

# FINITE ELEMENT METHOD - Assignment 5

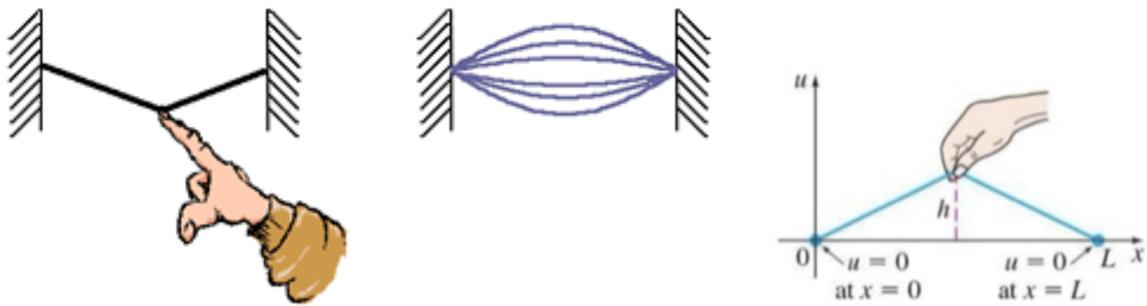
(Due by 11:59 pm on 2023/11/27 Mon.)

陳啟璋 (Chen, Chi-Wei)

Student ID: r11521614

## Problem 1

A nylon string of length  $L = 5\text{cm}$  and density  $\rho = 1.14\text{g/cm}^3$ , fixed at both ends with tension  $P = 10\text{N}$ , is plucked at its midpoint. The initial displacement distribution of the string is denoted as  $u(x, t = 0)$  function as shown below. Then, the string is released without initial velocities:



**1.1 Please determine the subsequent motion history of the string  $u(x, t)$  by your FEM code.**

### 1.1.1 初始條件和材料性質

- 繩子長度  $L = 5\text{ cm}$
- 繩子密度  $\rho = 1.14 \text{ g/cm}^3$
- 繩子固定在兩端的張力  $P = 10 \text{ N}$
- 初始提升的高度  $h = 0.01 \text{ m}$
- 繩子寬度  $w = 0.0001 \text{ m}$
- 斷面截面積  $A = 10^{-8} \text{ m}^2$
- 楊氏係數  $E = 125000 \text{ GPa}$ 
  - 由應力應變推得：

$$\epsilon = \frac{\Delta L}{L} = \frac{\sqrt{(L/2)^2 + h^2} - L/2}{L/2} = 8 \times 10^{-6}$$

$$\sigma = \frac{P}{A} = \frac{10}{10^{-8}} = E\epsilon$$

$$E = 1.25 \times 10^{14}$$

```
*PARAMETER  
num-dim: 2  
*MATPROP  
b-plane-strain: 1
```

```

young's-modulus: 1250000000000000
poisson's-ratio: 0.3
density: 1.14
beta1: 0.5
beta2: 0.5
dt: 0.000001
nstp: 60
nprt: 1
nmod: 5

```

## 1.1.2 網格

- 兩端固定
- 在  $x = 0.025$  m 紿定初始提升的高度  $h = 0.01$  m



```

num-prescribed-disp: 20
node#-dof#-disp:
1 1 0.0
1 2 0.0
2 1 0.0
2 2 0.0
3 1 0.0
3 2 0.0
4 1 0.0
4 2 0.0
5 1 0.0
5 2 0.0
6 1 0.0
6 2 0.0
7 1 0.0
7 2 0.0
8 1 0.0
8 2 0.0
9 1 0.0
9 2 0.0
10 1 0.0
10 2 0.0

```

## 1.1.3 根據題目給定繩子在時間 $t = 0$ 時初始位移分佈

$$u(x, t = 0) = \sum_{n=1}^{\infty} C_n \sin \frac{n\pi x}{L} \cos \frac{nc\pi t}{L}$$

```

L = 0.05; % 繩子的長度
h = 0.01; % 初始提升的高度
t = 0; % 初始時間

% 計算波速 c (根據給定的張力和密度)
c = sqrt(10 / 1.14e3);

% 遍歷所有節點
for i = 1:nnode
    displacements = 0.0; % 初始化位移值

    % 進行級數求和以計算初始位移
    for n = 1:1000
        term = (8 * h / (pi^2 * n^2)) * ... % 每一項的計算
            sin(n * pi / 2) * ...
            sin(n * pi * coor(1, i) / L) * ...
            cos(n * c * pi * t / L);
        displacements = displacements + term; % 將各項加總得到最終位移
    end
end

```

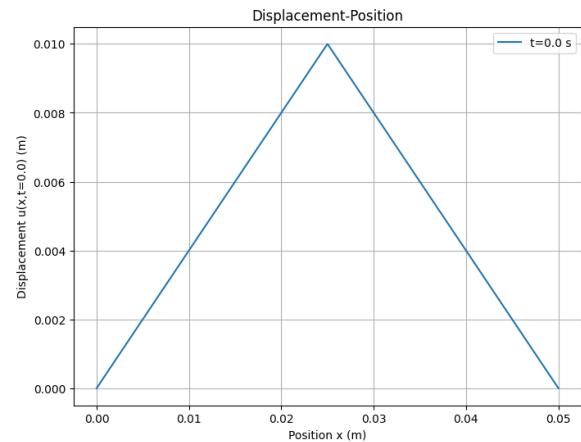
```

    end

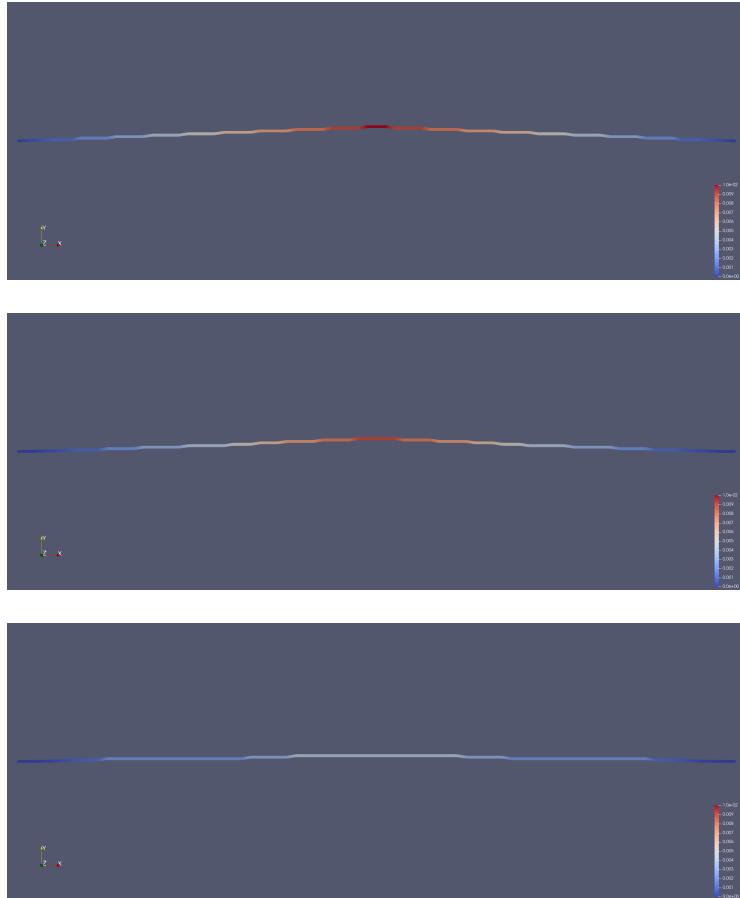
    uc((i - 1) * ndime + 2, 1) = displacements; % 存儲在位移向量中的對應節點的位移值
end

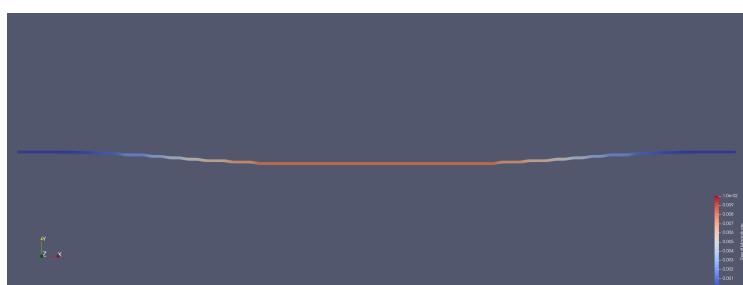
```

