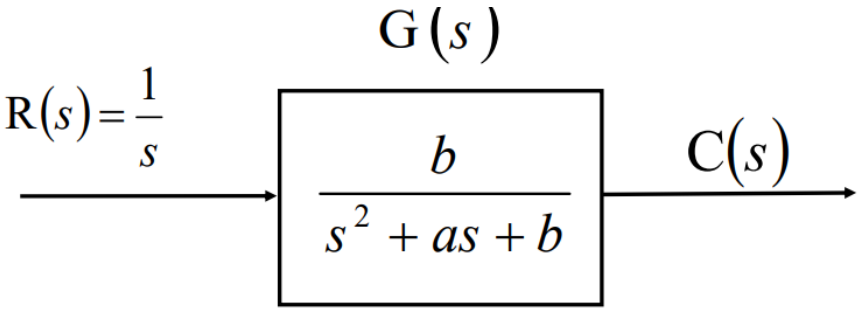
期末報告:四階系統的動態響應

109303570 李芳誼、109303567 陳薇如

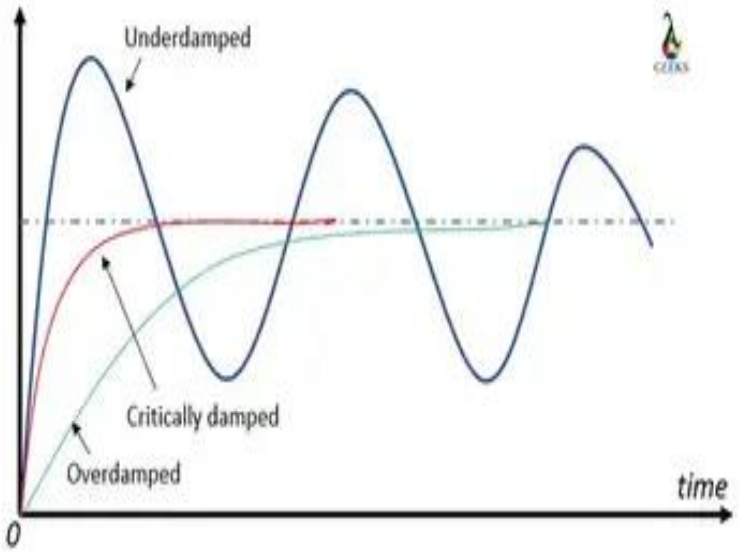
前言

在工程領域中，控制系統通常為高階系統，而分析高階系統最主要的方式，是將高階系統先轉換成較簡易的一階或二階系統進行分析。因此，若要明白四階系統，則可先從較低階之系統著手，再延伸至四階系統。

我們可以先設立一個二階系統如下:



接著我們將透過這個二階系統介紹三種不同情況的響應，如圖一顯示，分別為臨界阻尼、過阻尼、無阻尼的響應情形。

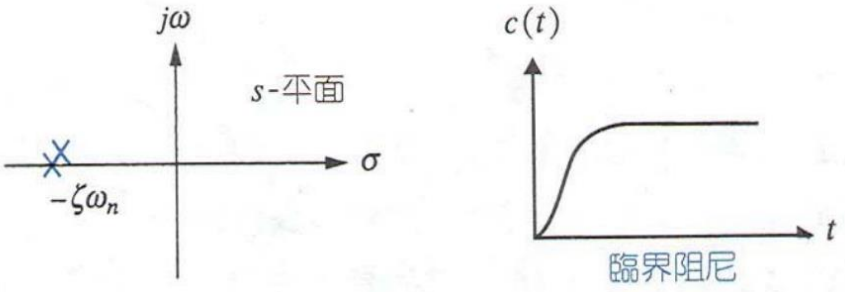


圖一、系統中不同的響應情形

1. 臨界阻尼 critical-damped

臨界阻尼是系統響應的理想狀況，此情形發生在阻尼係數到達臨界值時，由圖一可觀察出系統能夠在最短的時間內收斂，並且不震盪。

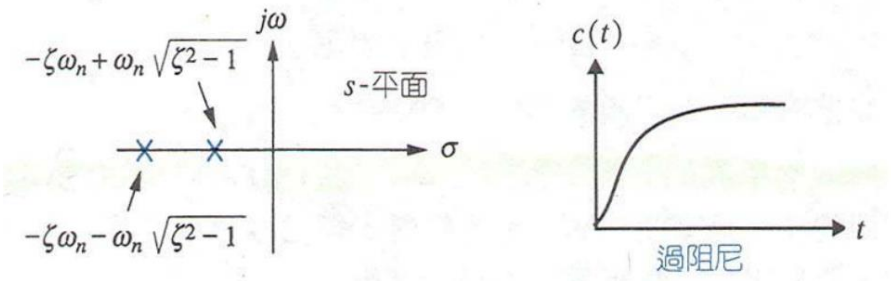
以二階系統而言，此時$G(s)=\frac{\omega_n^2}{S^2+2\omega_n S+\omega_n^2}$，阻尼比$\zeta=1$，所以$S=-\zeta\omega_n$



2. 過阻尼 overdamped

當阻尼係數大於臨界阻尼時，即為過阻尼。由圖一可得知，雖然過阻尼並無震盪現象，但收斂時間明顯較臨界阻尼還要久，反應較慢。

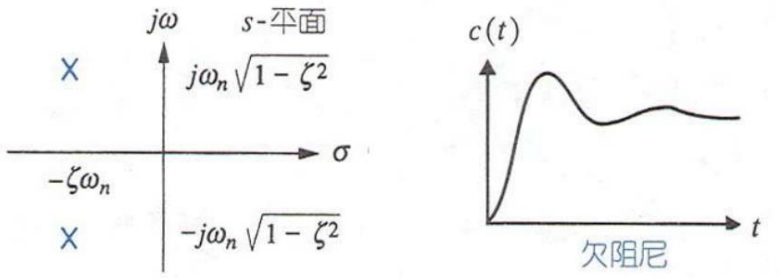
以二階系統而言，此時$G(s)=\frac{\omega_n^2}{S^2+2\zeta\omega_n S+\omega_n^2}$，阻尼比$\zeta>1$，所以$S=-\zeta\omega_n\pm\omega_n\sqrt{\zeta^2-1}$



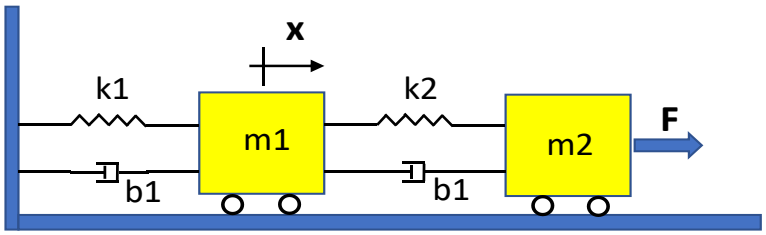
3. 欠阻尼 under-damped

當阻尼係數小於臨界阻尼時，即為欠阻尼或低阻尼。在此情況下系統將因為阻尼係數過小而產生震盪現象，如圖一。

以二階系統而言，此時$G(s)=\frac{\omega_n^2}{S^2+2\zeta\omega_n S+\omega_n^2}$，阻尼比$0<\zeta<1$，所以$S=-\zeta\omega_n\pm j\omega_n\sqrt{\zeta^2-1}$



數學模型推導



假設 m2 位移為 y，並分別以 m1 和 m2 為節點列出下面兩個式子：

$$\mathrm{m1}\ddot{x} + (b1 + b2)\dot{x} + (k1 + k2)x = b2(\dot{y}) + k2(y)$$
$$\mathrm{m2}(\ddot{y}) + \mathrm{b2}(\dot{y}) + \mathrm{k2(y)} + \mathrm{F} = \mathrm{b2}(\dot{x}) + \mathrm{k2(x)}$$

進行拉普拉斯轉換：

$$[\mathrm{m1}S^2 + (b1 + b2)S + (k1 + k2)]x = [b2(S) + k2](y)$$
$$[\mathrm{m2}(S^2) + \mathrm{b2(S)} + \mathrm{k2}]\mathrm{y} + \mathrm{F} = [\mathrm{b2(S)} + \mathrm{k2}](\mathrm{x})$$

將 y 代換掉：

$$x = \frac{[b2(S) + k2](y)}{[\mathrm{m1}S^2 + (b1 + b2)S + (k1 + k2)]}$$

$$\mathrm{y} = \mathrm{x}\frac{[\mathrm{m1}S^2 + (b1 + b2)S + (k1 + k2)]}{[b2(S) + k2]}$$

$$\mathrm{F} = [\mathrm{m2}(S^2) + \mathrm{b2(S)} + \mathrm{k2}](\mathrm{x})\frac{[\mathrm{m1}S^2 + (b1 + b2)S + (k1 + k2)]}{[b2(S) + k2]} - [\mathrm{b2(S)} + \mathrm{k2}](\mathrm{x})$$

$$= \left\{[\mathrm{m2}(S^2) + \mathrm{b2(S)} + \mathrm{k2}]\frac{[\mathrm{m1}S^2 + (b1 + b2)S + (k1 + k2)]}{[b2(S) + k2]} - [\mathrm{b2(S)} + \mathrm{k2}]\right\}x$$

最後整理並寫出$\frac{F}{X}$：

$$\frac{F}{x} = \left\{[\mathrm{m2}(S^2) + \mathrm{b2(S)} + \mathrm{k2}]\frac{[\mathrm{m1}S^2 + (b1 + b2)S + (k1 + k2)]}{[b2(S) + k2]} - [\mathrm{b2(S)} + \mathrm{k2}]\right\}$$

程式流程與程式碼說明

由於電腦性能原因，我們這次報告採用 maplotlib 而非 vpython 作為製圖工具。

1. 時域分析程式:

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. from scipy import signal

4. #---系統參數-----
5. m1, b1, k1 = 1,1,1
6. m2, b2, k2 = 1,1,1

7. #---轉移函數係數----
8. A0=m1*m2
9. A1=b2*m1+b1*m2+b2*m2
10. A2=k2*m1+b1*b2+k1*m2+k2*m2
11. A3=k2*b1+b2*k1
12. A4=k1*k2

13. B0=k2
14. B1=b2

15. #---轉移函數設定----
16. num = [B1,B0]
17. den = [A0,A1,A2,A3,A4]
18. system = signal.TransferFunction(num, den)

19. #---時軸設定-----
20. period = 2 * 10
21. f = 1 / period
22. w = 2 * np.pi * f
23. N = 3
24. sample = 50
25. t = np.linspace(0, period*N, num=period*N*sample,endpoint=False)
```

```
26. #---Step response------
27. t_step, y_step = signal.step(system, T = t, N = period*N*sample)

