1. (Gram–Schmidt process) 有一向量空間基底$S_1 = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n\}$, 從建構一組單範正交基底$S_2 = \{\boldsymbol{e}_1, \boldsymbol{e}_2, \dots, \boldsymbol{e}_n\}$, 其中

$$\boldsymbol{e}_i^T \boldsymbol{e}_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

- 選取$\boldsymbol{u}_1 = \boldsymbol{v}_1$, 將其正規化, 得到第一個單位向量$\boldsymbol{e}_1 = \boldsymbol{u}_1/\|\boldsymbol{u}_1\|$
- 選取$\boldsymbol{v}_2$投影至子空間span$\{\boldsymbol{e}_1\}$, 得到殘差向量$\boldsymbol{u}_2$, 再正規化得$\boldsymbol{e}_2$.

$$\boldsymbol{u}_2 = \boldsymbol{v}_2 - (\boldsymbol{v}_2^T\boldsymbol{e}_1)\boldsymbol{e}_1 \xRightarrow[\text{正規化}]{} \boldsymbol{e}_2 = \boldsymbol{u}_2/\|\boldsymbol{u}_2\|$$

- 選取$\boldsymbol{v}_3$投影至子空間span$\{\boldsymbol{e}_1, \boldsymbol{e}_2\}$, 得到殘差向量$\boldsymbol{u}_3$, 再正規化得$\boldsymbol{e}_3$.

$$\boldsymbol{u}_3 = \boldsymbol{v}_3 - (\boldsymbol{v}_3^T\boldsymbol{e}_1)\boldsymbol{e}_1 - (\boldsymbol{v}_3^T\boldsymbol{e}_2)\boldsymbol{e}_2 \xRightarrow[\text{正規化}]{} \boldsymbol{e}_3 = \boldsymbol{u}_3/\|\boldsymbol{u}_3\|$$

- 選取$\boldsymbol{v}_4$投影至子空間span$\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3\}$, 得到殘差向量$\boldsymbol{u}_4$, 再正規化得$\boldsymbol{e}_4$.

$$\boldsymbol{u}_4 = \boldsymbol{v}_4 - (\boldsymbol{v}_4^T\boldsymbol{e}_1)\boldsymbol{e}_1 - (\boldsymbol{v}_4^T\boldsymbol{e}_2)\boldsymbol{e}_2 - (\boldsymbol{v}_4^T\boldsymbol{e}_3)\boldsymbol{e}_3 \xRightarrow[\text{正規化}]{} \boldsymbol{e}_4 = \boldsymbol{u}_4/\|\boldsymbol{u}_4\|$$

- …
- 選取$\boldsymbol{v}_r$投影至子空間span$\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_{r-1}\}$, 得到殘差向量$\boldsymbol{u}_r$, 再正規化得$\boldsymbol{e}_r$.

$$\boldsymbol{u}_r = \boldsymbol{v}_r - \sum_{i=1}^{r-1}(\boldsymbol{v}_r^T\boldsymbol{e}_i)\boldsymbol{e}_i \xRightarrow[\text{正規化}]{} \boldsymbol{e}_r = \boldsymbol{u}_r/\|\boldsymbol{u}_r\|$$

表格 1 Gram–Schmidt 算法

	殘差向量	正交向量
$r = 1$	$\boldsymbol{u}_1 = v_1$	$e_1 = \dfrac{\boldsymbol{u}_1}{\|\boldsymbol{u}_1\|}$
$2 \le r \le n$	$\boldsymbol{u}_r = \boldsymbol{v}_r - \sum_{i=1}^{r-1}(\boldsymbol{v}_r^T\boldsymbol{e}_i)\boldsymbol{e}_i$	$e_r = \dfrac{\boldsymbol{u}_r}{\|\boldsymbol{u}_r\|}$

更多細節可以參考這裡和這裡, 閱讀時要注意, 不同參考資料中, 變數名稱定義會有不同.
撰寫一 python 程式(hw1.py)實作 Gram–Schmidt process, 函數宣告如下:

```
def gram_schmidt(S1: np.ndarray):
    """
    Parameters
    ----------
    S1 : np.ndarray
        A m x n matrix with columns that need to be orthogonalized using Gram-
Schmidt process.
        It is assumed that vectors in S = [v1 v2 ... vn] are linear independent.

    Returns
    -------
    S2 : np.ndarray
        S2 = [e1 e2 ... en] is a mxn orthogonal matrix such that span(S1)=span(S2)

    """
```

```
    S2 = np.zeros (S1.shape)
    # write your code here
    return S2
```