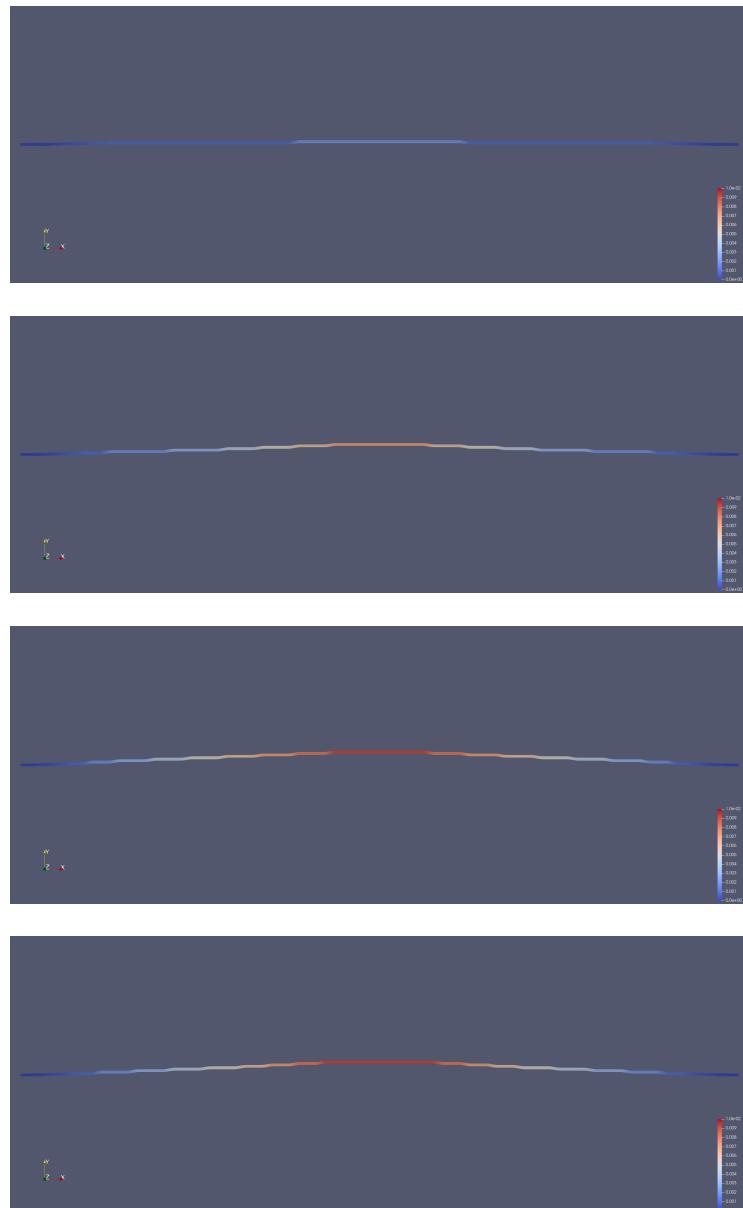
- 這段程式碼中，主要的循環是為每個節點計算初始位移。使用了一個包含多個項的級數求和來模擬繩子的初始位移狀態。這個級數公式是在繩子固定在兩端並且中間有一個高度為  $h$  的提升時的自由振動情況下對位移的解析解。每個節點的位移值是通過將多個項相加求和而得到的。
- 這段程式碼中使用的公式在  $t = 0$  時的特定結果，這種情況下繩子的振動模式和初始位移是通過此公式給出的，並使用 `coor(1, i)` 來表示每個節點的位置對應於繩子的空間坐標。
- 最終，將各節點的初始位移存入 `uc` 位移向量中。



#### 1.1.4 使用 ParaView 可視化 Newmark Method 計算隨時間變化之位移值 $u(x, t)$







**1.2 The exact solution of  $u(x, t)$  is:**

$$u(x, t) = \sum_{n=1}^{\infty} C_n \sin \frac{n\pi x}{L} \cos \frac{nc\pi t}{L}$$

where  $C_n = \frac{8h}{\pi^2 n^2} \sin \frac{n\pi}{2}$  when  $n$  is an odd number ( $C_n = 0$  when  $n$  : even number),  $c$  is  $\sqrt{P/\rho}$ .

$$u(x, t) = \sum_{n=1}^{\infty} \frac{8h}{\pi^2 n^2} \sin \frac{n\pi}{2} \sin \frac{n\pi x}{L} \cos \frac{nc\pi t}{L}, c = \sqrt{\frac{10}{1.14 \times 10^3}}$$

### 1.2.1 Python code to calculate exact solution

這段 Python 程式碼是用來計算並可視化繩子在不同時間下的解析解。它利用了給定的精確解的公式來計算繩子的位移隨時間和位置的變化，並使用 Matplotlib 來繪製對應的位移-位置圖。

在程式中：

- `calculate_displacement` 函數計算了在指定位置 `x` 和時間 `t` 下的位移值，利用給定的精確解公式進行求和，可根據 `num_terms` 參數控制求和的項數。

- `L_value` 和 `h_value` 分別代表繩子的長度和提升的高度。
- `num_terms_to_sum` 參數用於控制在求和計算中使用的項數。
- `t_values` 是時間的範圍，定義了想要計算和展示精確解析解的時間點。
- `x_values` 則是在繩子長度範圍內生成的位置值，用於計算並顯示在不同位置下的位移。

程式將根據所選擇的時間值，在每個時間點上計算繩子在不同位置的位移，並將結果用折線圖顯示出來。每條曲線代表了不同時間下繩子位移隨位置的變化，使得可以直觀地觀察到繩子振動的情況。

```

import matplotlib.pyplot as plt
import numpy as np

def calculate_displacement(x, t, L, h, num_terms):
    c = np.sqrt(10 / 1.14e3)
    displacement = 0.0
    for n in range(1, num_terms + 1):
        term = (8 * h / (np.pi**2 * n**2)) * \
            np.sin(n * np.pi / 2) * \
            np.sin(n * np.pi * x / L) * \
            np.cos(n * c * np.pi * t / L)
        displacement += term
    return displacement

# Parameters
timeStep = 6e-2
L_value = 0.05 # m
h_value = 0.01 # m
num_terms_to_sum = 1000 # Change this number to sum more terms for a better approximation
t_values = np.arange(0, timeStep*10, timeStep) # Creating t values from 0 to 0.2 seconds

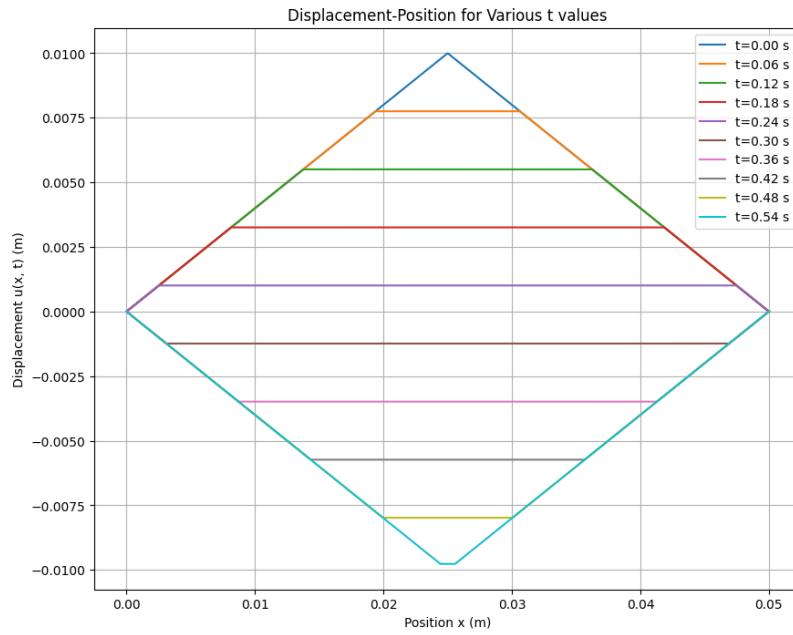
x_values = np.linspace(0, L_value, 1000) # Generating 1000 x values from 0 to L_value

plt.figure(figsize=(10, 8))

for t_value in t_values:
    displacements = [calculate_displacement(x, t_value, L_value, h_value, num_terms_to_sum) for x in x_values]
    plt.plot(x_values, displacements, label=f"t={t_value:.2f} s")

plt.title("Displacement-Position for Various t values")
plt.xlabel("Position x (m)")
plt.ylabel("Displacement u(x, t) (m)")
plt.legend()
plt.grid(True)
plt.show()

```



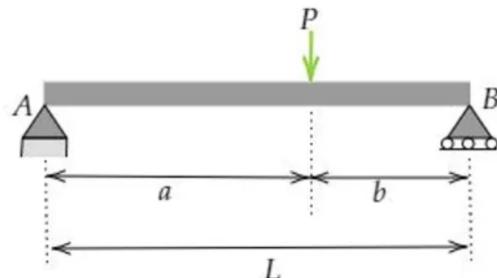
### 1.2.2 Is your solution solved by FEM close to the exact solutions? Please provide your comments.

從 FEM 計算的結果發現，位移與解析解非常接近，但在數值展示上有些微差異。解析解呈現有稜有角的特徵，而 FEM 計算則較為平滑連續。目前猜測這些差異可能來自幾個因素：

1. 網格分辨率：FEM 的結果受到網格分辨率的影響。由於我的網格分得不夠細，可能無法準確捕捉解的細節。
2. 時間積分參數： $\beta_1 = 1/2$  和  $\beta_2 = 1/2$  參數在時間積分方案中扮演著重要角色。不同的參數值可能會影響數值穩定性和解的行為。一開始我使用  $\beta_1 = 1/2$  和  $\beta_2 = 0.5$  結果計算值都無法收斂，最後選用  $\beta_1 = 1/2$  和  $\beta_2 = 1/2$  計算。
3. 數值穩定性：FEM 方法中涉及到的矩陣運算及求逆可能存在數值計算上的限制。特別是在使用 `pinv()` 函數時，由於其對奇異矩陣或具有高條件數的矩陣計算時的敏感性，可能導致解的誤差增加。
4. 通常 FEM 的結果會更接近真實情況且比較平滑，因為它考慮了物理系統的較多細節，而解析解大多是基於簡化模型和理想情況的結果。

## Problem 2

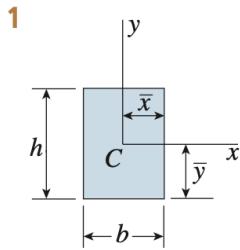
Consider a copper pinned-pinned beam with no initial motion subjected to a harmonic force:  $P(x, t) = P_0 \sin(\omega t)$  at  $x = a$ , as shown below.



