Application of Computational Intelligence in Engineering

Group 13

Minimize the Displacement by SA

Team members:

R09521245　李牧軒

R09521212　黃懷寬

R09521227　陳俊廷

R09521206　蕭宇東

R09521209　洪翌鈞

M10705334　林文柏

目錄

一、概述:

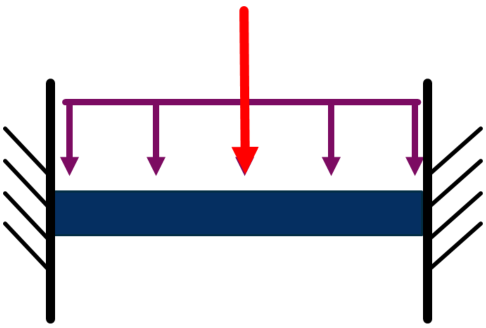
前言:

建築結構主要由梁、柱、牆三者所構成，每種元件的形狀、尺寸都至關重要，牽動著整體結構強度以及材料成本，一旦設計不良，輕者則會造成金錢浪費、以及可使用空間的壓縮；重者則會造成結構物的不安全性，而如何設計出安全、經濟、與美觀兼顧的結構物，正是結構技師所需具備的基本能力。

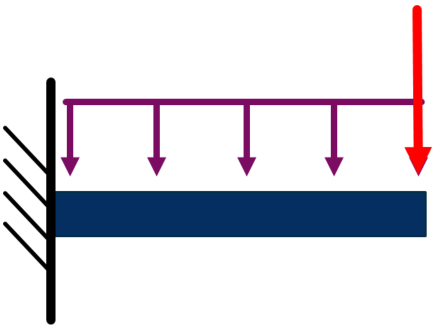
本次的研究重點為**懸臂梁、與兩端點固定梁的斷面尺寸分配**，計畫透過**模擬退火演算法**(Simulated Annealing，以下簡稱 SA)來執行梁的**斷面尺寸最佳化**，以此判斷在彈性情況下，如何能使這兩種梁**受力產生的位移降至最低**。

計畫內容:

我們希望能探討: 在固定的總體積與梁寬的情況(代表著固定的材料成本)，兩種類型的梁分別受到一組均部載重和集中載重(忽略自重)，要**如何配置斷面，才能使的梁受力後產生的變位最小化**。在兩端固定梁的案例中(如圖 1.1)，我們所觀測的點位在**梁中心**；至於懸臂梁案例(如圖 1.2)，觀測點則位在梁的**自由端端點**。



(圖 1.1)



(圖 1.2)

二、SA 演算法:

參數設定:

在本次研究中，懸臂梁與兩端固定梁皆以**同樣的參數**去執行 SA 最佳化，參數設定如下:

結構物基本參數設定(**除了節點數，其餘參數皆為定值**):

- 材料性質: E = 200 (GPa)
- 梁的總體積: V = 2 (m^3)
- 梁長: L = 10 (m)
- 梁寬: W = 0.5 (m)
- 集中載重: Force = 100 (KN)
- 均部載重: Load = 100 (KN/m) (以 2D 形式來執行計算。)
- **節點數**: n = 5、7、9、11 (代表將梁**均分成 n-1 段**，分別是 4、6、8、10 段。)

SA 演算法的**相關參數與設定**:

- 每一段的初始梁深為**平均分佈**。
- 每次步驟中，隨機抽取**三段梁深**來進行更換。

(起初，我們設定交換一半的梁深，發現若一次交換太多的梁深，會類似於隨機搜索，不利於最佳化結果，且非常費時。)

- 梁深的最小單位為10^{-4}(m)，也就是 1 毫米。

(起初，我們使用預設的10^{-8}(m)，然而發現過高的精度會導致頻繁更換差異不大的梁深，不利於最佳化結果。)

- **初始溫度、降溫公式&迭代次數、與步數**: 由 Taguchi Method 來選出最佳參數。

Taguchi Method:

參數:

- 初始溫度: T_0
- 降溫公式: $T(k) = T(k-1)/(1+\gamma\, T(k-1))$
- 迭代次數: $n = \alpha\,(T_0 - t) + \beta$
- 步數: Step

將降溫公式與迭代次數的[γ, α, β]合成一組參數來進行分析

	T_0	Schedule/Iteration [γ, α, β]	Step
Set 1	1000	[0.001,2,10]	1000
Set 2	400	[0.01,2.5,20]	500
Set 3	100	[0.0005,1,5]	300

(表 2.1)

Taguchi Array:

以 n=9 的兩端固定梁來執行分析，每種參數組合執行 20 次，得到**平均變位**和**標準差**，來判斷何組參數組為最佳解。

Experiment Number	Parameter 1	Parameter 2	Parameter 3	Mean	Std
1	1	1	1	**0.027**	0.006
2	1	2	2	**0.031**	0.004
3	1	3	3	**0.039**	0.0042
4	2	1	2	**0.026**	0.005
5	2	2	3	**0.029**	0.0061
6	2	3	1	**0.028**	0.0042
7	3	1	3	**0.0036**	0.0041
8	3	2	1	**0.026**	0.0049
9	3	3	2	**0.0032**	0.0042

(表 2.2)

參數選擇:

根據上述九種參數組合所得到的平均以及標準差，分別將相同參數的結果再進行一次平均，結果如下表格(表 2.3)。分別**選取平均位移最小的三種參數，作**

為本次研究的參數組合。

	Mean/Std		
	Parameter 1	**Parameter 2**	**Parameter 3**
Set 1	0.0323/0.00473	0.0297/0.00503	0.027/0.00503
Set 2	0.0277/0.0051	0.0287/0.005	0.0297/0.0044
Set 3	0.0313/0.0044	0.033/0.0042	0.00347/0.0048

(表 2.3)

- 初始溫度: T_0 = 400
- 降溫公式: T(k) = T(k-1)/(1+0.01*T(k-1))
- 迭代次數: n = 2.5*(T_0-t) + 20
- 步數: Step = 1000

模型分析:

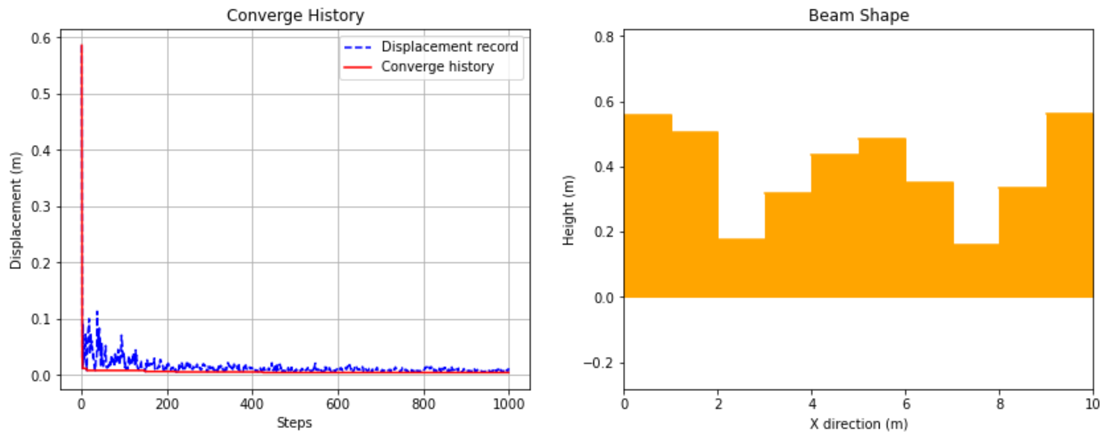
由於兩端固定梁與懸臂梁的梁深分布趨勢不同，以下依序介紹不同斷面數的兩端固定梁和懸臂梁的模型分析成果，並且會在之後的章節進行驗證和解釋。

- 兩端固定梁

SA 演算法進行最佳化分析後，會得到兩種類型的斷面分布結果，分別為**兩端與中間較深的對稱解**(形似英文字母的 W)與**梁深集中在一側不對稱解**。

➢ n=11(斷面數=10):

對稱解:

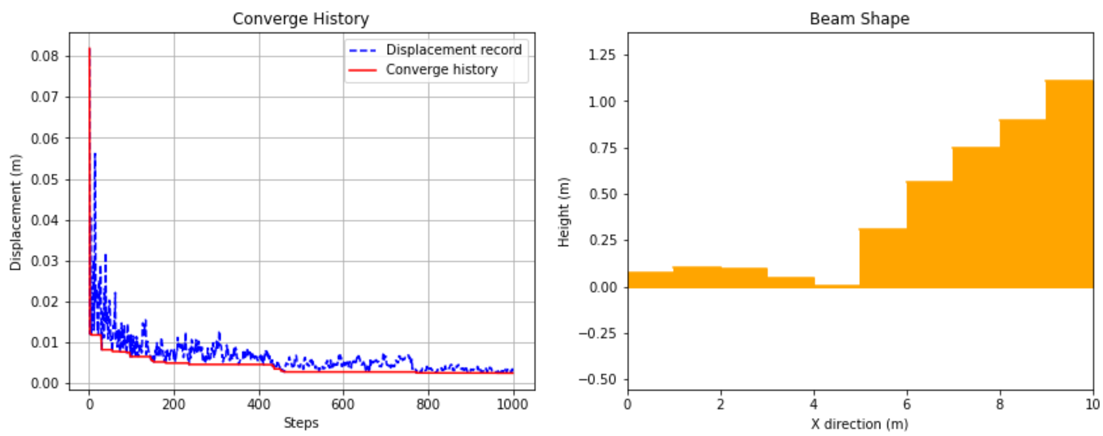


(圖 2.1)

Total volume 1.944 m^3 (總體積趨近於 2，結構物的參數設定控制)
optimal H list [0.559 0.505 0.175 0.32 0.436 0.484 0.351 0.162 0.333 0.563] (最佳化後的梁深分布，單位:m)
optimal Displacement 0.003 m (中點垂直位移量)

經過多次分析，最佳化位移多落在 0.004~0.006(m)。

不對稱解:



(圖 2.2)

Total volume 1.973 m^3

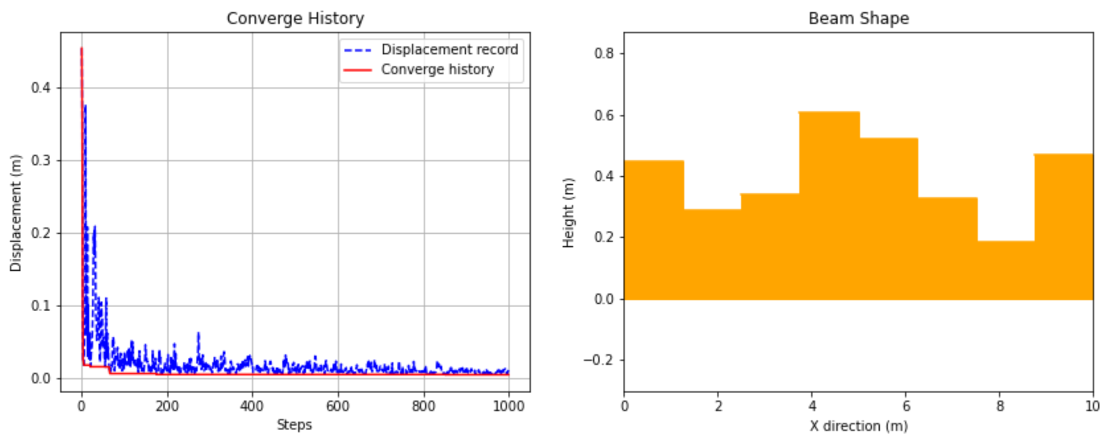
optimal H list [0.074 0.1 0.093 0.044 0.002 0.311 0.566 0.7
5 0.898 1.108]

optimal Displacement 0.003 m

經過多次分析，最佳化位移多落在 0.003~0.005(m)。

- n=9(斷面數=8):

對稱解:

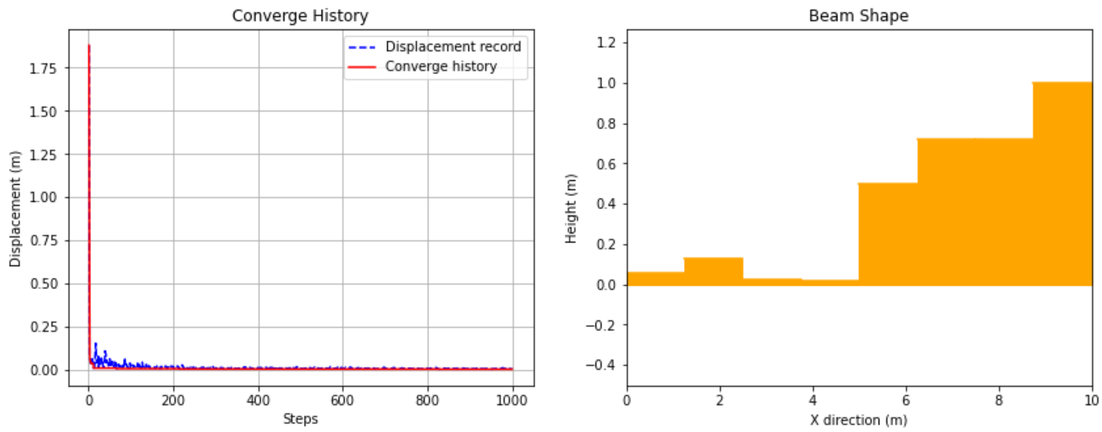


(圖 2.3)

Total volume 1.986 m^3

```
optimal H list [0.448 0.287 0.339 0.605 0.52 0.325 0.185 0.
468]
optimal Displacement 0.005 m
```

不對稱解:

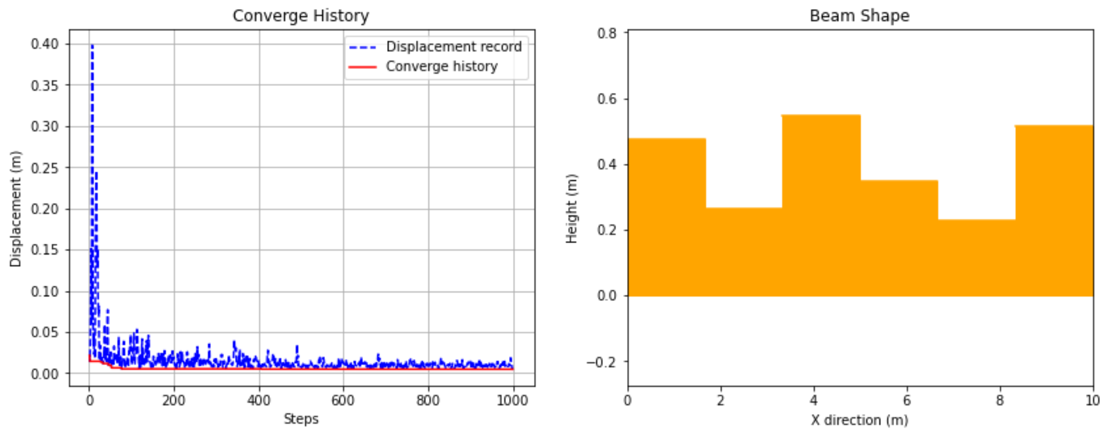


(圖 2.4)

```
Total volume 1.979 m^3
optimal H list [0.058 0.128 0.025 0.017 0.497 0.717 0.722
1.002]
optimal Displacement 0.004 m
```

- n=7(斷面數=6):

對稱解:



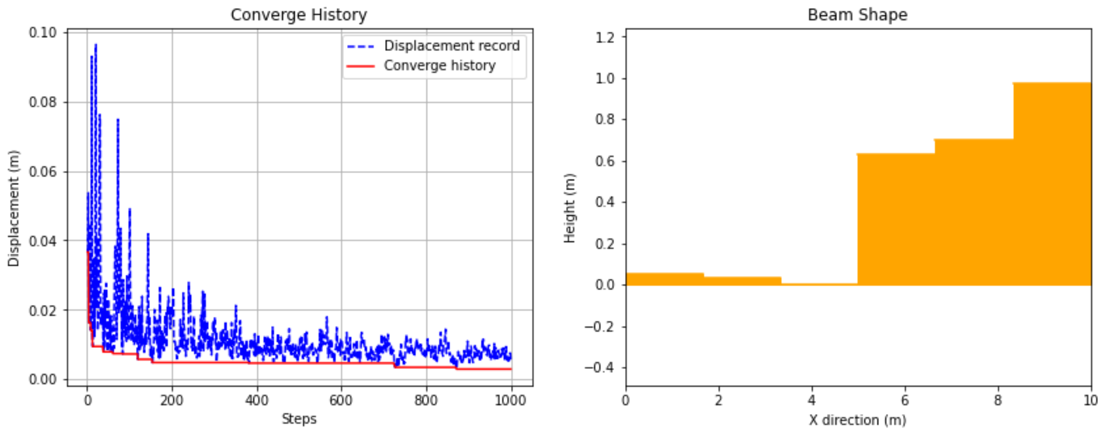
(圖 2.5)

```
Total volume 1.978 m^3
```

optimal H list [0.474 0.263 0.546 0.348 0.229 0.513]

optimal Displacement 0.005 m

不對稱解:



(圖 2.6)

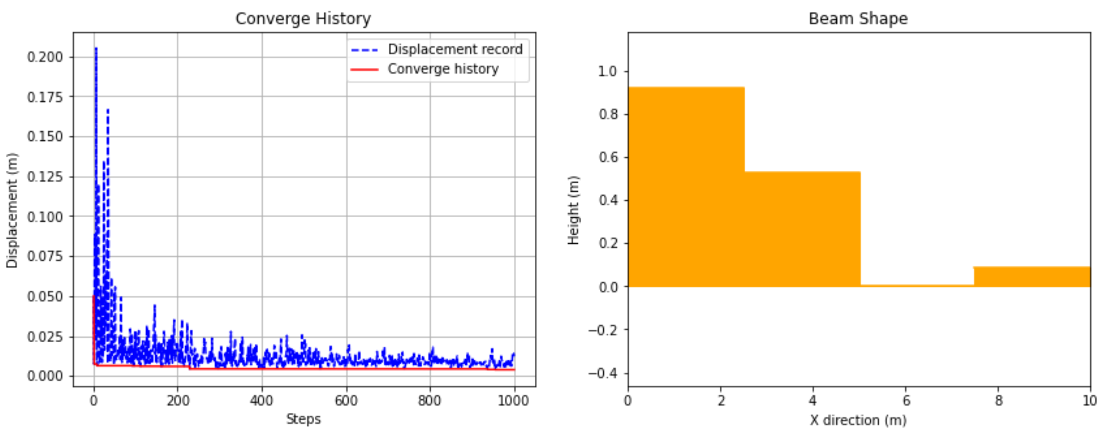
Total volume 1.992 m^3

optimal H list [0.05 0.032 0.002 0.628 0.703 0.975]

optimal Displacement 0.003 m

- n=5(斷面數=4):