2. 影像矩陣$A_{m\times n}$ 進行奇異值分解(SVD), 特徵值由大到小排列.

$$A_{m\times n} = U_{m\times n}\Sigma_{n\times n}V_{n\times n}^T$$

可以保留前r個成份, 得到近似影像

$$\bar{A}_{m\times n} = U_{m\times r}\Sigma_{r\times r}V_{n\times r}^T$$

兩張圖像的差異(失真)部分視為雜訊

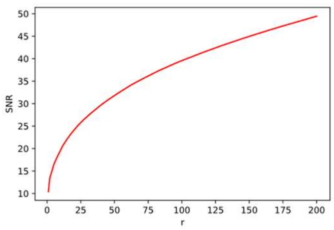
$$N_r = A - \bar{A}$$

原影像A的能量和雜訊的能量比值, 稱為訊號雜訊比(signal-to-noise ratio, SNR), 常以分貝(dB)形式呈現.

$$A_{SNR}[r] = 10 \times \log \frac{\|A\|_F^2}{\|N_r\|_F^2}$$

計算訊號能量請參考 class notebook: 內容庫/補充說明/Energy of a 2D Signal. 參考 hw2.py, 完成下列兩項功能. 當$1 \le r \le 200$,

- ✓ 以$A_{SNR}[r]$ vs. r作圖. 參考下圖結果.

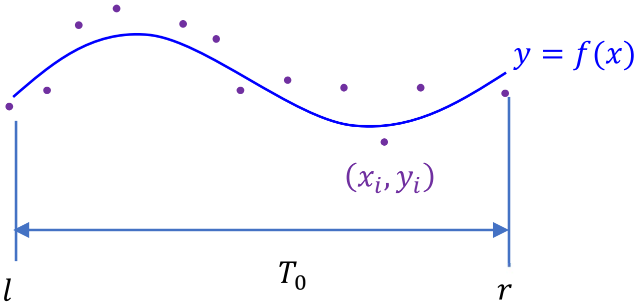


- ✓ 寫程式驗證

$$\|N_r\|_F^2 = \sum_{i=r+1}^{n} \lambda_i \quad \text{or} \quad \|\bar{A}\|_F^2 = \sum_{i=1}^{r} \lambda_i$$

where λ_i is the eigenvalue of $A^T A$.

3. 若我們只對輸入x在區間$[l, r]$取樣, 擬合函數$f(x)$只能預測$l \le x \le r$的輸出, 輸入在區間$[l, r]$外, 無法預測輸出值. 曲線擬合的任務: 當$l \le x \le r$, 找到最適合的曲線.



既然在$x < l$或$x > r$時, $f(x)$是什麼曲線都無所謂, 我們可把$f(x)$當週期函數, 其週期如下

$$T_0 = r - l = \frac{1}{f_0}$$

頻率為f_0的周期函數$f(x)$可以用富式級數表示如下:

$$y = a_0 + a_1 \cos \omega_0 x + b_1 \sin \omega_0 x + a_2 \cos 2\omega_0 x + b_2 \sin 2\omega_0 x + \cdots$$

若只用前n的成分來近似, 則

$$y = a_0 + \sum_{k=1}^{n} a_1 \cos k\omega_0 x + \sum_{k=1}^{n} b_1 \sin k\omega_0 x$$

$$= \begin{bmatrix} 1 & \cos \omega_0 x & \cdots & \cos n\omega_0 x & \sin \omega_0 x & \cdots & \sin n\omega_0 x \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \\ b_1 \\ \vdots \\ b_n \end{bmatrix} = \boldsymbol{\phi}^T(x)\boldsymbol{a}$$

上式, 輸入x經由函數$\phi(x)$轉換, 變成線性形式, 可用 least square 算法解$\boldsymbol{a}$.
若收集到m個樣本$D = \{(x_i, y_i)\}_{i=1}^{m}$, 把每個樣本代入上式, 寫成如下矩陣形式

