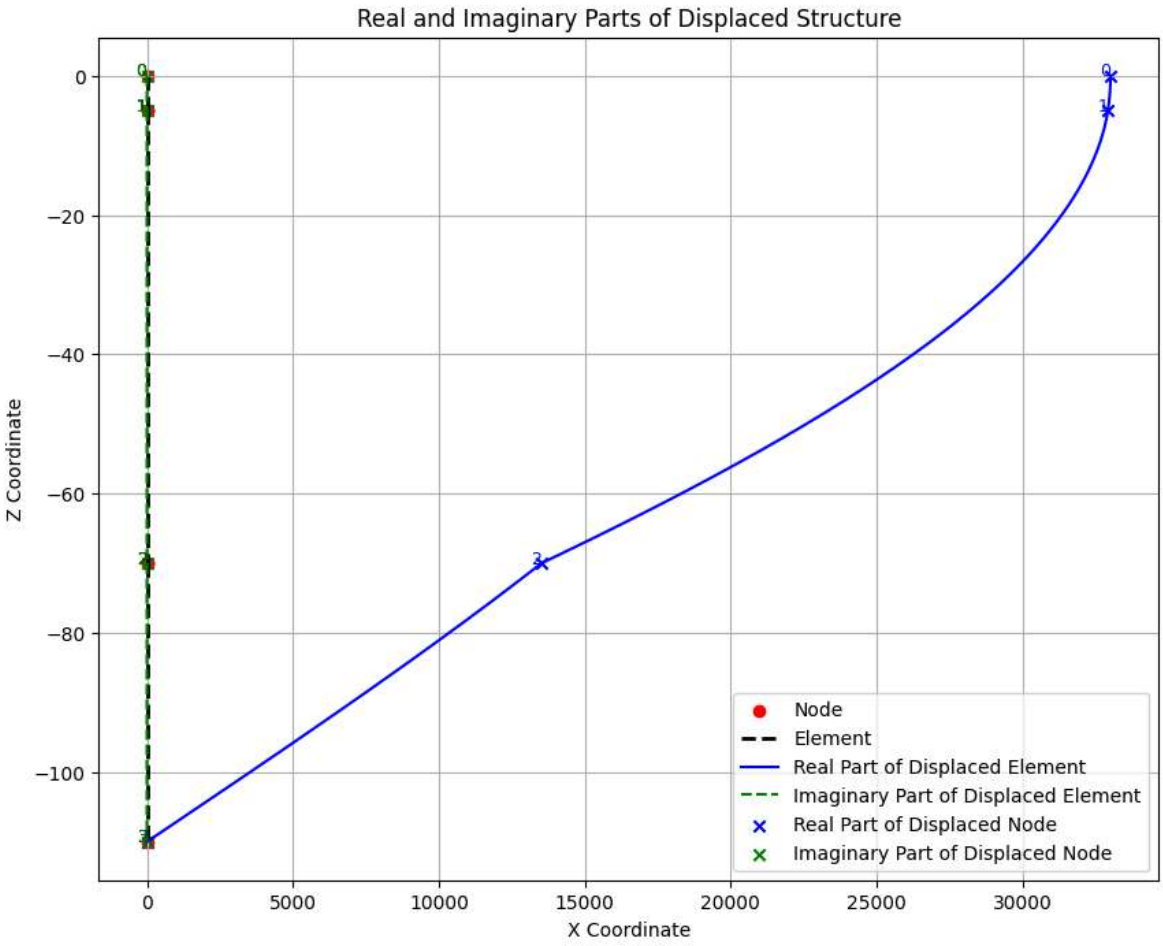
```
fig, axs = plt.subplots(3, sharex=True, figsize=(10,6))
axs[0].semilogy(omega_range1 / 2 / np.pi, abs(Det_M1), label='abs')
axs[1].plot(omega_range1 / 2 / np.pi, Det_M1.real/omega_range1**6, label='Real')
axs[1].plot(omega_range1 / 2 / np.pi, Det_M1.imag/omega_range1**6, label='Imag')
axs[2].plot(omega_range1 / 2 / np.pi, arg_dets1, label='argument')
axs[2].plot(omega_range1[:-1] / 2 / np.pi, abs(np.diff(arg_dets1))/np.pi, label=
axs[2].set_ylim([-np.pi, np.pi])

axs[0].scatter(f_m1, np.ones_like(omega_m1), marker='x', c='r', label='Eigenfreq
axs[1].scatter(f_m1, np.zeros_like(omega_m1), marker='x', c='r', label='Eigenfre
axs[2].scatter(f_m1, np.zeros_like(omega_m1), marker='x', c='r', label='Eigenfre

for ax in axs:
    ax.grid()
    ax.legend()
    ax.set_xlim([0, f_max1])
axs[0].set_ylabel('Value')
axs[1].set_ylabel('Value')
axs[2].set_ylabel('Phase')
axs[2].set_xlabel('Frequency [Hz]')
axs[2].set_yticks([-np.pi, 0, np.pi], ['-$\\mathrm{\\pi}$', 0, '$\\mathrm{\\pi}$
fig.suptitle('Natural frequencies (normalised)')
plt.tight_layout()

N = 10
print(f'The first {N} natural frequencies are: {f_m1[:N]}')
```

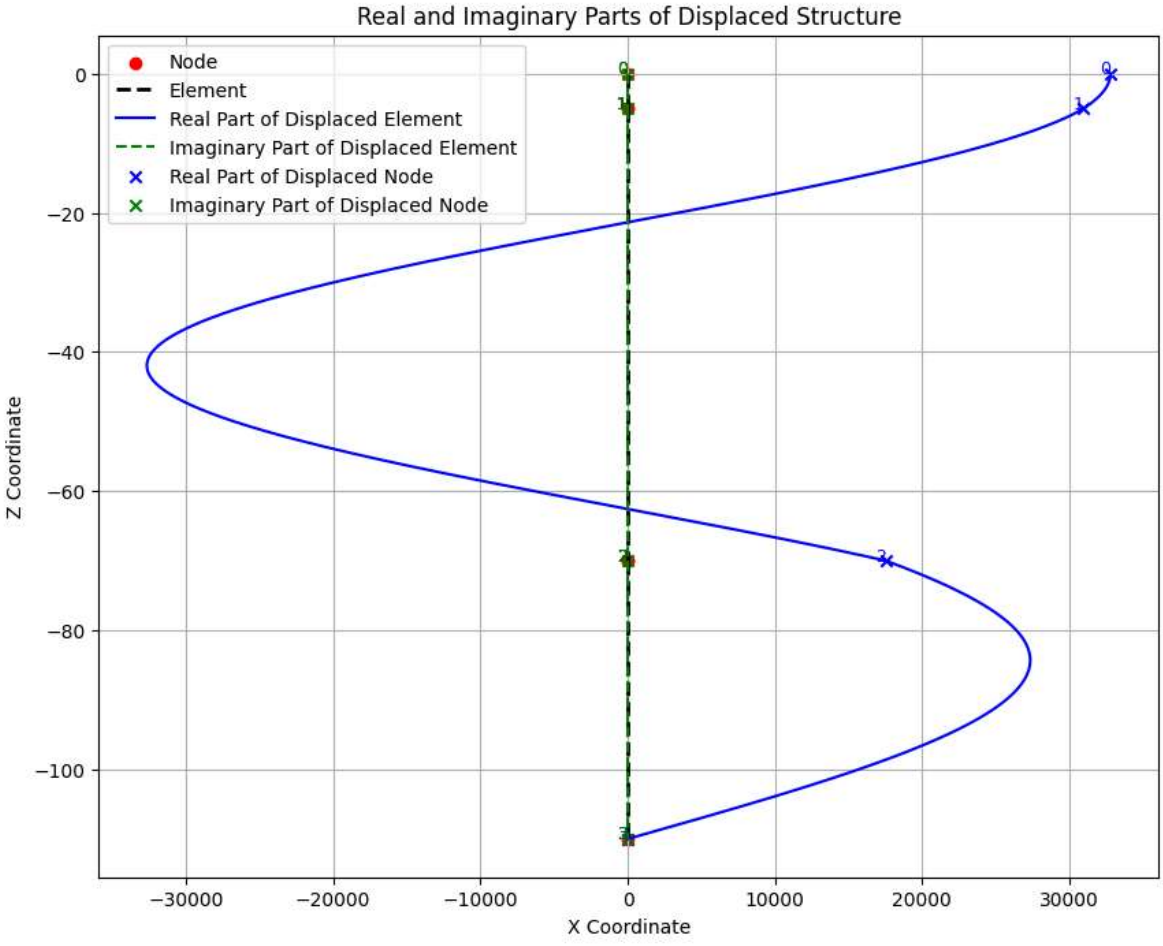
```
The first 10 natural frequencies are: [ 0.63562324  2.92014046  4.18691567  6.477
51586  7.7312669   8.93687311
 10.03522988 12.49800316 13.59461092 16.05221878]
```

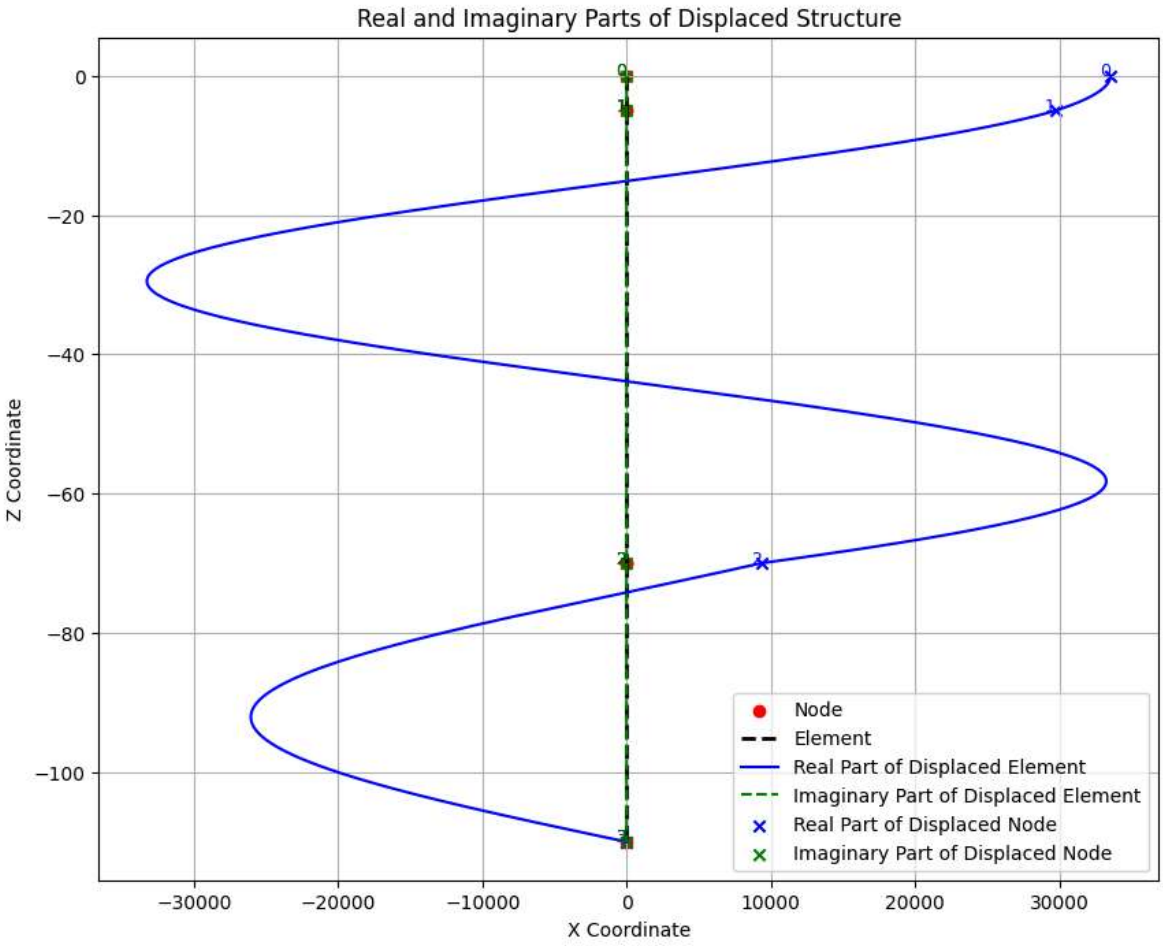


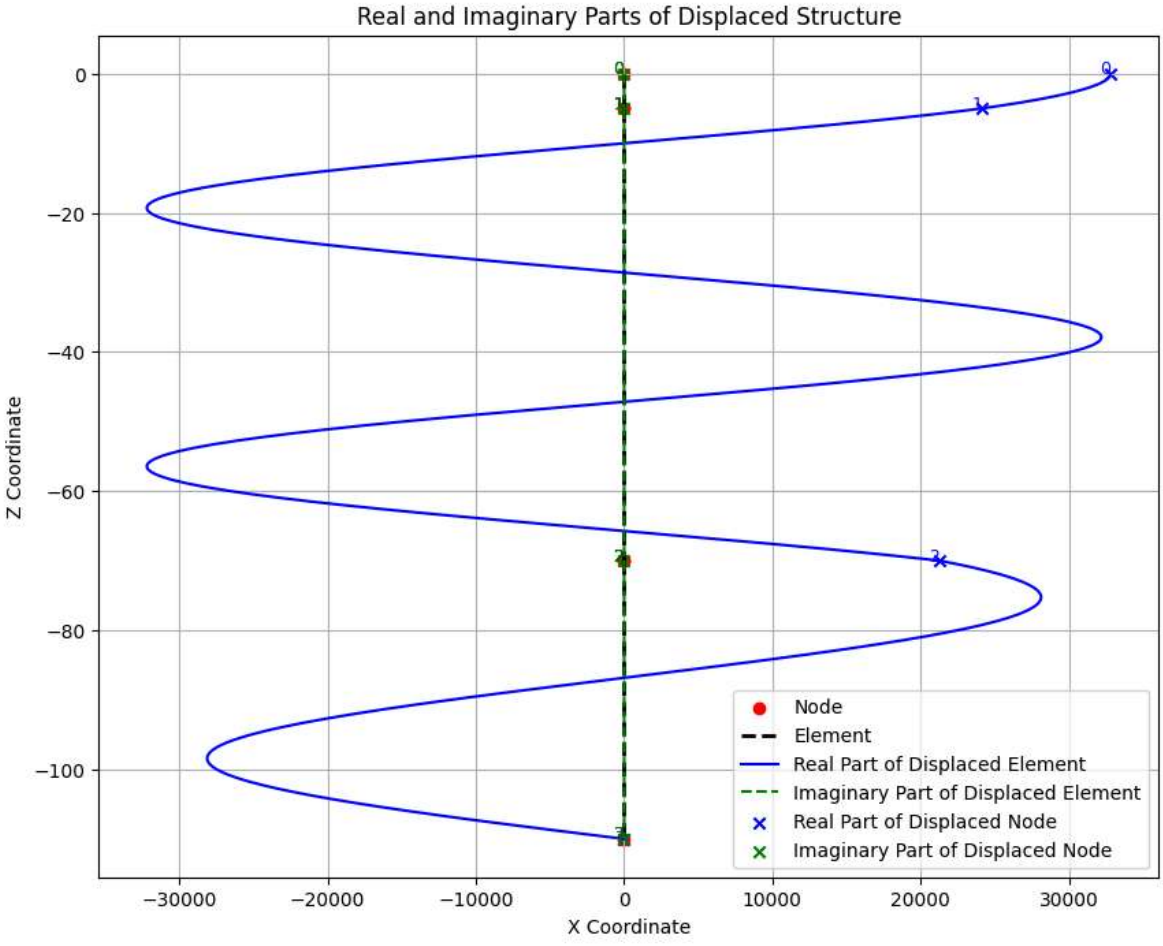
In [6]:
```
#Normalising the modes of vibration and plotting the first 10

def get_local_coordinates(elements,num):
    coor_local = {}
    for element in elements:
        coor_local[element.id] = element.L*np.linspace(0,1,num)
    return coor_local
```