28. #---Square response------
29. input_square = signal.square(w * t)
30. t_square,y_square,x_square =
    signal.lsim(system,input_square,T=t)

31. #---作圖-----
32. #<step>
33. plt.subplot(211)
34. plt.title("4rd system step response")
35. input_step=[1] * t.size
36. plt.plot(t_step,input_step , color='gray', label="Input: Step")
37. plt.plot(t_step, y_step, color='blue', label="Output: Step")
38. plt.xlabel('time(sec)')
39. plt.ylabel( 'Amplitude')
40. plt.grid()
41. plt.legend()
42. #<square>
43. plt.subplot(212)
44. plt.title("4rd system square response")
45. plt.plot(t_square, input_square, color='gray', label="Input:
    Square")
46. plt.plot(t_square, y_square,color='r', label="Output: Square")
47. plt.xlabel('time(sec)')
48. plt.ylabel('Amplitude')
49. plt.grid()
50. plt.legend()
51. plt.show()
```

第 7-18 行，由數學模型推導出的轉移函式。

第 19-25 行，輸入波之必要參數，包含頻率、周期，以及時軸設定。

第 26-27 行，步階訊號的輸入訊號和響應。

第 28-30 行，方波訊號的輸入訊號和響應。

第 32-41 行，步階訊號的作圖。

第 43-51 行，方波訊號的作圖。

2. 頻域分析程式:

```
1. import numpy as np
2. import matplotlib.pyplot as plt
3. from scipy import signal

4. #---系統參數-----
5. m1, b1, k1 = 1,1,1
6. m2, b2, k2 = 1,1,1

7. #---預計要模擬的參數-----
8. my_list=[0.01,0.1,1,10,100]
9. label="k1/k2="

10.#---製作不同參數的轉移函數---
11.system=[]
12.for i in my_list:
13.#---轉移函數係數----
14.    k1=1
15.    k2=k1/i
16.    A0=m1*m2
17.    A1=b2*m1+b1*m2+b2*m2
18.    A2=k2*m1+b1*b2+k1*m2+k2*m2
19.    A3=k2*b1+b2*k1
20.    A4=k1*k2

21.    B0=k2
22.    B1=b2

23.    #---轉移函數設定----
24.    num = [B1,B0]
25.    den = [A0,A1,A2,A3,A4]
```

```
26.     system1 = signal.TransferFunction(num, den)
27.     system.append(system1)

28. #---x 軸設定-----
29. f = np.logspace(-2, 3, num=1000)
30. w = 2 * np.pi * f

31. #---y 軸響應(大小、相位)-----
32. mag=[]
33. phase=[]
34. for sys in system:
35.     w, mag1, phase1 = signal.bode(sys, w)
36.     mag.append(mag1)
37.     phase.append(phase1)

38. #---大小/頻率-----
39. plt.subplot(211)
40. for i in range(len(my_list)):
41.     plt.semilogx(w, mag[i], label=label+str(my_list[i]))
42. plt.title("4th system Bode diagram")
43. plt.xlabel('frequency(rad/sec)')
44. plt.ylabel('magnitude(db)')
45. plt.legend()
46. plt.grid()