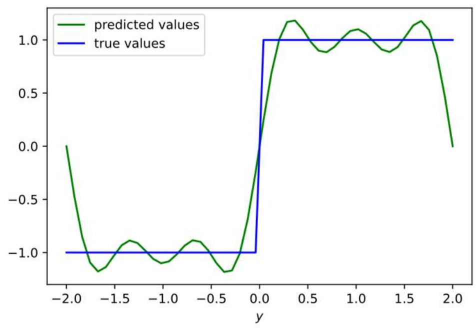
$$\begin{bmatrix} \boldsymbol{\phi}^T(x_1) \\ \boldsymbol{\phi}^T(x_2) \\ \vdots \\ \boldsymbol{\phi}^T(x_m) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \\ b_1 \\ \vdots \\ b_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \Rightarrow X\boldsymbol{a} = \boldsymbol{y}$$

則$\boldsymbol{a} = V\Sigma U^T \boldsymbol{y}$. 此處, $X = U\Sigma V^T$ (short SVD).
撰寫 python 程式(hw3.py), 取$n = 5$, 以富式級數回歸擬合方波.

```
pts = 50
x = np.linspace(-2, 2, pts)
y = np.zeros(x.shape)

# square wave
pts2 = pts // 2
y[0:pts2] = -1
```

```
y[pts2:] = 1

T0 = np.max(x) - np.min(x)
f0 = 1.0 / T0
omega0 = 2.0 * np.pi * f0
n = 5

# step1: generate X=[1 cos(omega0 x) cos(omega0 2x) ... cos(omega0 nx) sin(omega0 x)
sin(omega0 2x) ... sin(omega0 nx)]
# step2: SVD of X => X=(U)(S)(V^T)
# step3: a = U @ S^-1 @ V^T @ y
# write your code here

y_bar = X @ a
plt.plot(x, y_bar, 'g-')
plt.plot(x, y, 'b-')
plt.xlabel('x')
plt.xlabel('y')
plt.show()
```

預期結果



4. 資料前處理是機器學習的重要步驟, 目標是改善資料的品質, 提高系統準確率. 主要內容是對資料進行清理(cleaning), 轉換(transforming)和整合, 以便在訓練之前好準備. 其中有一種方式是將數值集合$X = \{x_1, x_2, \dots, x_n\}$, 重新映射到範圍$[a, b]$, 轉換方式如下:

$$x_i \leftarrow a + \frac{x_i - \min(X)}{\max(X) - \min(X)} \times (b - a)$$

其中, $\max(X)$和$\min(X)$分別是X的最大值和最小值. 請用 python 副程式實現上述功能, 處理 1D 或 2D 陣列的數值變換. 用以下例子做說明.

✓ 輸入是 1D 向量.

$[3, 5, 1, 4]$映射到區間$[5, 7]$ $\Longrightarrow$ $[6, 7, 5, 6.5]$

✓ 輸入是 2D 向量, 又分為兩種情況, 以 row 或 column 向量為變換的對象.

row-wise:

$$\begin{bmatrix} 1 & 4 & 3 & 6 \\ 6 & 7 & 9 & 5 \\ 11 & 14 & 13 & 16 \end{bmatrix} \text{映射到區間 } [0,1] \Longrightarrow \begin{bmatrix} 0 & 0.6 & 0.4 & 1 \\ 0.25 & 0.5 & 1 & 0 \\ 0 & 0.6 & 0.6 & 1 \end{bmatrix}$$

column-wise:

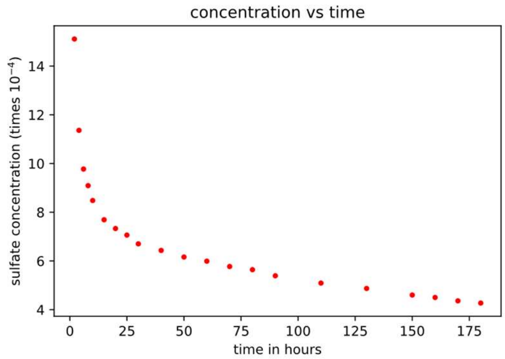
$$\begin{bmatrix} 1 & 4 & 3 & 6 \\ 6 & 7 & 9 & 5 \\ 11 & 14 & 13 & 16 \end{bmatrix} \text{映射到區間 } [0,1] \Longrightarrow \begin{bmatrix} 0 & 0 & 0 & 0.1 \\ 0.5 & 0.3 & 0.6 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