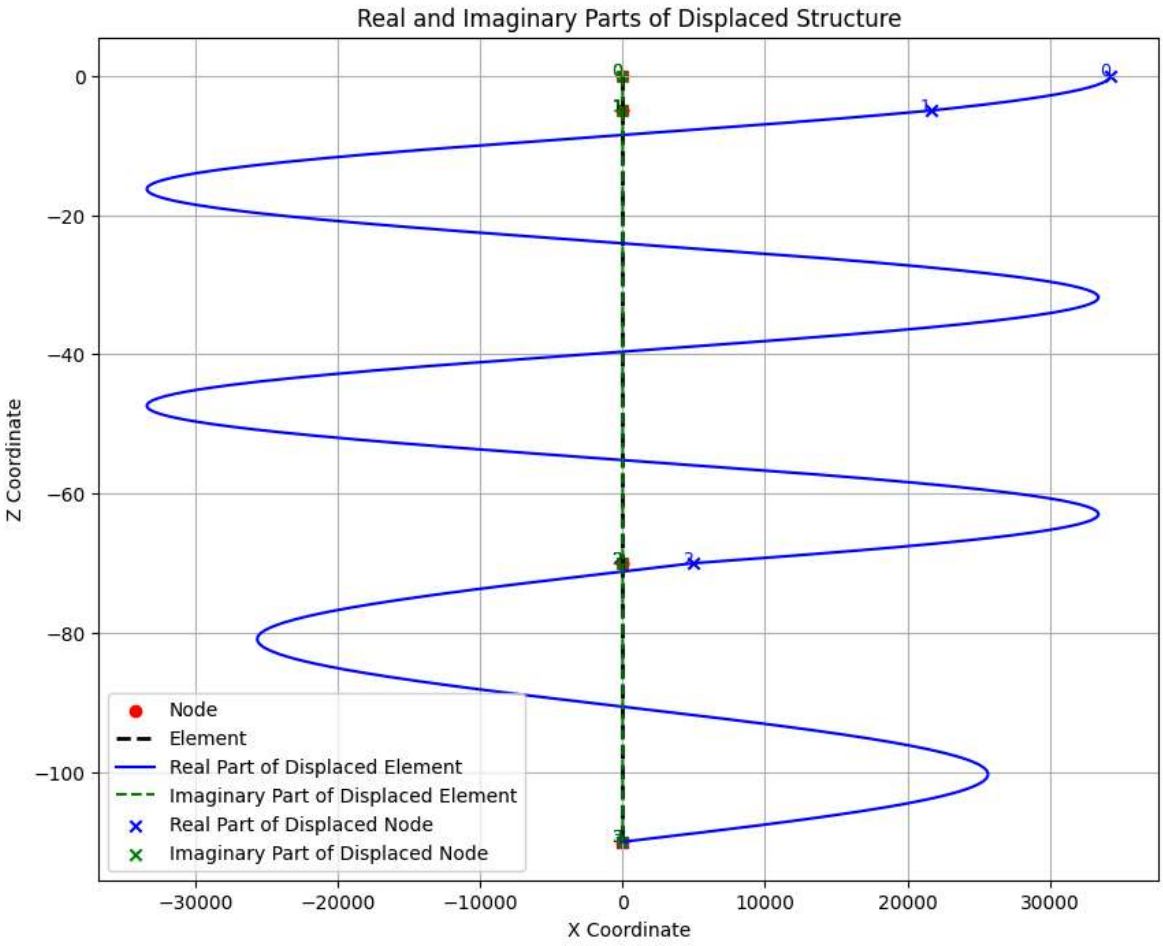
```
def Normalize_modes(Omega, displacement, displacement_local, local_coordinates,
    disp_norm = {}
    disp_local_norm = {}
    for omega in Omega:
        Gamma = 0
        for element in elements:
            key = list(element.element_types.keys())[0]
            rho = element.element_types[key].rho
            A = element.element_types[key].A
            Gamma += rho * A * np.trapezoid(displacement_local[omega][element.id

        disp_norm[omega] = {elem: disp / np.sqrt(Gamma) for elem, disp in displa
        #Local normalised displacements
        disp_local_norm[omega] = {elem: disp / np.sqrt(Gamma) for elem, disp in
    return disp_norm, disp_local_norm

#firstly create a dictionary for storing the displacements of each mode
disp_dict = {}
disp_local_dict = {}
num = 8000
for omega in omega_m1:
    Kc_global = s1.GlobalConstrainedStiffness(omega)
    u_free = s1.SolveUfree(Kc_global[:-1,:-1], -Kc_global[:-1,-1])
    u_free = np.concatenate((u_free, np.array([1])))
    u_elem = s1.FullDisplacement(u_free)
    disp = s1.ElementDisplacements(u_elem, omega, num)
    disp_dict[omega] = disp
    # get local displacements
    disp_local = s1.ElementDisplacements(u_elem, omega,num,local_axes=True)
    disp_local_dict[omega] = disp_local

local_coordinates = get_local_coordinates(elems1, num)
disp_norm, disp_local_norm = Normalize_modes(omega_m1, disp_dict, disp_local_dic

for omega in omega_m1[:10]:
    s1.PlotElementDisplacements(disp_norm[omega], scale=1e7)
```

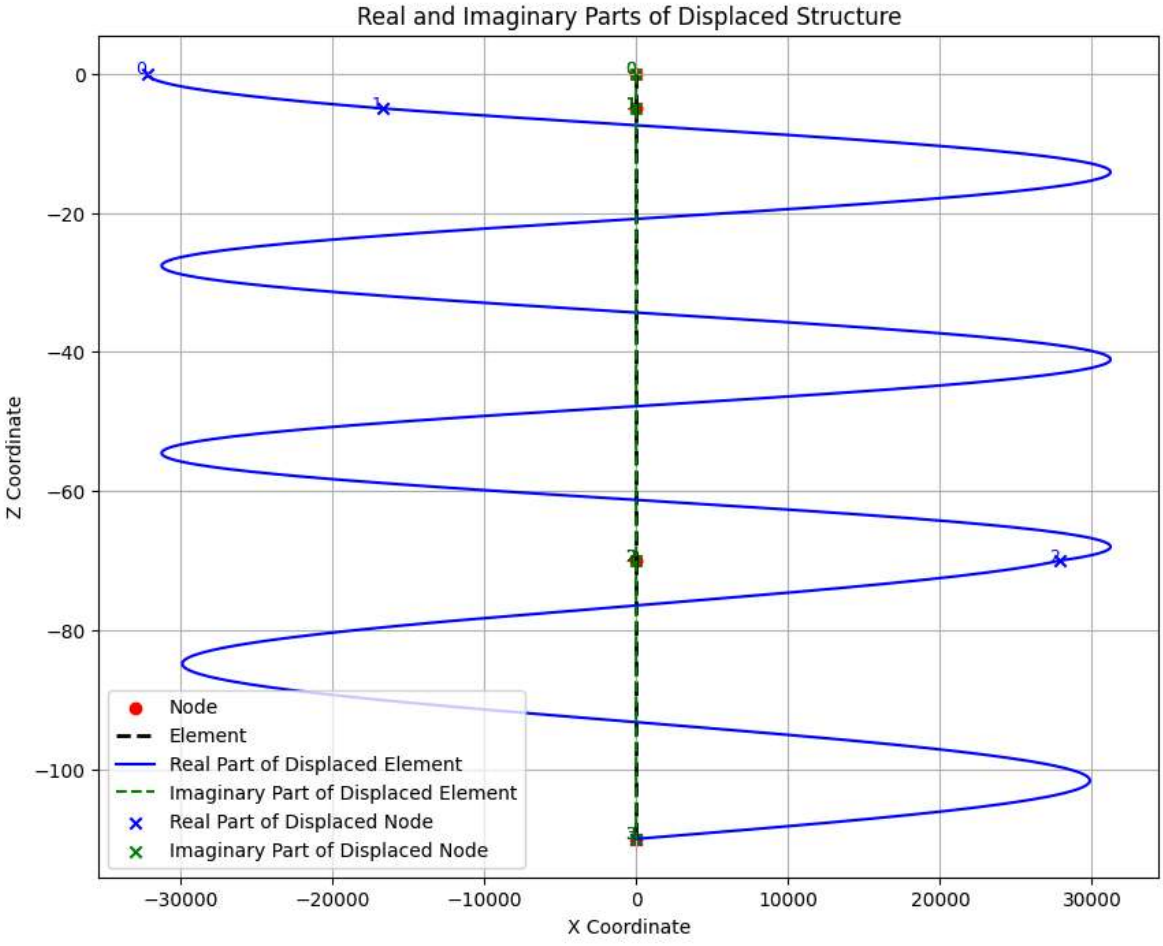

Real and Imaginary Parts of Displaced Structure
Z Coordinate
X Coordinate
Node
Element
Real Part of Displaced Element
Imaginary Part of Displaced Element
Real Part of Displaced Node
Imaginary Part of Displaced Node

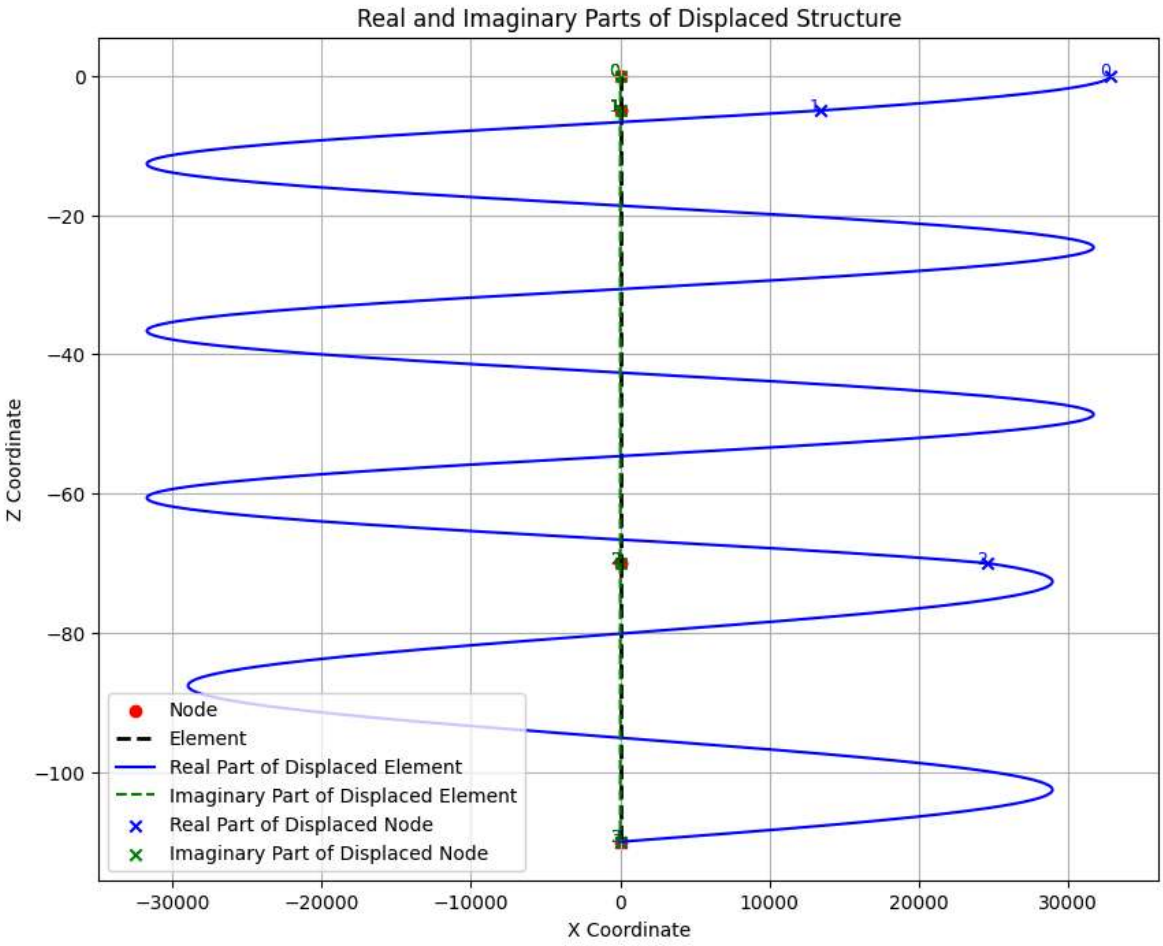

Real and Imaginary Parts of Displaced Structure
Z Coordinate
X Coordinate
Node
Element
Real Part of Displaced Element
Imaginary Part of Displaced Element
Real Part of Displaced Node
Imaginary Part of Displaced Node

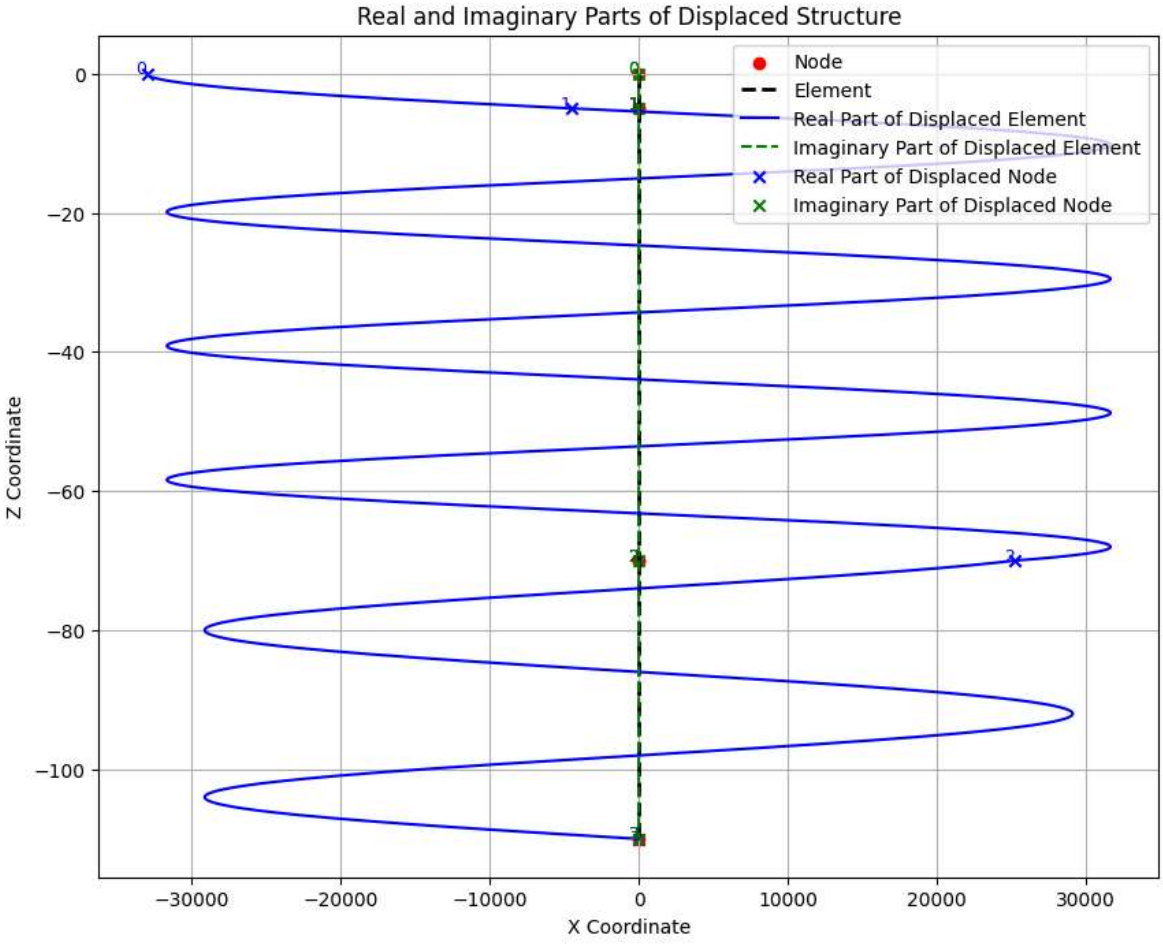

Real and Imaginary Parts of Displaced Structure
Z Coordinate
X Coordinate
0
−20
−40
−60
−80
−100
−30000
−20000
−10000
0
10000
20000
30000
Node
Element
Real Part of Displaced Element
Imaginary Part of Displaced Element
Real Part of Displaced Node
Imaginary Part of Displaced Node