47. #---相位/頻率-----
48. plt.subplot(212)
49. for i in range(len(my_list)):
50.     plt.semilogx(w, phase[i], label=label+str(my_list[i]))
51. plt.xlabel('frequency(rad/sec)')
52. plt.ylabel('phase(degree)')
53. plt.legend()
54. plt.grid()
55. fig = plt.gcf()
56. fig.set_size_inches(18,12)
57. plt.show()
```

第 10-27 行，將不同參數的的轉移函式存入叫 system 的 list 中。

第 28-30 行，頻率軸設定。

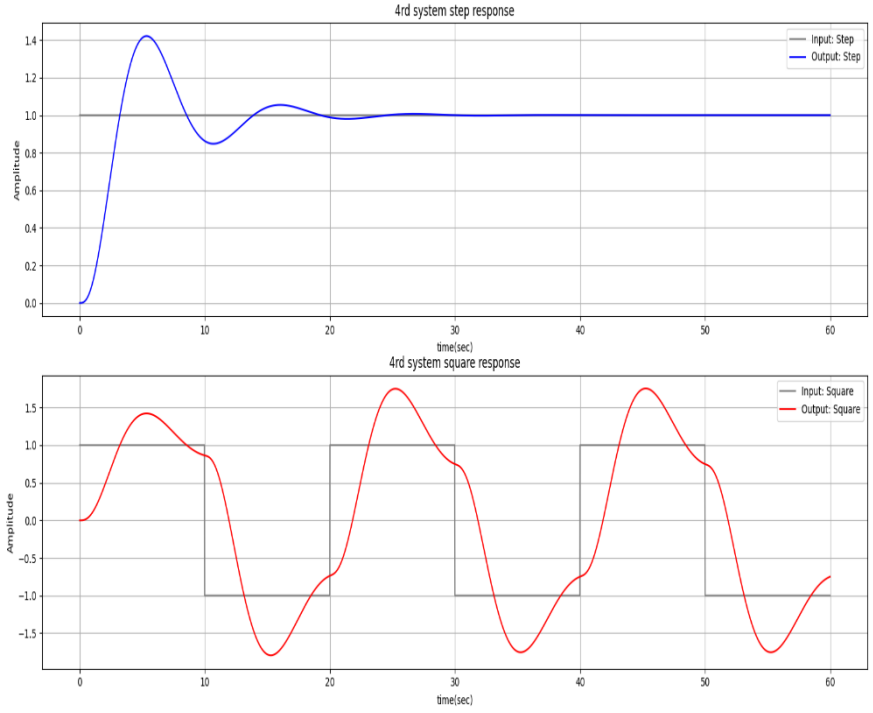
第 32-37 行，製作兩個圖的 y 軸的響應，並存在 list 中。

第 39-46 行，使用 list 製作大小、頻率圖。

第 58-56 行，使用 list 製作相位、頻率圖。

模擬結果分析

在討論參數之間的關係之前，我們有六個參數可以調整，分別是 m1、m2、b1、b2、k1 和 k2。為了測試波形輸出，我們將所有參數設置為 1。這樣可以讓我們在初始狀態下觀察系統的行為，並檢測任何可能的問題或改進的空間。



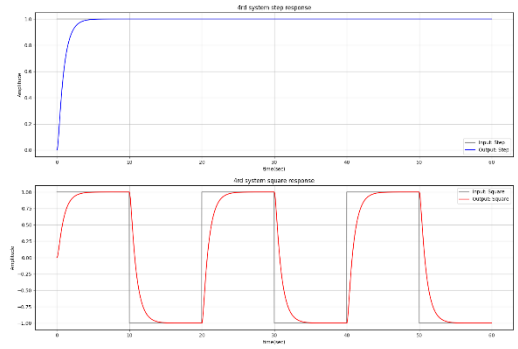
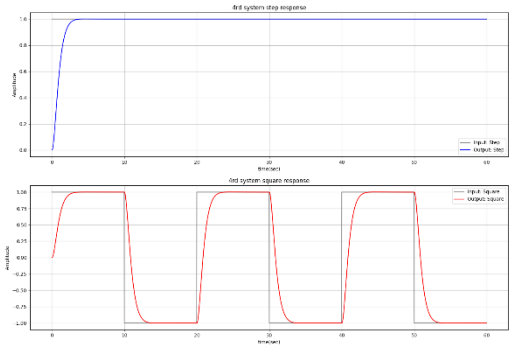
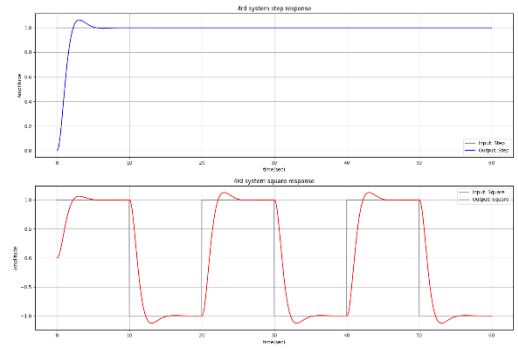
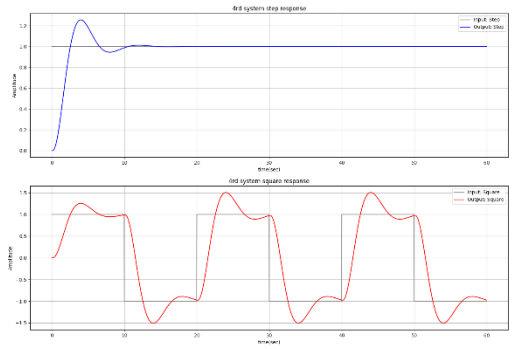
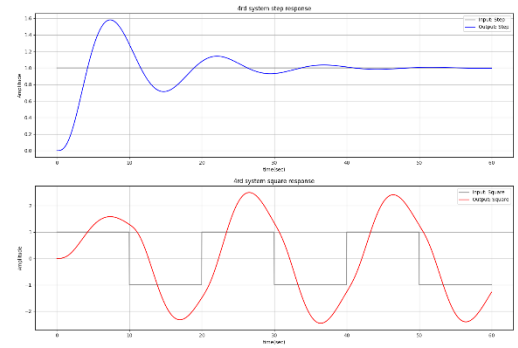
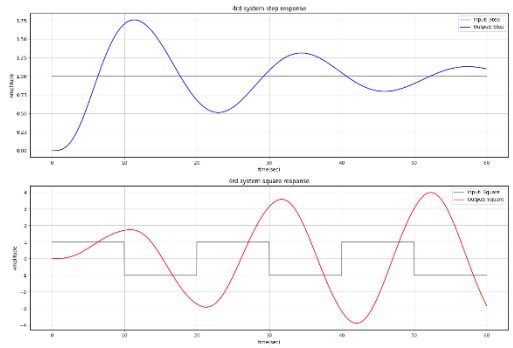
透過觀察上方的單位步階函數圖形以及下方輸入連續方波的圖形，我們可以藉由這些圖形來判斷系統對輸入信號的響應情況。這些圖形提供了關於系統行為和性能的重要資訊，使我們能夠了解系統在不同輸入條件下的運作方式和特性。

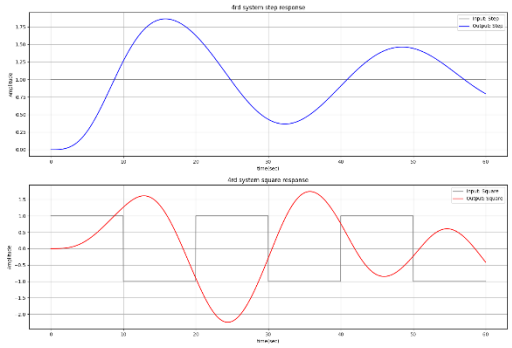
接下來，我們將分析分為三個部分。第一部分是針對相同類型係數的分析，我們將探討這些係數之間的關係和對系統行為的影響。第二部分是針對相同類型係數的比值進行分析，我們將研究這些比例對系統性能的影響程度。透過這兩部分的分析，可以更深入地了解系統各參數之間的相互作用。最後一部分為第三部分，主要講解 Vpython 模擬。

(1) 相同類型係數分析

為了探索不同參數對系統響應的影響，我們將進行一系列實驗。首先，我們將固定 b1、b2 和 k1、k2 的值，只變動 m1 和 m2。這樣可以讓我們觀察在不同 m 值下系統的行為變化。通過這樣的設計，我們能夠逐步理解 m 參數對系統的影響程度。接著，我們將持續進行類似的實驗，來研究 b 和 k 對系統的影響，以完整地分析系統的行為。

- 變更參數 m

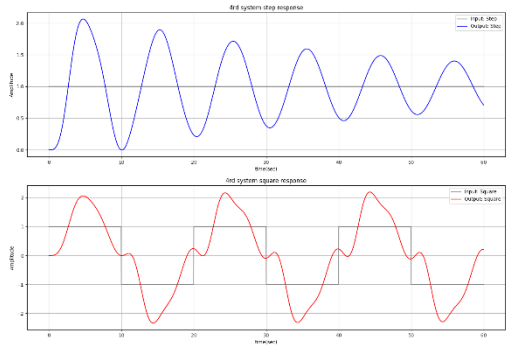
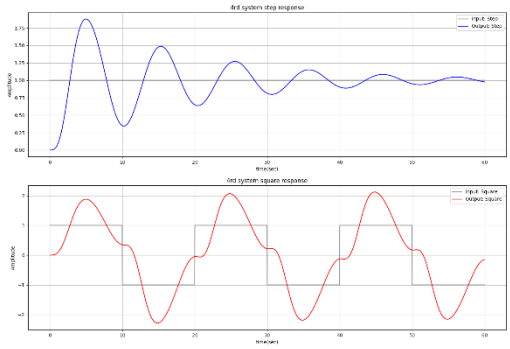
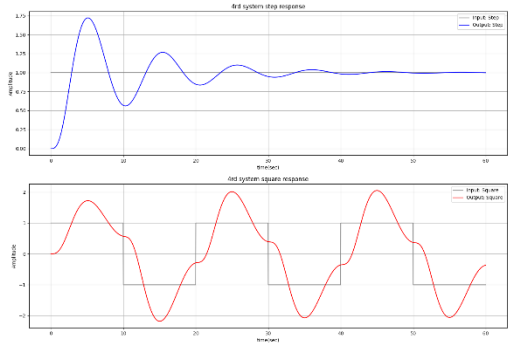
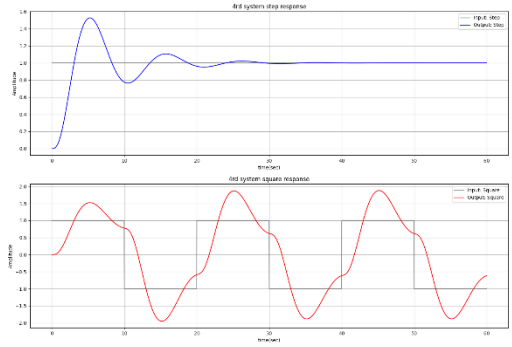
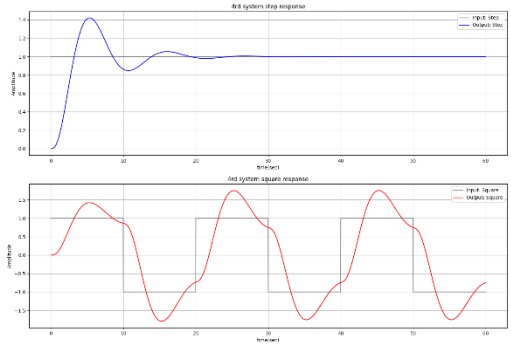
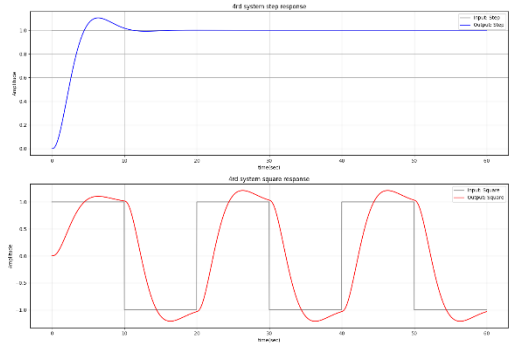
	
m1 = m2 = 0.05	m1 = m2 = 0.1
	
m1 = m2 = 0.2	m1 = m2 = 0.5
	
m1 = m2 = 2	m1 = m2 = 5

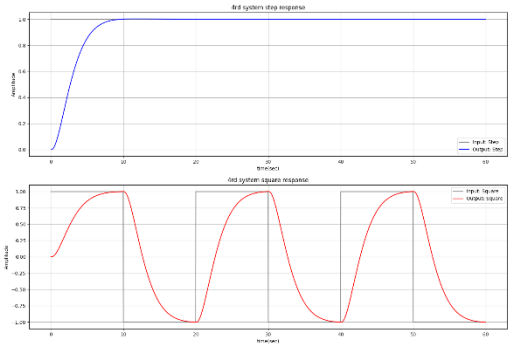
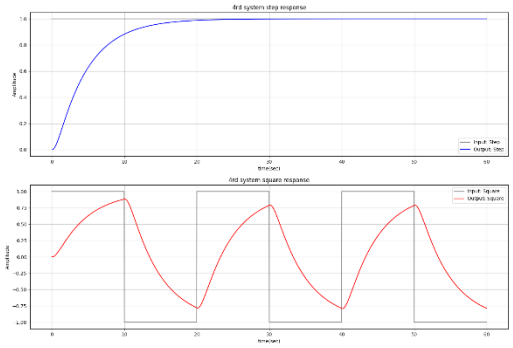
	
m1 = m2 = 10	

觀察圖中響應之曲線會發現，若是將 m 設得太小，如前兩張圖將 m1 和 m2 設為 0.05 和 0.1，系統響應出現了過阻尼的情況，此時阻尼比小於 1，響應曲線沒有什麼震盪情形且一開始就呈現直驅向上趨勢。當 m1 和 m2 設為 0.2 時，響應出現了震盪，並且隨著 m 值從 5 到 10 越來越大，震盪也越來越大，系統收斂時間明顯變長許多。

總的來說，系統的質量參數 m1 和 m2 對於系統的響應具有一定的影響力。較小的質量值導致系統呈現過阻尼行為，而較大的質量值則引發響應的震盪現象並延長收斂時間。

- 變更參數 b

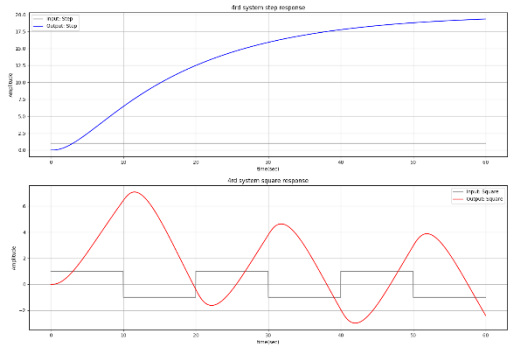
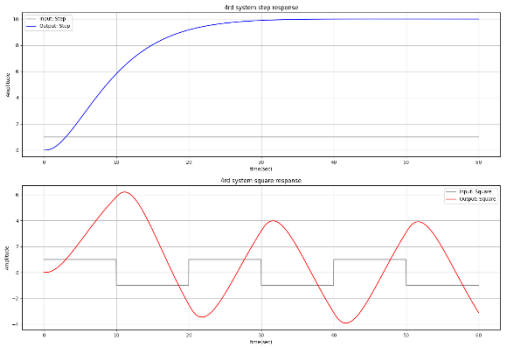
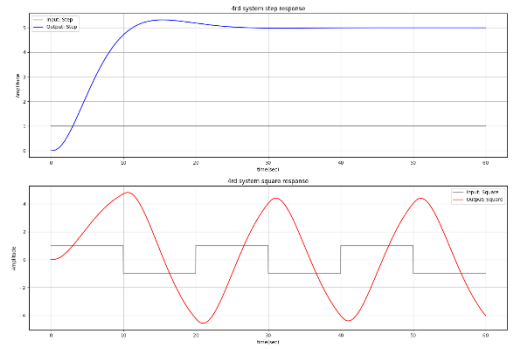
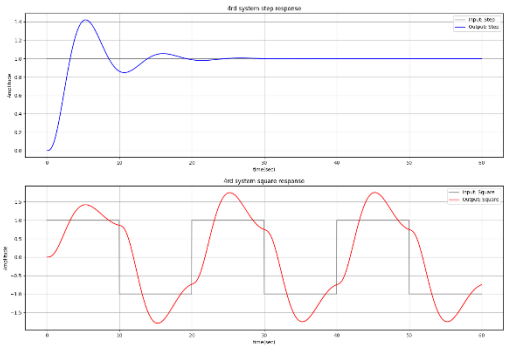
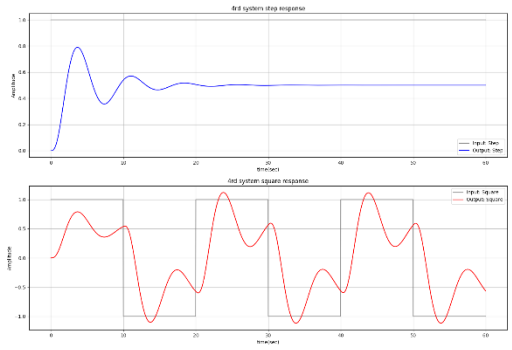
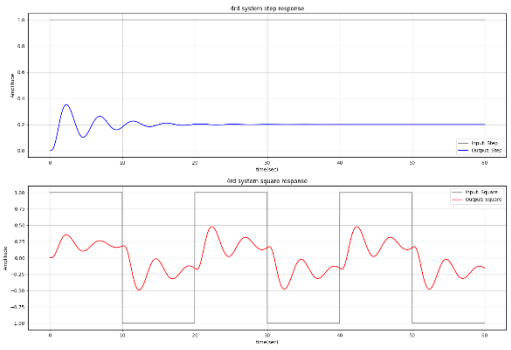
	
b1 = b2 = 0.1	b1 = b2 = 0.3
	
b1 = b2 = 0.5	b1 = b2 = 0.8
	
b1 = b2 = 1	b1 = b2 = 2

