Advanced Engineering Mathematics (I)

工程數學 (一)

Homework Assignment 2

Differential Equations and Applications

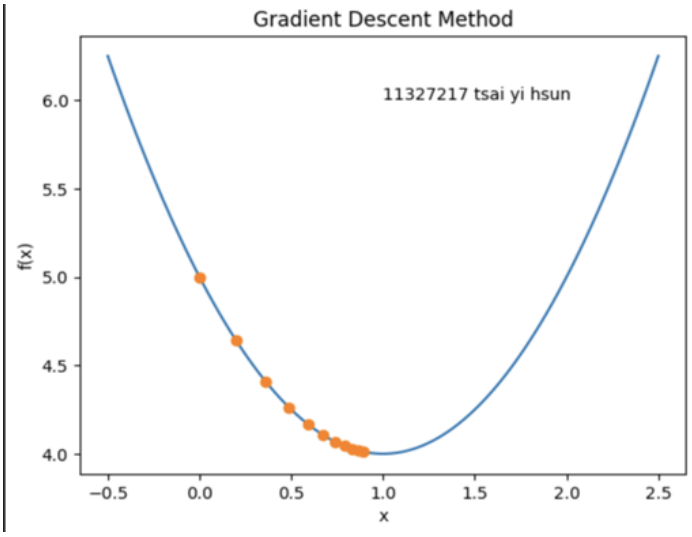
學號： 11327217

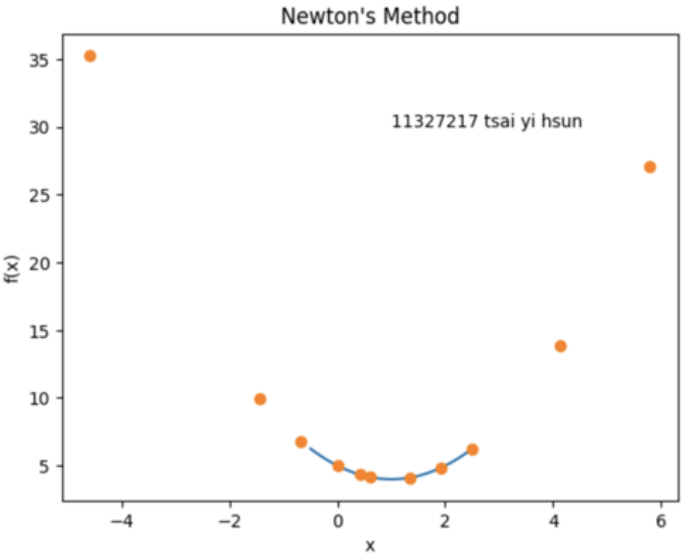
姓名：蔡易勳

指導教授：張元翔

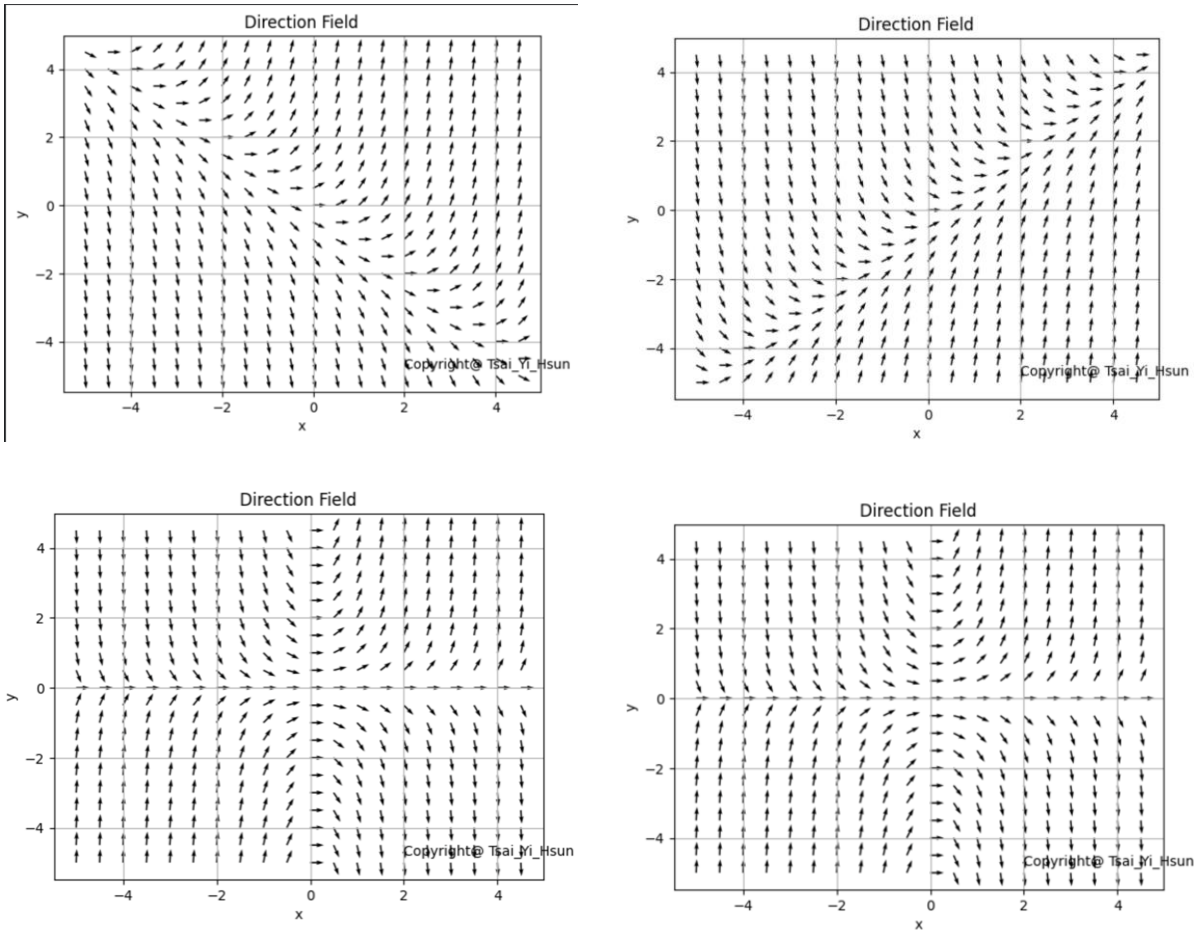
中原大學資訊工程系

【Problem 1】


Gradient Descent Method
11327217 tsai yi hsun
f(x)
x
6.0
5.5
5.0
4.5
4.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Newton's Method
11327217 tsai yi hsun
f(x)
x
35
30
25
20
15
10
5
−4
−2
0
2
4
6