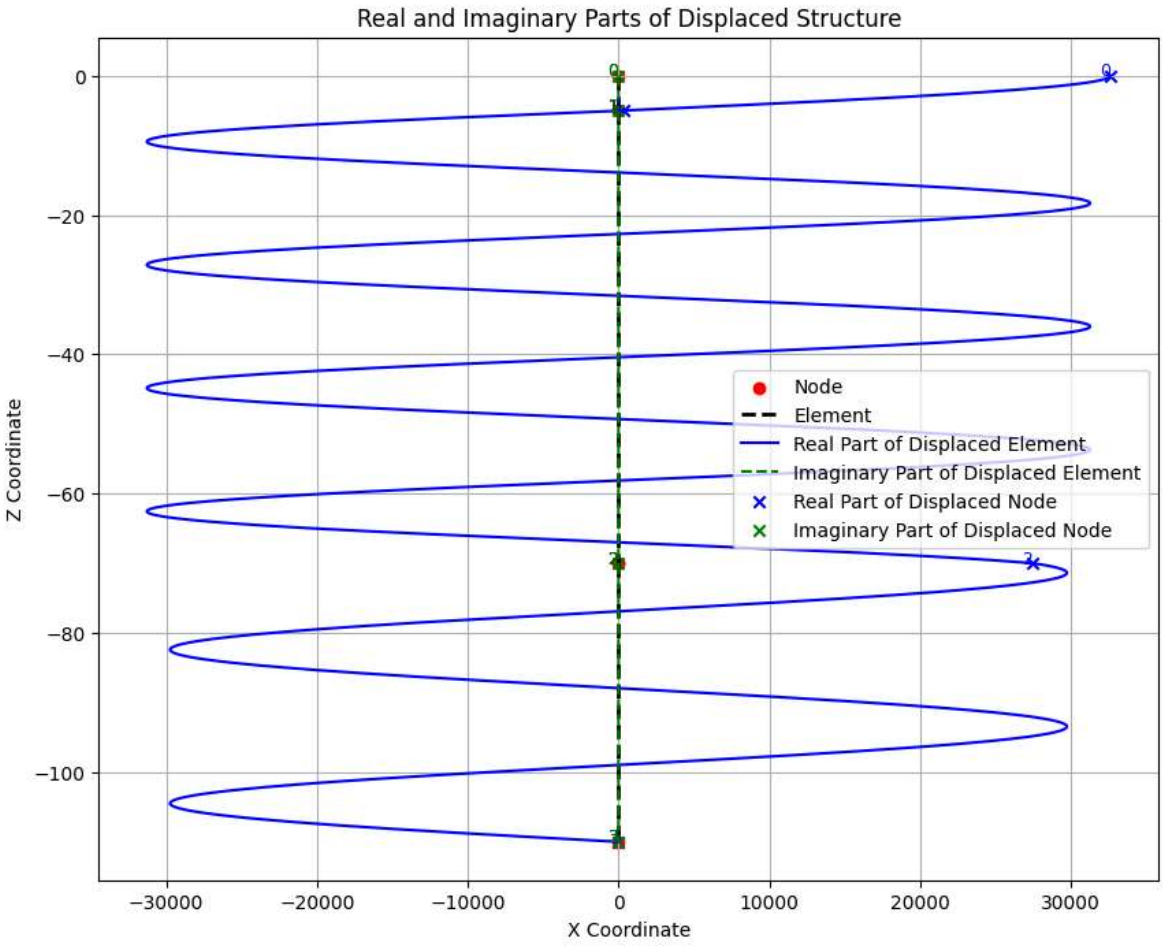

Real and Imaginary Parts of Displaced Structure
Z Coordinate
X Coordinate
0
−20
−40
−60
−80
−100
−30000
−20000
−10000
0
10000
20000
30000
Node
Element
Real Part of Displaced Element
Imaginary Part of Displaced Element
Real Part of Displaced Node
Imaginary Part of Displaced Node

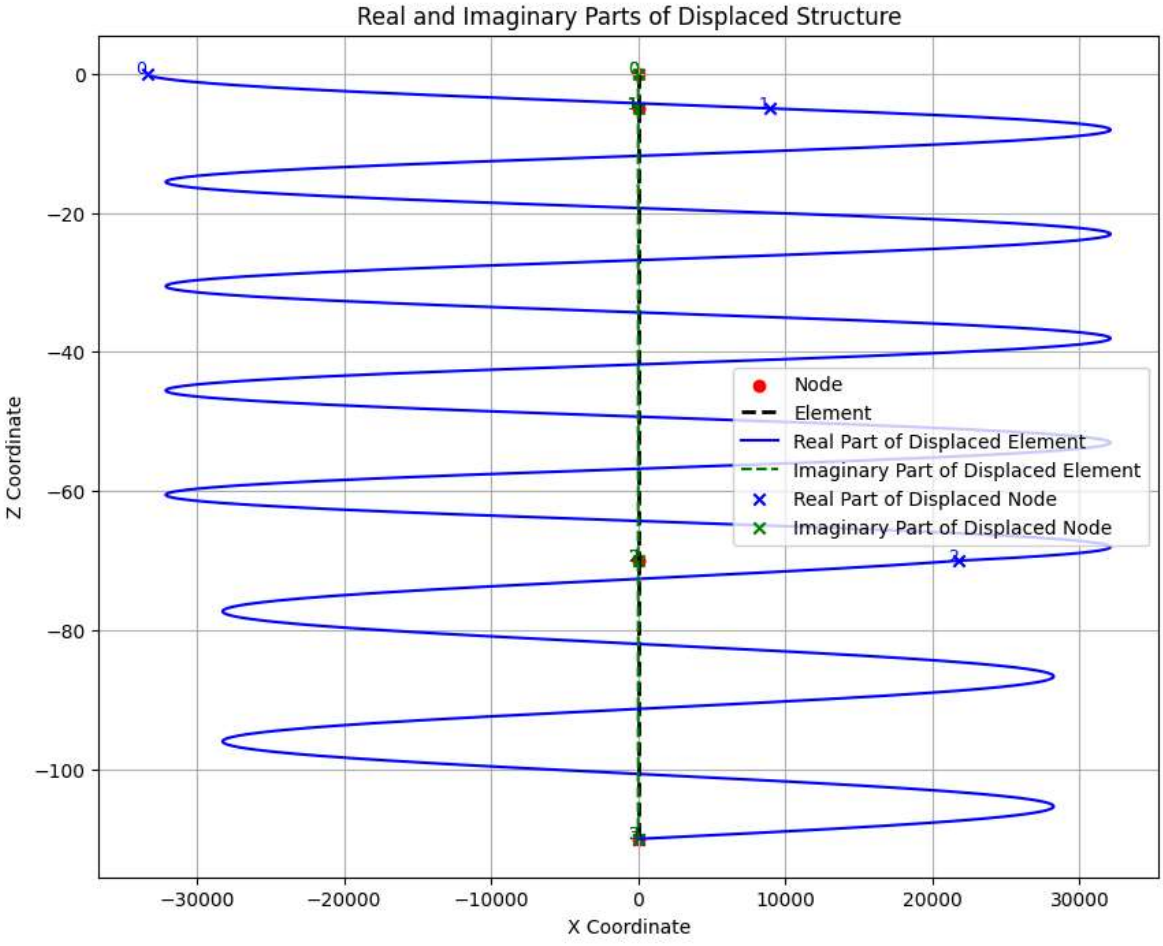

Real and Imaginary Parts of Displaced Structure
Z Coordinate
X Coordinate
0
−20
−40
−60
−80
−100
−30000
−20000
−10000
0
10000
20000
30000
Node
Element
Real Part of Displaced Element
Imaginary Part of Displaced Element
Real Part of Displaced Node
Imaginary Part of Displaced Node


Real and Imaginary Parts of Displaced Structure
Z Coordinate
X Coordinate
0
−20
−40
−60
−80
−100
−30000
−20000
−10000
0
10000
20000
30000
Node
Element
Real Part of Displaced Element
Imaginary Part of Displaced Element
Real Part of Displaced Node
Imaginary Part of Displaced Node


Real and Imaginary Parts of Displaced Structure
Z Coordinate
X Coordinate
Node
Element
Real Part of Displaced Element
Imaginary Part of Displaced Element
Real Part of Displaced Node
Imaginary Part of Displaced Node

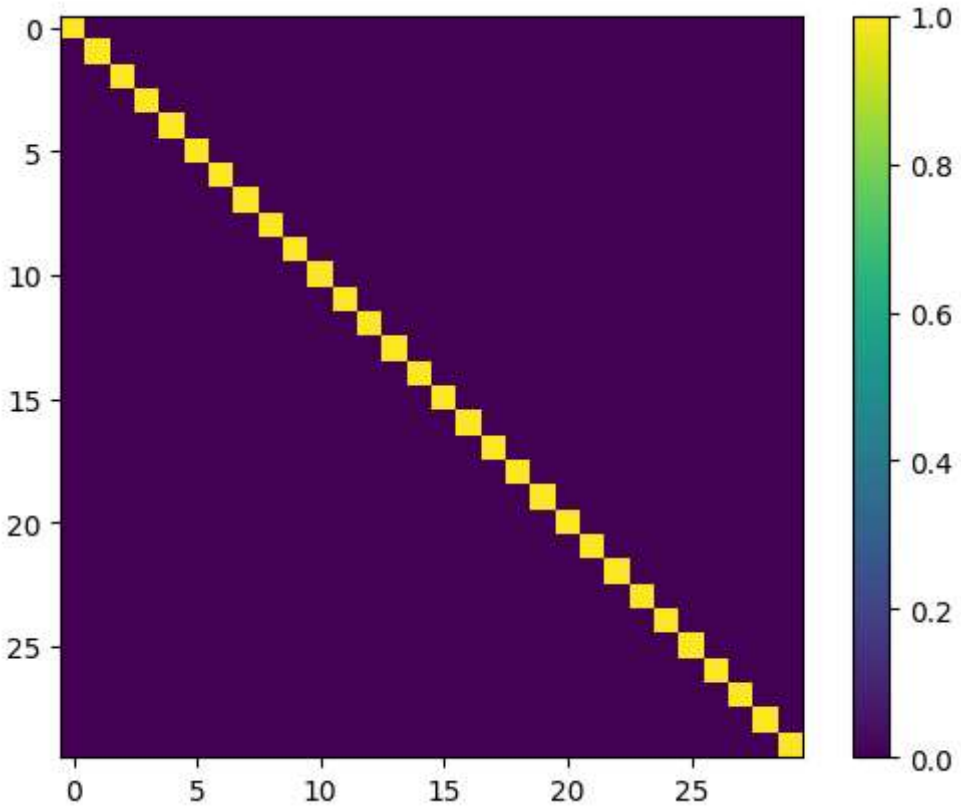

Real and Imaginary Parts of Displaced Structure
Z Coordinate
X Coordinate
Node
Element
Real Part of Displaced Element
Imaginary Part of Displaced Element
Real Part of Displaced Node
Imaginary Part of Displaced Node





In [7]:
```
#Checking the orthogonality of the modes
```

```
def Check_Orthogonality(omega, disp_local_norm, local_coordinates, elements):
    N = len(omega)
    Ortho_matrix = np.zeros((N, N), dtype=complex)
    for ii, omega_1 in enumerate(omega):
        for jj, omega_2 in enumerate(omega):
            for element in elements:
                key = list(element.element_types.keys())[0]
                rho = element.element_types[key].rho
                A = element.element_types[key].A
                Ortho_matrix[ii, jj] += rho * A * np.trapezoid(disp_local_norm[c
    return Ortho_matrix

Ortho_coeff = Check_Orthogonality(omega_m1[:30], disp_local_norm, local_coordina

plt.imshow(abs(Ortho_coeff))
plt.colorbar();
```



In case soil damping is included, the natural frequencies become complex and the modes of vibration will be damped (modal damping). The PyDynSM package does not support that, because it finds the natural frequencies using the phase shifts in the argument of the determinant of the global stiffness matrix for specific omegas. The damping influences the phase shift, which is why this method is not able to locate natural frequencies when damping is included.

Exercise 3: Frequency response functions

To find the frequency response functions (FRFs) of the layered soil system, it will be subjected to a unit amplitude earthquake motion (displacement-wise). The response at the top of layer 1 and at the top of layer 3 will be calculated. The maximum considered

frequency was chosen as 100 Hz, to find a balance between completeness and the ability to visualise the results.

From now on, material damping will be considered. This is incorporated in the model using material damping according to:

$$\tilde{G} = G(1 + 2\xi i)$$

in which ξ is the material loss factor [-], i the imaginary unit and G is the shear modulus [MPa]. The material loss factor can be determined as:

$$\xi = \frac{\eta\omega}{2G}$$

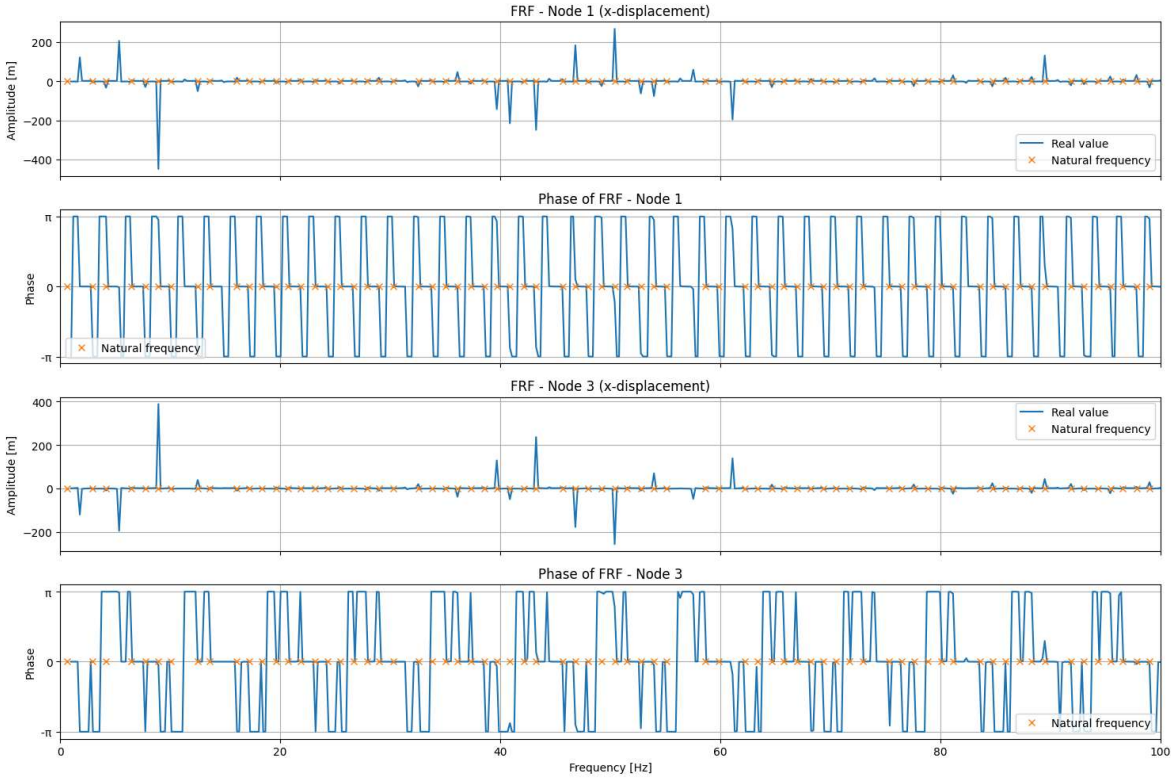
in which η is the provided damping coefficient in Ns/m^2 and ω is chosen as the first natural (angular) frequency.