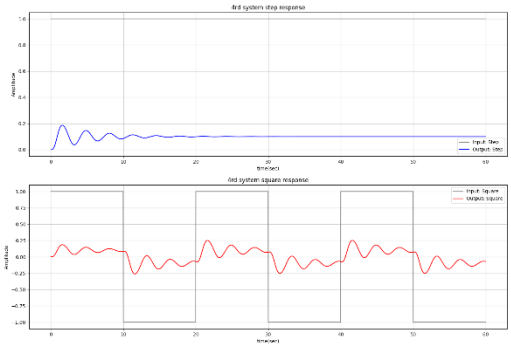
	
b1 = b2 = 3	b1 = b2 = 5

b 值大小對響應造成的影響和 m 值大小對響應造成的影響完全相反，我們一開始一樣將 b 值設的很小，設在 0.1 和 0.3，但響應曲線出現了明顯震盪，當我們把 b 值慢慢從 0.5 調小到 2 左右時，震盪明顯減少，系統收斂速度變快。在 b 值為 3 時，響應呈現出類似於臨界阻尼之情況，不僅沒有震盪，系統也快速收斂，是一個很漂亮的曲線。然而隨著 b 值調大來到 5 時，系統雖然仍無震盪，但收斂時間卻增加了，為過阻尼之情形。

總的來說，系統的阻尼參數 b1 和 b2 對於系統的響應具有一定的影響力。較小的阻尼值將導致系統呈現震盪，適當的阻尼值讓系統減少震盪並快速收斂，然而若阻尼值過大則將使系統收斂時間增加。

- 變更參數 k

	
k1 = k2 = 0.05	k1 = k2 = 0.1
	
k1 = k2 = 0.2	k1 = k2 = 1
	
k1 = k2 = 2	k1 = k2 = 5

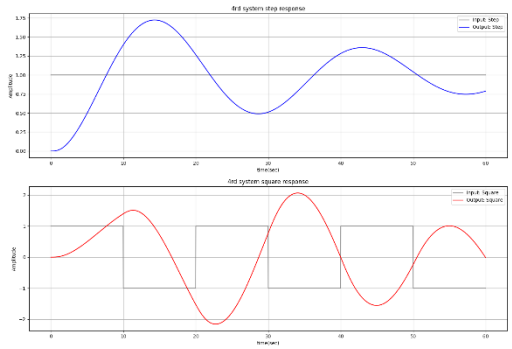
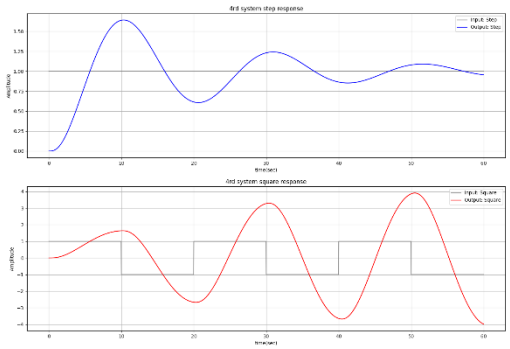
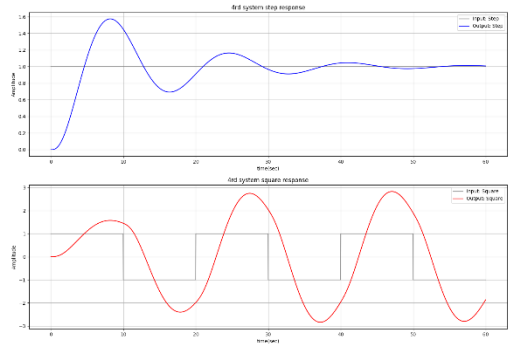
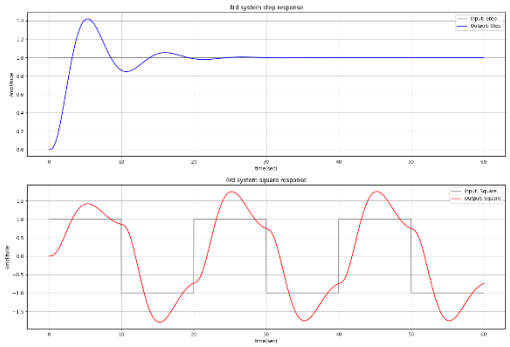
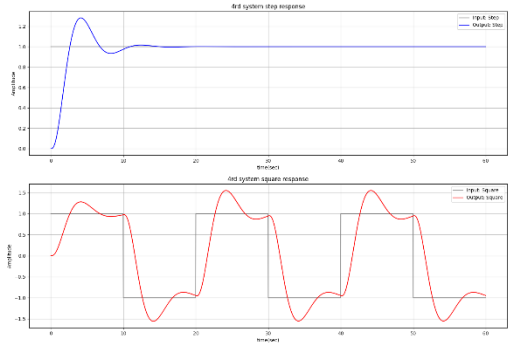
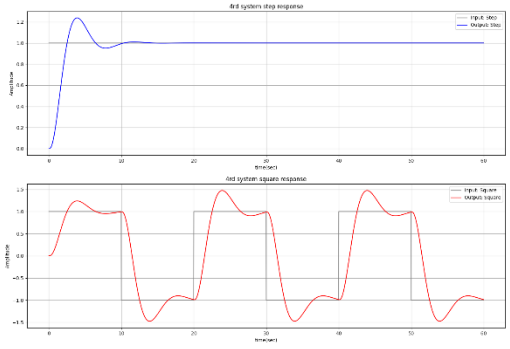


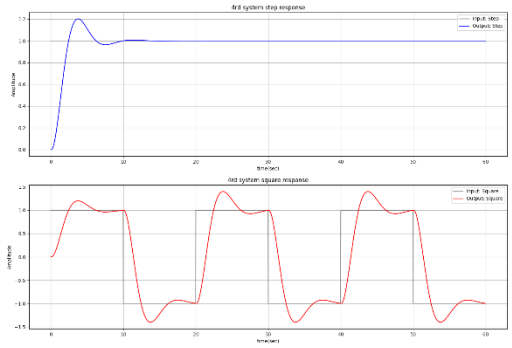
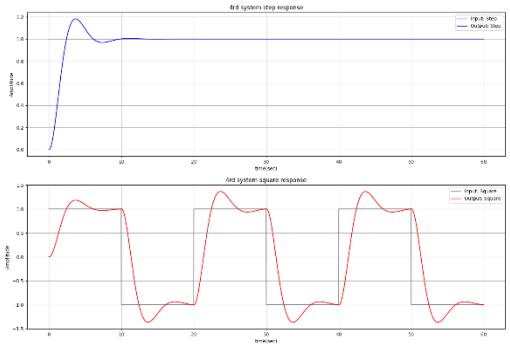
k1 = k2 = 10

由圖中可以觀察出當 k 值在 0.05 時，雖然響應之曲線並無震盪，但收斂時間特別長，是目前實驗下來最慢收斂的情況。當我們把 k 值慢慢調大，會發現系統開始出現震盪，且震盪頻率增加，但收斂所需時間卻變少了。這是因為 k 值增加將使系統自然頻率也跟著變大，因此響應時震盪頻率也增加。除此之外，k 值的增加也會使系統剛性增強，讓系統回復的時間大幅縮短。

(2) 相同類型係數的比值分析

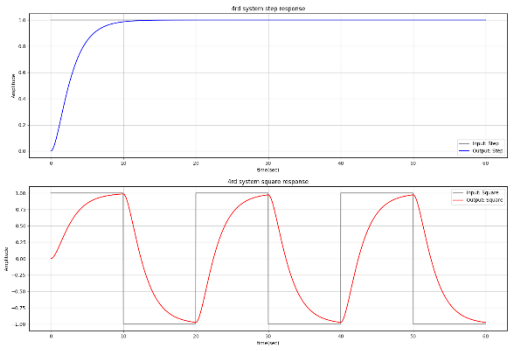
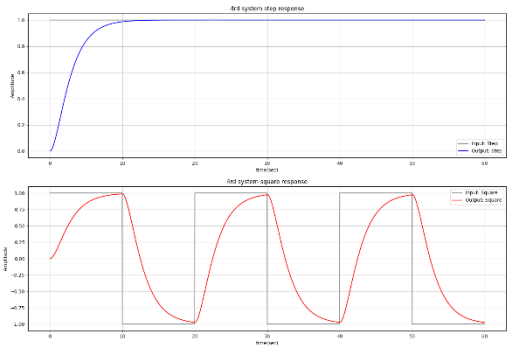
- 變更參數 m1/m2

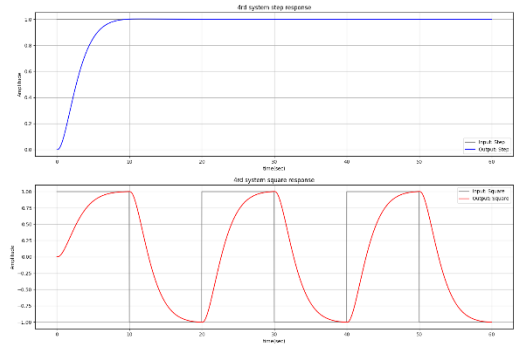
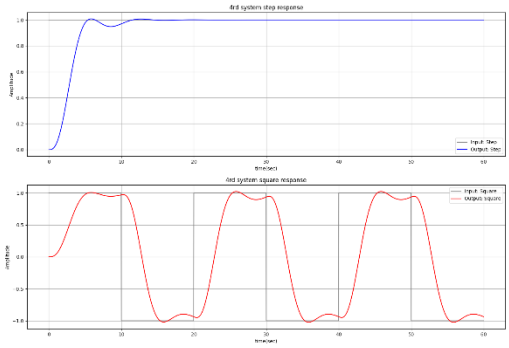
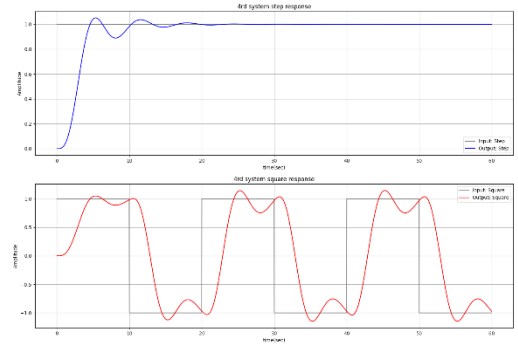
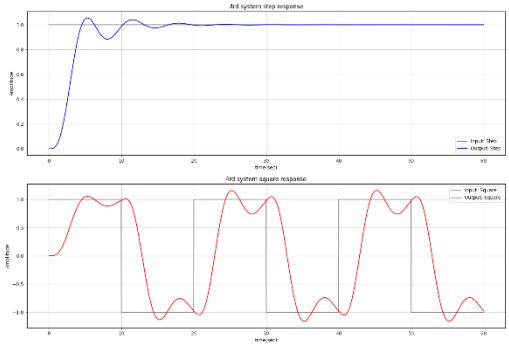
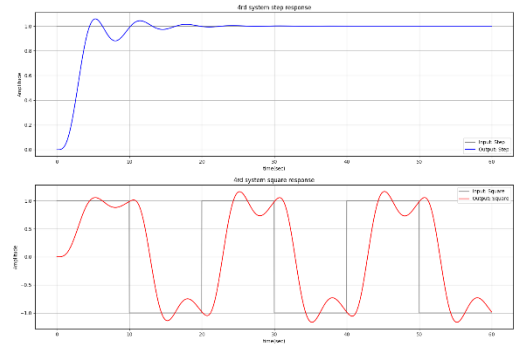
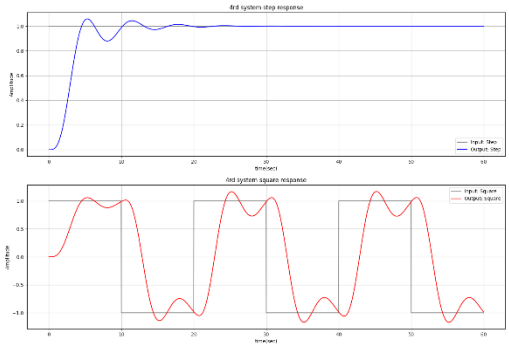
	
m1/m2 = 0.1	m1/m2 = 0.2
	
m1/m2 = 1/3	m1/m2 = 1
	
m1/m2 = 3	m1/m2 = 5

	
m1/m2 = 10	m1/m2 = 20

一開始我們將 m1/m2 的值設的很小，只有 0.1，發現產生出的響應震盪非常大，方波的部分則是呈現弦波的樣子。隨著 m1/m2 的值慢慢調大變成 1 之後的響應震盪開始變小，收斂時間變少了，而原本弦波的樣子也逐漸呈現出方波的樣子。

- 變更參數 b1/b2

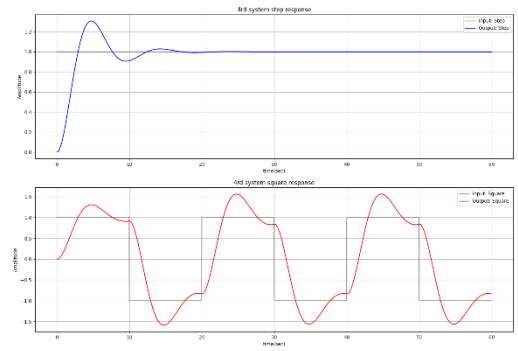
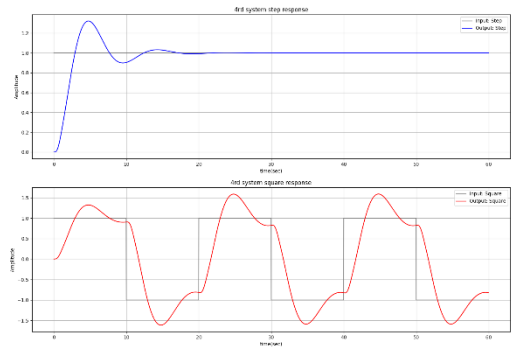
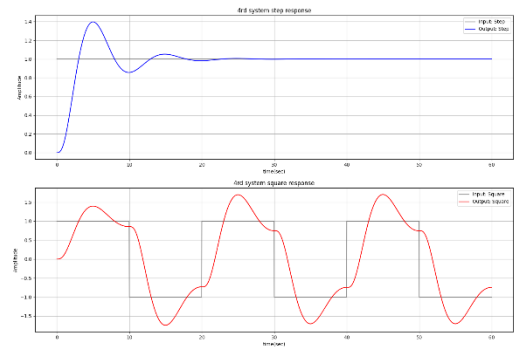
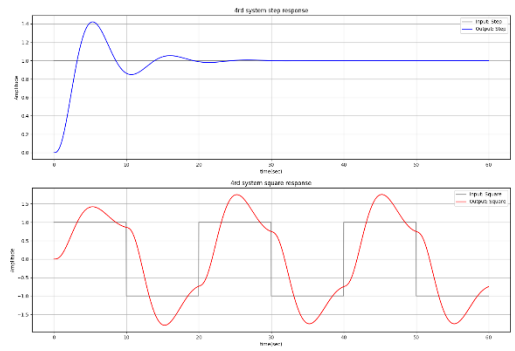
	
b1/b2 = 0.01	b1/b2 = 0.1

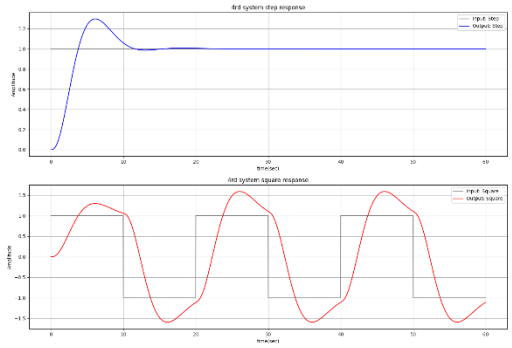
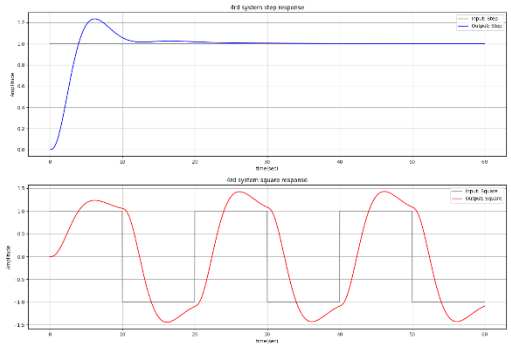
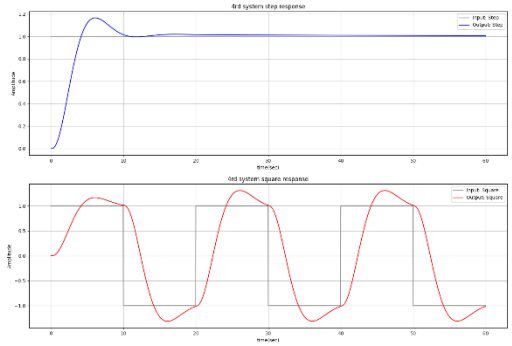
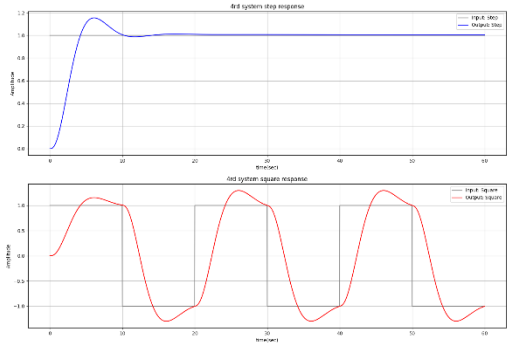
	
b1/b2 = 1	b1/b2 = 10
	
b1/b2 = 100	b1/b2 = 200
	
b1/b2 = 500	b1/b2 = 1000

b1/b2 我們一樣由小的值開始往上測試，從 0.01 到 1 左右響應皆呈現過阻尼的結果，且方波顯示並不完整，但從 10 開始一直到 1000，兩

阻尼的值差距變大之後，響應的圖形其實都長得類似，曲線開始出現震盪，且方波變得更完整了一點。

- 變更參數 k1/k2

	
k1/k2 = 0.01	k1/k2 = 0.08
	
k1/k2 = 0.5	k1/k2 = 1

	
k1/k2 = 5	k1/k2 = 10
	
k1/k2 = 50	k1/k2 = 100

k1/k2 在我們測試的參數中，都呈現低阻尼的型態，當 k1/k2 小於 1 時，系統響應震盪多了一些，方波則是隨著 k1/k2 的值變大而變得比較平滑和穩定。

(3) Vpython 模擬

我們修改了前述製作方波響應的程式碼，作為 Vpython 動態物件模擬的輸入及響應。