【Problem 2】



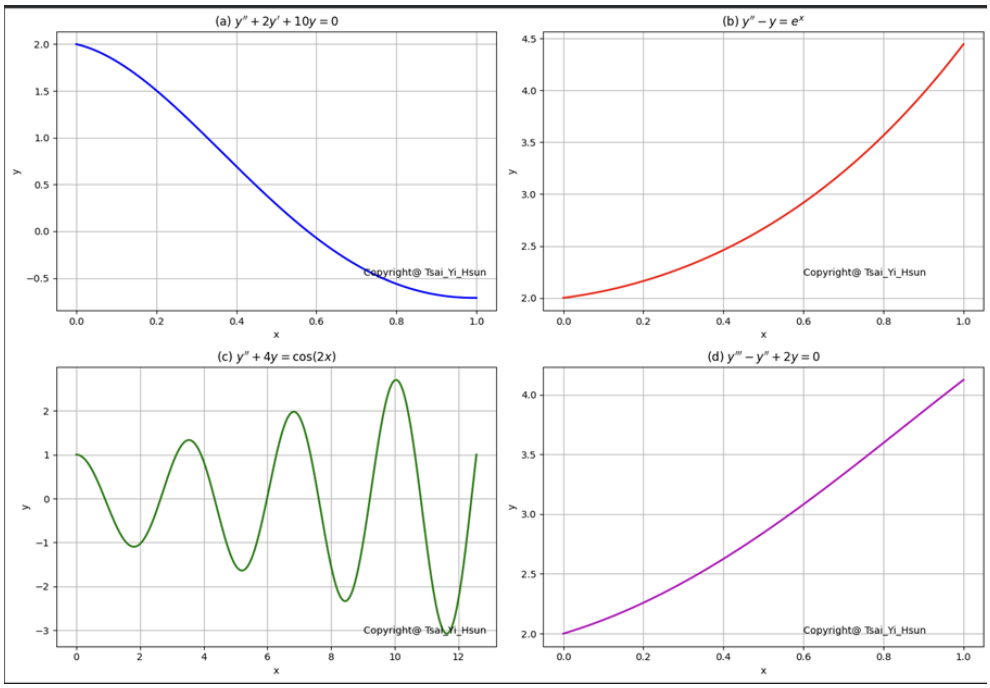
My Findings In Details:

本圖為利用方向場方式所繪製的向量分佈圖，用來觀察平面上各點的變化趨勢。從結果可以看出，整體圖形在水平方向與垂直方向皆呈現規律性的變化，箭頭方向具有重複出現的結構，顯示系統在不同區域中具有相似的行為模式。

在部分區域中，箭頭接近水平排列，代表該區域的變化幅度較小，整體趨勢相對穩定；而在其他區域，箭頭傾斜程度較大，表示變化方向較為明顯，方向場的走向也更加集中。這種分布使整張圖形成類似波動狀的結構。此外，程式中對向量進行正規化處理，使所有箭頭長度一致，只呈現方向資訊，避免因數值大小不同而影響視覺判斷。這樣的做法讓整體趨勢更清楚，也更容易比較不同位置之間的方向差異。

整體而言，透過方向場的視覺化方式，可以直觀地了解平面上各點的方向變化情形，即使不進行進一步的數值計算，也能快速掌握整體行為與分布特性，對理解系統的整體趨勢相當有幫助。

【Problem 3】



(手寫推導)

(a) Aux eq $\Rightarrow m^2 + 2m + 10 = 0$

$\Rightarrow m_{1,2} = \frac{-2 \pm \sqrt{4-40}}{2} = \frac{-2 \pm \sqrt{-36}}{2} = -1 \pm 3i \quad (\alpha = -1, \beta = 3)$

Case III : $y_c = e^{-x}(C_1 \cos 3x + C_2 \sin 3x)$

$y_c(0) = 2 \Rightarrow C_1 = 2$

$y_c'(0) = -1 \Rightarrow C_2 = 1$

$\Rightarrow y = \frac{1}{3} e^{-x} \sin 3x$

(b) 求 y_c : Aux eq $m^2 - 1 = 0 \quad m_{1,2} = 1, -1$

Case I : $y_c = C_1 e^x + C_2 e^{-x}$

求 y_p : $g(x) = e^x$) ∴

設 $y_p = Ae^x$

設 $y_p = Axe^x$ $\quad y_p' = Ae^x + Axe^x$

$y_p'' = Ae^x + Ae^x + Axe^x$

$= 2Ae^x + Axe^x$

代入

$\Rightarrow 2Ae^x + Axe^x - Axe^x = e^x$

$\Rightarrow 2Ae^x = e^x \Rightarrow A = \frac{1}{2}$

$\Rightarrow y_p = \frac{1}{2} xe^x$

General Sol :

