ASP Assignment

Simulation of Method of Steepest Descent

姓名：潘俊宏 學號：N96111037

目錄：

1. 參數表

a_1, a_2	$v_1(n), v_2(n)$	$\mathbf{Q}$	$\mathbf{v}(n)$
誤差分量係數	誤差分量	$\mathbf{R}$特徵向量矩陣	權重$\mathbf{w}(n)$在特徵向量中的投影分量

$\mathbf{w}(n)$	$\mathbf{R} = E[\mathbf{x}\mathbf{x}^T]$	$J(\mathbf{w}(n))$	λ_1, λ_2	$\chi(\mathbf{R}) = \dfrac{\lambda_1}{\lambda_2}$	$\mathbf{J}_{min}$
權重向量	輸入訊號的自相關矩陣	成本函數	$\mathbf{R}$特徵值	eigenvalue spread	最小成本函數

2.實驗規格與實際模擬

for a constant $J(n)$, the value of $v_1(n)$ and $v_2(n)$ form an elipse $\lambda_1 v_1^2 + \lambda_2 v_2^2 = J(n) - J_{min}$

2–1 Varying eigenvalue spread $\chi(\mathbf{R})$, fixed step–size $\mu = 0.3$, keep $\sigma_u^2 = 1$

(一) case 1

$a_1 = -0.1950,\ a_2 = 0.95$	$\lambda_1 = 1.1,\ \lambda_2 = 0.9$	$\chi(\mathbf{R}) = 1.22$	$\mathbf{J}_{min} = 0.0965$

(二) case 2

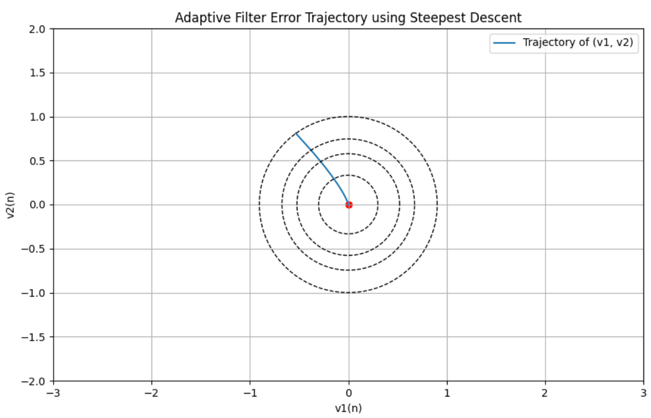
$a_1 = -0.9750,\ a_2 = 0.95$	$\lambda_1 = 1.5,\ \lambda_2 = 0.5$	$\chi(\mathbf{R}) = 3$	$\mathbf{J}_{min} = 0.0731$

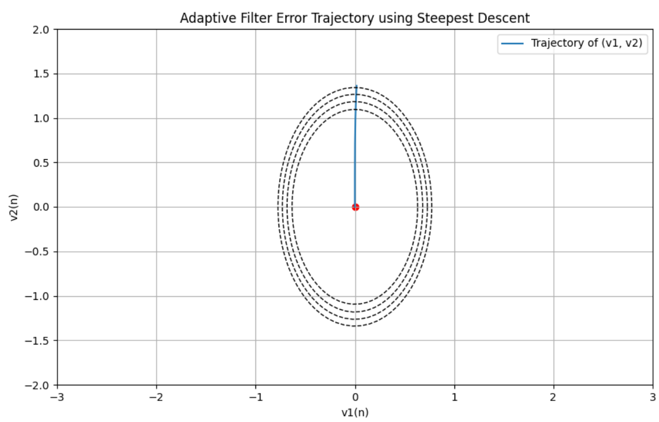
(三) case 3

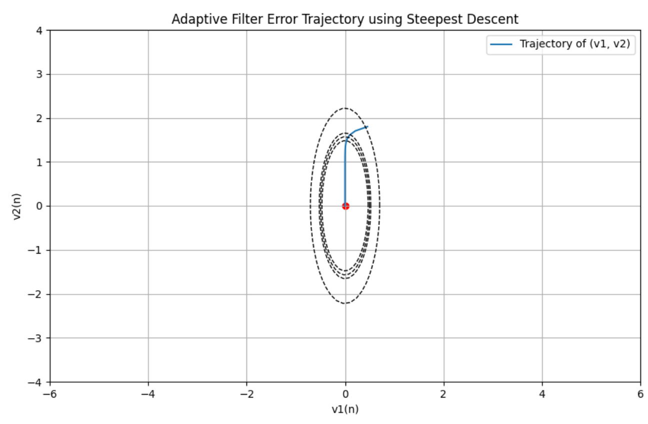
$a_1 = -1.5955,\ a_2 = 0.95$	$\lambda_1 = 1.818,\ \lambda_2 = 0.182$	$\chi(\mathbf{R}) = 10$	$\mathbf{J}_{min} = 0.0322$