In [8]:
```
#Finding the responses of the specific degrees of freedom in a certain frequency

FRF_seism_freq_max = 100
FRF_seism_freq = np.linspace(1, FRF_seism_freq_max, 500)
FRF_seism_omega = FRF_seism_freq * 2 * np.pi
FRF_seism_toplayer1 = np.zeros(len(FRF_seism_omega), dtype=complex)
FRF_seism_toplayer3 = np.zeros(len(FRF_seism_omega), dtype=complex)

#Adding a unit displacement at the bottom of the lowest layer
nodes1[-1].prescribe_node(x=1)

#Adding non-zero damping now
eta = [eta_1, eta_2, eta_3]
for i,elem in enumerate(elems1):
    soil_section_damping = {'rho': rho[i], 'A': A_s, 'G': G_list[i], 'ksi': eta[
    elem.SetSection('Shear Beam', soil_section_damping)

s1.run_connectivity() #this is needed after fixing a node

for i,omega in enumerate(FRF_seism_omega):
    Kc_global = s1.GlobalConstrainedStiffness(omega)
    Fc_global = s1.GlobalConstrainedForce(omega)
    u_free = s1.SolveUfree(Kc_global, Fc_global)
    #first free dof (index 0) is the x-displacement of node 1 (index 0)
    FRF_seism_toplayer1[i] = u_free[0]
    #third free dof (index 2) is the x-displacement of node 3 (index 2)
    FRF_seism_toplayer3[i] = u_free[2]
```

```
Successfully added element of type: Shear Beam to Element 0
Successfully added element of type: Shear Beam to Element 1
Successfully added element of type: Shear Beam to Element 2
```

In [9]:
```
#Plotting the FRFs

f_m_below_f_max_FRF = f_m1[f_m1 < FRF_seism_freq_max]
fig, axes = plt.subplots(4, 1, figsize=(15, 10), sharex=True)
axes[0].plot(FRF_seism_freq, np.real(FRF_seism_toplayer1), label='Real value')
# axes[0].plot(FRF_seism_freq, np.imag(FRF_seism_toplayer1), label='Imaginary va
# axes[0].plot(FRF_seism_freq, np.abs(FRF_seism_toplayer1), label='Absolute valu
axes[0].set_title('FRF - Node 1 (x-displacement)')
axes[0].set_ylabel('Amplitude [m]')
```

```
axes[1].plot(FRF_seism_freq, np.angle(FRF_seism_toplayer1))
axes[1].set_title('Phase of FRF - Node 1')

axes[2].plot(FRF_seism_freq, np.real(FRF_seism_toplayer3), label='Real value')
# axes[2].plot(FRF_seism_freq, np.imag(FRF_seism_toplayer3), label='Imaginary va
# axes[2].plot(FRF_seism_freq, np.abs(FRF_seism_toplayer3), label='Absolute valu
axes[2].set_title('FRF - Node 3 (x-displacement)')
axes[2].set_ylabel('Amplitude [m]')

axes[3].plot(FRF_seism_freq, np.angle(FRF_seism_toplayer3))
axes[3].set_title('Phase of FRF - Node 3')
axes[3].set_xlabel('Frequency [Hz]')

for i in [1, 3]:
    axes[i].set_ylabel('Phase')
    axes[i].set_ylim([-1.1 * np.pi, 1.1 * np.pi])
    axes[i].set_yticks([-np.pi, 0, np.pi], ['-$\\mathrm{\\pi}$', 0, '$\\mathrm{\

for ax in axes:
    ax.plot(f_m_below_f_max_FRF, np.zeros(len(f_m_below_f_max_FRF)), 'x', label=
    ax.legend(loc='best')
    ax.set_xlim(0, FRF_seism_freq_max)
    ax.grid()

fig.tight_layout()
```



The frequency response functions of both node 1 and node 3 show large (displacement) amplifications for certain frequencies. It can be seen that those frequencies are almost the same frequencies for both cases: roughly speaking, a group at frequencies around 0-10 Hz, a group around 40-60 Hz and a few peaks between 80 and 100 Hz. Most of the larger amplifications have a different sign for both nodes, which can be explained by the fact that the mode shapes peak at those nodes, but have a different sign only. Most amplifications occur at a natural frequency (two peaks between 0 and 10 Hz and two

peaks around 60 Hz do not). The fact that some frequencies only show a peak at one of the nodes can be explained by the corresponding mode shape (of that natural frequency) having a node (zero amplitude) at the other node.

The plotted phase plots show whether the two nodes move in-phase or out-of-phase with each other, i.e. whether their displacement is in the same or in a different direction. For frequencies where both plots show the same phase or one shows π and the other shows $-\pi$, the movements are in-phase. In other cases, the movements are out-of-phase.

Exercise 4: Governing equations of the offshore wind turbine

The offshore wind turbine can be split up into two segments: the top segment (above the water surface) and the submerged segment. The top part is not subjected to any loading (wind force is not considered here), the lower part is excited by sea waves.

Equations of motion

Both sections can be modelled as an Euler-Bernoulli beam. The equation of motion for the top segment is:

$$\rho_t A_t \frac{\partial^2 w_1(z,t)}{\partial t^2} + E_t I_t \frac{\partial^4 w_1(z,t)}{\partial z^4} = 0$$

in which ρ_t is the density of the turbine [kg/m^3], A_t is the cross-sectional area of the turbine [m^2], $w_1(z,t)$ is the horizontal displacement [m] on the interval $H_w < z < H_w + H_t$, t is the time [s], E_t is the Young's modulus of the turbine [N/m^2] and I_t its moment of inertia [m^4].

The equation of motion for the segment submerged in water is:

$$\rho_t A_t \frac{\partial^2 w_2(z,t)}{\partial t^2} + E_t I_t \frac{\partial^4 w_2(z,t)}{\partial z^4} = f_H(z,t)$$

in which $w_2(z,t)$ is the horizontal displacement [m] on the interval $0 < z < H_w$ [m] and $f_H(z,t)$ the distributed wave force acting on the turbine [N/m].

The following expressions are used: $A_t = \frac{1}{4}\pi(D_{out}^2 - D_{inner}^2)$ and $I_t = \frac{\pi}{64}(D_{out}^4 - D_{inner}^4)$. For the wave forces acting on the turbine, Morison equation can be used, using only the inertial part, which is given by:

$$f_H(z,t) = \frac{1}{4}\pi D^2 \rho_w \frac{\partial u}{\partial t} + \frac{1}{4}\pi D^2 C_a \rho_w \left(\frac{\partial u}{\partial t} - \frac{\partial^2 w_2}{\partial t^2} \right)$$

in which D is an equivalent diameter of displaced mass, ρ_w is the density of water, u is the velocity of the waves [m/s] and C_a is an additional mass factor. When implementing this force, the details will be elaborated.

The rotation of an Euler-Bernoulli beam is given by $\varphi(z,t) = -\frac{\partial u}{\partial z}$, the bending moment by $M(z,t) = -E_t I_t \frac{\partial^2 u}{\partial z^2}$ and the shear force by $V(z,t) = -E_t I_t \frac{\partial^3 u}{\partial z^3}$.

Boundary conditions

The equations of motion are fourth-order in space, which means that there should be two boundary condition at the soil surface and two at the top of the offshore wind turbine.

At the soil surface ($z = 0$), a rigid connection with the soil (excited by waves) can be assumed, with no relative displacement. Soil-structure interaction is neglected:

$$w_2(z=0,t) = u_1(z_1=0,t)$$

Secondly, the rotation is zero:

$$-\frac{\partial w_2}{\partial z}\Big|_{z=0} = 0$$

At the top, the top mass with mass M and rotational inertia J can be incorporated as follows: No description has been provided for this image

$$M\frac{\partial^2 w_1}{\partial t^2}\Big|_{z=H_w+H_t} = \sum_i F_i \rightarrow M\frac{\partial^2 w_1}{\partial t^2}\Big|_{z=H_w+H_t} = E_t I_t \frac{\partial^3 w_1}{\partial z^3}\Big|_{z=H_w+H_t}$$

$$J\frac{\partial^3 w_1}{\partial z \partial t^2}\Big|_{z=H_w+H_t} = \sum_i M_i \rightarrow J\frac{\partial^3 w_1}{\partial z \partial t^2}\Big|_{z=H_w+H_t} = E_t I_t \frac{\partial^2 w_1}{\partial z^2}\Big|_{z=H_w+H_t}$$

Interface conditions

No description has been provided for this image

Following from the fact that the equations of motion are fourth-order in space, there should be four interface conditions at the interface, which is at sea surface. All four field variables can be equated to each other (in which minus signs and the bending stiffness can be divided out):

$$w_1(z=H_w,t) = w_2(z=H_w,t)$$

$$\frac{\partial w_1}{\partial z}\Big|_{z=H_w} = \frac{\partial w_2}{\partial z}\Big|_{z=H_w}$$

$$\frac{\partial^2 w_1}{\partial z^2}\Big|_{z=H_w} = \frac{\partial^2 w_2}{\partial z^2}\Big|_{z=H_w}$$

$$\frac{\partial^3 w_1}{\partial z^3}\Big|_{z=H_w} = \frac{\partial^3 w_2}{\partial z^3}\Big|_{z=H_w}$$

Exercise 5: Dynamical response of the offshore wind turbine

Defining the forcing

The offshore wind turbine is subjected to two loads: the wave force which is directly acting on the structure and the seismic motion at bedrock level (indirectly). They can be assumed uncorrelated.

Wave force

The power spectral density of the wave force can be determined using the JONSWAP spectrum, which is given by:

$$S = \alpha g^2 (2\pi)^{-4} f^{-5} e^{-\beta\left(\frac{f}{f_p}\right)^{-4}} \gamma^{e^{-\frac{1}{2}\left(\frac{\frac{f}{f_p}-1}{\sigma}\right)^2}}$$

in which:

- $\alpha = 0.076\tilde{F}^{-0.22}$
- $g = 9.81$ [m/s²]
- f: frequency
- $\beta = \frac{5}{4}$
- f_p: peak frequency
- $\gamma = 3.3$
- $\sigma = \begin{cases} 0.07 & f \leq f_p \\ 0.09 & f > f_p \end{cases}$

For the calculation of f_p using the non-dimensional peak frequency $\tilde{f}_p$ and the non-dimensional fetch $\tilde{F}$, the following three formulas are used:

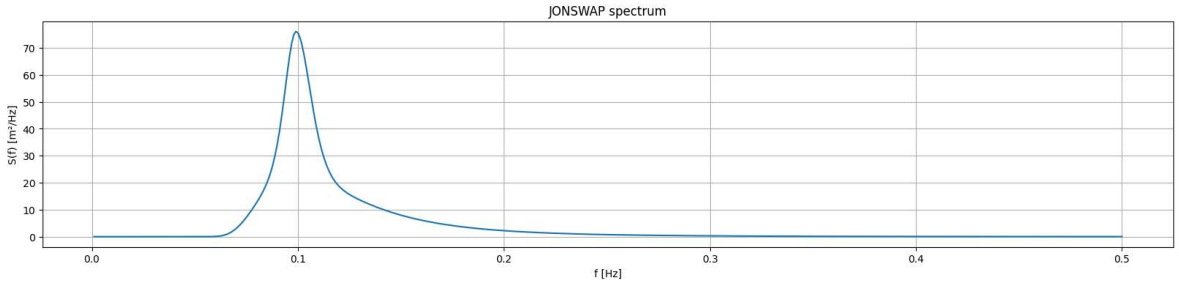
- $\tilde{f}_p = \frac{U_{10} f_p}{g} \rightarrow f_p = \frac{\tilde{f}_p g}{U_{10}}$
- $\tilde{F} = \frac{gF}{U_{10}^2}$
- $\tilde{f}_p = 3.5\tilde{F}^{-0.33}$

with U_{10} the average wind speed at a reference height of 10 meter [m/s] and fetch F [m]. The parameters from load case 2 of the assignment of Unit 2 of this course (Dynamics of Structures Subjected to Wind & Waves) can be used:

- $U_{10} = 23$ [m/s]
- $F = 200 \cdot 10^3$ [m]

In [10]:
```
#Defining the power spectral density for the wave force (JONSWAP spectrum)

g = 9.81 #m/s²
beta = 5 / 4
gamma = 3.3
sigma_l = 0.07
sigma_u = 0.09
U_10 = 23 #m/s
F = 200E3 #m
f_max_JS = 0.5 #Hz

F_tilde = g * F / U_10**2
```

```
f_p_tilde = 3.5 * F_tilde**(-0.33)
f_p = f_p_tilde * g / U_10
alpha = 0.076 * F_tilde**(-0.22)

freq_waves = np.linspace(0.001, f_max_JS, 500)

def JONSWAP(f):
    sigma = sigma_l * np.ones_like(f)
    sigma[f > f_p] = sigma_u
    return alpha * g**2 * (2 * np.pi)**(-4) * f**(-5) * np.exp(-beta * (f / f_p)

plt.figure(figsize=(20, 4))
plt.plot(freq_waves, JONSWAP(freq_waves))
plt.title('JONSWAP spectrum')
plt.xlabel('f [Hz]')
plt.ylabel('S(f) [m²/Hz]')
plt.grid()
```

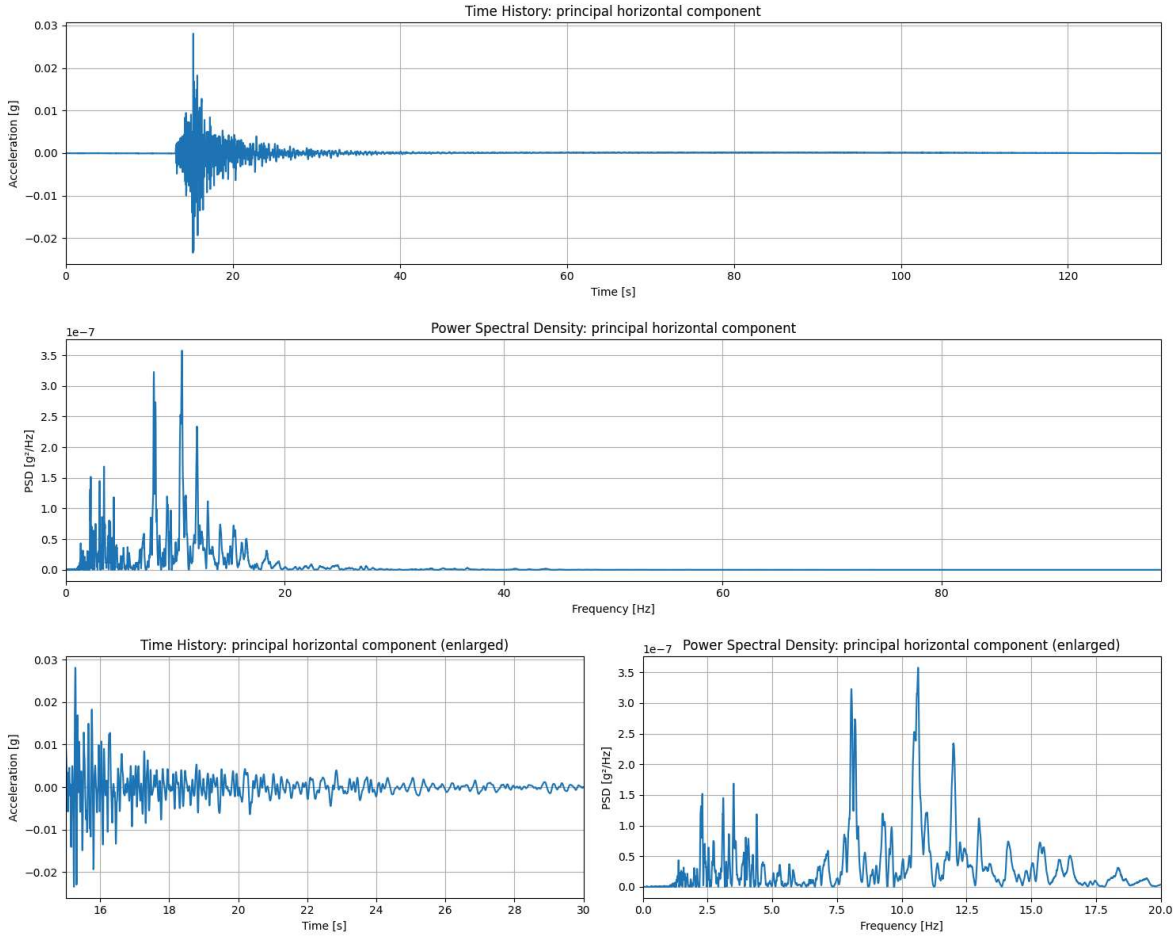


Seismic motion

The seismic motion at bedrock level can be taken as the principal horizontal component of a Groningen signal at approximately 100 meter depth that took place on 8 January 2018. Its power spectral density was already provided in the notebook `PSD.ipynb` and is repeated here. For the plots of the (filtered) FFT magnitude spectrum and the auto-correlation function, the reader is referred to the provided notebook.