函數要求如下:

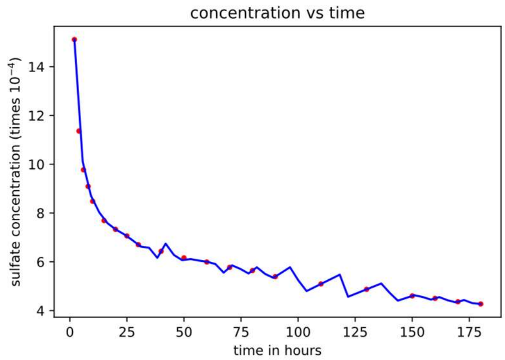
```
def scale_to_range(X: np.ndarray, to_range=(0,1), byrow = False):
    """
    Parameters
    ----------
    X:
        1D or 2D array

    to_range: default to (0,1).
        Desired range of transformed data.

    byrow: default to False
        When working with a 2D array, true to perform row mapping;
        otherwise, column mapping. Ignore if X is 1D.

    """
    a, b = to_range
    Y = np.zeros(X.shape)
    # write your code here
    return Y
```

5. 資料集 hw5.csv 是測量 Brunhilda 狒狒血液中硫酸鹽濃度與時間的關係. 請先點開連結, 閱讀對資料資料集的說明

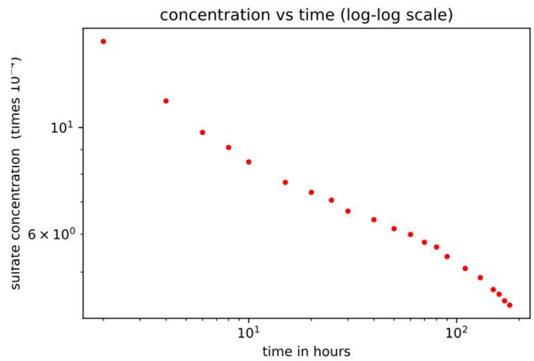
- ✓ 請以[硫酸鹽濃度 vs 時間]作圖. 預期結果如下



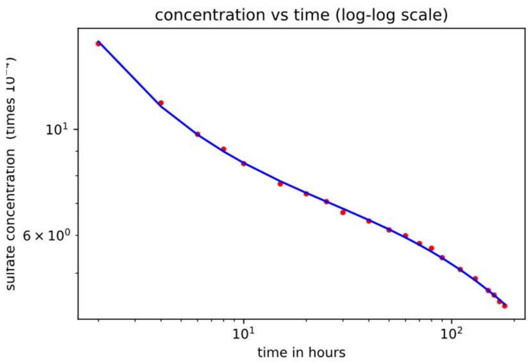
- ✓ 請找一個回歸方法, 並以上圖為基礎, 畫出預測曲線. 解釋你找到的迴歸是好是壞, 以及原因. 下圖是老師找到的預測曲線, 你畫出藍色曲線會和我的不同.



- ✓ 請以[硫酸鹽濃度對數 vs 時間對數]作圖. 預期結果如下



- ✓ 建立濃度對數與時間對數的迴歸曲線, 並以上圖為基礎加上此曲線.下圖是老師找到的預測曲線, 你找出的藍色預測曲線可能和我的不同.

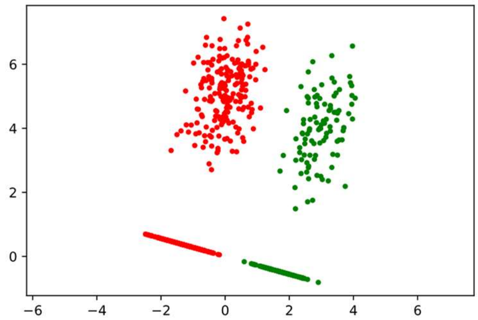


concentration vs time (log-log scale)
sulfate concentration (times 10^{-})
10^1
6×10^0
10^1
10^2
time in hours

6. 參考講義線性判別分析(Linear discriminant analysis, LDA)推導. 利用 2 維常態分布產生兩個類別資料

```
# class 1
mean1 = np.array([0, 5])
sigma1 = np.array([[0.3, 0.2],[0.2, 1]])
N1 = 200
X1 = np.random.multivariate_normal(mean1, sigma1, N1)

# class 2
mean2 = np.array([3, 4])
sigma2 = np.array([[0.3, 0.2],[0.2, 1]])
N2 = 100
X2 = np.random.multivariate_normal(mean2, sigma2, N2)
```

找出向量$\boldsymbol{w}$, 使得資料點在$\boldsymbol{w}$上的投影, 不同類別間距盡量分散, 同類別緊密. 並畫出資料點在$\boldsymbol{w}$上的投影. 參考下圖.