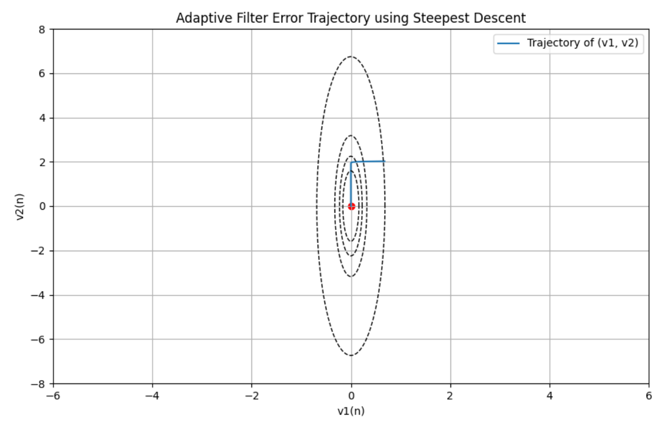
(四) case 4

$a_1 = -1.9114,\ a_2 = 0.95$	$\lambda_1 = 1.908,\ \lambda_2 = 0.0198$	$\chi(\mathbf{R}) = 100$	$\mathbf{J}_{min} = 0.0038$









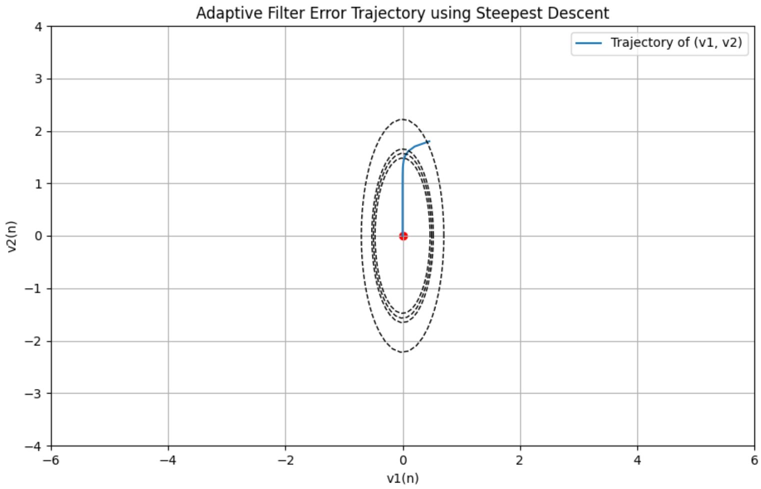
2–2 Varying step–size μ. fixed $\chi(\mathbf{R})$ = 10

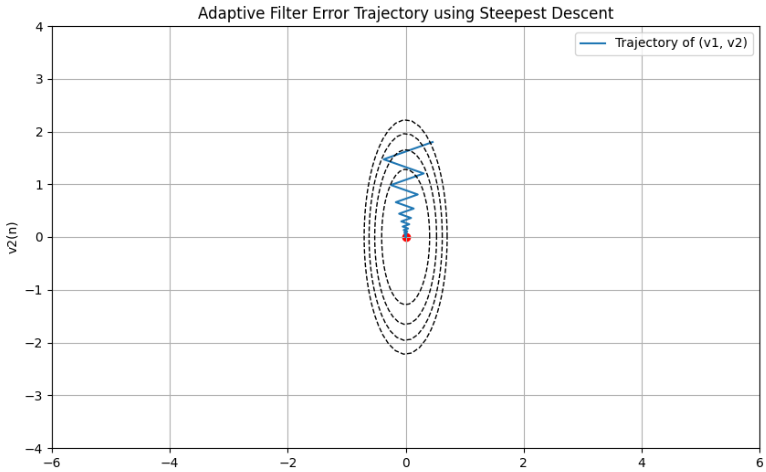
（一） case 1

$a_1 = -0.1950,\ a_2 = 0.95$	$\mu = 0.3$	$\lambda_1 = 1.818,\ \lambda_2 = 0.182$	$a_1 = -1.5955,\ a_2 = 0.95$