```
from vpython import *
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal


g = 9.8  # 重力加速度 9.8 m/s^2
size = 0.05  # 球半徑 0.05 m
L = 0.5  # 彈簧原長 0.5m
k = 10  # 彈簧力常數 10 N/m
m = 0.1  # 球質量 0.1 kg
Fg = m * vector(0, -g, 0)  # 球所受重力向量


#---------場地------------
L = 2
scene = canvas(width=800,height=500, center=vector(0,-L*0.8,0),
    range=1.2*L)
floor = box(length=4,height=0.02, width=1.6, opacity=0.2)
floor.pos = vector(0, -L*1.5, 0)
wall = box(length=0.02, height=2, width=1, color=color.white,
    opacity=1)
wall.pos = vector(-2,-L*1.1,0)

#----------第一組------------
#方塊 m1 物件
object = box(length=0.6, height=0.6, width=0.6,
    color=color.white, opacity=1)
object.pos = vector(-0.4,-1.35*L,0)
#彈簧 k1 物件
spring = helix(radius=0.08, thickness=0.04)
spring.pos= vector(-2,-1.28*L,0)
spring.color = vector(0.7,0.5, 0.2)
```

```
spring.axis = 1.6 * vector(1,0,0)
#阻尼器 B1(外殼)
damperB = cylinder(radius=0.08, thickness=0.04)
damperB.pos = vector(-2, -1.45*L, 0)
damperB.color=vector(0.4,0.1,0.2)
damperB.axis = 0.6 * vector(1, 0, 0)
#阻尼器 b1
damperb = cylinder(radius=0.04, thickness=0.04)
damperb.pos =vector(-1.4, -1.45*L, 0)
damperb.color = vector(0.3, 0.5, 0.1)
damperb.axis = 1*vector(1, 0, 0)

#----------第二組---------------
#彈簧 k2 物件
spring2 = helix(radius=0.08, thickness=0.04)
spring2.pos = vector(-0.1,-1.28*L,0)
spring2.color = vector(0.7,0.5,0.2)
spring2.axis = 1.6 * vector(1,0,0)
#方塊 m2 物件
object2 = box(length=0.6, height=0.6, width=0.6,
   color=color.white, opacity=1)#物體
object2.pos = vector(1.5, -1.35*L, 0)
#阻尼器 c2
damperc = cylinder(radius=0.04, thickness=0.04)
damperc.pos = vector(0.5, -1.45*L, 0)
damperc.color=vector(0.3,0.3,0.6)
damperc.axis = 1 * vector(1, 0, 0)
#阻尼器 C2(外殼)
damperC = cylinder(radius=0.08, thickness=0.04)
damperC.pos = vector(-0.1, -1.45*L, 0)
damperC.color = vector(0.6,0.5, 0.2)
damperC.axis = 0.6* vector(1, 0, 0)