7. 計算成本函數$J(\boldsymbol{w})$的極值, 我們常用梯度下降法

$$\boldsymbol{w}_{i+1} \leftarrow \boldsymbol{w}_i - \alpha \nabla J(\boldsymbol{w})|_{\boldsymbol{w}=\boldsymbol{w}_i}$$

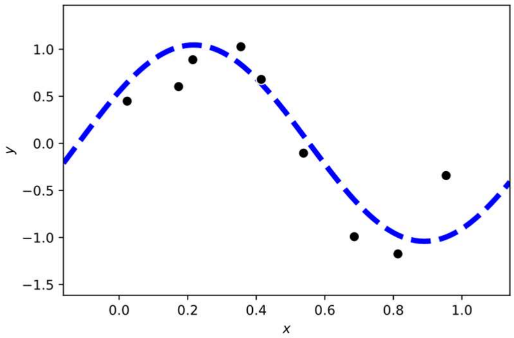
計算$J(\boldsymbol{w})$極值時的權重向量$\boldsymbol{w}$. 上述公式必須算出梯度$\nabla J(\boldsymbol{w})$, 一般會用下列兩種方法:

- 解析法: 以偏導數技巧算出$\nabla J(\boldsymbol{w})$閉式解
- 數值法: $\nabla J(\boldsymbol{w}) = \left[\frac{\partial J(\boldsymbol{w})}{\partial w_1} \quad \frac{\partial J(\boldsymbol{w})}{\partial w_2} \quad \cdots \quad \frac{\partial J(\boldsymbol{w})}{\partial w_n}\right]^T$

  可用極小的數值ϵ, 計算$\nabla J(\boldsymbol{w})$的第k個分量

$$\frac{\partial J(\boldsymbol{w})}{\partial w_k} \cong \frac{J(w_1, \dots, w_k + \epsilon, \dots, w_n) - J(w_1, \dots, w_k, \dots, w_n)}{\epsilon}$$

參考下圖



黑色小圓圈為樣本點, 經過評估後, 嘗試以弦波(sine wave)做預測曲線. 弦波通式如下

$$y = w_1 + w_2 \sin(w_3 x + w_4)$$

以殘差

$$e_i = y_i - w_1 - w_2 \sin(w_3 x_i + w_4)$$

平方和為成本函數

$$J(w_1, w_2, w_3, w_4) = \sum_{i=1}^{m} e_i^2 = \sum_{i=1}^{m} (y_i - w_1 - w_2 \sin(w_3 x_i + w_4))^2$$

撰寫 python 程式(hw7.py), 分別以❶解析法和❷數值法進行曲線擬合.

提示:

- 解析法計算梯度

$$\frac{\partial J(\boldsymbol{w})}{\partial w_1} = -\sum_{i=1}^{m} 2e_i$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_2} = -\sum_{i=1}^{m} [2e_i \sin(w_3 x_i + w_4)]$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_3} = -\sum_{i=1}^{m} [2e_i x_i \cos(w_3 x_i + w_4)]$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_4} = \cdots \text{do it by yourself}$$

- 數值法計算梯度

$$\frac{\partial J(\boldsymbol{w})}{\partial w_1} = \frac{J(w_1 + 10^{-8}, w_2, w_3, w_4) - J(\boldsymbol{w})}{10^{-8}}$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_2} = \frac{J(w_1, w_2 + 10^{-8}, w_3, w_4) - J(\boldsymbol{w})}{10^{-8}}$$