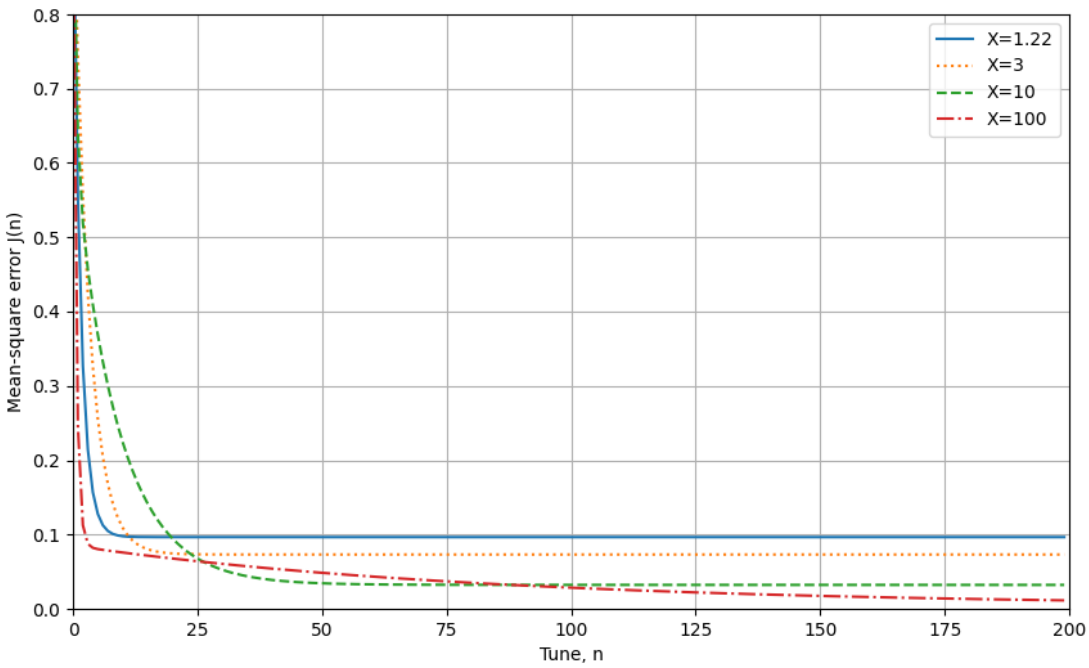
（二） case 2

$a_1 = -0.1950,\ a_2 = 0.95$	$\mu = 1$	$\lambda_1 = 1.818,\ \lambda_2 = 0.182$	$a_1 = -1.5955,\ a_2 = 0.95$





2–3 The learning curve when $\chi(\mathbf{R})$ increase and step–size is fixed



3.附錄

3-1基於2-1case延伸的代碼

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

# AR model parameters
a1, a2 = -0.1950, 0.95

# Simulation settings
N = 1000 # Number of samples
mu = 0.3  # Learning rate
sigma_u = 1  # Noise standard deviation
J_min = 0.0965
lambda1 = 1.1
lambda2 = 0.9
# Initialize the AR process
v1 = np.zeros(N)
v2 = np.zeros(N)
J  = np.zeros(N)
for n in range(0, N):
    # Use only past values of u to predict the current u[n]
    v1[n] = -1/np.sqrt(2)*((1-mu*lambda1)**n)*(a1+a2)
    v2[n] = -1/np.sqrt(2)*((1-mu*lambda2)**n)*(a1-a2)
    J[n] = J_min+lambda1*v1[n]**2+lambda2*v2[n]**2
# Calculate error vectors for plotting
def plot_ellipse(lambda1, lambda2, delta_J, color='black'):
    theta = np.linspace(0, 2 * np.pi, 100)
    r = np.sqrt(delta_J / (lambda1 * np.cos(theta)**2 + lambda2 * np.sin(theta)**2))
    v1 = r * np.cos(theta)
    v2 = r * np.sin(theta)
    plt.plot(v1, v2, color=color, linestyle='dashed', linewidth=1)
increments = [0.1,0.3, 0.5, 0.9]  # 不同的能量水平增量

# Plotting the trajectory of the error vectors
plt.figure(figsize=(10, 6))
plt.plot(v1, v2, label='Trajectory of (v1, v2)')
plt.scatter(v1[-1], v2[-1], color='red')  # End point
for inc in increments:
    plot_ellipse(lambda1, lambda2, inc)
plt.title('Adaptive Filter Error Trajectory using Steepest Descent')
plt.xlabel('v1(n)')
plt.ylabel('v2(n)')
plt.grid(True)
plt.xlim(-3, 3)
plt.ylim(-2, 2)
plt.legend()
plt.show()
```