#系統參數
m1 = 1
m2 = 1
b1 = 1
b2 = 1
```

```
k1 = 1
k2 = 1
#轉移函數係數
A4 = m1*m2
A3 = b2*m1+b1*m2+b2*m2
A2= k2*m1+b1*b2+k1*m2+k2*m2
A1 = k2*b1+b2*k1
A0 = k1 * k2
#X2
B1 = b2
B0 = k2
#X1
C2 = m1
C1 = b1+b2
C0 = k1+k2

t = 0.0001
dt = 0.002
inputAmplitude = 1.0
period = 2 * 10
f = 1 / period
w = 2 * np.pi * f
N = 3
sample = 50
t = np.linspace(0, period*N, num=period*N*sample,endpoint=False)

num1 = [B1,B0]
den1 = [A0,A1,A2,A3,A4]
system1 = signal.TransferFunction(num1, den1)
input_square = signal.square(w * t)
t_square,y_square1,x_square =
    signal.lsim(system1,input_square,T=t)


num2 = [B1,B0]
den2 = [C2,C1,C0]
system2 = signal.TransferFunction(num2, den2)
t_square,y_square2,x_square =
    signal.lsim(system2,input_square,T=t)
```

```
94. for i in range(t.size):
95.     rate(100)
96.     #---------spring 和 damper 的連動------------
97.     object2.pos = vector(1.2+0.5*y_square1[i], -1.35*L,0)
98.     object.pos = vector(-0.4+0.5*y_square2[i], -1.35*L,0)

99.     damperC.pos.x = object.pos.x
100.    damperc.pos.x = object.pos.x

101.    damperb.axis = vector((1.4+object.pos.x),0,0)
102.    damperc.axis = vector((object2.pos.x - object.pos.x),0,0)