由於斷面數只剩 4，故無法明顯歸類為對稱解或不對稱解:



(圖 2.7)

Total volume 1.925 m^3

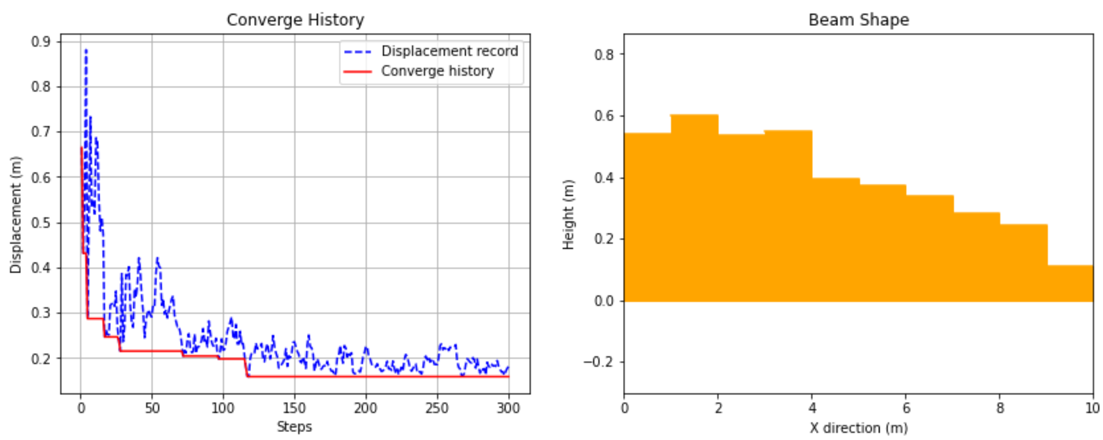
optimal H list [0.923 0.528 0.003 0.086]

optimal Displacement 0.004 m

● 懸臂梁

不同於兩端固定梁有兩種最佳解，經過 SA 演算法分析，懸臂梁的梁深最佳分布皆是呈現: **梁深隨著固定端往自由端遞減**。

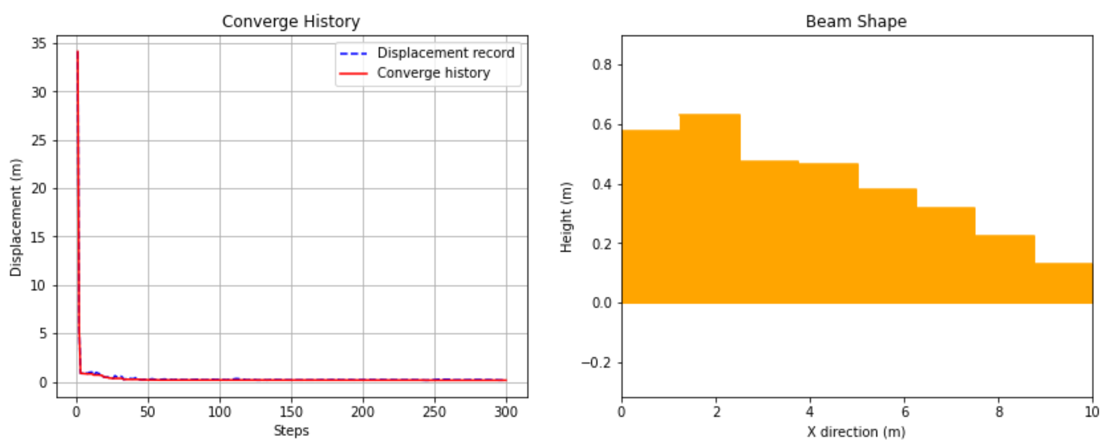
➢ n=11(斷面數=10):



(圖 2.8)

Total volume 1.981 m^3
optimal H list [0.538 0.602 0.535 0.549 0.394 0.373 0.338 0.28 0.244 0.109]
optimal Displacement 0.159 m (自由端的垂直位移)

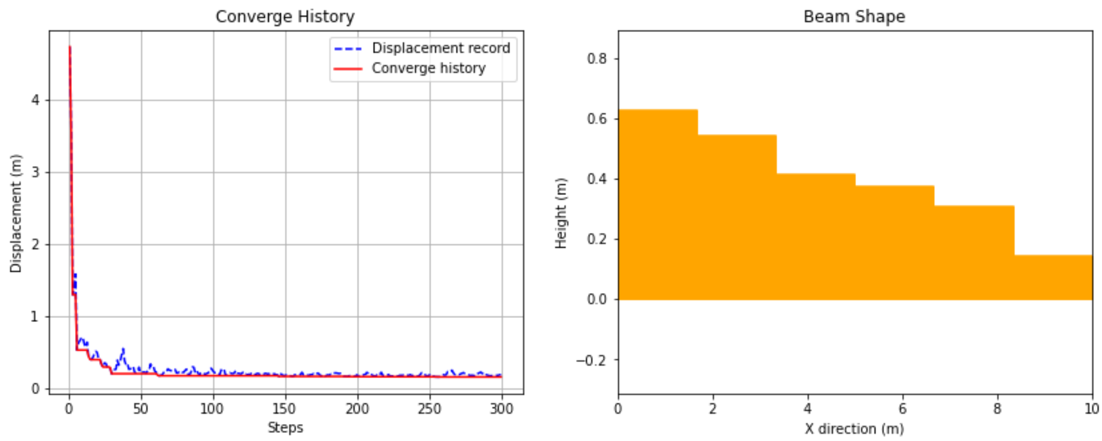
➢ n=9(斷面數=8):



(圖 2.9)

Total volume 1.999 m^3
optimal H list [0.579 0.632 0.473 0.464 0.381 0.317 0.223 0.129]
optimal Displacement 0.151 m