### 2.1 Determine the steady motion $u(x, t)$ the beam by your FEM code.

#### 2.1.1 初始條件和一些材料性質

- 楊氏係數  $E = 130 \text{ GPa}$
- 斷面慣性矩  $I = \frac{0.03^4}{12} = 6.75 \times 10^{-8}$



**Rectangle (Origin of axes at centroid)**

$$A = bh \quad \bar{x} = \frac{b}{2} \quad \bar{y} = \frac{h}{2}$$

$$I_x = \frac{bh^3}{12} \quad I_y = \frac{hb^3}{12} \quad I_{xy} = 0 \quad I_P = \frac{bh}{12}(h^2 + b^2)$$

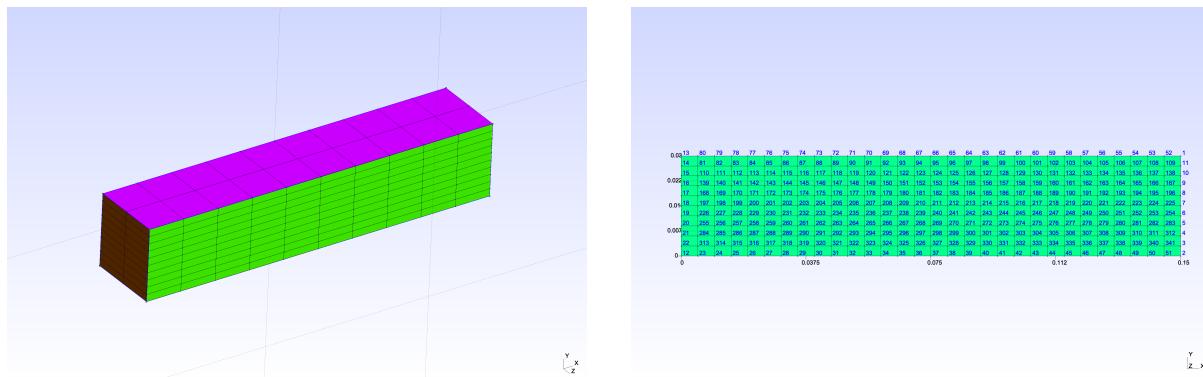
- 密度  $\rho = 8.96 \text{ g/cm}^3$
- 長度  $L = 5 \text{ cm}$
- 寬度  $b = 3 \text{ cm}$
- 高度  $h = 3 \text{ cm}$
- 斷面截面積  $A = 0.03^2 = 9 \times 10^{-4} \text{ m}^2$
- 簡支樑的模態頻率 ( $\omega_n$ ) 為：

$$\omega_n = a\beta_n^2 = a \left( \frac{n\pi}{L} \right)^2 = \sqrt{\frac{EI}{\rho A}} \left( \frac{n\pi}{L} \right)^2$$

- $\omega_n = \sqrt{\frac{EI}{\rho A}} \left( \frac{n\pi}{L} \right)^2 = \sqrt{\frac{130 \times 10^9 \cdot 6.75 \times 10^{-8}}{8.96 \times 10^3 \cdot 0.03^2}} \left( \frac{n\pi}{0.15} \right)^2$
- $\omega_1 = 14469.90$
- $\omega = 0.1 \omega_1 = 1446.99$

```
*PARAMETER
num-dim: 2
*MATPROP
b-plane-strain: 1
young's-modulus: 130000000000
poisson's-ratio: 0
density: 8.96
beta1: 0.5
beta2: 0.5
dt: 0.0001
nstp: 50
npert: 1
nmod: 5
```

## 2.1.2 網格



- 左端 (node=12)：在此節點， $x$  和  $y$  軸的自由度被限制，即固定不動。
- 右端 (node=2)：在此節點， $x$  軸的自由度是可移動的，而  $y$  軸的自由度是被固定的。

```

num-prescribed-disp: 3
node#-dof#-disp:
12 1 0.0
12 2 0.0
2 2 0.0

```

### 2.1.3 根據題目給定簡支梁隨時間變化的受力

- 在  $a = b = L/2$  紿定初始向下之力

$$P(a = L/2, t) = P_0 \sin(\omega t)$$

- 首先設定了初始條件。其中，使用了週期性的正弦函數  $P(a = L/2, t) = P_0 \sin(\omega t)$ ，表示在樑中心施加一個以時間為變數的向下力。

```

%% 初始位移 u(x,t=0)
% P(a,t=0) = P0*sin(wt) = 0
t = 0;
P0 = -10; % Prescribed force value
w1 = 14469.9;
w = 0.1*w1;

```

- 接下來，程式進入 Newmark 方法的計算。在每個時間步驟中，程式會更新時間  $t$ ，計算出當前時間點的力  $P$ ，並在 `rpres` 中更新該力。這代表將受力情況隨時間變化考慮進模擬中，透過不同時間點的正弦函數值  $P$  來模擬時間變化下的外力作用。

```

%% Newmark Method
for i = 1:nstp

    % Update rpres
    t = t + dt;
    P = P0*sin(w*t);
    % Update rpres inside the loop
    rpres(node * ndime - (ndime - dof)) = P;

    % Remaining computation with Newmark Method...
    % ... (omitted for brevity)
end

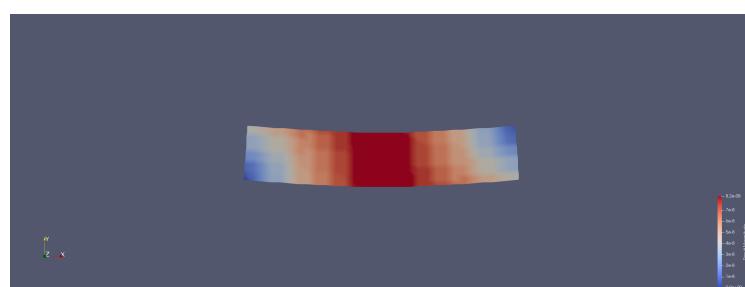
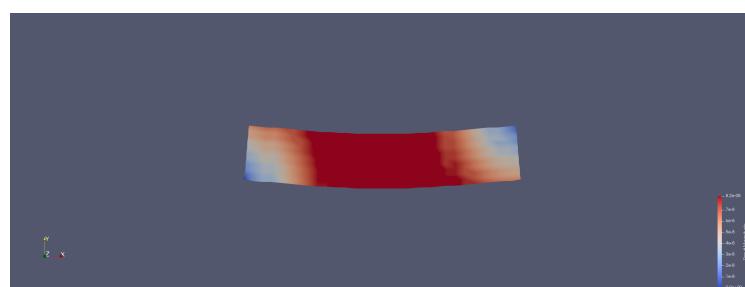
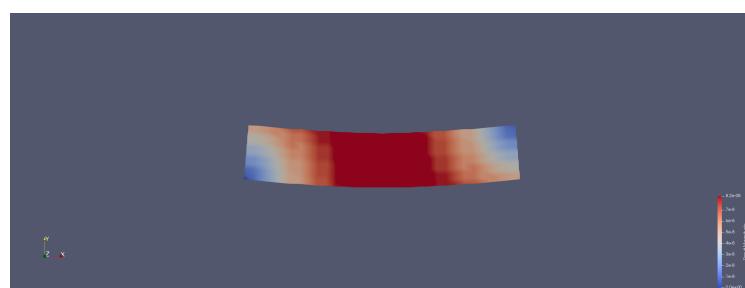
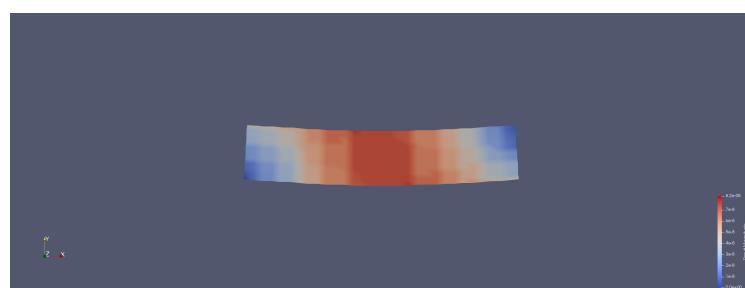
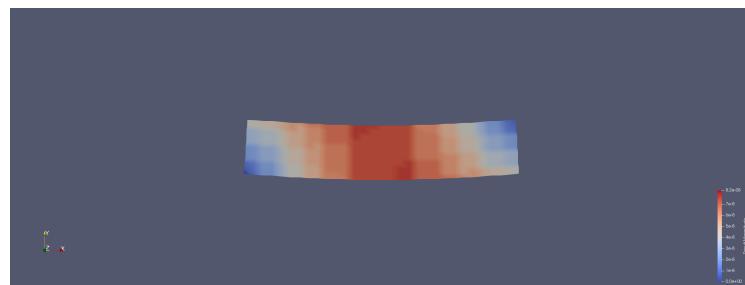
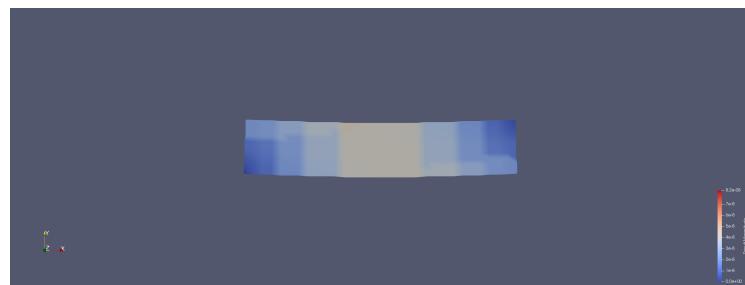
```