In [11]:
```
#Defining the power spectral density for the seismic motion

file_path = 'NL.G192..ALL.2018-01-08.dat'
data = np.loadtxt(file_path)

time = data[:, 0]
accel = data[:, 1]
name = 'principal horizontal component'

N = len(accel)
dt = time[1] - time[0]
fs = 1/dt
cutoff = 0.1
order = 4

# Apply high-pass filter to remove low frequency oscillation
nyq = 0.5 * fs           # Nyquist frequency
normal_cutoff = cutoff / nyq
b, a = butter(order, normal_cutoff, btype='high', analog=False)
accel_filtered = filtfilt(b, a, accel)

# Calculate correlation
corr = correlate(accel_filtered, accel_filtered, mode='full') * dt # dt as the a
```

```
lags = np.arange(-len(accel_filtered) + 1, len(accel_filtered)) * dt

# divide by total bin length
normalization = np.array([N - abs(lag) for lag in range(-N + 1, N)]) * dt # dt h
corr = corr / normalization

# Power Spectral Density as fourier transform of the correlation function
N_psd = len(corr)
PSD_seism_0 = abs(rfft(corr)) * dt # no multiplication with 2 due to using negat
freq_seism = rfftfreq(len(corr), dt)

# plot
fig = plt.figure(figsize=(15, 12))
ax1 = plt.subplot2grid((3, 2), (0, 0), colspan=2)
ax2 = plt.subplot2grid((3, 2), (1, 0), colspan=2)
ax3 = plt.subplot2grid((3, 2), (2, 0))
ax4 = plt.subplot2grid((3, 2), (2, 1))

ax1.plot(time, accel)
ax1.set_title(f'Time History: {name}')
ax1.set_xlabel('Time [s]')
ax1.set_ylabel('Acceleration [g]')
ax1.set_xlim(0, time[-1])
ax1.grid(True)

ax2.plot(freq_seism, PSD_seism_0)
ax2.set_title(f"Power Spectral Density: {name}")
ax2.set_xlabel("Frequency [Hz]")
ax2.set_ylabel("PSD [g²/Hz]")
ax2.set_xlim(0, freq_seism[-1])
ax2.grid(True)

ax3.plot(time, accel)
ax3.set_title(f'Time History: {name} (enlarged)')
ax3.set_xlabel('Time [s]')
ax3.set_ylabel('Acceleration [g]')
ax3.set_xlim(15, 30)
ax3.grid(True)

ax4.plot(freq_seism, PSD_seism_0)
ax4.set_title(f"Power Spectral Density: {name} (enlarged)")
ax4.set_xlabel("Frequency [Hz]")
ax4.set_ylabel("PSD [g²/Hz]")
ax4.set_xlim(0, 20)
ax4.grid(True)
plt.tight_layout()
```



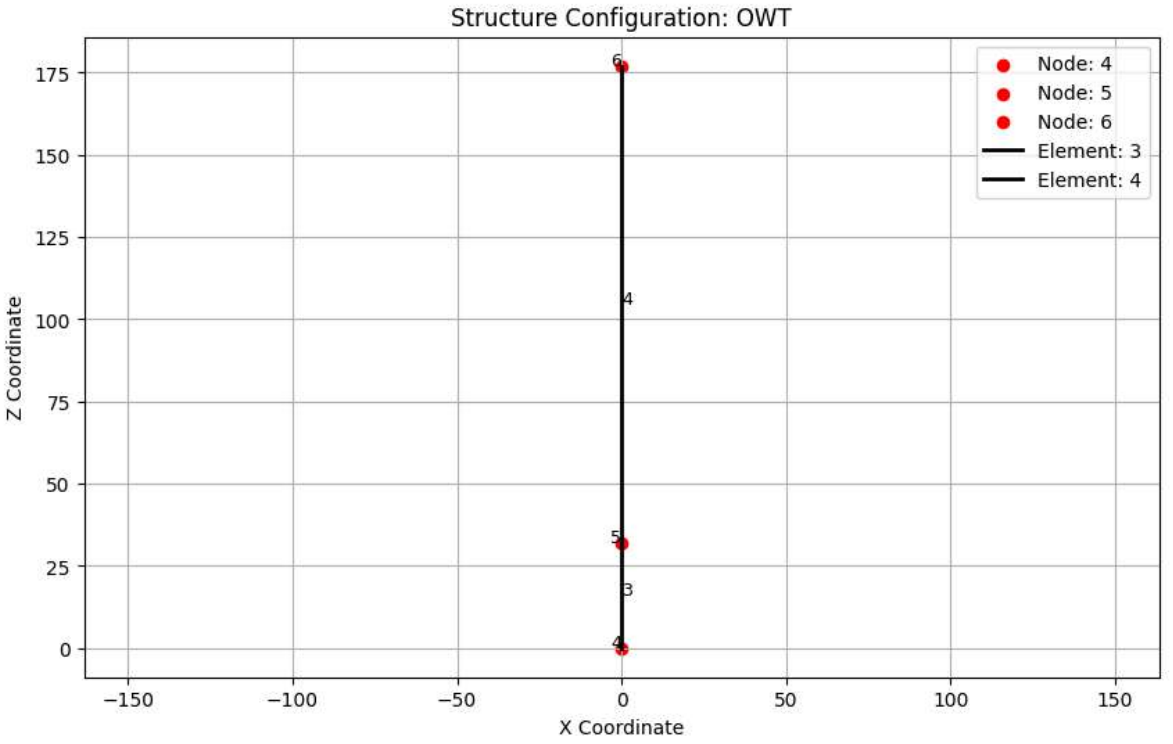
Implementation of the offshore wind turbine in PyDynSM

The offshore wind turbine can be modeled using the PyDynSM Python package. This is done below, using "EulerBernoulli Beam foundation attachments" as element type.

In [12]:
```
#Creating the offshore wind turbine

#Instantiating class
s2 = PDM.Assembler('OWT')

#Creating nodes
nodes2 = []
H_list2 = [0, H_w, H_w + H_t]

for H in H_list2:
    node = s2.CreateNode(0, H, dof_config=['x', 'phi_y'])
    nodes2.append(node)

nodes2[0].fix_node('x', 'phi_y') #x is fixed, because it should be unforced (so

#Creating elements
elems2 = []

A_t = 1 / 4 * np.pi * (D_out**2 - D_inner**2)
I_t = 1 / 64 * np.pi * (D_out**4 - D_inner**4)
W_t = E_t * I_t / 10
owt_section1 = {'rho': rho_t, 'A': A_t, 'E': E_t, 'Ib': I_t, 'Wb': W_t} #no damp
owt_section2 = owt_section1.copy()
owt_section2['Pm2'] = M
owt_section2['J2'] = J
```

```
elems2.append(s2.CreateElement([nodes2[0], nodes2[1]]))
elems2[-1].SetSection('EulerBernoulli Beam', owt_section1)
elems2.append(s2.CreateElement([nodes2[1], nodes2[2]]))
elems2[-1].SetSection('EulerBernoulli Beam foundation attachments', owt_section2

s2.PlotStructure(plot_elements=True)

f_max2 = 400 #Hz
omega_range2 = np.linspace(1, f_max2 * 2 * np.pi, 1000)
```

```
Assembler 'OWT' successfully initialised
Successfully added element of type: EulerBernoulli Beam to Element 3
Successfully added element of type: EulerBernoulli Beam foundation attachments to
Element 4
```

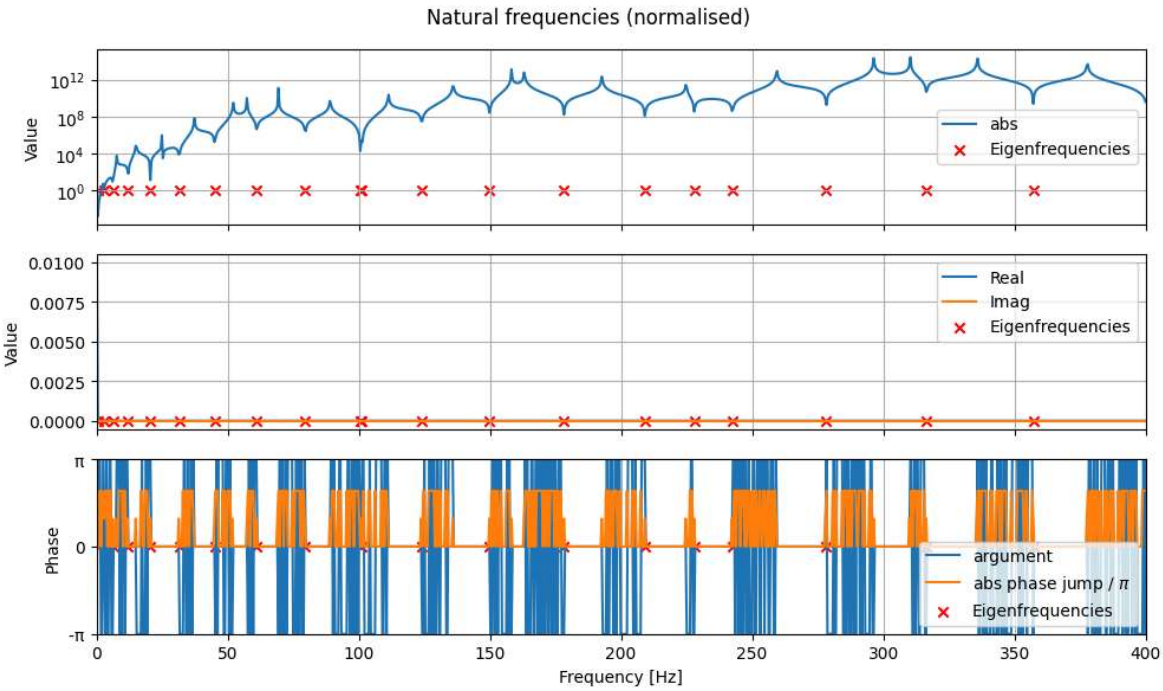


In [13]:

```
#Finding the natural frequencies and visualising that process and the results

s2.run_connectivity()

omega_m2 = np.array(find_eigen_frequencies(omega_range2, s2))
omega_m2 = remove_close_roots(omega_m2)
f_m2 = omega_m2 / (2 * np.pi)

Det_M2 = np.array([det_func(omega, s2) for omega in omega_range2])
arg_dets2 = np.angle(Det_M2)

fig, axs = plt.subplots(3, sharex=True, figsize=(10,6))
axs[0].semilogy(omega_range2 / 2 / np.pi, abs(Det_M2), label='abs')
axs[1].plot(omega_range2 / 2 / np.pi, Det_M2.real/omega_range2**6, label='Real')
axs[1].plot(omega_range2 / 2 / np.pi, Det_M2.imag/omega_range2**6, label='Imag')
axs[2].plot(omega_range2 / 2 / np.pi, arg_dets2, label='argument')
axs[2].plot(omega_range2[:-1] / 2 / np.pi, abs(np.diff(arg_dets2))/np.pi, label=
axs[2].set_ylim([-np.pi, np.pi])

axs[0].scatter(f_m2, np.ones_like(omega_m2), marker='x', c='r', label='Eigenfreq
axs[1].scatter(f_m2, np.zeros_like(omega_m2), marker='x', c='r', label='Eigenfre
axs[2].scatter(f_m2, np.zeros_like(omega_m2), marker='x', c='r', label='Eigenfre
```

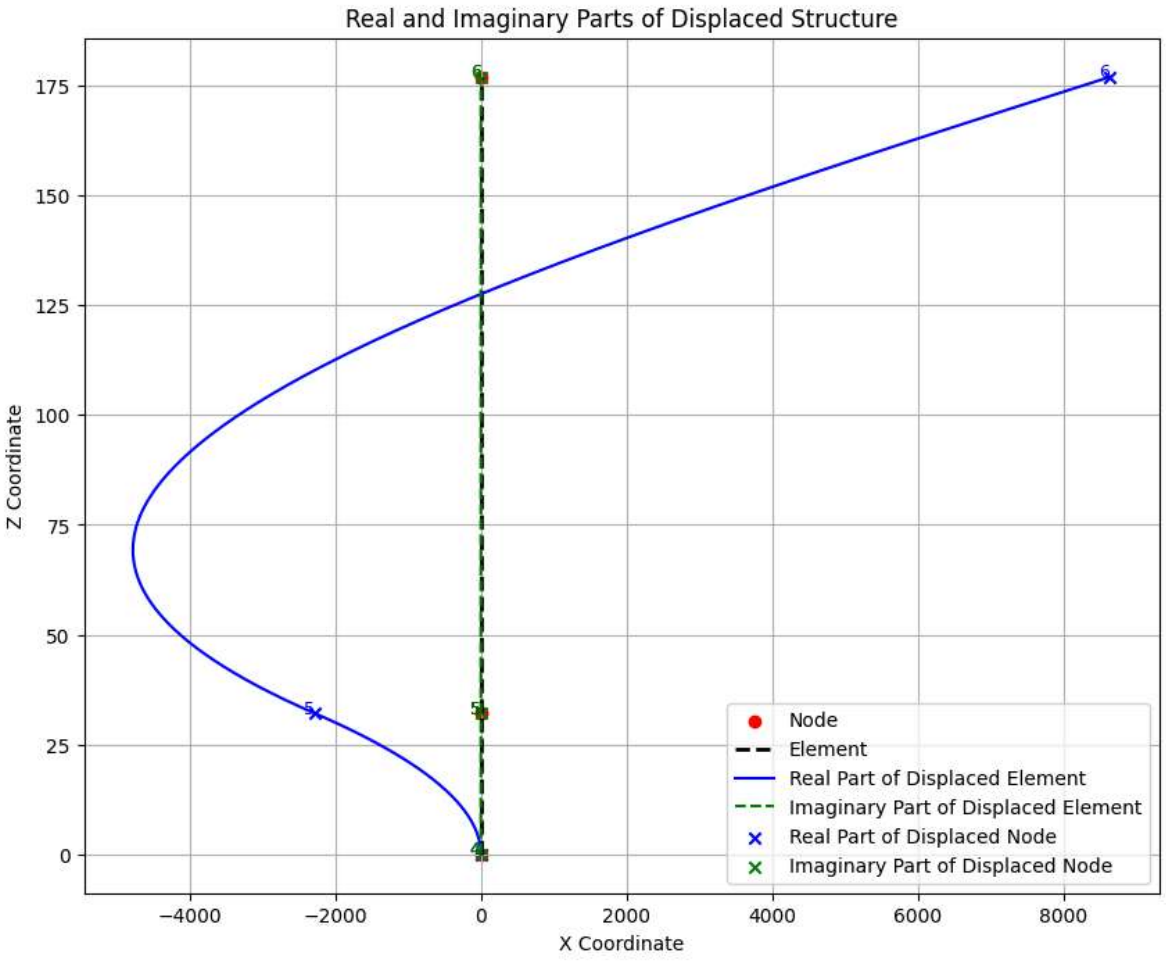
```
for ax in axs:
    ax.grid()
    ax.legend()
    ax.set_xlim([0, f_max2])
axs[0].set_ylabel('Value')
axs[1].set_ylabel('Value')
axs[2].set_ylabel('Phase')
axs[2].set_xlabel('Frequency [Hz]')
axs[2].set_yticks([-np.pi, 0, np.pi], ['-$\\mathrm{\\pi}$', 0, '$\\mathrm{\\pi}$
fig.suptitle('Natural frequencies (normalised)')
plt.tight_layout()

N = 10
print(f'The first {N} natural frequencies are: {f_m2[:N]}')
```

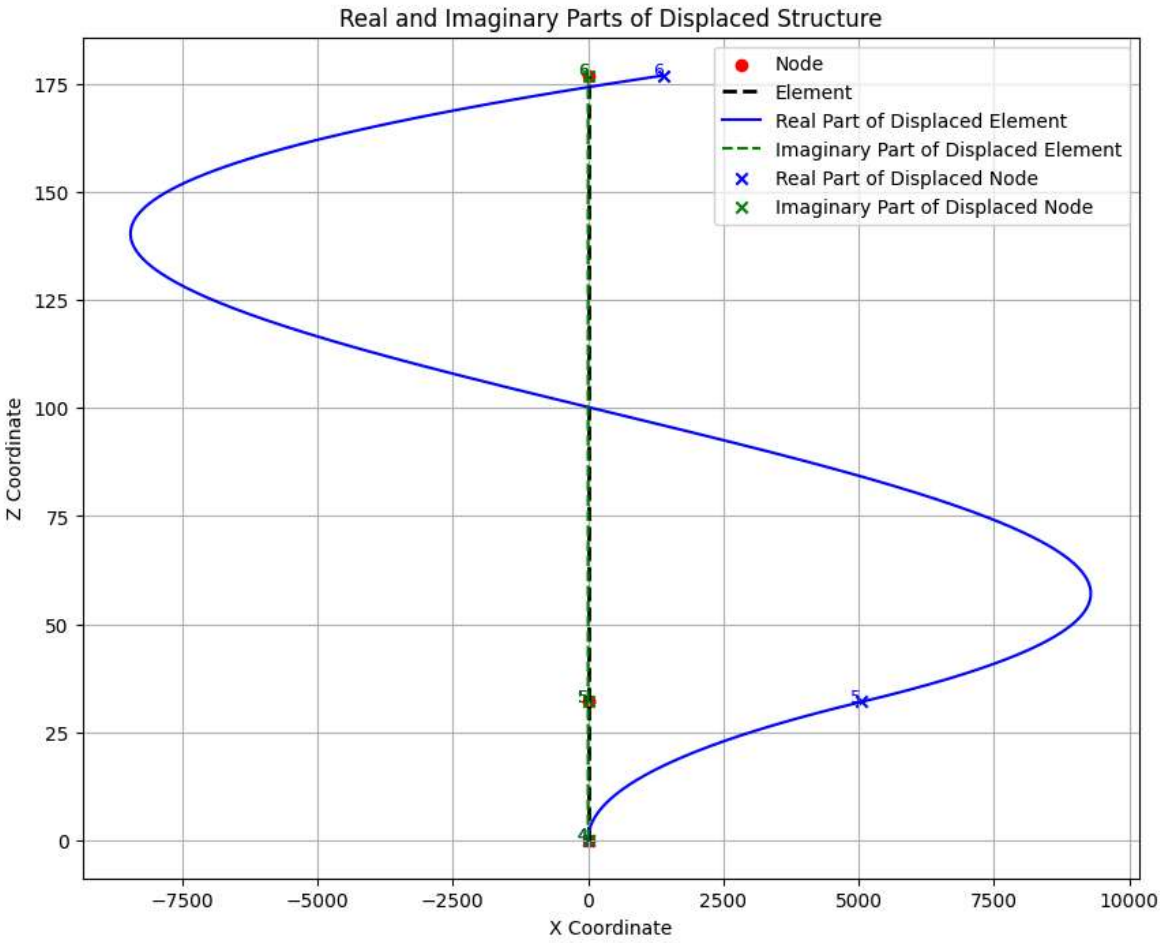
```
The first 10 natural frequencies are: [  0.58428912   3.02026855   6.30726849   1
2.08006804  20.57523125
  31.57105613  45.11912835  61.14139408  79.63774973 100.6064627 ]
```