$\cdots$ and so on

```
dataset = pd.read_csv('data/hw7.csv').to_numpy(dtype = np.float64)
x = dataset[:, 0]
y = dataset[:, 1]

# parameters for our two runs of gradient descent
w = np.array([-0.1607108,  2.0808538,  0.3277537, -1.5511576])

alpha = 0.05
max_iters = 500
# cost function
#     J(w0, w1, w2, w3) = sum(y[i] - w0 - w1 * sin(w2 * x[i] + w3))^2
for _ in range(1, max_iters):
    pass
    # remove the above pass and write your code here
    # calculate gradient of cost function by using partial derivative(使用偏導數計算
梯度)
    # update rule: w =  w - alpha * gradient_of_cost


xmin,xmax,ymin,ymax = scatter_pts_2d(x, y)
xt = np.linspace(xmin, xmax, 100)
yt1 = w[0] + w[1] * np.sin(w[2] * xt + w[3])

w = np.array([-0.1607108,  2.0808538,  0.3277537, -1.5511576])
for _ in range(1, max_iters):
    pass
    # remove the above pass and write your code here
    # calculate gradient of cost function by using numeric method(使用數值法計算梯度)
    # update rule: w =  w - alpha * gradient_of_cost

xt = np.linspace(xmin, xmax, 100)
yt2 = w[0] + w[1] * np.sin(w[2] * xt + w[3])
# plot x vs y; xt vs yt1; xt vs yt2
```

提示:

- 解析法計算梯度

$$\frac{\partial J(\boldsymbol{w})}{\partial w_1} = -\sum_{i=1}^{m} 2e_i$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_2} = -\sum_{i=1}^{m} [2e_i \sin(w_3 x_i + w_4)]$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_3} = -\sum_{i=1}^{m} [2e_i x_i \cos(w_3 x_i + w_4)]$$

$$\frac{\partial J(\boldsymbol{w})}{\partial w_4} = \cdots \text{do it by yourself}$$

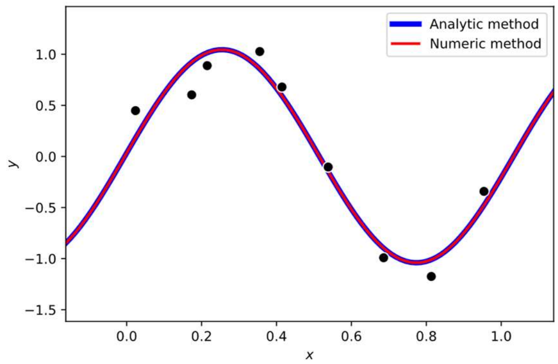
- 數值法計算梯度

$$\frac{\partial J(\boldsymbol{w})}{\partial w_1} = \frac{J(w_1 + 10^{-8}, w_2, w_3, w_4) - J(\boldsymbol{w})}{10^{-8}}$$

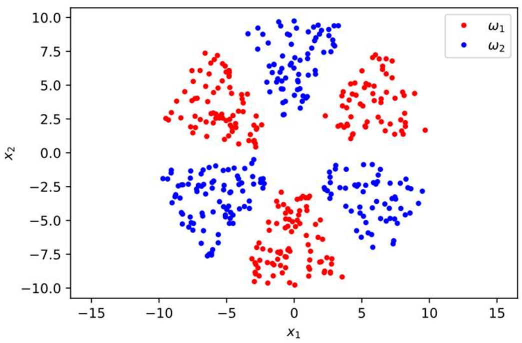
$$\frac{\partial J(\boldsymbol{w})}{\partial w_2} = \frac{J(w_1, w_2 + 10^{-8}, w_3, w_4) - J(\boldsymbol{w})}{10^{-8}}$$

$\cdots$ and so on

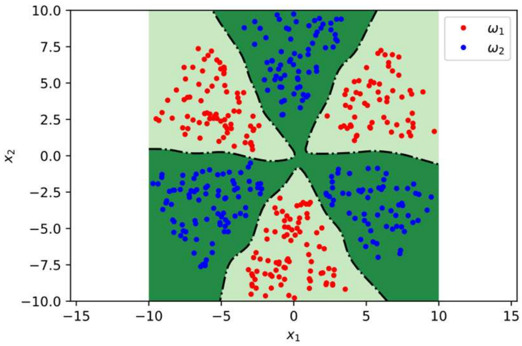
預期結果



8. 二元分類資料集 hw8.csv, 其類別為ω_1和ω_2, 並作圖如下



請找一方法, 例如: SVM, 神經網路, adaboost, 邏輯回歸, ..., 對上述資料分類, 並將結果作圖.
預期結果參考下圖:



說明: 分類方法和參數不同, 分類邊界會和上圖不一樣. 選用色彩可自行定義.