103.    spring2.pos.x = object.pos.x
104.    spring2.axis = vector(object2.pos.x - object.pos.x,0,0)
105.    spring.axis = vector(object.pos.x - wall.pos.x,0,0)
```

第 11-55 行，製作兩個方塊作為 m1m2、兩個彈簧作為 k1k2、兩個阻尼作為 b1b2。

第 56-75 行，製作了兩個轉移函數的參數，分為 Xo 與 Fi 的轉移函數，及 X2 與 Fi 的轉移函數。

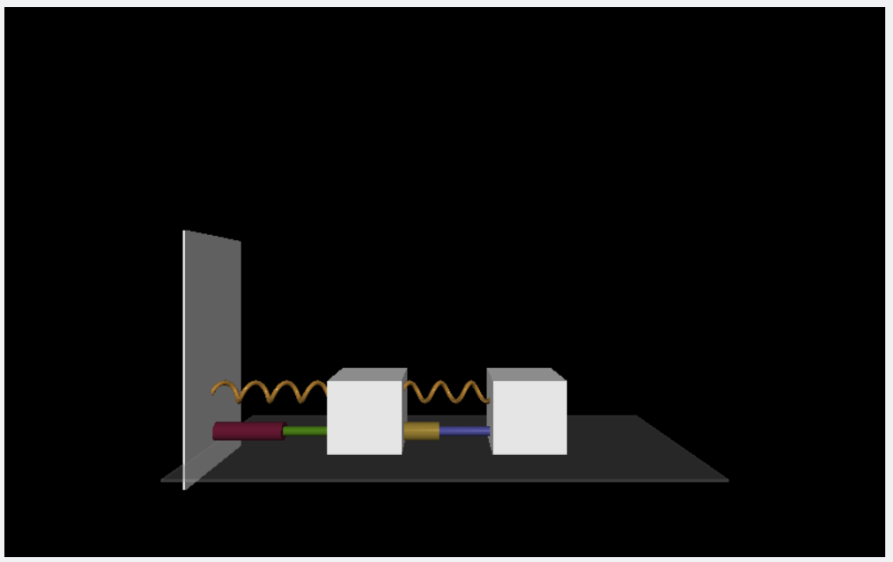
第 76-84 行，設定時軸需要的參數。

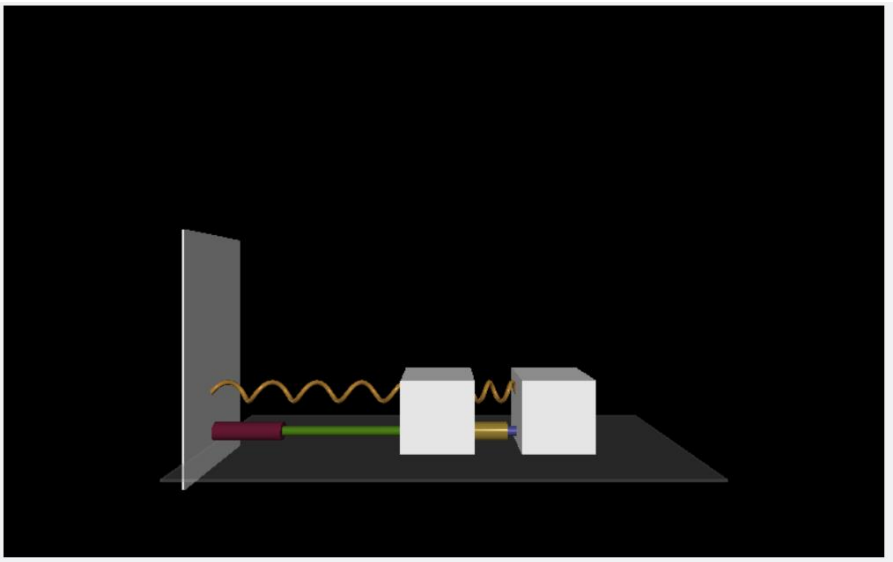
第 85-89 行，製作第一個轉移函數與 Xo 之響應。

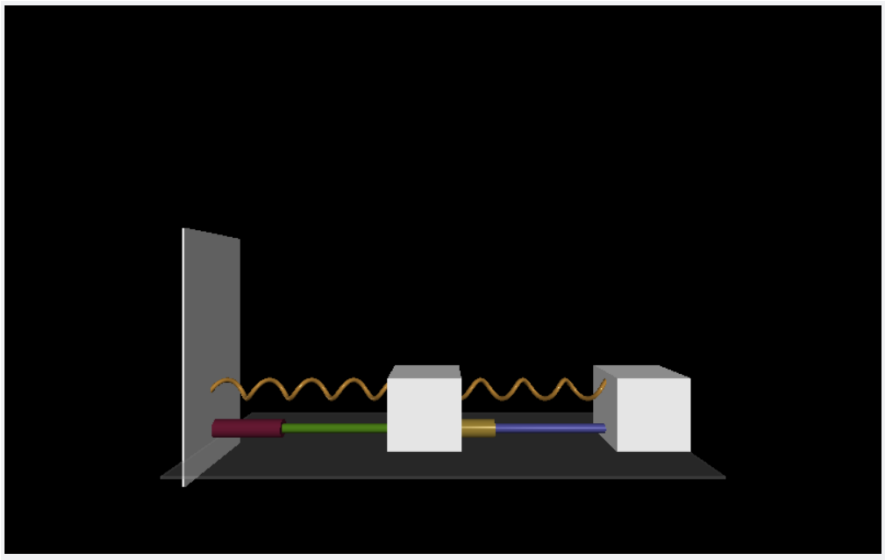
第 90-93 行，製作第二個轉移函數與 X2 之響應。

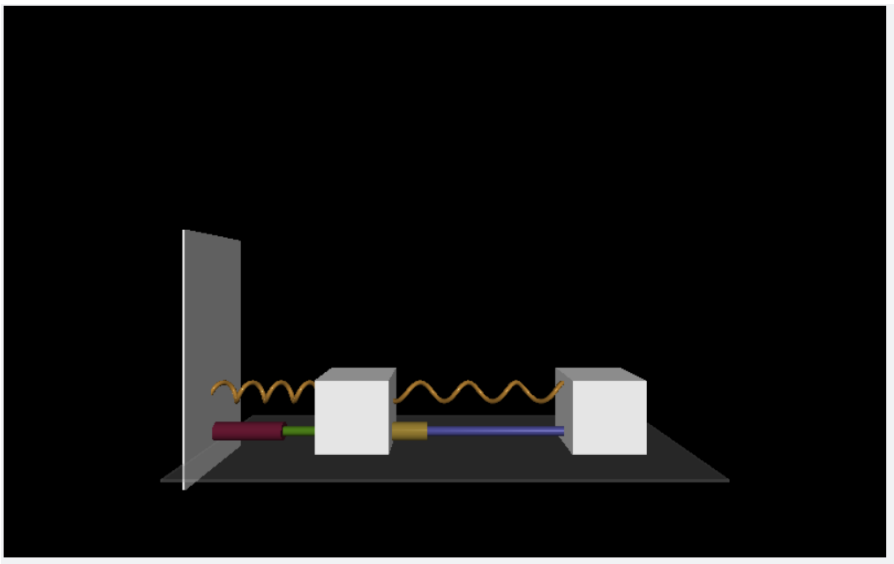
第 94-105 行，用 rate()設定速度，並且設定 m2 的移動、以及需要跟隨 m2 位置改變長度的 k1k2 及 b1b2。

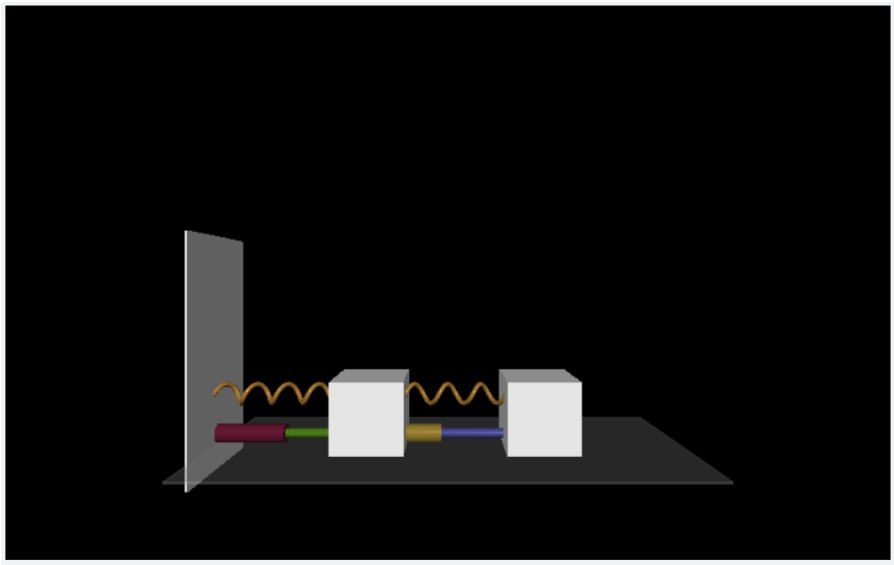
下圖為成果展示，移動順序按下圖安排順序。





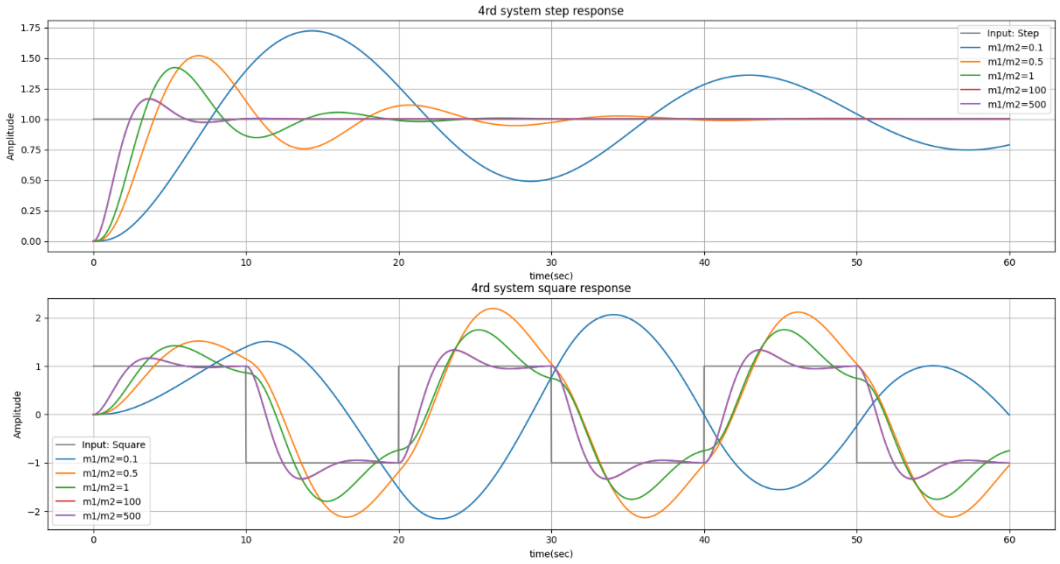




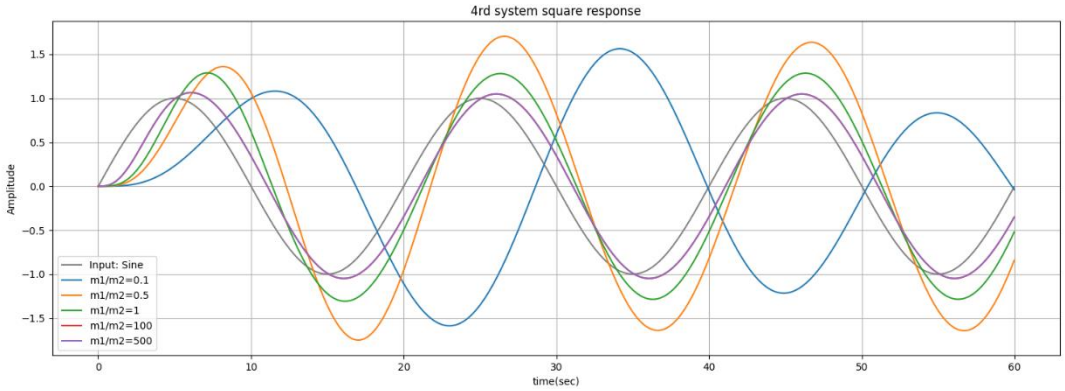


結果討論

我們將上述所有參數變化對系統的影響彙整並繪製於一張圖中，透過將所有變化情形綜合比較的方式，能夠讓我們更清楚地觀察趨勢和比較不同參數對系統響應的影響。



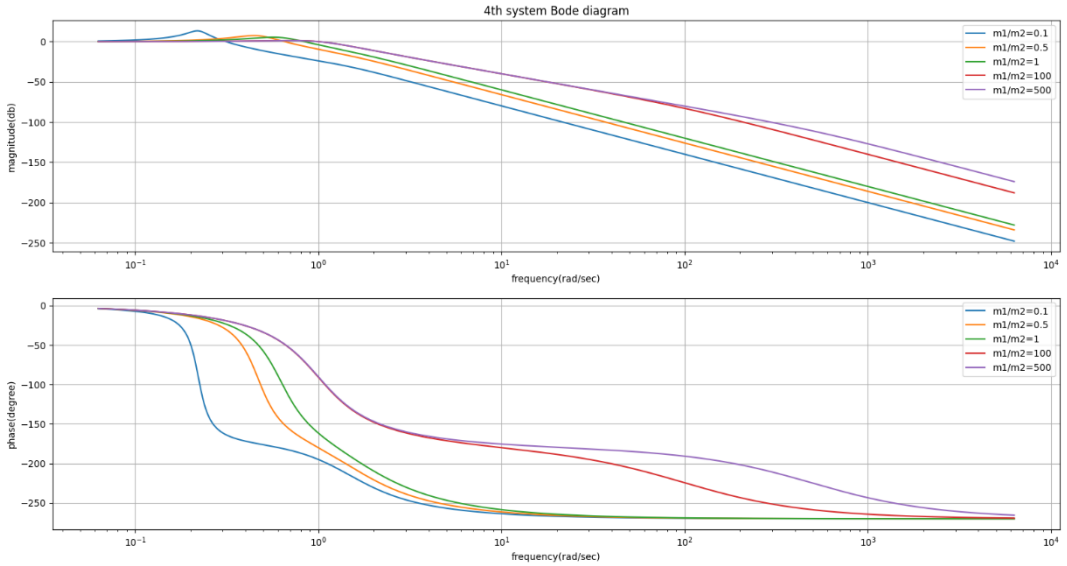
m1/m2 = [0.1, 0.5, 1, 100, 500]



sin 波輸入

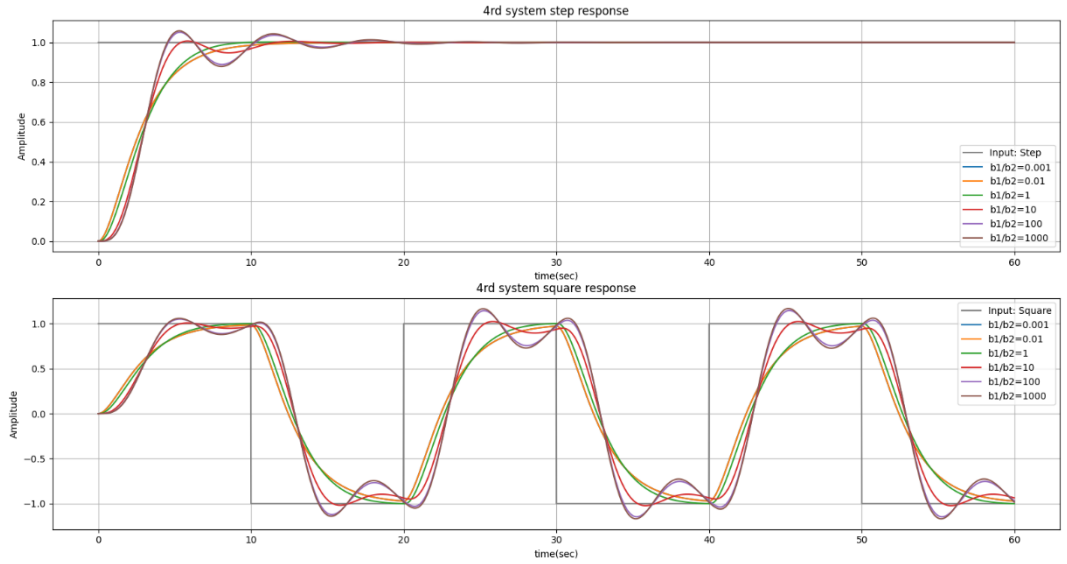
當以 sin 波作為輸入時，系統的響應呈現愈來愈大的震盪，並且沒有明顯的收斂趨勢，顯示系統非常不穩定。特別是當 m1/m2 設為 0.1 時，相位延遲現象最為明顯，而當 m1/m2 設為 0.5 時，系統的響應震盪幅度達到最大。這些結果暗示了質量比值對於系統輸入為 sin

波時的動態行為和穩定性有著重要的影響。

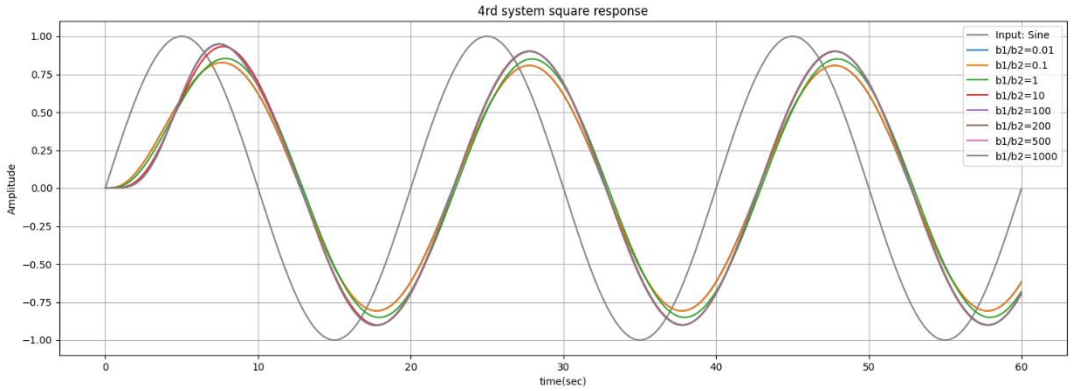


波德圖

由波德圖可以看出圖形響應之增益會先些微增加在減少，而相位也和之前一樣是延遲的，這些和先前的圖所得到的結論相符合。



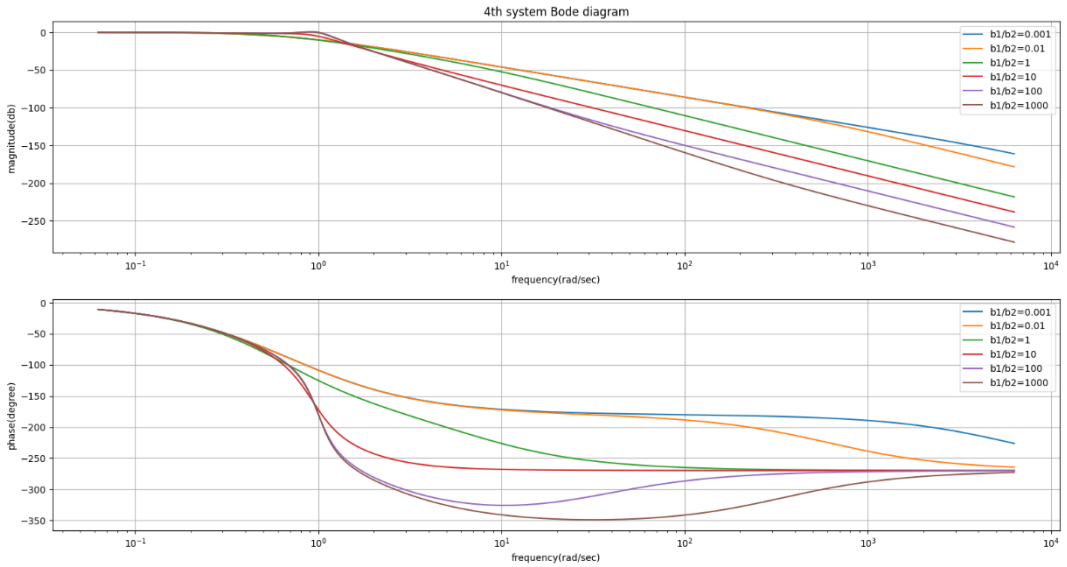
b1/b2 = [0.01, 0.1, 1, 10,100, 1000]



sin 波輸入

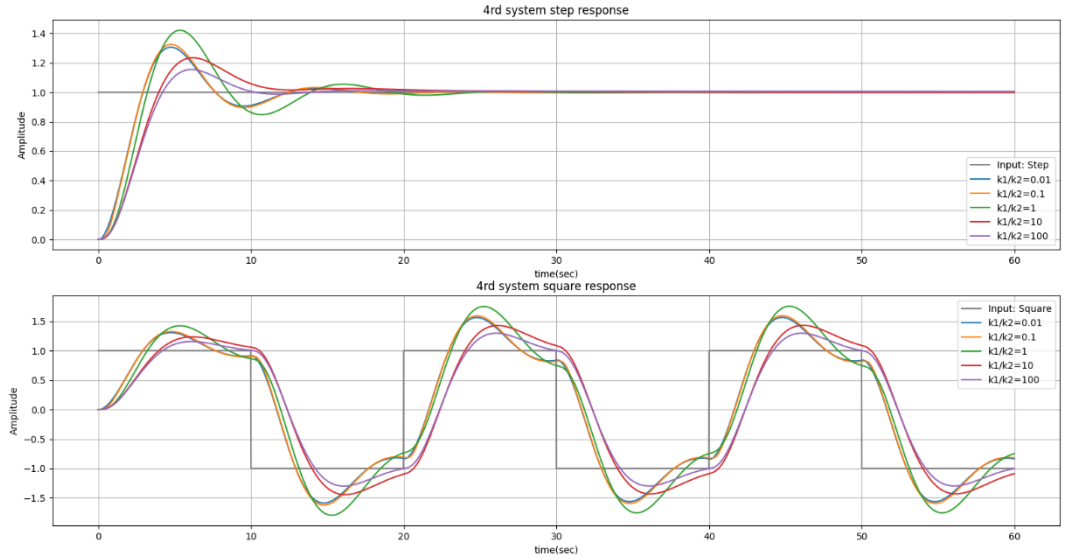
在以 sin 波作為輸入時，系統的響應呈現非常不穩定的情況，並且輸出波形並沒有趨於穩定的趨勢，同時存在相位延遲。然而，不論我們如何設定 b1 和 b2 的值，輸出的波形卻呈現相似的特徵。這暗示著系統的阻尼係數對於響應的穩定性和波形的形狀影響有限，可能存