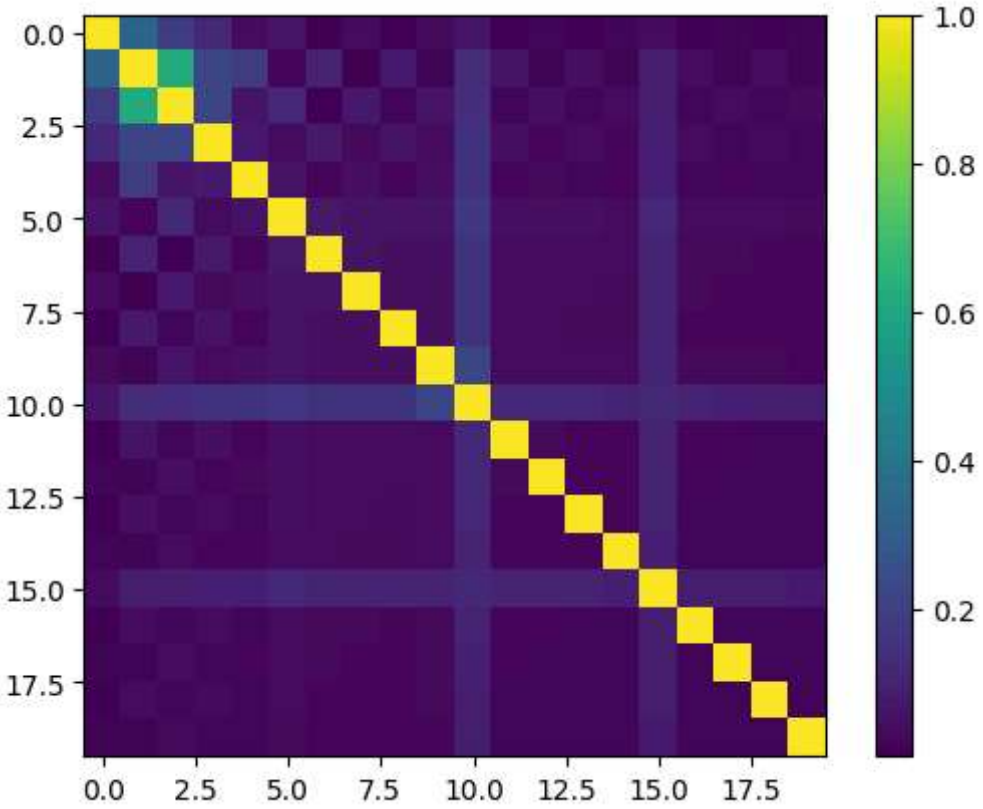
In [14]:
```
#Normalising the modes of vibration and plotting the first 2

#firstly create a dictionary for storing the displacements of each mode
disp_dict = {}
disp_local_dict = {}
num = 8000
for omega in omega_m2:
    Kc_global = s2.GlobalConstrainedStiffness(omega)
    u_free = s2.SolveUfree(Kc_global[:-1,:-1], -Kc_global[:-1,-1])
    u_free = np.concatenate((u_free, np.array([1])))
    u_elem = s2.FullDisplacement(u_free)
    disp = s2.ElementDisplacements(u_elem, omega, num)
    disp_dict[omega] = disp
    # get local displacements
    disp_local = s2.ElementDisplacements(u_elem, omega,num,local_axes=True)
    disp_local_dict[omega] = disp_local

local_coordinates = get_local_coordinates(elems2, num)

def Normalize_modes2(Omega, displacement, displacement_local, local_coordinates,
```

```
    disp_norm = {}
    disp_local_norm = {}
    for omega in Omega:
        Gamma = 0
        for element in elements:
            key = list(element.element_types.keys())[0]
            rho = element.element_types[key].rho
            A = element.element_types[key].A
            Gamma += rho * A * np.trapezoid(displacement_local[omega][element.id
        Gamma += M * displacement_local[omega][element.id][1][-1]**2 #"element"

        disp_norm[omega] = {elem: disp / np.sqrt(Gamma) for elem, disp in displa
        #Local normalised displacements
        disp_local_norm[omega] = {elem: disp / np.sqrt(Gamma) for elem, disp in
    return disp_norm, disp_local_norm

disp_norm, disp_local_norm = Normalize_modes2(omega_m2, disp_dict, disp_local_di

for omega in omega_m2[:2]:
    s2.PlotElementDisplacements(disp_norm[omega], scale=1e7)
```





In [15]:
```
#Checking the orthogonality of the modes

def Check_Orthogonality2(omega, disp_local_norm, local_coordinates, elements):
    N = len(omega)
    Ortho_matrix = np.zeros((N, N), dtype=complex)
    for ii, omega_1 in enumerate(omega):
        for jj, omega_2 in enumerate(omega):
            for element in elements:
                key = list(element.element_types.keys())[0]
                rho = element.element_types[key].rho
                A = element.element_types[key].A
                Ortho_matrix[ii, jj] += rho * A * np.trapezoid(disp_local_norm[c
            Ortho_matrix[ii, jj] += M * disp_local_norm[omega_1][element.id][1][
    return Ortho_matrix

Ortho_coeff = Check_Orthogonality2(omega_m2[:30], disp_local_norm, local_coordin

plt.imshow(abs(Ortho_coeff))
plt.colorbar();
```



The stripes in the visualisation of the orthogonality of the modes are due to the point mass M and its moment of inertia J. When only including J, two stripes (9 and 15; both horizontal and vertical) are visible, as well as the high off-diagonal values around the first modes. When only including M, stripes at 9 and 14 (not 15) become visible. When excluding both, a perfect graph is visible like for the layered soil system.

Creating the power spectral densities of the inputs

For the seismic motion, the provided power spectral density (PSD) can be used which is about the acceleration of the bedrock. It will be limited to a maximum frequency of 50 Hz, to avoid long computation times for frequencies that are not expected to create a response in the large and rigid structure. (A motion which happens 50 times per seconds seems unlikely to happen.)

For the power spectral density of the wave force, first a PSD of the wave heights is created using the provided JONSWAP spectrum, which was implemented in a function. For the inertia force according to the Morison equation, it holds that:

$$f_H(z,t) = \frac{1}{4}\pi D^2 \rho_w \frac{\partial u}{\partial t} + \frac{1}{4}\pi D^2 C_a \rho_w \left(\frac{\partial u}{\partial t} - \frac{\partial^2 w_2}{\partial t^2} \right)$$

$$f_H(z,t) = A_t C_M \rho_w \frac{\partial u}{\partial t} - A_t C_a \rho_w \frac{\partial^2 w_2}{\partial t^2}$$

with No description has been provided for this image
$C_M = 1 + C_a$. For the meaning of the other variables, see Exercise 4. The equations are about displaced mass, so the area should be taken as: $\frac{1}{4}\pi(D_{out}^2 - D_{inner}^2) = A_t$ as defined previously. When calculating the significant wave height from the JONSWAP

spectrum and using that to calculate the Keulegan Carpenter (KC) number for deep water (which is assumed, to be able to use the easier formula), according to the formula $KC = \pi \frac{H}{D}$ with H the significant wave height and D the outer diameter of the offshore wind turbine, a KC number of 2.15 comes out. This means that the wave force indeed is inertia dominated so drag can safely be neglected, as was said in the assignment. Using a graph that relates KC numbers with the inertia coefficient C_M, it can be read that for an (assumed) smooth cylinder, the value of C_M is around 2.05. $C_M = 1 + C_a$, and because the coefficient for added mass C_a has an upper bound of 1, it will be given that upper bound value.

First, the JONSWAP spectrum should be converted to a PSD of the wave particle acceleration. The horizontal wave particle velocity and acceleration in regular "undeep" water are given by:

$$u = \zeta_a \omega \frac{\cosh(k(d+z))}{\sinh(kd)} \cos(kx - \omega t)$$

$$\dot{u} = \zeta_a \omega^2 \frac{\cosh(k(d+z))}{\sinh(kd)} \sin(kx - \omega t)$$

with $\zeta_a = \frac{H}{2}$, and H is what is represented by the JONSWAP spectrum. Furthermore, k is the wave number and z the considered depth (z = -1 m is one meter under water). When only considering the amplitude of the wave and not the space and time dependent part, the relation between the wave height H and the amplitude of the particle acceleration $\dot{u}$ reduces to:

$$\dot{u}_a = \frac{H}{2} \omega^2 \frac{\cosh(k(d+z))}{\sinh(kd)}$$

It means that the PSD of the wave particle acceleration is given by $S_{\dot{u}_a} = \alpha_1^2 S_{JS}$ in which $\alpha_1 = \frac{\omega^2}{2}$. The part of the formula with the wave number k is not considered, because it results in overflows in the sinh and cosh function and there was no time to solve this (it was quickly tried using exponential functions and dividing the fraction by certain exponentials to end up with only negative powers, but this did not work; see commented code below). On top of that, it was at first sight unsure how to implement the acceleration that is changing with z (depth). Due to those two facts, a constant unit distributed wave force along the offshore wind turbine will be assumed.

The second step relates this acceleration to the force according to the Morison equation: $S_F = \alpha_2^2 S_{\dot{u}_a}$ with $\alpha_2 = A_t C_M \rho_w$. This S_F will be used in the following calculations and is called `PSD_waves_distr_force` in the code.

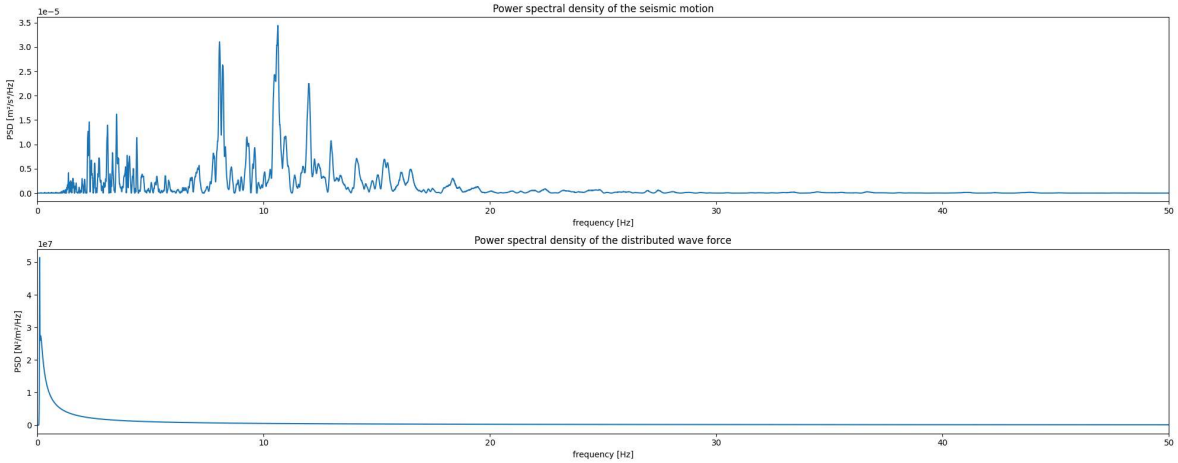
The other part of the inertia force, which is depending on the acceleration of the structure, can be incorporated in the system by increasing the cross-sectional density with $A_t C_a \rho_w$. Keeping the cross-sectional area at A_t, the original mass density should be increased with $C_a \rho_w$.

In [16]:
```
#Calculating the power spectral densities (PSDs) in the correct units for both i
```

```python
#Removing zero frequency value and limit to relevant frequencies
FRF_freq_max = 50
mask = (freq_seism > 0) & (freq_seism < FRF_freq_max)
FRF_freq = freq_seism[mask]
FRF_omega = FRF_freq * 2 * np.pi


#Converting PSD in [g²/Hz] to unit [m²/s⁴/Hz]
PSD_seism = PSD_seism_0[mask] * g**2


PSD_waves_height = JONSWAP(FRF_freq) #[m²/Hz]
alpha_1 = FRF_omega**2 / 2 #[s⁻²]
PSD_waves_acceleration = alpha_1**2 * PSD_waves_height #[m²/s⁴/Hz]

m_0 = np.trapezoid(PSD_waves_height, FRF_freq)
H_s = 4 * np.sqrt(m_0)
KC = np.pi * H_s / D_out
print(f'KC = {KC:.2f} [-]') #-> C_M > 2 for smooth cylinder

C_M = 2.05 #[-]
C_a = np.min([C_M - 1, 1]) #[-]
print(f'C_a = {C_a} [-]')
rho_w = 1025 #[kg/m³]
alpha_2 = A_t * C_M * rho_w #[kg/m]
PSD_waves_distr_force = alpha_2**2 * PSD_waves_acceleration #[kg²/s⁴/Hz] = [kg²m

plt.figure(figsize=(20, 8))
plt.subplot(2, 1, 1)
plt.plot(FRF_freq, PSD_seism)
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD [m²/s⁴/Hz]')
plt.title('Power spectral density of the seismic motion')
plt.xlim(0, FRF_freq_max)

plt.subplot(2, 1, 2)
plt.plot(FRF_freq, PSD_waves_distr_force)
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD [N²/m²/Hz]')
plt.title('Power spectral density of the distributed wave force')
plt.xlim(0, FRF_freq_max)
plt.tight_layout()


# #Calculation of wavenumbers (k)
# def omega_wave_number(k, g, d, omega):
#     return k * g * np.tanh(k * d) - omega**2

# k = np.zeros_like(FRF_omega)
# for i,om in enumerate(FRF_omega):
#     k[i] = fsolve(omega_wave_number, k[i - 1], args=(g, H_w, om))[0]

# diff = omega_wave_number(k, g, H_w, FRF_omega)
# print(len(diff[diff > 1e-10]) == 0)

# k_L = k[500:505] #here the original expression overflows
# k_L = k[869:875] #here the changed expression overflows
# zz = H_w / 2
# print(np.cosh(k_L * (H_w + zz)) / np.sinh(k_L * H_w))
```

```
# print((1 + np.exp(-k_L * (2 * H_w + 2 * zz))) / (np.exp(-k_L * zz) - np.exp(-2
```

```
KC = 2.15 [-]
C_a = 1.0 [-]
```



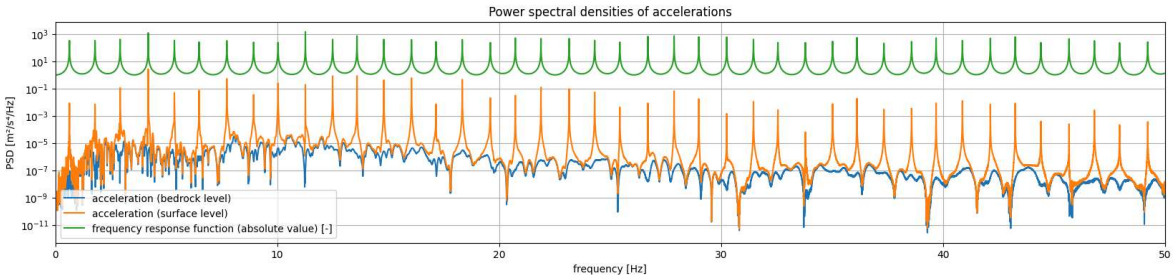
Determination of acceleration at the top of the offshore wind turbine

Only the (translational) acceleration at surface level is considered to transfer the response of the layered soil system to the offshore wind turbine model, because a rotational degree of freedom is not present in the shear beam definition that is used for the layered soil system. Otherwise, the response through surface rotational acceleration could have been added to the found response at the end of this section.