$y = C_1 e^x + C_2 e^{-x} + \frac{1}{2} xe^x$

(c) Aux eq : $m^2 + 4 = 0 \Rightarrow m = \pm 2i \quad (\alpha = 0, \beta = 2)$

$y_c = C_1 \cos(2x) + C_2 \sin(2x)$

設 $y_p = Ax \sin(2x)$ $\quad y_p' = A\sin(2x) + 2Ax\cos(2x)$

$y_p' = 4A\cos(2x) - 4Ax\sin(2x)$

$y_p'' + 4y_p = \cos(2x) \Rightarrow A = \frac{1}{4}$

$4A\cos(2x) \quad y_p = \frac{1}{4} x \sin(2x)$

$\Rightarrow y = C_1 \cos(2x) + C_2 \sin(2x) + \frac{1}{4} x \sin(2x)$

$y_c(0) = 1 \Rightarrow C_1 = 1$

$y_c'(0) = 0 \Rightarrow C_2 = 0$

$\Rightarrow y = \cos(2x) + \frac{x}{4} \sin(2x)$

(d) Aux eq $m^3 - m^2 + 2 = 0$

$\Rightarrow (m+1)(m^2 - 2m + 2) = 0$

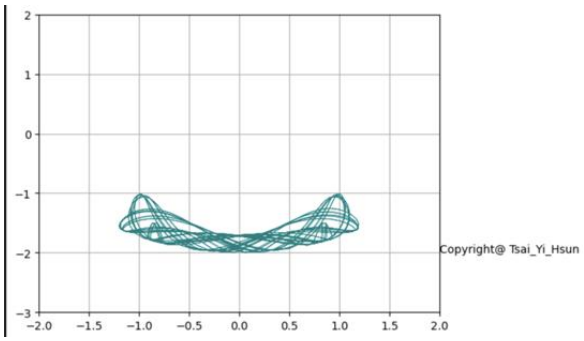
$m_{1,2,3} = -1, 1 \pm i \quad (\alpha = 1, \beta = 1)$

Case I and III

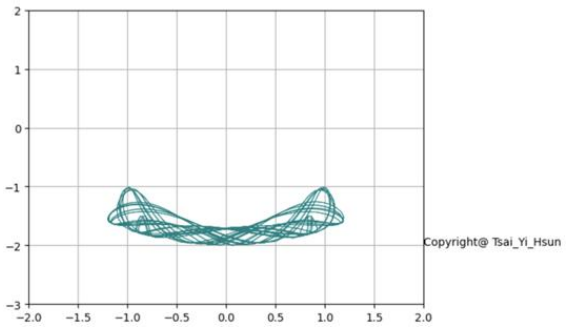
$\Rightarrow y = C_1 e^{-x} + e^x (C_2 \cos x + C_3 \sin x)$

【Problem 4】

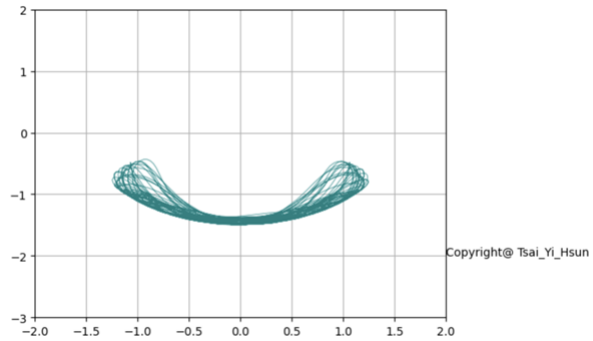
(a)



(b)



(c)