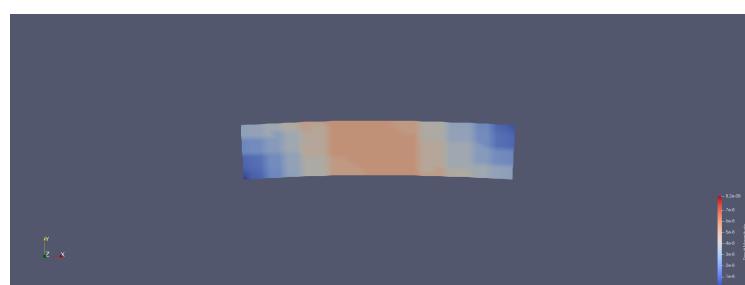
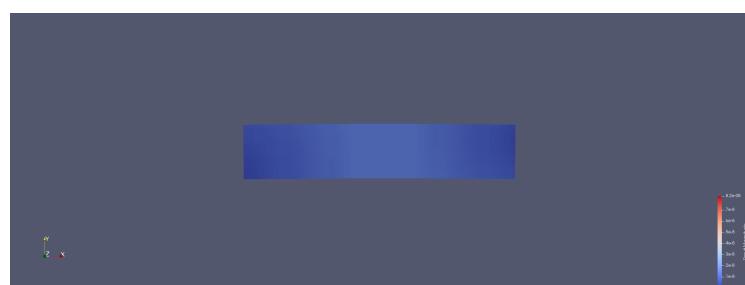
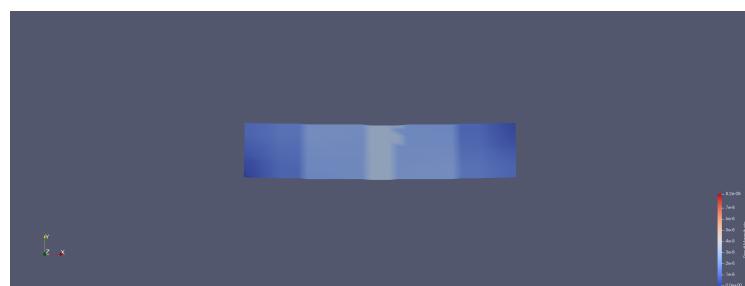
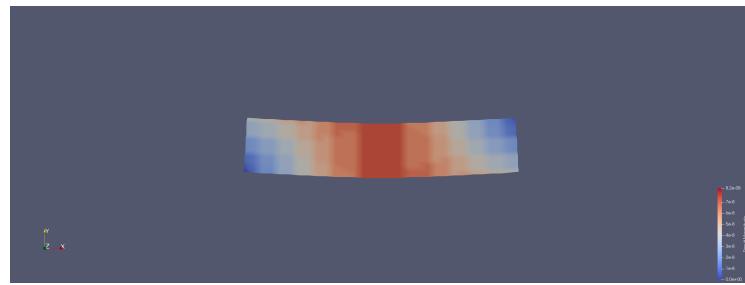
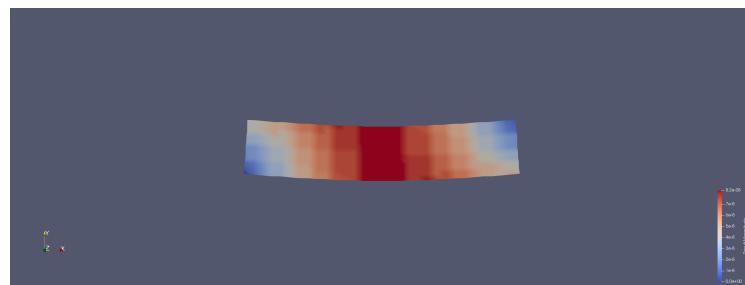
在每個時間步驟中，Newmark 方法用於計算位移、速度和加速度。其中的 `rpres` 會根據當前時間點的力值 `P` 進行更新，以反映在模擬中考慮到時間變化的受力情況。最後，若達到指定的輸出頻率，程式將生成 VTK 文件以保存當前時間點的計算結果。

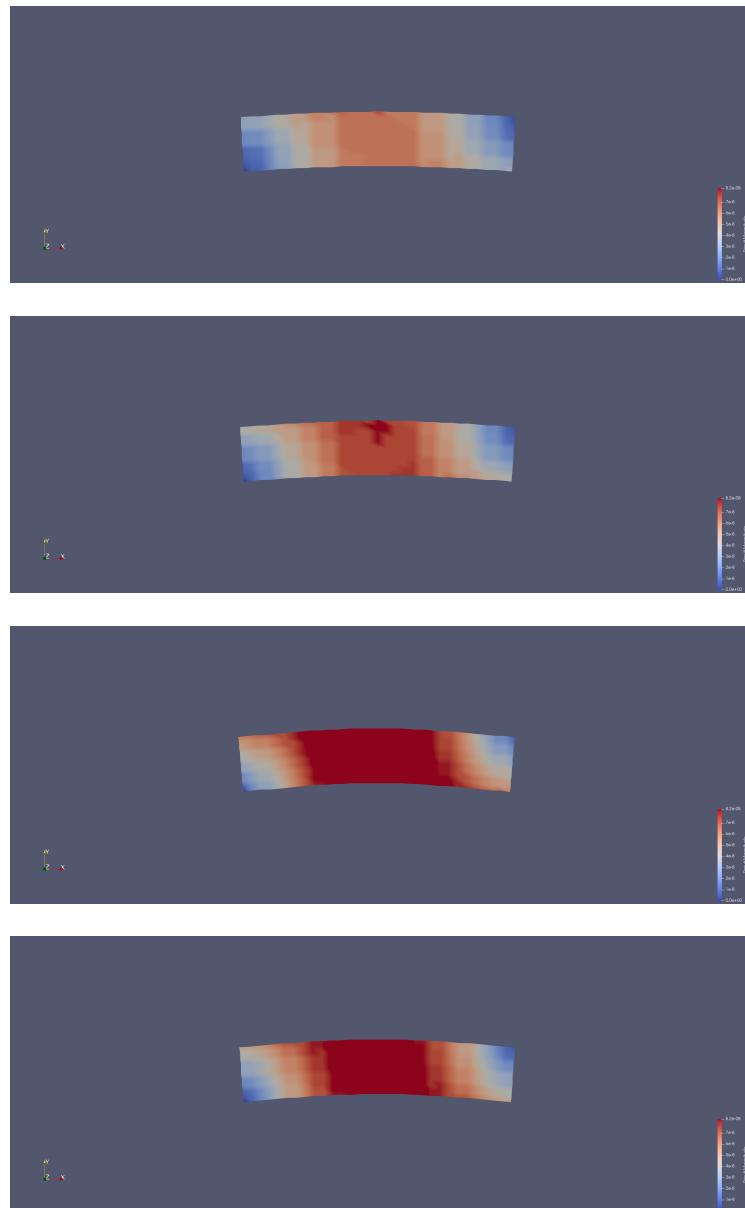
這樣的設計允許模擬在不同時間點受到不同大小的外力影響，透過 `rpres` 的更新，Newmark 方法能夠在計算過程中即時考慮到這些外力的變化，使得計算結果更加真實地反映出受力情況對樑的影響。

### 2.1.4 使用 ParaView 可視化 Newmark Method 計算隨時間變化之位移值 $u(x, t)$









## 2.2 The exact solution of $u(x, t)$ is:

$$u(x, t) = \frac{2P_0}{\rho AL} \sum_{n=1}^{\infty} \frac{1}{\omega_n^2 - \omega^2} \sin \frac{n\pi a}{L} \sin \frac{n\pi x}{L} \sin \omega t$$

where  $\omega_n$  is the natural frequency of the n-th mode of such beam,  $\rho$  is the density of the copper material, and  $A$  is the cross-sectional area of the beam. Please also apply the following data for your analysis:  $P_0 = 10N$ ,  $\omega = 0.1 \omega_1$ , Young's modulus of copper  $E = 130 GPa$ , the mass density of copper  $\rho = 8.96 g/cm^3$ , the width and height of the beam cross-section are both 3 cm, and  $L$  is 15 cm.

### 2.2.1 Python code to calculate exact solution

這段 Python 程式碼主要是用來計算和繪製受力簡支梁的解析解。首先使用了給定的公式，計算了  $u(x, t)$  的精確解，該解釋了在給定梁的材料特性和受力情況下，梁的位移狀態隨時間和位置的變化。

在程式中：

- 定義了所需的物理參數和材料性質，如梁的長度  $L$ 、橫截面寬度和高度、截面積  $A$ 、楊氏模數  $E$ 、密度  $\rho$  等。
- `calculate_omega_n` 函數計算了梁的自然頻率，其中使用了材料的物理特性和梁的幾何尺寸。

- `calculate_displacement_exact_given_frequencies` 函數則按照給定的精確解公式計算了  $u(x, t)$  的解析解。它包含了對不同  $n$  值的和式計算，以表示正弦函數的總和，並隨著位置和時間的變化對位移進行了計算。
- 最後的部分是生成和繪製了不同時間點下，梁在不同位置上的位移狀態。程式迭代計算了給定時間點下的解析解的位移值，並以圖表的方式呈現了這些位移值隨位置變化的情況。