Acceleration at surface level due to seismic motion

In [17]:
```
#Calculating the FRF of unit seismic motion (acceleration) at bedrock level vs.

FRF_seism_surface = np.zeros(len(FRF_omega), dtype=complex)

for i,omega in enumerate(FRF_omega):
    #Adding unit acceleration at bedrock level
    nodes1[-1].prescribe_node(x=1/-omega**2)
    s1.run_connectivity() #this is needed after fixing a node
    Kc_global = s1.GlobalConstrainedStiffness(omega)
    Fc_global = s1.GlobalConstrainedForce(omega)
    u_free = s1.SolveUfree(Kc_global, Fc_global)
    #first free dof (index 0) is the x-displacement of node 1 (index 0)
    FRF_seism_surface[i] = u_free[0] * -omega**2

#Units: [m²/s⁴/Hz] = [m/s²/(m/s²)]² * [m²/s⁴/Hz]
#Units: [m²/s⁴/Hz] = [     -     ]² * [m²/s⁴/Hz]
PSD_seism_surface = np.abs(FRF_seism_surface)**2 * PSD_seism

plt.figure(figsize=(20, 4))
plt.plot(FRF_freq, PSD_seism, label='acceleration (bedrock level)')
plt.plot(FRF_freq, PSD_seism_surface, label='acceleration (surface level)')
plt.semilogy(FRF_freq, np.abs(FRF_seism_surface), label='frequency response func
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD [m²/s⁴/Hz]')
plt.title('Power spectral densities of accelerations')
plt.grid()
```

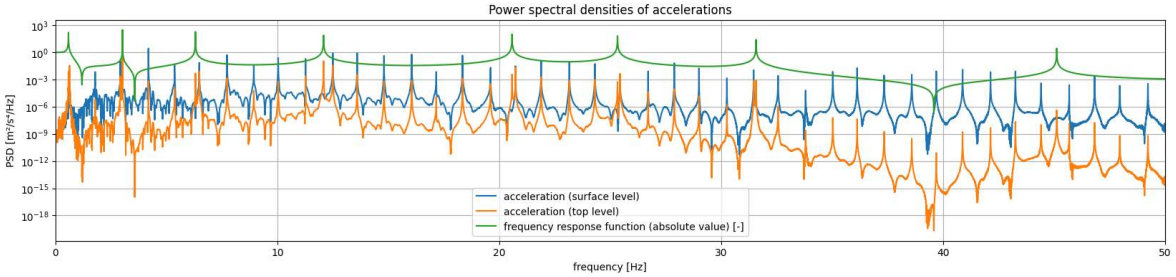
```
plt.legend()
plt.xlim(0, FRF_freq_max);
```



Acceleration at the top due to both loads

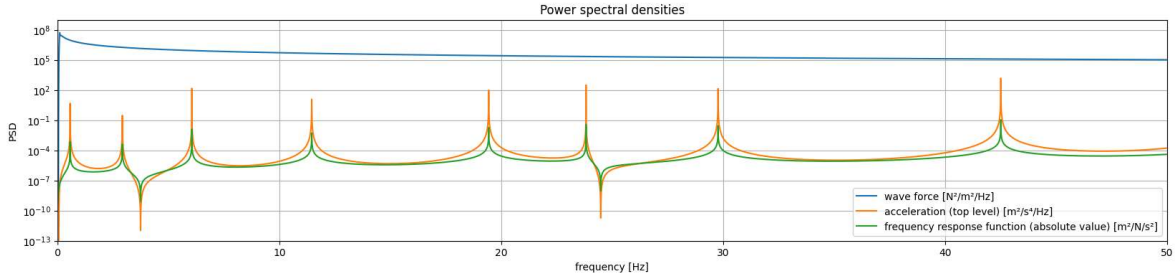
For both loads, a frequency response function can be calculated which relates the load to an acceleration at the top of the offshore wind turbine.

In [18]:
```
#Calculating the FRF of unit seismic motion (acceleration) at surface level vs.

FRF_seism_top = np.zeros(len(FRF_omega), dtype=complex)
FRF_seism_base_M = np.zeros(len(FRF_omega), dtype=complex)

for i,omega in enumerate(FRF_omega):
    #Adding unit acceleration at surface level
    nodes2[0].prescribe_node(x=1/-omega**2)
    s2.run_connectivity() #this is needed after fixing a node
    Kc_global = s2.GlobalConstrainedStiffness(omega)
    Fc_global = s2.GlobalConstrainedForce(omega)
    F_global = s2.GlobalForce(omega)
    u_free = s2.SolveUfree(Kc_global, Fc_global)
    K_global = s2.GlobalStiffness(omega)
    support_reactions = s2.SupportReactions(K_global, u_free, F_global)

    #third free dof (index 2, in total 4 free dofs) is the x-displacement of the
    FRF_seism_top[i] = u_free[2] * -omega**2
    #second support reaction (index 1, in total 2 support reactions) is the bend
    FRF_seism_base_M[i] = support_reactions[1]

#Units: [m²/s⁴/Hz] = [m/s²/(m/s²)]² * [m²/s⁴/Hz]
#Units: [m²/s⁴/Hz] = [     -     ]² * [m²/s⁴/Hz]
PSD_seism_top = np.abs(FRF_seism_top)**2 * PSD_seism_surface

plt.figure(figsize=(20, 4))
plt.plot(FRF_freq, PSD_seism_surface, label='acceleration (surface level)')
plt.plot(FRF_freq, PSD_seism_top, label='acceleration (top level)')
plt.semilogy(FRF_freq, np.abs(FRF_seism_top), label='frequency response function
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD [m²/s⁴/Hz]')
plt.title('Power spectral densities of accelerations')
plt.grid()
plt.legend()
plt.xlim(0, FRF_freq_max);
```



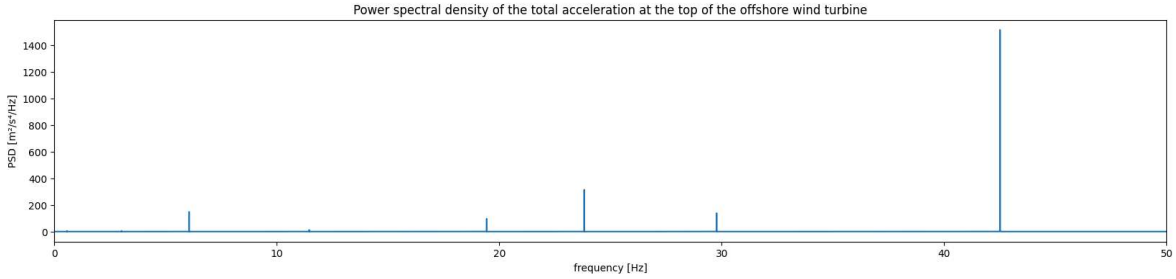
In [19]:
```
#Calculating the FRF of unit wave load (distributed force) along the submerged p

FRF_waves_top = np.zeros(len(FRF_omega), dtype=complex)
FRF_waves_base_M = np.zeros(len(FRF_omega), dtype=complex)

#Adding a unit distributed force along the lower half of the offshore wind turbi
elems2[0].AddDistributedLoad(z=1)
#... and removing the unit acceleration at the bottom
nodes2[0].prescribe_node(x=0)

owt_section1_added_mass = owt_section1.copy()
owt_section1_added_mass['rho'] += C_a * rho_w
owt_section2_added_mass = owt_section2.copy()
owt_section2_added_mass['rho'] += C_a * rho_w

elems2[0].SetSection('EulerBernoulli Beam', owt_section1_added_mass)
elems2[1].SetSection('EulerBernoulli Beam foundation attachments', owt_section2_

s2.run_connectivity() #this is needed after adding a load and changing geometry

for i,omega in enumerate(FRF_omega):
    Kc_global = s2.GlobalConstrainedStiffness(omega)
    Fc_global = s2.GlobalConstrainedForce(omega)
    F_global = s2.GlobalForce(omega)
    u_free = s2.SolveUfree(Kc_global, Fc_global)
    K_global = s2.GlobalStiffness(omega)
    support_reactions = s2.SupportReactions(K_global, u_free, F_global)
    #third free dof (index 2, in total 4 free dofs) is the x-displacement of the
    FRF_waves_top[i] = u_free[2] * -omega**2
    #second support reaction (index 1, in total 2 support reactions) is the bend
    FRF_waves_base_M[i] = support_reactions[1]

#Units: [m²/s⁴/Hz] = [m/s²/(N/m)]² * [N²/m²/Hz]
#Units: [m²/s⁴/Hz] = [  m²/N/s²  ]² * [N²/m²/Hz]
PSD_waves_top = np.abs(FRF_waves_top)**2 * PSD_waves_distr_force

plt.figure(figsize=(20, 4))
plt.plot(FRF_freq, PSD_waves_distr_force, label='wave force [N²/m²/Hz]')
plt.plot(FRF_freq, PSD_waves_top, label='acceleration (top level) [m²/s⁴/Hz]')
plt.semilogy(FRF_freq, np.abs(FRF_waves_top), label='frequency response function
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.title('Power spectral densities')
plt.grid()
plt.legend()
plt.xlim(0, FRF_freq_max)
plt.ylim(10e-14, 10e8);
```

```
Successfully added element of type: EulerBernoulli Beam to Element 3
Successfully added element of type: EulerBernoulli Beam foundation attachments to
Element 4
```



In [20]:
```
#Calculating and plotting the resulting PSD of the acceleration at the top of th

#Units: [m²/s⁴/Hz] = [m²/s⁴/Hz] + [m²/s⁴/Hz]
PSD_top = PSD_seism_top + PSD_waves_top

plt.figure(figsize=(20, 4))
plt.plot(FRF_freq, PSD_top)
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD [m²/s⁴/Hz]')
plt.title('Power spectral density of the total acceleration at the top of the of
plt.xlim(0, FRF_freq_max)

std_top = np.sqrt(1 / np.pi * np.trapezoid(PSD_top, FRF_omega))
print(std_top)
```

```
4.264874715226855
```



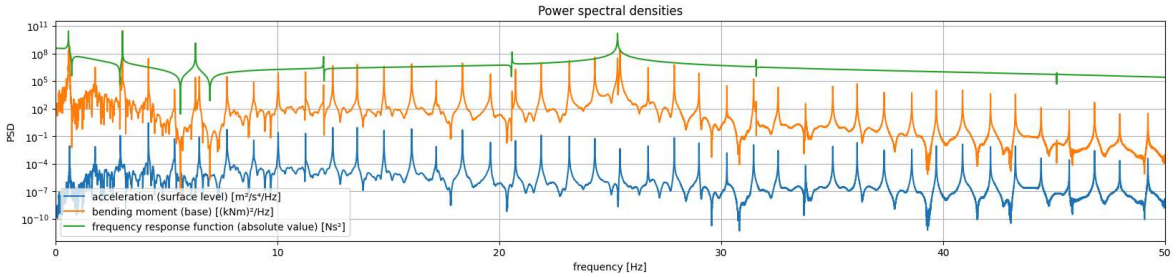
Determination of bending moment at the base of the offshore wind turbine

Bending moment at the base due to both loads

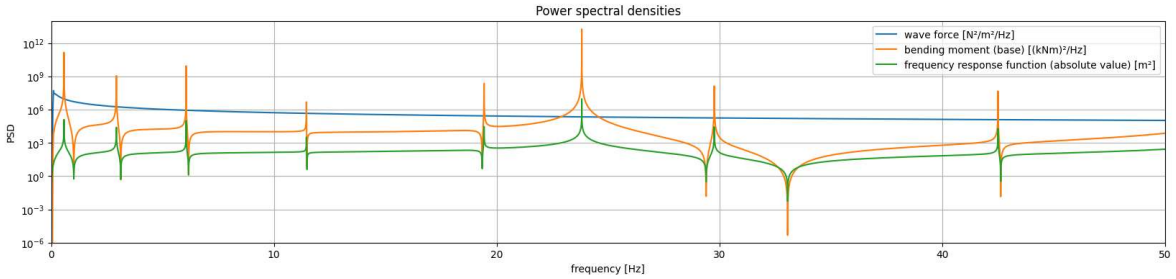
For both loads, a frequency response function can be calculated which relates the load to a bending moment at the base of the offshore wind turbine. To save computation time and changing the unit excitations back and forth between acceleration and distributed force, the FRF and resulting PSD were already computed above.

In [21]:
```
#Calculating and plotting the PSD of the response of the bending moment at the b

#Units: [(kNm)²/Hz] = [Nm/(m/s²)]² * [m²/s⁴/Hz] * [kN/N]²
#Units: [(kNm)²/Hz] = [   Ns²   ]² * [m²/s⁴/Hz] * [kN/N]²
PSD_seism_base_M = np.abs(FRF_seism_base_M)**2 * PSD_seism_surface * (1e-3)**2 #

plt.figure(figsize=(20, 4))
plt.plot(FRF_freq, PSD_seism_surface, label='acceleration (surface level) [m²/s⁴
plt.plot(FRF_freq, PSD_seism_base_M, label='bending moment (base) [(kNm)²/Hz]')
plt.semilogy(FRF_freq, np.abs(FRF_seism_base_M), label='frequency response funct
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.title('Power spectral densities')
```

```
plt.grid()
plt.legend()
plt.xlim(0, FRF_freq_max);
```

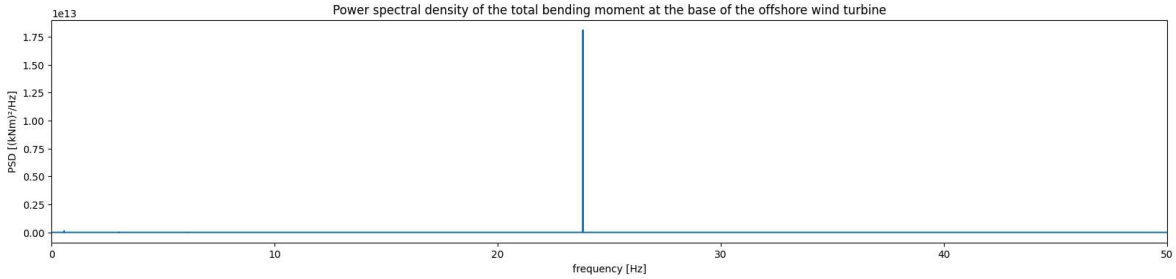


In [22]:
```
#Calculating and plotting the PSD of the response of the bending moment at the b

#Units: [(kNm)²/Hz] = [Nm/(N/m)]² * [N²/m²/Hz] * [kN/N]²
#Units: [(kNm)²/Hz] = [   m²   ]² * [N²/m²/Hz] * [kN/N]²
PSD_waves_base_M = np.abs(FRF_waves_base_M)**2 * PSD_waves_distr_force * (1e-3)*

plt.figure(figsize=(20, 4))
plt.plot(FRF_freq, PSD_waves_distr_force, label='wave force [N²/m²/Hz]')
plt.plot(FRF_freq, PSD_waves_base_M, label='bending moment (base) [(kNm)²/Hz]')
plt.semilogy(FRF_freq, np.abs(FRF_waves_base_M), label='frequency response funct
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD')
plt.title('Power spectral densities')
plt.grid()
plt.legend()
plt.xlim(0, FRF_freq_max)
plt.ylim(10e-7, 10e13);
```



In [23]:
```
#Calculating and plotting the resulting PSD of the bending moment at the base of

#Units: [(kNm)²/Hz] = [(kNm)²/Hz] + [(kNm)²/Hz]
PSD_base_M = PSD_seism_base_M + PSD_waves_base_M

plt.figure(figsize=(20, 4))
plt.plot(FRF_freq, PSD_base_M)
plt.xlabel('frequency [Hz]')
plt.ylabel('PSD [(kNm)²/Hz]')
plt.title('Power spectral density of the total bending moment at the base of the
plt.xlim(0, FRF_freq_max)

std_base = np.sqrt(1 / np.pi * np.trapezoid(PSD_base_M, FRF_omega))
print(std_base)
```

```
373685.8566200594
```



Responses: mean and standard deviation

For both acceleration at the top and bending moment at the base of the offshore wind turbine, it is a priori known that the mean is zero. This is because the both forces have a zero value at zero frequency, meaning that there is no static part of the forcing and only (dynamic) oscillations around the zero mean. This can be seen in the PSDs in the Defining the forcing section. On top of that, in the time history of the seismic motion a zero mean can be observed.

The standard deviations were computed above and are 4.3 m/s² for the top acceleration and 374 MNm for the bending moment at the base.

Exceedance of safety thresholds

For the maximum allowable acceleration at the top of the offshore wind turbine, a value of 0.2 [g] = 1.96 [m/s²] is given. When using the calculated standard deviation and a normal distribution, the probability of exceedance of an acceleration of 1.96 [m/s²] is found as 65%.