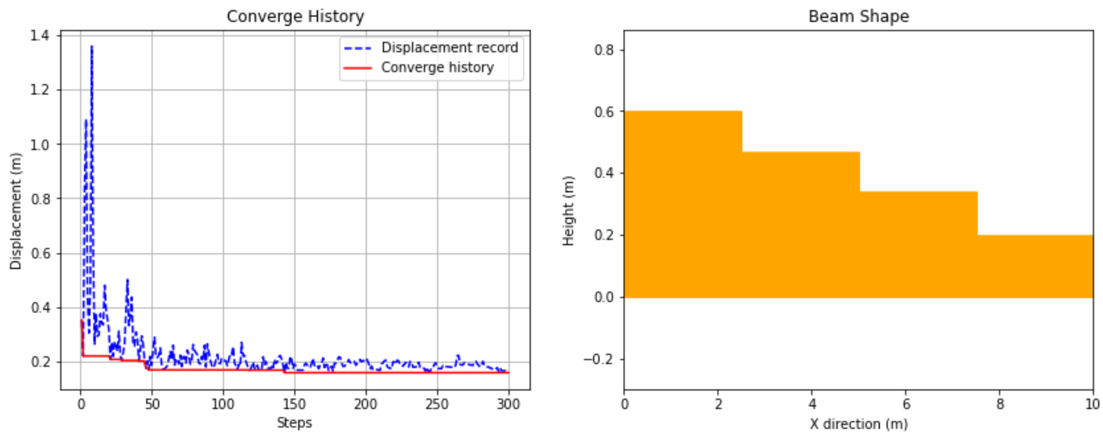
➢ n=7(斷面數=6):



(圖 2.10)

Total volume 1.999 m^3
optimal H list [0.625 0.542 0.411 0.374 0.305 0.142]
optimal Displacement 0.154 m

➢ n=5(斷面數=4):



(圖 2.11)

Total volume 1.998 m^3
optimal H list [0.596 0.467 0.339 0.196]
optimal Displacement 0.158 m

過程與結果討論:

參數與設定的修改:

- 每一個 Step 中，交換梁深的數量:

起初，我們設定每個 step 交換半數的梁深，然而發現若一次交換太多的梁深，會類似於隨機搜索，不利於最佳化結果，且非常費時，因此調整成每 Step 交換 3 個梁深。

- 精度調整

原本使用 python 預設的10^{-8}(m)，然而發現過高的精度會導致產生許多差異不大的梁深，會不利於最佳化結果，因此調整精度為10^{-3}(m)，也就是 1 毫米。

- 容忍下一步結果並非更好的程度:

容忍的概率為:random() < exp(-delta***μ**/t)，μ越大，代表容忍下一步變糟的程度降低，雖然調大μ能跑出較低的位移，但會花更多時間，權衡後，我們使用的μ為500。

- 隨機產生梁深的機率分布:

我們嘗試過以**平均分布**、**常態分佈**、與 **Beta 分布**，發現差異不大，最終以平均分布來產生梁深。

結果討論:

- 兩端固定梁有兩種類型的解，經過多次分析的數據平均，我們發現:若容忍下一步結果並非更好的概率係數 μ 越大(也就是越偏好只接受更佳解)，有越高的機率會出現對稱解。結果如下表(表 2.4):

Symmetric Possibility (%)	μ= 500	μ= 2000
N=7	10	40
N=9	40	60

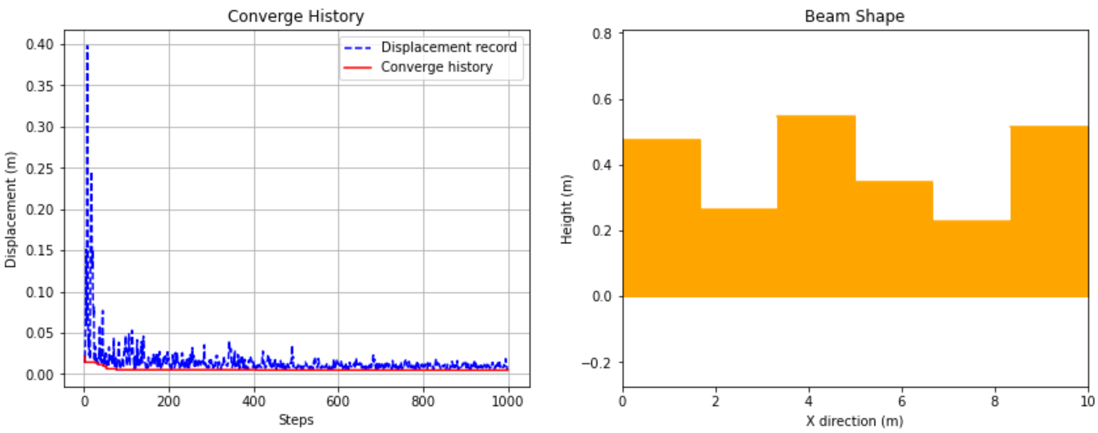
(表 2.4)

- 分析懸臂梁時，由於不像兩端固定梁有兩種類型的解，相對容易找出最佳解，此外，在相同的n(斷面數)之下，SA 演算法分析懸臂梁的時間明顯多於兩端固定梁，因此在調整懸臂梁參數時，主要以減少耗時為主要目的。

驗證:

為了驗證我們的 SA 演算法的計算正確性，使用了 etabs 來進行力學分析，並且將結果和最佳化得到解做比對，分別 n=7(斷面數為 6)時的懸臂梁、與兩端固定梁的兩種結果。

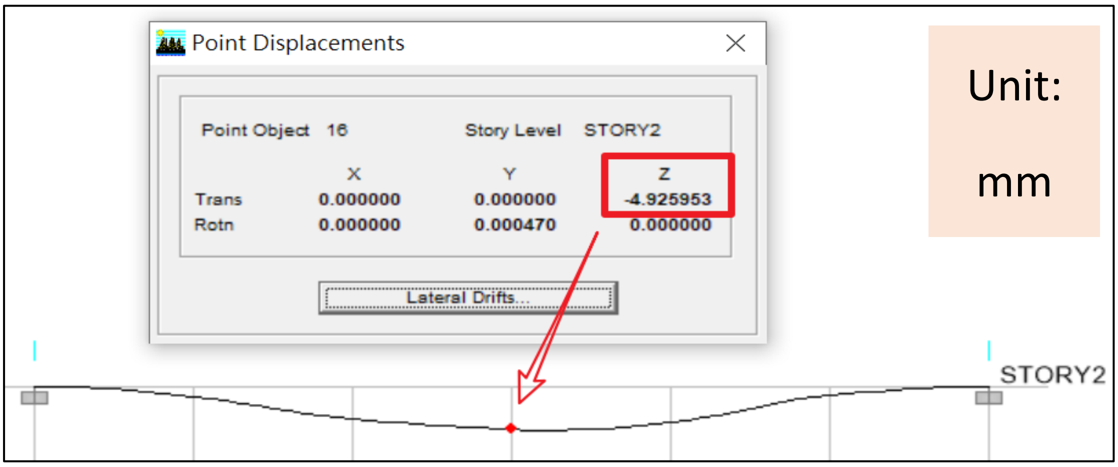
➢ 兩端固定梁的對稱解:



(圖 2.12)

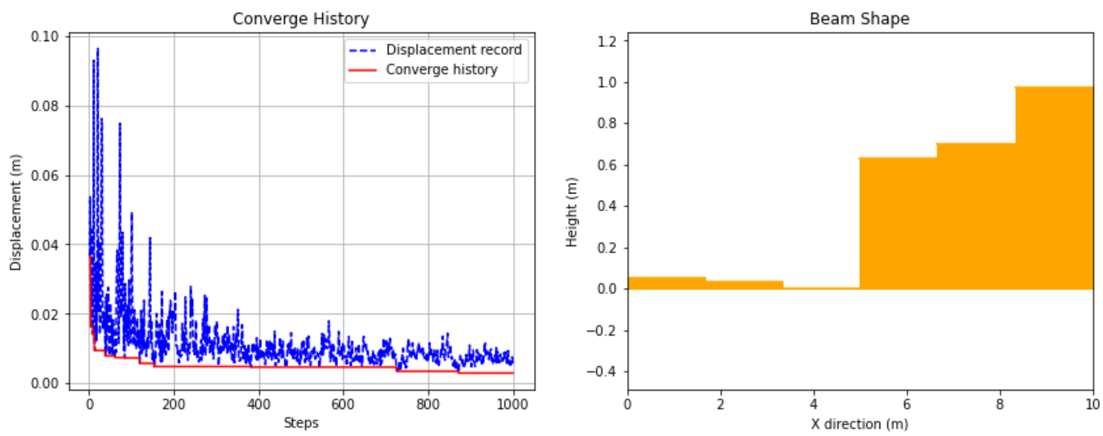
```
optimal H list [0.474 0.263 0.546 0.348 0.229 0.513]
optimal Displacement 0.005 m
```

Etabs 建模結果:



(圖 2.13)

➢ 兩端固定梁的不對稱解:



(圖 2.14)

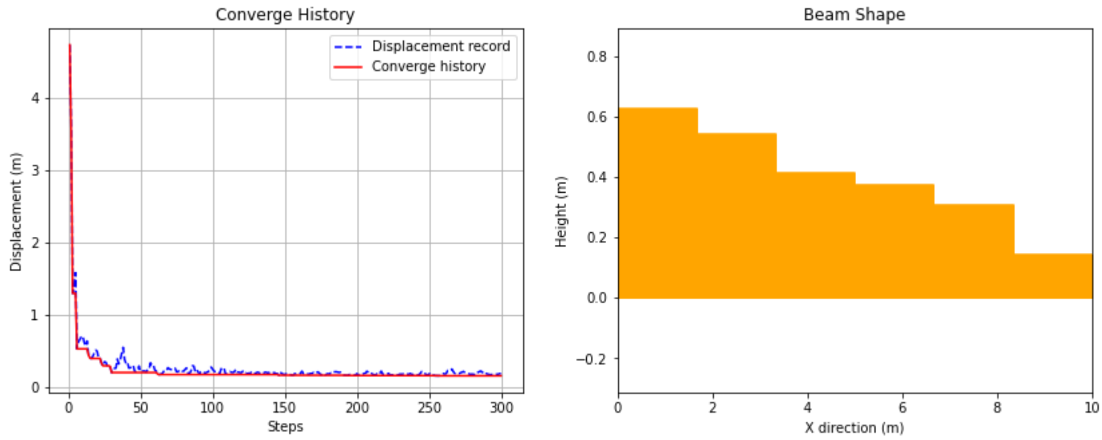
```
optimal H list [0.05 0.032 0.002 0.628 0.703 0.975]

optimal Displacement 0.003 m
```

Etabs 建模結果:

(圖 2.15)

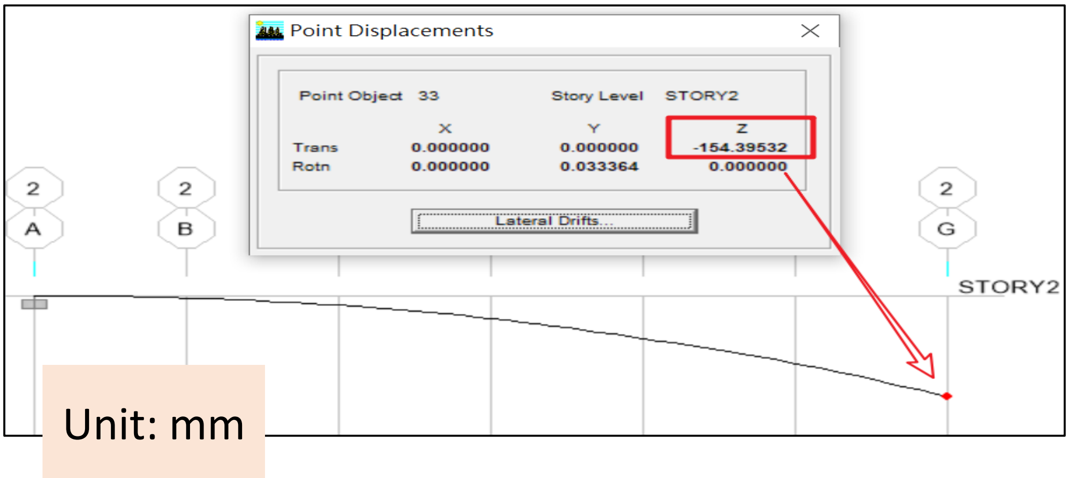
➢ 懸臂梁



(圖 2.16)

```
optimal H list [0.625 0.542 0.411 0.374 0.305 0.142]
optimal Displacement 0.154 m
```

Etabs 建模結果:



(圖 2.17)

由上述三組比對可以驗證出，本研究的演算法程式計算是正確且合理的。

三、結論:

本次研究的結果，懸臂梁的部分非常符合現實設計，梁深由固定端往自由端遞減。然而兩端固定梁的不對稱解，則較不符合現實情況，這是因為: 本次研究是探討兩端固定梁的中心垂直變位，而非最大變位。觀察圖 2.15 可以發現: 雖然不對稱解的中心變位極少，但整體的最大變位卻非常大，這情況若發生在現實的鋼梁或者混凝土梁，會造成非常嚴重的破壞。若要做實務設計，得把最大變位也列入考量，才不會因為設計不良而造成結構物的破壞，進而威脅到生命與財產。

四、分工:

學號:	姓名:	分工內容
R09521245	李牧軒	撰寫程式碼
R09521212	黃懷寬	參數調整、結果分析
R09521209	洪翌鈞	參數調整、結果分析
M10705334	林文柏	使用 Etabs 驗證結果
R09521227	陳俊廷	整理歸納與書面報告
R09521206	蕭宇東	PPT 製作與口頭報告

五、程式碼:

本組選擇以 python 來執行 SA 演算法。

Main.py:

```
from math import inf
from random import random
import Solve
import SA
import Input
import numpy as np
import Plot



n = 11  # number of nodes (Odd number)
type_ = 1  # [ 1 for two fixed end beam ,others for cantilever]
force = 100  # concentrate load (kN)
load = 100  # weight load (kN/m)

total_length = 10  # Toatal length (m)
l = total_length / (n-1)  # element length
W = 0.5  # width(constant)
V = 2  # contraint volumn
constraint = (V/W)/l  # total height contraint (constraint total volumn)



# random initalize elements height list (m)
H = Input.random_H(n-1, constraint)

# SA algoritm parameter
SA_setting = {
    't0': 300,  # Intital Temperature
    'steps': 1000,  # Iteration number
}

# Beam Parameter
```

```
config = {
    'NNOD': n,
    'NBC': n-1,
    'NMAT': 1,
    'NSEC': n-1,
    'NNE': 2,
    'IFORCE': 2,
    'COOR': np.linspace(0, total_length, n).reshape(1, -1),
    'NFIX': Input.NFIX(n, type_),
    'EXLD': Input.EXLD(n, force, type_),
    'IDBC': Input.IDBC(n-1),
    'PROP': np.array([[200e6], [0], [0], [0], [0]]),
    'SECT': Input.SECT(n-1, W, H),
    'FEF': Input.FEF(n-1, 1, load),
}

# SA main code
dis_record, dis_record_best = [], []
best = inf
loc = n-3 if type_ == 1 else -2
for step in range(SA_setting['steps']):
    if step == 0:
        t = SA_setting['t0']
        Delta = Solve.Execute(**config)[loc].item()
    else:
        # cooling schedule
        t = SA.schedule(t)
    # n_iter function
    n_iter = SA.n_iter(SA_setting['t0'], t)
    for iter in range(round(n_iter)):
        H_next = SA.change(H, constraint, 3)
        config['SECT'] = Input.SECT(n-1, W, H_next)
        Delta_next = Solve.Execute(**config)[loc].item()
        diff = abs(Delta_next) - abs(Delta)
        if diff < 0 or random() < SA.accept(diff*500, t):
            H = H_next
            break
```

```
    config['SECT'] = Input.SECT(n-1, W, H)

    Delta = Solve.Execute(**config)[loc].item()

    # save the best solution
    if abs(Delta) < best:
        H_best = H
        best = abs(Delta)

    dis_record.append(abs(Delta))
    dis_record_best.append(best)

    print(f'Step {step+1:3d}: Displacement {abs(Delta):.3f} m')

print('-------------------------')
print(f'Total volumn {(H_best*W*1).sum():.3f} m^3')
print('optimal H list', H_best)
print('optimal Displacement {:.3f} m'.format(best))
Plot.plot(SA_setting['steps'], dis_record, dis_record_best,
          config['COOR'], n, total_length, H_best)
```

Input.py:

```
import numpy as np
from math import floor
import random

def NFIX(n, type_):
    NFIX = np.zeros([2, n])
    if type_ == 1:
        NFIX[:, [0, -1]] = -1
    else:
        NFIX[:, 0] = -1
    return NFIX

def EXLD(n, force, type_):
    EXLD = np.zeros([2, n])
    if type_ == 1:
        node = floor(n/2)
        EXLD[0, int(node)] = -force
    else:
        EXLD[0, -1] = -force
    return EXLD

def IDBC(e):
    IDBC = np.zeros([5, e])
    IDBC[2, :] = 1
    IDBC[3, :] = np.arange(1, e+1)
    IDBC[0] = np.arange(1, e+1)
    IDBC[1] = IDBC[0] + 1
    return IDBC.astype('int')

def SECT(e, W, h_list):
    h_array = np.array(h_list)
    SECT = np.zeros([5, e])
    I_array = W * h_array**3 / 12
    SECT[1, :] = I_array
    return SECT
```

```
def FEF(e, l, load):
    FEF = np.zeros([4, e])
    FEF[[0, 2], :] = load*l / 2
    FEF[1, :] = load * l**2 / 12
    FEF[3, :] = -load * l**2 / 12
    return FEF

def random_H(e, constraint):
    upper = constraint / (e-1)
    while True:
        h_list = np.random.uniform(0.001, upper, e)
        if h_list.sum() <= constraint:
            break
    h_list = np.round(h_list, 3)
    return h_list
```

SA.py:

```
import numpy as np
import random
from math import *

def schedule(T_pre):  # cooling schedule
    return T_pre / (1+0.001*T_pre)

def accept(delta, t): return exp(-delta/t)  # Acceptance probability function

def n_iter(t0, t): return 1.5*(t0-t) + 10  # n_iter function

def change(A_list, constraint, num):
    # sample elements to change the Height
    ele = random.sample([_ for _ in range(len(A_list))], num)
    idx = [i for i in range(len(A_list)) if i not in ele]
    upper = constraint - sum(A_list[idx])
    A_copy = A_list.copy()
    while True:
        A_copy[ele] = np.random.uniform(0.001, upper, num)
        if A_copy.sum() <= constraint:
            break
    A_copy = np.round(A_copy, 3)
    return A_copy
```