這段程式碼提供了一個基於精確解的模型，以理論方式探討了在不同時間點下梁的位移狀態，同時也提供了一種方式來比較解析解和 FEM 結果之間的差異，從而評估模型的準確性。

```

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Constants and parameters
P0 = -10 # N
E_copper = 130e9 # Young's modulus in Pa
rho_copper = 8.96e3 # Density in kg/m^3
L = 0.15 # Length in meters
width = 0.03 # Width of cross-section in meters
height = 0.03 # Height of cross-section in meters
A = width * height # Cross-sectional area
I = 6.75e-8

def calculate_omega_n(n, E, I, rho, A, L):
    return np.sqrt((E * I) / (rho * A)) * (n * np.pi / L)**2

omega = 0.1 * calculate_omega_n(1, E_copper, I, rho_copper, A, L) # Frequency in terms of omega_1

# Calculate the exact solution for u(x, t) using given natural frequencies
def calculate_displacement_exact_given_frequencies(x, t, num_terms):
    displacement = 0.0
    a = L/2
    for i in range(1, num_terms + 1):
        term = 1 / (calculate_omega_n(i, E_copper, I, rho_copper, A, L) ** 2 - omega ** 2) * \
            np.sin(np.pi * i * a / L) * \
            np.sin(np.pi * i * x / L) * np.sin(omega * t)

        displacement += term
    return (2 * P0) / (rho_copper * A * L) * displacement

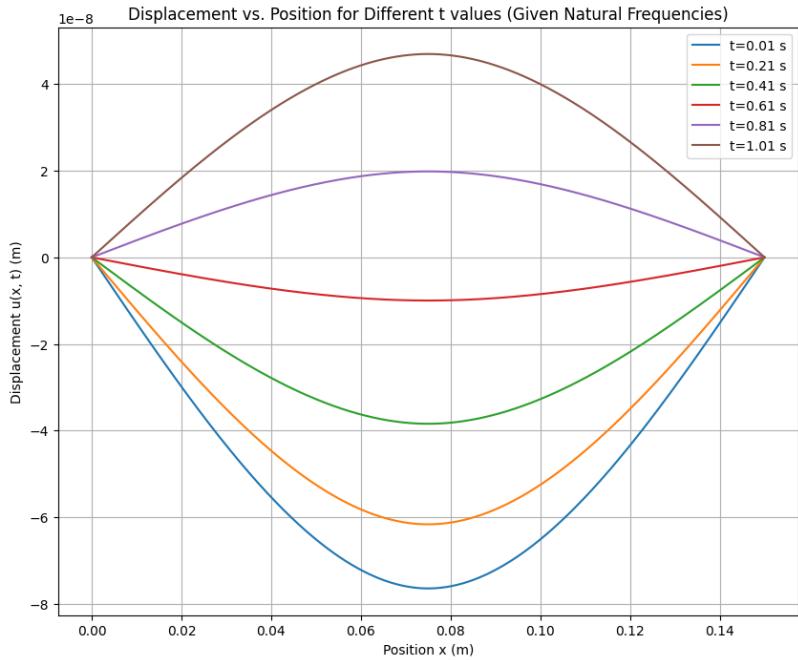
# Generate x values
x_values = np.linspace(0, L, 1000) # Generating x values from 0 to L

# Specific times for displacement calculation
t_values = np.arange(0.01, 1.1, 0.2) # Generating t values from 0 to 0.2 seconds with step 0.1 seconds

# Plotting for each t_value
plt.figure(figsize=(10, 8))
for t_value in t_values:
    displacements_exact = [calculate_displacement_exact_given_frequencies(x, t_value, 500) for x in x_values]
    plt.plot(x_values, displacements_exact, label=f't={t_value:.2f} s')

plt.title("Displacement vs. Position for Different t values (Given Natural Frequencies)")
plt.xlabel("Position x (m)")
plt.ylabel("Displacement u(x, t) (m)")
plt.legend()
plt.grid(True)
plt.show()

```



### 2.2.2 Is your solution solved by FEM close to the exact solutions? Please provide your comments.

從 FEM 計算的結果發現：

1. **時間積分參數設置 ( $\beta_1 = 1/2$  和  $\beta_2 = 1/2$ )**：這些參數決定了 Newmark 法中時間積分的性質， $\beta_1 = 1/2$  和  $\beta_2 = 1/2$  的取值在理論上可以影響算法的擬靜態 (quasi-static) 和動態 (dynamic) 行為。一般而言，這樣的參數設置是為了實現對於給定問題的穩定且高效的數值模擬。
2. **位移值的大小**：觀察到的位移值稍微大於解析解，但仍然在相同的數量級內。這可能與數值積分方法、網格劃分、以及時間步長等數值參數的設定有關。另一種可能的原因是，FEM 方法在離散化空間和時間的過程中使用 Euler-Bernoulli 的方法，而這些誤差可能會在計算過程中累積。
3. **振動模態的一致性**：觀察到振動模態與解析解一致。這意味著 FEM 方法在描述結構動態響應方面是準確的，即使在位移值上有一些微小的差異。
4. **網格劃分的影響**：我目前使用的網格可能並不足夠細緻，這可能對模擬結果有所影響。在有限元方法中，網格的分辨率對於結果的準確性和精度至關重要。更細緻的網格通常能夠提供更準確的解，但這同時也會增加計算的時間和成本。
5. **數值穩定性**：FEM 方法中涉及到的矩陣運算及解求逆可能存在數值計算上的限制。特別是在使用 `pinv()` 函數時，由於其對奇異矩陣或具有高條件數的矩陣計算時的敏感性，可能導致解的誤差增加。

## Problem 3

Suppose we could convert the beam in Problem 2 to a fixed-free axial bar with initial axial displacement condition  $u_0(x) = u(x, t = 0) = \frac{F_0x}{EA}$  caused by an suddenly axial tensile force  $F_0$  (applied and removed immediately). Consider  $F_0 = 1$  N, and please determine the vibration solution of the mechanical system:



### 3.1 Apply your own FEM code.

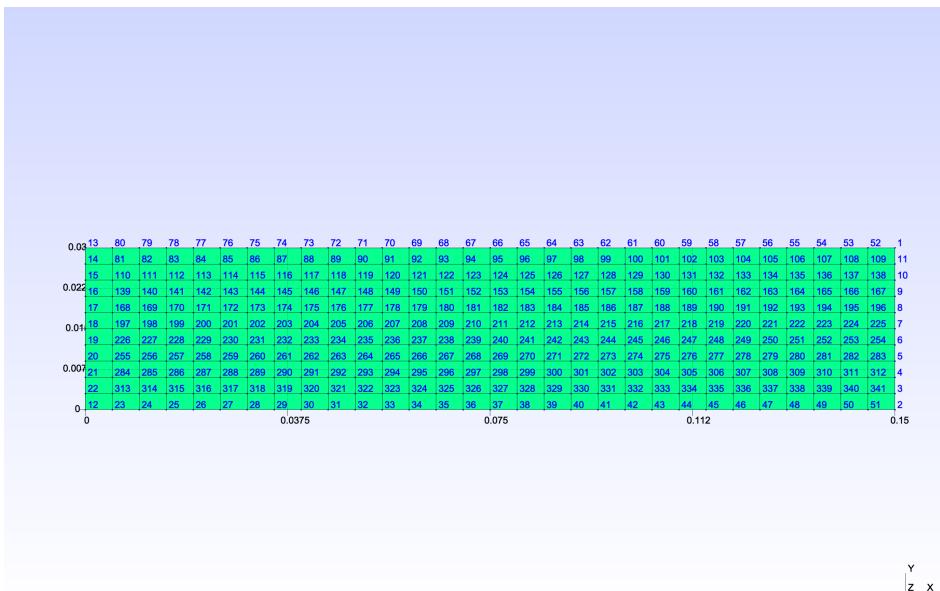
#### 3.1.1 初始條件和一些材料性質

與第二題所使用的梁參數一致，僅改變固定端的邊界條件

- 長度  $L = 5 \text{ cm}$
- 寬度  $b = 3 \text{ cm}$
- 高度  $h = 3 \text{ cm}$
- 斷面截面積  $A = 0.03^2 = 9 \times 10^{-4} \text{ m}^2$
- 密度  $\rho = 8.96 \text{ g/cm}^3$
- 軸向張力  $F_0 = 1 \text{ N}$
- 楊氏係數  $E = 130 \text{ GPa}$

```
*PARAMETER
num-dim: 2
*MATPROP
b-plane-strain: 1
young's-modulus: 130000000000
poison's-ratio: 0.3
density: 8.96
beta1: 0.5
beta2: 0.5
dt: 0.00002
nstp: 100
nprt: 1
nmod: 5
```

### 3.1.2 網格



- 左端：在此節點， $x$  和  $y$  軸的自由度被限制，即固定不動。

```
num-prescribed-disp: 20
node#-dof#-disp:
12 1 0.0
12 2 0.0
13 1 0.0
13 2 0.0
14 1 0.0
14 2 0.0
15 2 0.0
15 1 0.0
```

```

16 2 0.0
16 1 0.0
17 2 0.0
17 1 0.0
18 2 0.0
18 1 0.0
20 1 0.0
20 2 0.0
21 1 0.0
21 2 0.0
22 1 0.0
22 2 0.0

```

### 3.1.3 根據題目給定懸臂樑在時間 $t = 0$ 時初始位移分佈

$$u_0(x) = u(x, t = 0) = \frac{F_0 x}{EA}$$

在這個問題中，懸臂樑在  $t = 0$  時刻受到應力  $F_0$  而有了一個軸向位移  $u_0(x)$ 。程式碼中的 `u_0(x)` 函數計算了在空間中每個位置  $x$  上的初始位移值，具體步驟如下：

#### 1. 定義初始位移函數：

- `u_0 = @(x) (F0 * x) / (E * A);`：此函數定義了  $u_0(x)$ ，即初始位移在空間中  $x$  處的值，根據公式  $\frac{F_0 x}{EA}$  計算而得。

#### 2. 計算每個節點的初始位移：

- `for i = 1:nnode` 迴圈：對於每個節點。
- `x = coor(1, i);`：獲取節點的  $x$  座標。
- `uc((i - 1) * ndime + 2, 1) = u_0(x);`：將該節點  $x$  座標位置的初始位移  $u_0(x)$  設置為該節點的  $y$  方向的位移值。
- `calculate_Cn = @(n) (integral(@(x) u_0(x) .* sin((2 * n + 1) * pi * x / (2 * L)), 0, L)) * 2 / L;`：此函數計算  $C_n$  係數，該係數用於模擬振動問題的求解。