在其他因素或參數對於系統響應的影響更為顯著，或者系統本身在這個特定的輸入下呈現較複雜的行為。因此，我們需要更深入地分析系統的特性和其他參數的影響，以更好地理解和解釋這種現象。

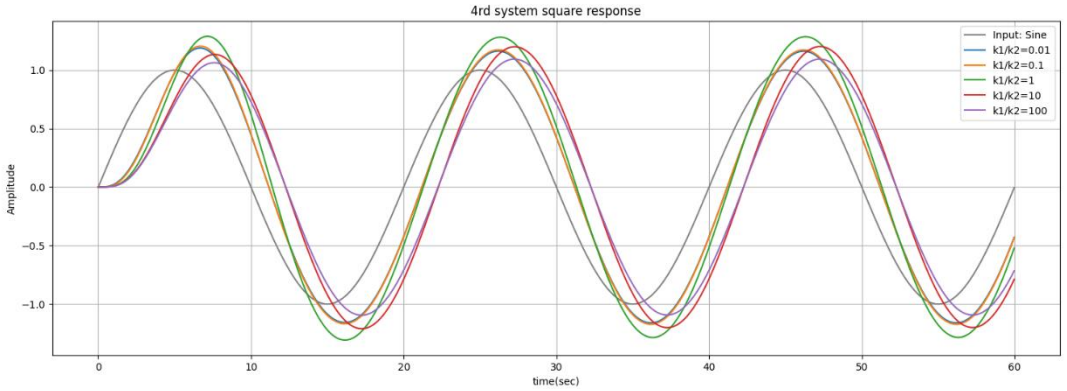


波德圖

從波德圖上可以看出輸出的響應延遲，和先前繪出的圖形結論相符。此外也可以看出系統響應之增益不太明顯。

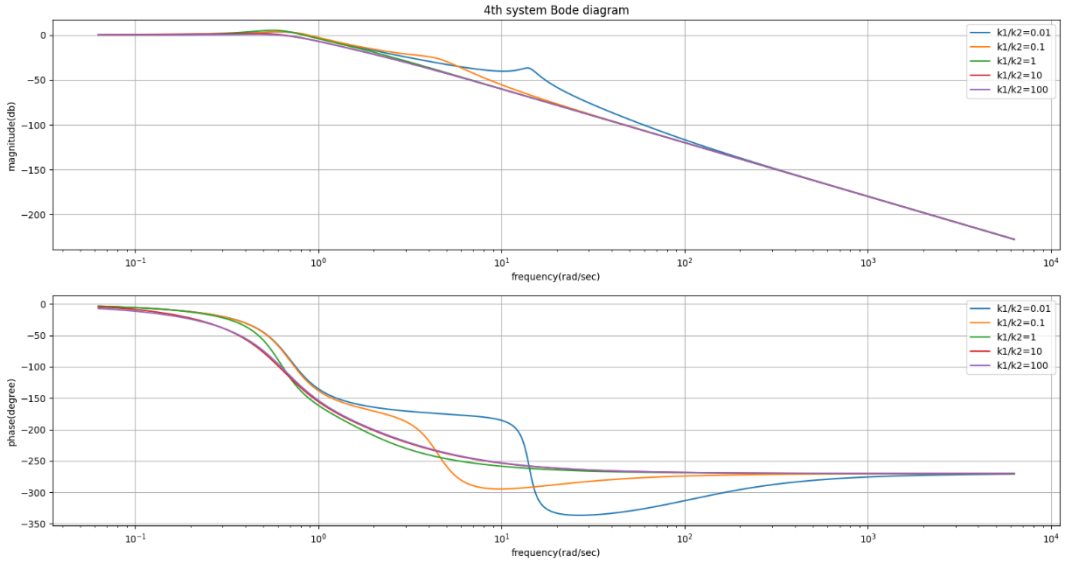


k1/k2 = [0.01, 0.1, 1, 10, 100]



sin 波輸入

由圖中可以觀察出 k1/k2 為 100 時，輸出震盪最小，但是相位的延遲最久，而 k1/k2 為 1 時，響應震盪最大。不管 k1/k2 的值設為多少，輸出結果都和之前一樣很不穩定。



波德圖

透過波德圖可以知道系統響應和之前一樣都會延遲，也就是輸入訊號的變化會在系統響應經過一段時間後才改變，除此之外，也可以觀察出系統響應的增益也不明顯。

結論

系統的質量會影響系統的慣性，如果質量越大，會使系統慣性增大，系統相對會變得比較遲鈍，若要使系統收斂會需要較長的時間。反之，質量越小，系統對外力的響應會變得比較敏感，系統慣性較小。彈簧常數會影響系統的剛性，較大的彈簧常數會使系統剛性增大，對外力造成的變形量會有更好的抵抗能力，響應速度也較快。反之，彈簧常數若越小，系統剛性減弱，響應速度也會變慢。阻尼係數

則是影響了系統的阻尼特性，若阻尼係數越大，系統對於震動的抑制能力更好，響應速度比較慢，但避震效果較好。反之，若阻尼係數越小，響應速度快，但震動也會變得比較大，因為系統消散能量的能力隨著阻尼係數減少而變小了。

總而言之，這三個參數的變化對系統響應有著密切的關聯。質量、彈簧常數和阻尼係數的調整將對系統的特性產生影響，包括穩定性、響應速度和振幅等。因此，通過調整這些參數，我們可以改變系統的特性以滿足不同的應用需求。

分工

一開始我們先一起討論好期末報告的整體架構，並進行分工。主要程式碼由陳薇如負責，而數據分析和報告呈現主要由李芳誼負責。

心得

109303570 李芳誼

這次的報告需要解四階系統，是我們從來沒有遇到過的，和上課內容或是作業相比，變得更難更複雜，需要運用到整學期所學的知識才能完成，因此我們花了非常多的時間在這份報告上，從報告的架構、程式碼的撰寫、資料的分析等等，每個步驟都與上課所學習習相

關，完成這份作業也算是將整學期的知識融換貫通後的成果。

在分析數據時，如果遇到數據出現異常，就會和組員一起討論，並偵錯，有時候是改改參數，有時候更甚至會動到整個程式碼。不管如何，我們都希望這次報告能夠更完整，所以中間所做的修改和討論都很值得，也讓我們獲益良多。

109303567 陳薇如

在製作個個報告的過程中，其實我遇到了許多困難，因此其實花了非常多的時間才完成。一開始，我使用的是和解二階相同的方式，將 y’、y”、y’”、y””依序寫出，並且在 vpython 上運行。但後來在研究頻域圖時，才知道如何使用 transfer function 這個解題方式，然而由於新的方法實在非常方便簡單，我們將原本的需要大約十多秒才能跑出一張圖的 vpython 換成使用 maplotlib，並且把時域的解題方式也改成了使用 transfer function，因此，我們才得以提高了效率，將非常多張的圖片快速產出。我覺得在這次報告中我學習到了在程式撰寫及計算上的整合及應用，受益良多。