For the maximum allowable bending/normal stress at the base of the offshore wind turbine, a value of 80% of 355 [N/mm²] is given. When calculating the probability of exceedance of this yield limit stress, first the static load of $(M + \rho_t \cdot A_t \cdot (H_w + H_t)) \cdot g$ that causes a static normal stress should be deducted from the limit value. Furthermore, the bending moment should be transformed into a stress. This is done using the formula:

$$\sigma = \frac{M_z z}{I_{zz}}$$

in which M_z, the bending moment about the z axis [Nm], is the found standard deviation, $z = \frac{D_{out}}{2}$ is the distance from the gravity center of the cross section to the outmost point [m] and I_{zz} is the moment of inertia [m⁴].

Then, using this standard deviation and a normal distribution, it can be found that the probability of exceedance of a stress of 80% of 355 = 284 [N/mm²] is 0.2%.

For both calculations, a factor 2 is used to account for the fact that exceeding negative -0.2 [g] and -284 [N/mm²] also results in failure.

In [24]:
```
print(f'acceleration: {2 * (1 - norm(0, std_top).cdf(0.2 * g)) * 100:.3f}%')

weight = (M + rho_t * A_t * (H_w + H_t)) * g #[N]
normal_stress_weight = weight / A_t * 1e-6 #[N/mm²]
```

```
print(normal_stress_weight)

std_base_sigma = (std_base * 1e3) * (D_out / 2) / I_t * 1e-6
print(std_base_sigma)
print(f'yield stress: {2 * (1 - norm(0, std_base_sigma).cdf(0.80 * 355 - normal_
```

```
acceleration: 64.549%
17.68136415264938
83.99835552725808
yield stress: 0.152%
```

Assumption of stationarity and ergodicity

Stationarity means that the statistical properties of the process do not change over time, e.g. the process is time invariant.

In the time series of both horizontal components of a Groningen earthquake (taken from the Matlab file `Ex01_Signals_comparison.m` from Lecture 5), it can be seen that the processes certainly are not time invariant: the amplitudes and thus variances (a statistical property) change over time. Amplitudes clearly increase at the arrival of surface waves around t=5 seconds.

 No description has been provided for this image

On the other hand, wide-sense stationarity, as is mentioned in the assignment, is a correct assumption for Groningen earthquakes. They require a constant mean and a correlation function that only depends on the time difference between two considered time instances (Stanley Chan, ECE 302: Lecture A.5 Wide Sense Stationary Processes, 2020). In fact, the mean is constant with value 0, while the correlation can be assumed to only depend on the time difference.

Ergodicity means that two collections of the random variable far apart in the sequence are practically independent (Eric Zivot, 4 Time Series Concepts, 2022). Or, in other words, one sufficiently long realisation is assumed to contain information about the entire process (Andrei Metrikine, Random vibrations L1, 2025). In general, this assumption cannot be made, as the process is not very long and only a part of the time series of the ground acceleration above does not contain information about the entire time series.

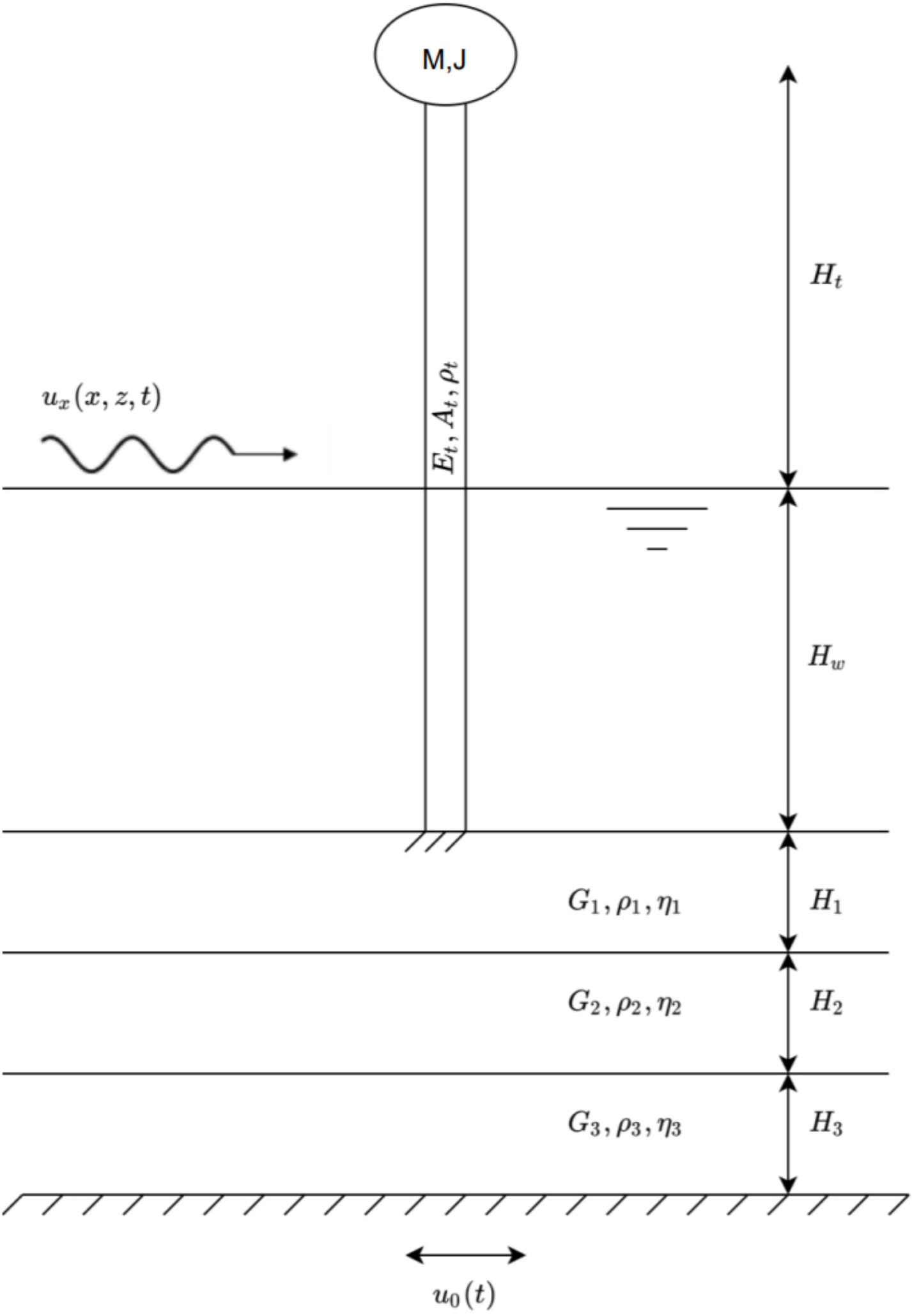
Correlation of vertical seismic excitation

If one would also consider seismic excitation acting in the vertical direction at the base of the offshore wind turbine, one could assume that the horizontal and vertical PSDs of the seismic motion are uncorrelated in the case of Groningen earthquakes.

This can best be shown using the Matlab file `Ex02_Fourier.m` from Lecture 5, in which the PSDs of North-South motion (major principal axis) and the vertical motion can be compared. For the measurements at different stations in the province of Groningen, e.g. FRB2 (Froombosch-2), HKS (Hoeksmeer) and STDM (Stedum), it can clearly be seen that frequencies at which relatively large peaks in one spectrum occur do not come with a (relatively) large amplitude in the other spectrum. As an example, the Hoeksmeer

measurement is shown (left: North-South, right: vertical). The meanings of the abbreviations of the locations were taken from Table 2 on page 17 of "Validatie van het seismisch netwerk van het KNMI in Groningen", available for download here.

 No description has been provided for this image

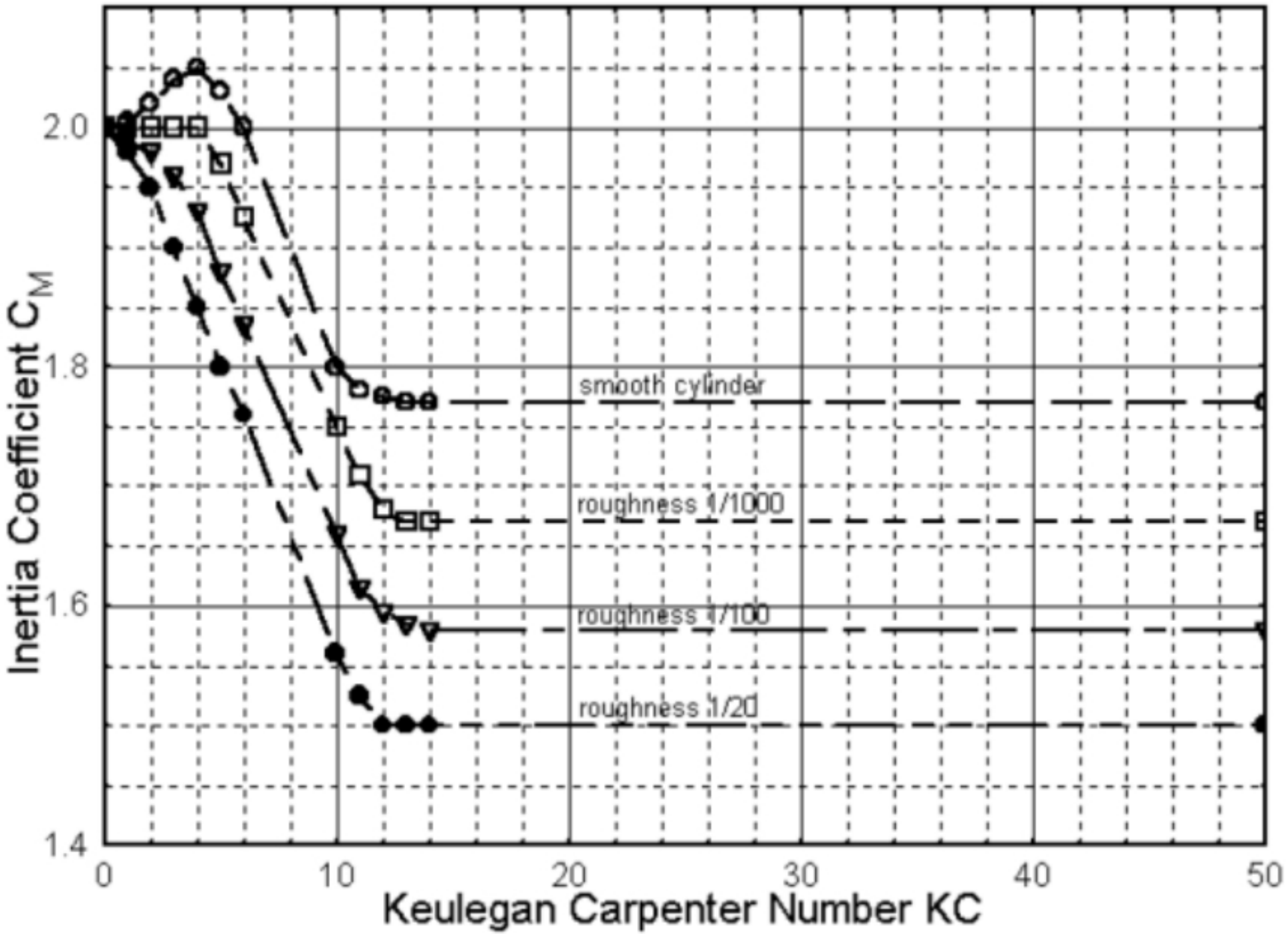

M,J
H_t
E_t, A_t, ρ_t
$u_x(x,z,t)$
H_w
G_1, ρ_1, η_1
H_1
G_2, ρ_2, η_2
H_2
G_3, ρ_3, η_3
H_3
$u_0(t)$

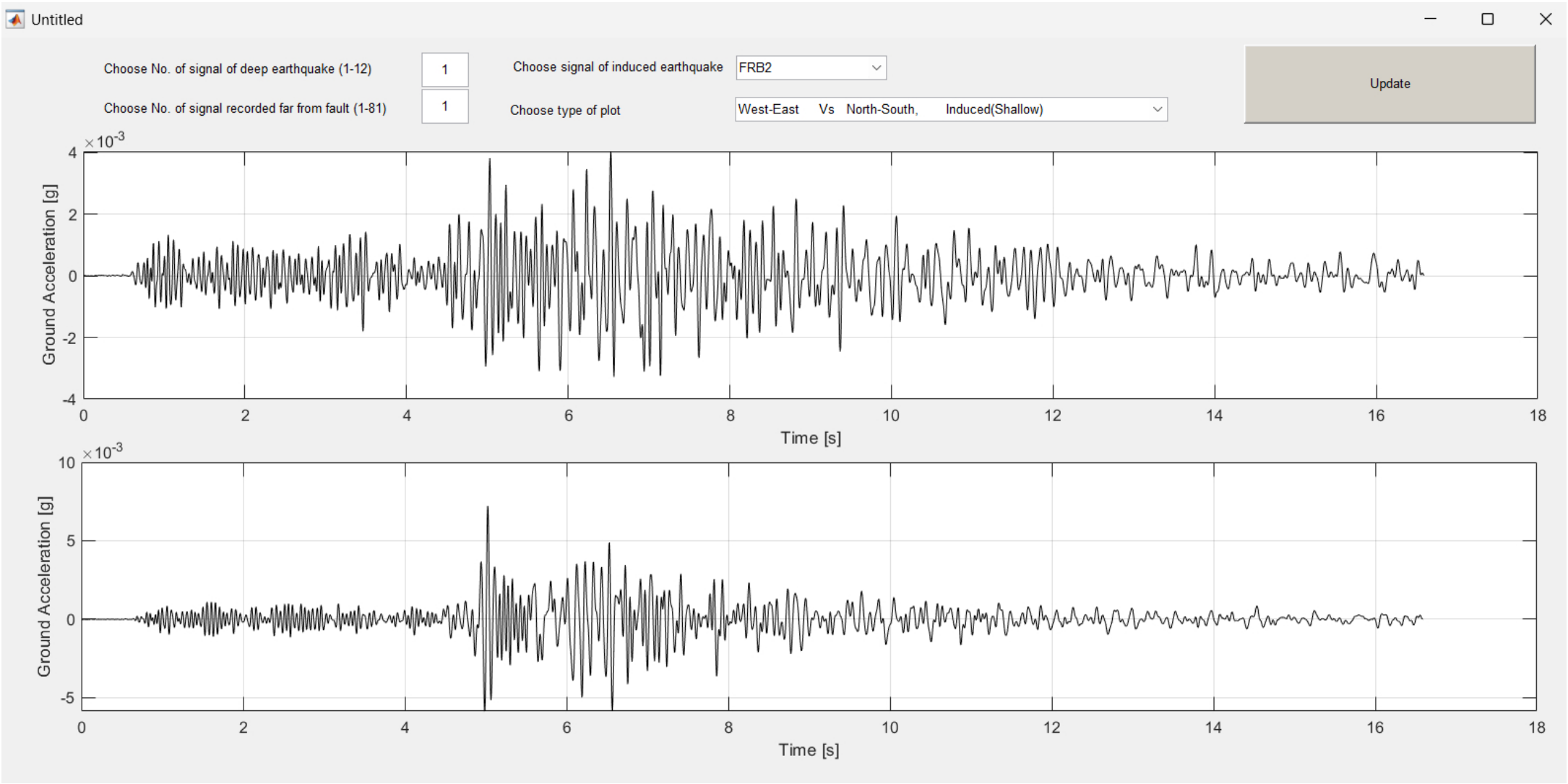
$$M \frac{\partial^2 w}{\partial t^2} = -V$$

$$M \frac{\partial^2 w}{\partial t^2} = EI \frac{\partial^3 w}{\partial z^3}$$

$$J \frac{\partial^2 \varphi}{\partial t^2} = -M$$

$$J \frac{\partial^2 \varphi}{\partial t^2} = EI \frac{\partial^2 w}{\partial x^2}$$


Inertia Coefficient C_M
2.0
1.8
1.6
1.4
0
10
20
30
40
50
Keulegan Carpenter Number KC
smooth cylinder
roughness 1/1000
roughness 1/100
roughness 1/20


Untitled
Choose No. of signal of deep earthquake (1-12)
1
Choose No. of signal recorded far from fault (1-81)
1
Choose signal of induced earthquake
FRB2
Choose type of plot
West-East Vs North-South, Induced(Shallow)
Update
×10-3
Ground Acceleration [g]
Time [s]
×10-3
Ground Acceleration [g]
Time [s]

Untitled

Choose No. of signal of deep earthquake (1-12) 1

Choose No. of signal recorded far from fault (1-81) 1

Choose signal of induced earthquake HKS

Choose type of plot North-South Vs Vertical, Induced(Shallow)

Update