這段程式碼將在空間中的每個節點位置  $x$  上計算並設置了初始位移值，利用了懸臂樑在  $t = 0$  時刻的初始軸向位移條件。

```

%% 初始位移 u(x,t=0)
t = 0;
F0 = 1; % Prescribed force value
L = 0.15;
A = 0.03 * 0.03;
E = mate(2);
rho = mate(4);
c = sqrt(E / rho);

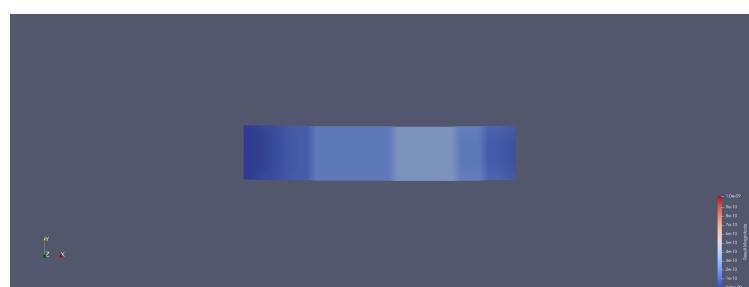
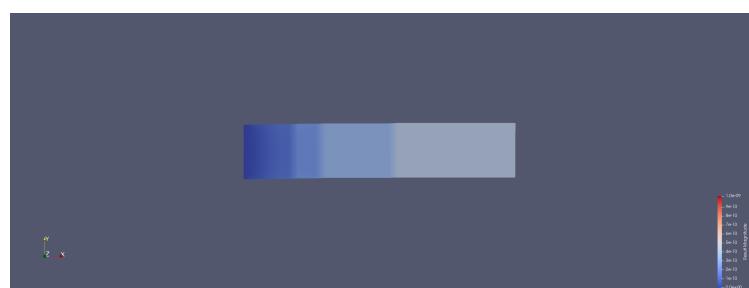
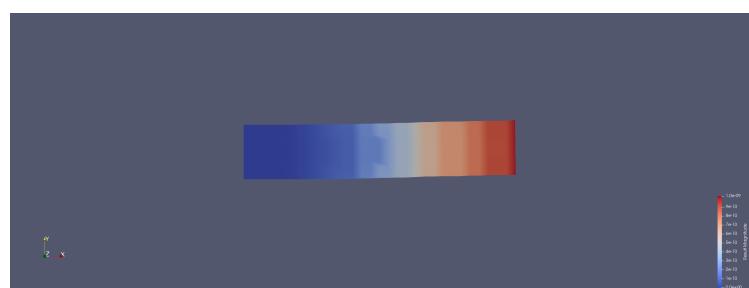
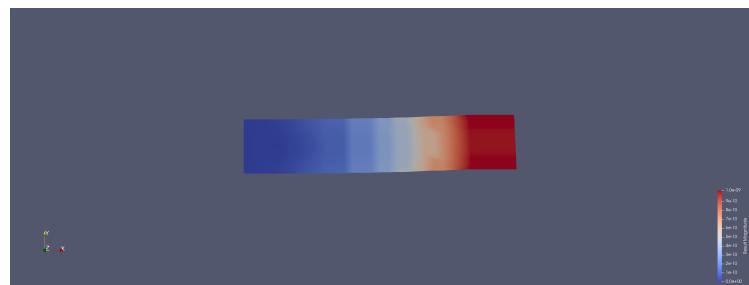
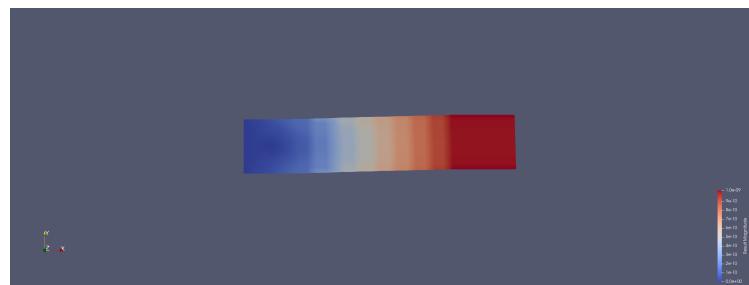
% Define the initial displacement function u_0(x)
u_0 = @(x) (F0 * x) / (E * A);

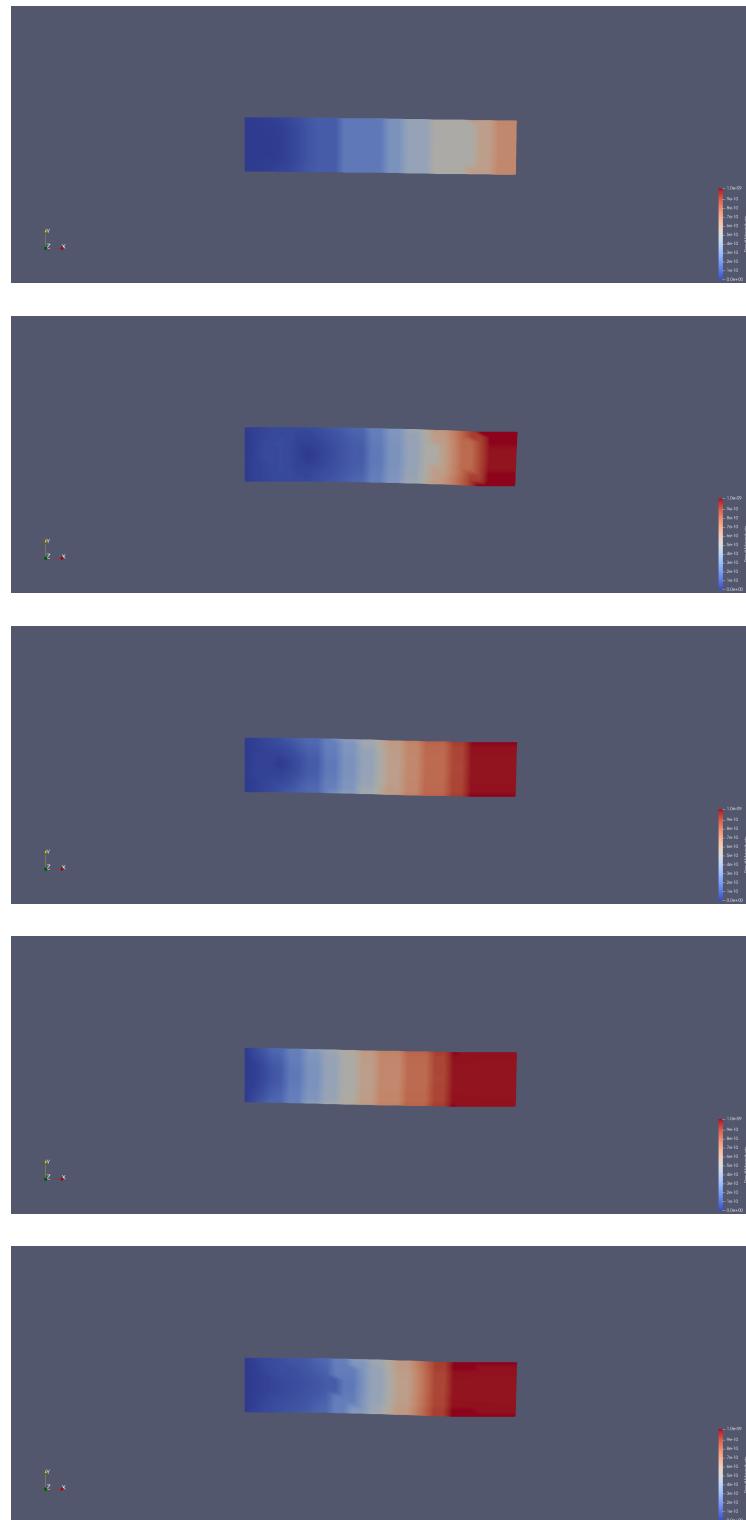
% Calculate Cn coefficients
calculate_Cn = @(n) (integral(@(x) u_0(x) .* sin((2 * n + 1) * pi * x / (2 * L)), 0, L)) * 2 / L;

for i = 1:nnode
    x = coor(1, i);
    uc((i - 1) * ndime + 2, 1) = u_0(x);
end

```

### 3.1.4 使用 ParaView 可視化 Newmark Method 計算隨時間變化之位移值 $u(x, t)$





**3.2 Compare your solution in (1) with the exact solution:**

$$u(x, t) = \sum_{n=0}^{\infty} C_n \sin \frac{(2n+1)\pi x}{2L} \cos \frac{(2n+1)\pi ct}{2L}$$

where  $C_n = \frac{2}{L} \int_0^L u_0(x) \sin \frac{(2n+1)\pi x}{2L} dx$ , and  $c = \sqrt{E/\rho}$ .

**3.2.1 Python code to calculate exact solution**

這段 Python 程式碼用於計算懸臂樑在不同時間點下的位移  $u(x, t)$ ，這是由非穩態線性波動方程式給出的解。這個解是通過將初始位移  $u_0(x)$  在時間  $t$  中的變化轉化為波動形式，然後通過傅立葉級數展開來近似。

1. `def u_0(x)` : 定義了初始位移函數  $u_0(x)$ ，該函數返回給定位置  $x$  的初始位移值，即  $u_0(x) = \frac{F_0x}{EA}$ 。
2. `def calculate_Cn(n)` : 這是一個函數，用於計算  $C_n$  係數，通過積分計算波動方程的傅立葉展開係數。
3. `def calculate_displacement(x, t, num_terms)` : 該函數計算了在給定時間  $t$  和位置  $x$  下的位移值  $u(x, t)$ 。它使用了  $C_n$  係數並應用了傅立葉展開來近似位移。
4. 生成  $x$  和  $t$  的數值以進行計算。
5. 繪製不同時間下位移  $u(x, t)$  的圖表。