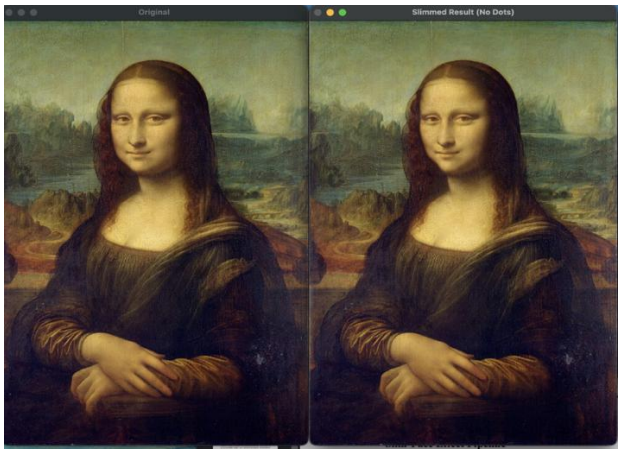
My Findings In Details:

從雙擺的模擬結果可以看出，它的運動比一般單擺複雜很多。一開始看起來還算規律，但隨著時間增加，擺動會逐漸變得混亂，特別是第二個擺，常常出現突然加速或方向改變的情況。

在運動過程中，能量會在兩個擺之間不斷轉移，導致整體運動狀態持續變化，沒有固定的週期。第二個擺球畫出的軌跡也不是重複的圖形，而是形成不規則的路徑，顯示出系統的非週期性。

另外，雙擺對初始條件非常敏感，只要一開始有些微差異，後續的運動結果就會有很大不同。這也說明雙擺是一個典型的混沌系統。整體而言，這次模擬讓我更直觀地理解非線性與混沌運動的特性。

【Problem 5】


Original
Slimmed Result (No Dots)


Original
Slimmed Result (No Dots)


Original
Slimmed Result (No Dots)


Original
Slimmed Result (No Dots)

Discuss My Findings In Details:

本實作利用 dlib 的 68 點人臉特徵點，搭配 OpenCV 的影像變形技術，成功在不破壞五官的情況下，達到臉型變瘦的效果。實驗中只針對下顎與臉頰邊緣進行變形，避免影響嘴巴與鼻子等重要區域，因此整體看起來較自然。

在特徵點選擇上，使用臉頰靠外側的下顎點（第 3–6 與 10–13 點），能有效改變臉部寬度，但不會讓臉部結構失真。位移方向以鼻尖作為參考，讓臉型往中間收縮，比單純水平推擠更自然。

影像變形僅在特定半徑內進行，且位移量會隨距離遞減，使變形邊緣平滑，避免出現明顯的拉扯痕跡。使用 cv2.remap 進行重繪，在畫面品質與效能上都表現良好。

實驗結果顯示，slimming_factor 設在 0.1～0.2 之間時效果最自然，數值過大容易產生不真實的變形。整體而言，此方法不需使用深度學習模型，就能以較低成本實現穩定的瘦臉效果，適合作為影像處理課程的實作範例。