Solve.py:

```
import numpy as np
from math import *

# %%
def IDMAT(NFIX, NNOD, NDN):
    IDND = np.zeros([NDN, NNOD])
    a, b = 0, 0
    for i in range(NNOD):
        for j in range(NDN):
            if NFIX[j, i] == 0:
                a += 1
                IDND[j, i] = a
            elif NFIX[j, i] < 0:
                b -= 1
                IDND[j, i] = b
            else:
                IDND[j, i] = IDND[j, NFIX[j, i]-1]
    NEQ = np.max(IDND)
    return IDND, NEQ

# %%
def MEMDOF(IDBC, IDND, NDE, NBC, NDN):
    LM = np.zeros([NDE, NBC])
    for j in range(NBC):
        for i in range(NDE):
            a = ceil((i+1)/NDN)
            node = IDBC[a-1, j]
            k = (i+1) % NDN
            if k == 0:
                k = NDN
            LM[i, j] = IDND[k-1, node-1]
    return LM

# %%
def SEMIBAND(LM, NDE, NBC):
    A = LM.copy()
```

```
    max_LM = A.max(0)
    for i in range(NBC):
        for j in range(NDE):
            if LM[j, i] < 0:
                A[j, i] = max_LM[i]
    NSBAND = (max_LM - A.min(0) + 1).max(0)
    return NSBAND

# %%
def ELKE(NDE, IDBC, PROP, SECT, IB, RL):
    E = PROP[0, IDBC[2, IB]-1]
    NU = PROP[1, IDBC[2, IB]-1]
    A = SECT[0, IDBC[3, IB]-1]
    Iz = SECT[1, IDBC[3, IB]-1]
    Iy = SECT[2, IDBC[3, IB]-1]
    J = SECT[3, IDBC[3, IB]-1]

    EE = E*Iz/RL*np.array([[12/RL**2, 6/RL, -12/RL**2, 6/RL],
                           [6/RL, 4, -6/RL, 2],
                           [-12/RL**2, -6/RL, 12/RL**2, -6/RL],
                           [6/RL, 2, -6/RL, 4]])
    return EE

# %%
def ROTATION(COOR, IDBC, MN, NCO, NDE):
    CO = COOR[:NCO, IDBC[:2, MN]-1].T
    RL = np.sqrt(np.sum((CO[1, :] - CO[0, :])**2))
    ROT = np.eye(2)
    T = np.zeros([int(NDE), int(NDE)])
    M = 2
    for i in range(int(NDE/M)):
        dof = np.arange(M) + i*M
        s = dof[0]
        d = dof[-1]
        T[dof, s:d+1] = ROT
    return T, RL

# %%
```

```
def LOAD(EXLD, IDND, NDN, NNOD, NEQ):
    GLOAD = np.zeros([int(NEQ), 1])
    for j in range(NNOD):
        for i in range(NDN):
            if IDND[i, j] > 0:
                ind = int(IDND[i, j]) - 1
                GLOAD[ind] = GLOAD[ind] + EXLD[i, j]

    return GLOAD

# %%
def FORMKP(COOR, IDBC, PROP, SECT, LM, FEF, GLOAD, NNOD, NBC,
           NMAT, NSEC, NCO, NDN, NDE, NNE, NEQ, IFORCE):
    GLK = np.zeros([int(NEQ), int(NEQ)])

    for IB in range(NBC):
        T, RL = ROTATION(COOR, IDBC, IB, NCO, NDE)

        EE = ELKE(NDE, IDBC, PROP, SECT, IB, RL)
        LDOF = np.where(LM[:, IB] > 0)
        GDOF = LM[LDOF, IB].astype('int') - 1
        ELK = np.dot(np.dot((T.T), EE), T)

        GDOF = np.array(GDOF).reshape(-1)
        GLK_s = GLK[GDOF]
        ELK_s = ELK[LDOF]
        GLK_s[:, GDOF] = GLK_s[:, GDOF] + ELK_s[:, LDOF].squeeze()
        if IFORCE != 1:
            EFEQ = np.dot(-T.T, FEF[:, IB])
            GLOAD[GDOF] = GLOAD[GDOF] + EFEQ[LDOF].reshape(GLOAD[GDOF].shape)
        GLK[GDOF] = GLK_s
    return GLK, GLOAD

# %%
def SOLVE(GLK, GLOAD):
    return np.dot(np.linalg.inv(GLK), GLOAD)

# %%
```

```
def Execute(NNOD, NBC, NMAT, NSEC,  NNE, IFORCE, COOR, NFIX, EXLD, IDBC, PROP,
SECT, FEF):

    IPR = np.array([[1, 2, 2, 2, 3, 3], [2, 2, 3, 3, 3, 6]])
    NCO = IPR[0, 0]
    NDN = IPR[1, 0]
    NDE = NDN*NNE

    IDND, NEQ = IDMAT(NFIX, NNOD, NDN)
    LM = MEMDOF(IDBC, IDND, NDE, NBC, NDN)
    GLOAD = LOAD(EXLD, IDND, NDN, NNOD, NEQ)
    GLK, GLOAD = FORMKP(COOR, IDBC, PROP, SECT, LM, FEF, GLOAD,
                        NNOD, NBC, NMAT, NSEC, NCO, NDN, NDE, NNE, NEQ, IFORCE)
    DELTA = SOLVE(GLK, GLOAD)
    return DELTA
```

Plot.py:

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

def history(steps, record, best_record):
    x = np.arange(1, steps+1)
    plt.plot(x, record, 'b--', label='Displacement record')
    plt.plot(x, best_record, 'r-', label='Converge history')
    plt.xlabel('Steps')
    plt.ylabel('Displacement (m)')
    plt.title("Converge History")
    plt.grid(True)
    plt.legend()
    # plt.show()

def structure(coord, n, L, H):
    y = np.zeros(n)
    coord = coord.ravel()
    mean = np.mean(H)
    max_ = max(H)
    for i in range(n-1):
        xx = coord[i:i+2]
        yy = y[i:i+2] + H[i]
        plt.plot(xx, yy, 'orange')
        plt.fill_between(xx, yy, color='orange')
    plt.xlim(0, L)
    plt.ylim(-max_/2, max_ + mean/1.5)
    plt.xlabel('X direction (m)')
    plt.ylabel('Height (m)')
    plt.title('Beam Shape')
    # plt.show()

def plot(steps, record, best_record, coord, n, L, H):
    plt.figure(figsize=(14, 5))
    plt.subplot(1, 2, 1)
    history(steps, record, best_record)
    plt.subplot(1, 2, 2)
```

```
structure(coord, n, L, H)
plt.show()
```