這段程式碼通過數值方法計算了懸臂樑在不同時間下的位移狀態，利用了初始位移函數和傅立葉級數展開，以近似懸臂樑的振動行為。

```
import numpy as np
import matplotlib.pyplot as plt

# Constants and parameters
F0 = 1 # N
E_copper = 130e9 # Young's modulus in Pa
rho_copper = 8.96e3 # Density in kg/m^3
L = 0.15 # Length in meters
A = 0.03 * 0.03 # Cross-sectional area
c = np.sqrt(E_copper / rho_copper) # Wave speed

# Define the initial displacement function u_0(x):
def u_0(x):
    return (F0 * x) / (E_copper * A)

# Calculate Cn coefficients
def calculate_Cn(n):
    integrand = lambda x: u_0(x) * np.sin((2 * n + 1) * np.pi * x / (2 * L))
    integral = np.round(np.trapz(integrand(np.linspace(0, L, 1000)), dx=L / 1000), decimals=16)
    return 2 * integral / L

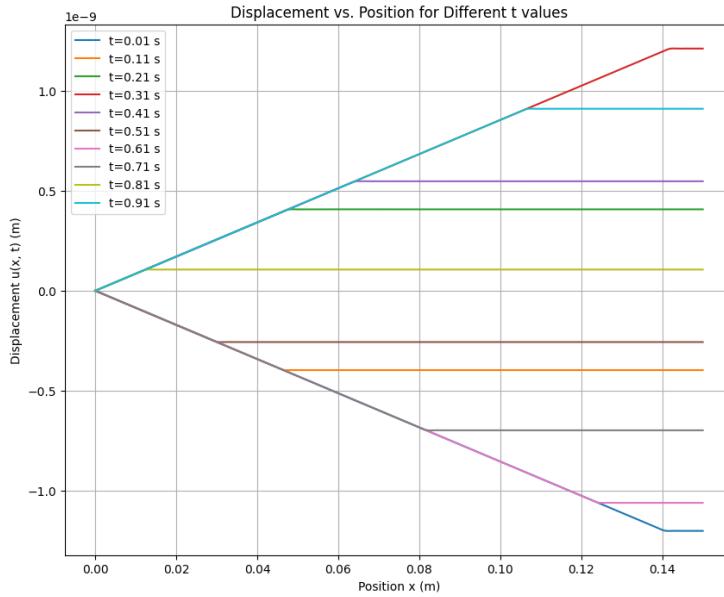
# Calculate displacement u(x, t)
def calculate_displacement(x, t, num_terms):
    displacement = 0.0
    for n in range(num_terms):
        Cn = calculate_Cn(n)
        term = Cn * np.sin((2 * n + 1) * np.pi * x / (2 * L)) * np.cos((2 * n + 1) * np.pi * c * t / (2 * L))
        displacement += term
    return displacement

# Generate x values
x_values = np.linspace(0, L, 1000) # Generating x values from 0 to L

# Specific times for displacement calculation
t_values = np.arange(0.01, 1.01, 0.1) # Generating t values from 0 to 1.1 seconds with step 0.2 seconds

# Plotting for each t_value
plt.figure(figsize=(10, 8))
for t_value in t_values:
    displacements = [calculate_displacement(x, t_value, 100) for x in x_values]
    plt.plot(x_values, displacements, label=f"t={t_value:.2f} s")

plt.title("Displacement vs. Position for Different t values")
plt.xlabel("Position x (m)")
plt.ylabel("Displacement u(x, t) (m)")
plt.legend()
plt.grid(True)
plt.show()
```



### 3.2.2 Is your solution solved by FEM close to the exact solutions? Please provide your comments.

觀察到 FEM 計算結果雖然與解析解的計算結果相近，但相對於解析解存在一些差異，特別是數值普遍較解析解來的大一點，以及解析解中出現的大轉折在 FEM 結果中較為平滑連續。有幾個可能的原因可以解釋兩者之間的差異：

1. **離散化誤差**：FEM 使用網格將連續問題離散化，如果網格劃分不夠細致，尤其在模態變化明顯的地方，可能導致位移計算的誤差。增加網格分辨率可能會更貼近解析解，但也會增加計算成本。
2. **時間積分方案的影響**：時間積分參數  $\beta_1$  和  $\beta_2$  影響 Newmark 法中的時間積分方式。這些參數會影響數值穩定性和解的精確性。更精準的時間積分方法可能需要更小的時間步長或調整參數。
3. **數值解析差異**：在某些情況下，FEM 可能無法精確地捕捉一些特殊振動模式或變化，這可能導致解的差異。
4. **數值穩定性**：FEM 方法中涉及到的矩陣運算及解求逆可能存在數值計算上的限制。特別是在使用 `pinv()` 函數時，由於其對奇異矩陣或具有高條件數的矩陣計算時的敏感性，可能導致解的誤差增加。

解析解有時是基於簡化的模型或假設，因此可能無法捕捉到結構的某些複雜行為或非線性效應。這樣的簡化可能導致解析解在特定情況下出現不連續性。相反，FEM 方法通常根據更現實和複雜的結構模型進行計算，並且能夠在模型的各個部分均勻地計算位移，這使得結果呈現出更為連續的特性。因此，解析解與FEM 方法的差異推測有可能是因為解析解所基於的模型簡化程度不同所導致的。

## Appendix A

### A.1 `HW05_1.m`

```

clc; clear;

%% 打開文件
filename = 'hw5-1';
inputFile = fopen(filename, 'r');

% 讀取輸入數據
[ndime, nnodes, nelms, nelnd, npres, ntrac, mate, coor, conn, pres, trac] = ReadInput_Dynamic(inputFile);

mate(4) = mate(4)*1000; % kg/m^3

mglob = GlobMass(ndime, nnodes, nelms, nelnd, mate, coor, conn);
kglob = GlobStif_Euler(ndime, nnodes, nelms, nelnd, mate, coor, conn); % 使用 "Euler-Bernoulli"
rglob = GlobTrac(ndime, nnodes, nelms, nelnd, ntrac, mate, coor, conn, trac); % 讀取 Traction
mpres = mglob;
kpres = kglob;

```

```

rpres = rglob;

% 取出 kpres 的特徵值並排序
% for i = 1:ndime*nnode
%     kpressort(i) = kpres(i,i);
% end
% kpressort = sort(kpressort);

%% 計算全局荷載向量(force unit N)
% 讀取 node force [node dof force]
% force = [287 2 10];
force = [];
nforce = size(force,1);
force = force';

for i = 1:nforce
    node = force(1, i); % Node where force is prescribed
    dof = force(2, i); % Degree of freedom (e.g., x=1, y=2, z=3)
    f_value = force(3, i); % Prescribed force value

    % Update rglob inside the loop
    rpres(node * ndime - (ndime - dof)) = f_value;
end

% 處理邊界條件 (Section 2.7 item 11)
% Prescribed displacements
for i = 1:npres
    ir = ndime*(pres(1,i)-1)+pres(2,i);
    for ic = 1:ndime*nnode
        mpres(ir,ic) = 0;
        mpres(ic,ir) = 0;
        kpres(ir,ic) = 0;
        kpres(ic,ir) = 0;
    end
    mpres(ir,ir) = 1.;
    rpres(ir) = pres(3,i);
end

%% Newmark Initial
beta1 = mate(5);
beta2 = mate(6);
dt = mate(7);
nstp = mate(8);
npnt = mate(9);
vc = zeros(nnnode*ndime,1);
uc = zeros(nnnode*ndime,1);

%% 初始位移 u(x,t=0)
L=0.05;
h=0.01;
t=0;
c = sqrt(10 / 1.14e3);
for i=1:nnnode
    displacements = 0.0;
    for n=1:1000
        term = (8 * h / (pi^2 * n^2)) * ...
            sin(n * pi / 2) * ...
            sin(n * pi * coor(1,i) / L) * ...
            cos(n * c * pi * t / L);
        displacements = displacements + term;
    end
    uc((i-1)*ndime+2,1) = displacements;
end

%% Newmark Method
mkpres = mpres+0.5*beta2*dt*dt*kpres;
ac = mkpres\(-kpres*uc+rpres);

iprt = 0;
indi = 1;
for i = 1:nstp

    t = t + dt;

```

```

an = mkpres\r(rpres-kpres*(uc+dt*vc+0.5*(1.-beta2)*dt*dt*ac));
vn = (vc+dt*(1.-beta1)*ac+dt*beta1*an);
un = (uc+dt*vc+(1.-beta2)*0.5*dt*dt*ac+0.5*beta2*dt*dt*an);
ac = an;
vc = vn;
uc = un;

if(iprt == nprt || iprt == 0)
    iprt = 0;

vtkShapeID = 9; % 設置VTK形狀 ID=9
% 一個 mod 做一個檔案
WriteVTKFile([filename,'_dyna_', num2str(i)], nnodes, ndime, nelem, nelnd, coor, conn, uc, vtkShapeID);

indi = indi+1;

end
iprt = iprt+1;
end

```

## A.2 HW05\_2.m

```

clc; clear;

%% 打開文件
filename = 'hw5-2_2D';
inputFile = fopen(filename, 'r');

% 讀取輸入數據
[ndime, nnodes, nelem, nelnd, npres, ntrac, mate, coor, conn, pres, trac] = ReadInput_Dynamic(inputFile);

mate(4) = mate(4)*1000; % kg/m^3

mglob = GlobMass(ndime, nnodes, nelem, nelnd, mate, coor, conn);
kglob = GlobStif_Euler(ndime, nnodes, nelem, nelnd, mate, coor, conn); % 使用 "Euler-Bernoulli"
rglob = GlobTrac(ndime, nnodes, nelem, nelnd, ntrac, mate, coor, conn, trac); % 讀取 Traction
mpres = mglob;
kpres = kglob;
rpres = rglob;

% 取出 kpres 的特徵值並排序
% for i = 1:ndime*nnodes
%     kpressort(i) = kpres(i,i);
% end
% kpressort = sort(kpressort);

%% 計算全局荷載向量(force unit N)
% 讀取 node force [node dof force]
force = [66 2 -10];
% force = [];
nforce = size(force,1);
force = force';

for i = 1:nforce
    node = force(1, i); % Node where force is prescribed
    dof = force(2, i); % Degree of freedom (e.g., x=1, y=2, z=3)
    f_value = force(3, i); % Prescribed force value
end

% 處理邊界條件 (Section 2.7 item 11)
% Prescribed displacements
for i = 1:npres
    ir = ndime*(pres(1,i)-1)+pres(2,i);
    for ic = 1:ndime*nnodes
        mpres(ir,ic) = 0;
        mpres(ic,ir) = 0;
        kpres(ir,ic) = 0;
        kpres(ic,ir) = 0;
    end
    mpres(ir,ir) = 1.;
    rpres(ir) = pres(3,i);
end