{程式碼}

```
import cv2
import dlib
import numpy as np

# 臉部特徵點模型檔案路徑
# 確保 'shape_predictor_68_face_landmarks.dat' 檔案存在
PREDICTOR_PATH = "shape_predictor_68_face_landmarks.dat"
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor(PREDICTOR_PATH)

def get_landmarks(im):
    # 偵測人臉並回傳 68 個特徵點座標。
    rects = detector(im, 1)
    if len(rects) == 0:
        return None, None

    # 假設只有一張臉，選擇第一張臉
    return np.matrix([[p.x, p.y] for p in predictor(im, rects[0]).parts()]), rects[0]

def apply_slimming(img, landmarks, slimming_factor=0.3):
    # 優化版：僅針對下顎邊緣進行變形，並保護嘴巴區域
    if landmarks is None: return img
```

```
        # 將 landmarks 轉換為整數座標
        pts = np.array(landmarks, dtype=np.int32)

        # 定義下顎線的關鍵點 (左臉 3-6 號, 右臉 10-13 號)
        left_cheek = pts[3:7]
        right_cheek = pts[10:14]

        # 鼻尖 (30 號) 作為推移的目標方向參考點
        nose_tip = pts[30]

        # 建立一個與原圖一樣大的偏移地圖 (Map)
        rows, cols = img.shape[:2]
        map_x = np.tile(np.arange(cols), (rows, 1)).astype(np.float32)
        map_y = np.tile(np.arange(rows), (cols, 1)).T.astype(np.float32)

        # 我們只針對下顎附近的像素進行位移
        # 為了避免嘴巴糊掉，我們設定變形半徑只影響邊緣
        radius = np.linalg.norm(pts[3] - pts[13]) / 6 # 縮小影響範圍

        # 左臉頰
        for i in range(len(left_cheek)):
            pt = left_cheek[i]
            # 計算推移方向：往鼻尖方向推一點
            direction = nose_tip - pt
            dist_to_nose = np.linalg.norm(direction)
            move_vec = (direction / dist_to_nose) * (dist_to_nose * slimming_factor * 0.5)

            # 局部扭曲邏輯
            mask = np.sqrt((map_x - pt[0])**2 + (map_y - pt[1])**2) < radius
            map_x[mask] -= move_vec[0] * (1 - np.sqrt((map_x[mask] - pt[0])**2 + (map_y[mask] -
pt[1])**2) / radius)
            map_y[mask] -= move_vec[1] * (1 - np.sqrt((map_x[mask] - pt[0])**2 + (map_y[mask] -
pt[1])**2) / radius)

        # 右臉頰 (邏輯同上)
        for i in range(len(right_cheek)):
            pt = right_cheek[i]
            direction = nose_tip - pt
```

```
            dist_to_nose = np.linalg.norm(direction)
            move_vec = (direction / dist_to_nose) * (dist_to_nose * slimming_factor * 0.5)

            mask = np.sqrt((map_x - pt[0])**2 + (map_y - pt[1])**2) < radius
            map_x[mask] -= move_vec[0] * (1 - np.sqrt((map_x[mask] - pt[0])**2 + (map_y[mask] - pt[1])**2) / radius)
            map_y[mask] -= move_vec[1] * (1 - np.sqrt((map_x[mask] - pt[0])**2 + (map_y[mask] - pt[1])**2) / radius)

    # 使用 OpenCV 的 remap 進行高品質重繪，這比手動 for 迴圈快且不糊
    output = cv2.remap(img, map_x, map_y, interpolation=cv2.INTER_LINEAR)
    return output

def draw_landmarks(img, landmarks):
    # 在圖片上繪製特徵點。
    if landmarks is None:
        return img

    # 修正處：將 matrix 轉換為 list，這樣 (x, y) 才能正確拆解
    for pt in landmarks.tolist():
        x, y = pt[0], pt[1]
        cv2.circle(img, (int(x), int(y)), 2, (0, 255, 0), -1)
    return img

# --- 主要執行區塊 ---
if __name__ == "__main__":
    IMAGE_PATH = 'input.jpg'
    img = cv2.imread(IMAGE_PATH)

    if img is None:
        print(f"錯誤: 無法讀取圖片 {IMAGE_PATH}")
    else:
        landmarks, face_rect = get_landmarks(img)
        if landmarks is not None:
            print("成功偵測到人臉，正在進行瘦臉處理...")

            # 將 matrix 轉為 numpy array 以供運算
            landmarks_points = np.array(landmarks)
```

```
        # 調整這個參數：0.1 輕微，0.5 明顯，0.8 強力
        SLIMMING_FACTOR = 0.13

        # 直接進行瘦臉，但不呼叫繪製點點的函數
        final_img = apply_slimming(img, landmarks_points, SLIMMING_FACTOR)

        # 顯示結果
        cv2.imshow("Original", img)
        cv2.imshow("Slimmed Result (No Dots)", final_img)

        print("處理完成！按任意鍵關閉視窗。")
        cv2.waitKey(0)
        cv2.destroyAllWindows()

        # 儲存一張沒有點點的乾淨結果圖
        cv2.imwrite("result_clean.jpg", final_img)
else:
        print("圖片中未偵測到人臉。")
```