```

```

end

%% Newmark Initial
beta1 = mate(5);
beta2 = mate(6);
dt = mate(7);
nstp = mate(8);
npert = mate(9);
vc = zeros(nnode*ndime,1);
uc = zeros(nnode*ndime,1);

%% 初始位移 u(x,t=0)
% P(a,t=0) = P0*sin(wt) = 0
t = 0;
P0 = f_value; % Prescribed force value
w1 = 14469.9;
w = 0.1*w1;

%% Newmark Method
mkpres = mpres+0.5*beta2*dt*dt*kpres;
ac = mkpres\(-kpres*uc+rpres);

iprt = 0;
indi = 1;
for i = 1:nstp

    % Update rpres
    t = t + dt;
    P = P0*sin(w*t);
    % Update rpres inside the loop
    rpres(node * ndime - (ndime - dof)) = P;

    % Calculate
    an = mkpres\rpres\(-kpres*(uc+dt*vc+0.5*(1.-beta2)*dt*dt*ac)); % Update `rpres`
    vn = (vc+dt*(1.-beta1)*ac+dt*beta1*an);
    un = (uc+dt*vc+(1.-beta2)*0.5*dt*dt*ac+0.5*beta2*dt*dt*an);
    ac = an;
    vc = vn;
    uc = un;

    if(iprt == npert)
        iprt = 0;

        vtkShapeID = 9; % 設置VTK形狀 ID=9
        % 一個 mod 做一個檔案
        WriteVTKFile([filename,'_dyna'], num2str(indi)], nnode, ndime, nelem, nelnd, coor, conn, uc, vtkShapeID);

        indi = indi+1;
    end
    iprt = iprt+1;
end

```

### A.3 HW05\_3.m

```

clc; clear;

%% 打開文件
filename = 'hw5-3_2D';
inputFile = fopen(filename, 'r');

% 讀取輸入數據
[ndime,nnode,nelem,nelnd,npres,ntrac,mate,coor,conn,pres,trac] = ReadInput_Dynamic(inputFile);

mate(4) = mate(4)*1000; % kg/m^3

mglob = GlobMass(ndime,nnode,nelem,nelnd,mate,coor,conn);
kglob = GlobStif_Euler(ndime,nnode,nelem,nelnd,mate,coor,conn); % 使用 "Euler-Bernoulli"
rglob = GlobTrac(ndime,nnode,nelem,nelnd,ntrac,mate,coor,conn,trac); % 讀取 Traction
mpres = mglob;

```

```

kpres = kglob;
rpres = rglob;

% 取出 kpres 的特徵值並排序
% for i = 1:ndime*nnode
%     kpresort(i) = kpres(i,i);
% end
% kpresort = sort(kpresa);

%% 計算全局荷載向量(force unit N)
% 讀取 node force [node dof force]
% force = [66 2 -10];
force = [];
nforce = size(force,1);
force = force';

for i = 1:nforce
    node = force(1, i); % Node where force is prescribed
    dof = force(2, i); % Degree of freedom (e.g., x=1, y=2, z=3)
    f_value = force(3, i); % Prescribed force value
end

% 處理邊界條件 (Section 2.7 item 11)
% Prescribed displacements
for i = 1:pres
    ir = ndime*(pres(1,i)-1)+pres(2,i);
    for ic = 1:ndime*nnode
        mpres(ir,ic) = 0;
        mpres(ic,ir) = 0;
        kpres(ir,ic) = 0;
        kpres(ic,ir) = 0;
    end
    mpres(ir,ir) = 1.;
    rpres(ir) = pres(3,i);
end

%% Newmark Initial
beta1 = mate(5);
beta2 = mate(6);
dt = mate(7);
nstp = mate(8);
npnt = mate(9);
vc = zeros(nnode*ndime,1);
uc = zeros(nnode*ndime,1);

%% 初始位移 u(x,t=0)
t = 0;
F0 = 1; % Prescribed force value
L = 0.15;
A = 0.03 * 0.03;
E = mate(2);
rho = mate(4);
c = sqrt(E / rho);

% Define the initial displacement function u_0(x)
u_0 = @(x) (F0 * x) / (E * A);

% Calculate Cn coefficients
calculate_Cn = @(n) (integral(@(x) u_0(x) .* sin((2 * n + 1) * pi * x / (2 * L)), 0, L)) * 2 / L;

for i = 1:nnode
    x = coor(1, i);
    uc((i - 1) * ndime + 2, 1) = calculateDisplacement(x, t, 100, L, calculate_Cn, c);
    uc((i - 1) * ndime + 2, 1) = u_0(x);
end

%% Newmark Method
mkpres = mpres+0.5*beta2*dt*dt*kpres;
ac = mkpres\(-kpres*uc+rpres);

iprt = 0;
indi = 1;
for i = 1:nstp
    % Calaulate

```

```

an = mkpres\r(pres-kpres*(uc+dt*vc+0.5*(1.-beta2)*dt*dt*ac));
vn = (vc+dt*(1.-beta1)*ac+dt*beta1*an);
un = (uc+dt*vc+(1.-beta2)*0.5*dt*dt*ac+0.5*beta2*dt*dt*an);
ac = an;
vc = vn;
uc = un;

if(iprt == nprt)
    iprt = 0;

vtkShapeID = 9; % 設置VTK形狀 ID=9
% 一個 mod 做一個檔案
WriteVTKFile([filename,'_dyna', num2str(indi)], nnode, ndime, nelnd, coor, conn, uc, vtkShapeID);

indi = indi+1;

end
iprt = iprt+1;
end

%% Calculate displacement u(x, t)
function displacement = calculateDisplacement(x, t, num_terms, L, calculate_Cn, c)
    displacement = 0.0;
    for n = 0:num_terms
        Cn = calculate_Cn(n);
        term = Cn * sin((2 * n + 1) * pi * x / (2 * L)) * cos((2 * n + 1) * pi * c * t / (2 * L));
        displacement = displacement + term;
    end
end

```

## Appendix B

### B.1 WriteVTKFile.m



輸出 \*.vtk file (擴充版\_2023-11-27)

```

function WriteVTKFile(filename, nnode, ndime, nelem, nelnd, coor, conn, uc, vtkShapeID, strain_stress_matrix)
    % Write VTK file with the given data

    % Open the file for writing
    fid = fopen([filename, '.vtk'], 'w');
    if fid == -1
        error('Cannot open the file for writing');
    end

    fprintf(fid, "# vtk DataFile Version 2.0\n");
    fprintf(fid, "Generated volume Mesh\n");
    fprintf(fid, "ASCII\n");
    fprintf(fid, "DATASET UNSTRUCTURED_GRID\n");
    fprintf(fid, "POINTS %d float\n", nnode);

    %% Coordinates
    if ndime == 2
        for k = 1:nnode
            fprintf(fid, '%f %f %f\n', coor(1, k), coor(2, k), 0.0); % 2D z=0.0
        end
    elseif ndime == 3
        for k = 1:nnode
            fprintf(fid, '%f %f %f\n', coor(1, k), coor(2, k), coor(3, k));
        end
    end

    %% Element connectivity
    fprintf(fid, "CELLS %d %d\n", nelem, nelem * (1 + nelnd));
    for k = 1:nelem
        formatString = '%d';

```

```

        for n = 1:nelnd
            formatString = [formatString, ' %d'];
        end
        formatString = [formatString, '\n'];
        fprintf(fid, formatString, nelnd, conn(1:nelnd, k) - 1);
    end

    %% VTK shape IDs
    fprintf(fid, "CELL_TYPES %d\n", nelem);
    for k = 1:nelem
        fprintf(fid, "%d\n", vtkShapeID); % VTK shape IDs
    end

    %% Output node values (e.g., u, stress, strain...)
    fprintf(fid, "POINT_DATA %d\n", nnodes);

    % Write "displacement": u_1, u_2, ...
    for j = 1:ndime
        fprintf(fid, "SCALARS u%d float\n", j);
        fprintf(fid, "LOOKUP_TABLE default\n");
        for k = 1:nnodes
            fprintf(fid, "%g\n", uc(ndime * (k - 1) + j, 1));
        end
    end

    % Check if strain_stress_matrix is provided
    if nargin == 10 && ~isempty(strain_stress_matrix)
        % Write "strain": e11, e12, e22
        for i = 1:3
            fprintf(fid, "SCALARS strain_%d float\n", i);
            fprintf(fid, "LOOKUP_TABLE default\n");
            for k = 1:nelem
                fprintf(fid, "%f\n", strain_stress_matrix(k, i+1));
            end
        end

        % Write "stress": s11, s12, s22
        for i = 1:3
            fprintf(fid, "SCALARS stress_%d float\n", i);
            fprintf(fid, "LOOKUP_TABLE default\n");
            for k = 1:nelem
                fprintf(fid, "%f\n", strain_stress_matrix(k, i+4));
            end
        end
    end

    % Close the file
    fclose(fid);
end

```

---

## Reference

[https://github.com/NTU-CAE-David/finite\\_element\\_method](https://github.com/NTU-CAE-David/finite_element_method)

[https://github.com/NTU-CAE-David/finite\\_element\\_method/tree/master/HW/HW05](https://github.com/NTU-CAE-David/finite_element_method/tree/master/HW/HW05)