

Assignment -3

Advanced DSP (EC6601)

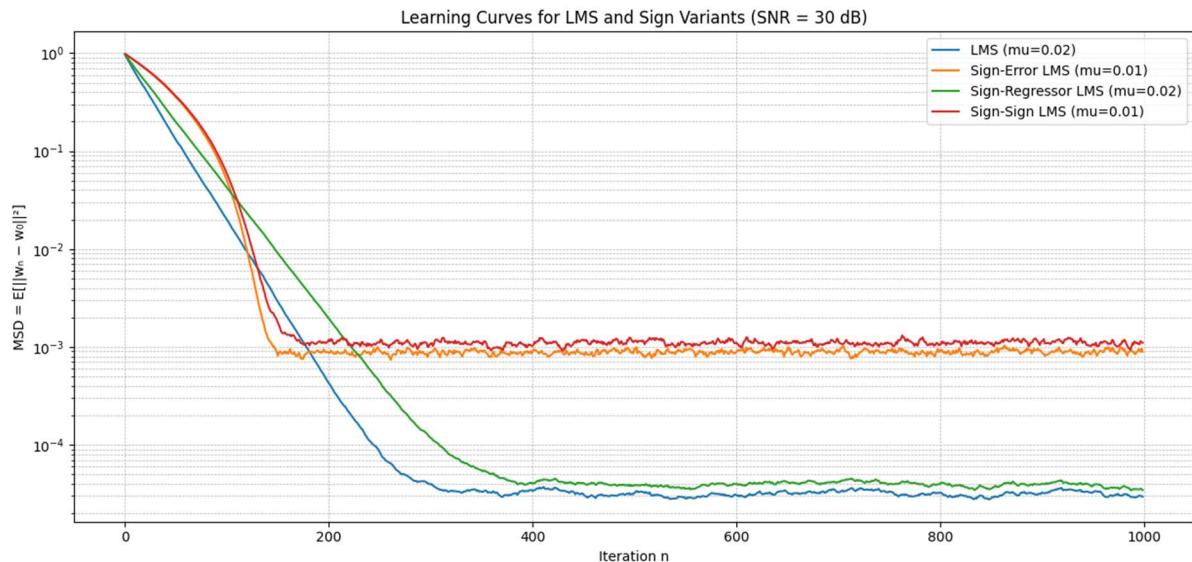
Manda Sreevalli

925EC5006

Executive PhD

1) Develop a Python or Matlab code for the simulation of a system identification problem. The parameter of the unknown system is $\bar{w}_o = [0.26, 0.93, 0.26]^T$. The input \bar{x}_n and output y_n relationship is linear and the noise in the output is independent and identically distributed with Gaussian distribution $\mathcal{N}(0, \sigma^2)$.

- a) Estimate the system parameter using the LMS algorithm and plot the mean square curve (MSE) w.r.t. the iteration $n=1, 2, 3, \dots$ for SNR=30dB.
b) Perform the step (a) for sign error-LMS, sign-regressor LMS and sign-sign LMS.



Script:

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

# =====
```

```

# SYSTEM PARAMETERS
# =====

w_true = np.array([0.26, 0.93, 0.26]) # true parameter
L = len(w_true)

SNR_dB = 30
N = 1000      # iterations
trials = 200   # ensemble runs

# Step-sizes (tunable)
mu_lms = 0.02
mu_se = 0.01  # sign-error
mu_sr = 0.02  # sign-regressor
mu_ss = 0.01  # sign-sign

# =====
# COMPUTE NOISE VARIANCE FROM SNR
# =====

sigma_x2 = 1.0 # each input sample variance
signal_power = np.dot(w_true, w_true) * sigma_x2
sigma2 = signal_power / (10 ** (SNR_dB / 10))

# =====
# STORAGE FOR MSD RESULTS
# =====

msd_lms = np.zeros(N)
msd_se = np.zeros(N)
msd_sr = np.zeros(N)
msd_ss = np.zeros(N)

# =====
# MAIN MONTE CARLO SIMULATION
# =====

for t in range(trials):

    # initialize weights
    w_lms = np.zeros(L)
    w_se = np.zeros(L)
    w_sr = np.zeros(L)
    w_ss = np.zeros(L)

```

```

for n in range(N):

    # Generate input vector and desired output
    x_n = np.random.normal(0, np.sqrt(sigma_x2), size=L)
    noise = np.random.normal(0, np.sqrt(sigma2))
    y_n = np.dot(w_true, x_n) + noise

    # -----
    # LMS UPDATE
    # -----
    e = y_n - np.dot(w_lms, x_n)
    w_lms += mu_lms * e * x_n

    # -----
    # SIGN-ERROR LMS
    # -----
    e = y_n - np.dot(w_se, x_n)
    w_se += mu_se * np.sign(e) * x_n

    # -----
    # SIGN-REGRESSOR LMS
    # -----
    e = y_n - np.dot(w_sr, x_n)
    w_sr += mu_sr * e * np.sign(x_n)

    # -----
    # SIGN-SIGN LMS
    # -----
    e = y_n - np.dot(w_ss, x_n)
    w_ss += mu_ss * np.sign(e) * np.sign(x_n)

    # Accumulate MSD
    msd_lms[n] += np.sum((w_lms - w_true)**2)
    msd_se[n]  += np.sum((w_se  - w_true)**2)
    msd_sr[n]  += np.sum((w_sr  - w_true)**2)
    msd_ss[n]  += np.sum((w_ss  - w_true)**2)

# Average over trials
msd_lms /= trials
msd_se  /= trials
msd_sr  /= trials
msd_ss  /= trials

# =====
# PLOTTING

```

```

# =====

plt.figure(figsize=(9, 5))
plt.semilogy(msd_lms, label=f"LMS ( $\mu$ ={mu_lms})")
plt.semilogy(msd_se, label=f"Sign-Error LMS ( $\mu$ ={mu_se})")
plt.semilogy(msd_sr, label=f"Sign-Regressor LMS ( $\mu$ ={mu_sr})")
plt.semilogy(msd_ss, label=f"Sign-Sign LMS ( $\mu$ ={mu_ss})")

plt.xlabel("Iteration n")
plt.ylabel("MSD =  $E[||w_n - w_0||^2]$ ")
plt.title("Learning Curves for LMS and Sign Variants (SNR = 30 dB)")
plt.grid(True, which="both", linestyle="--", linewidth=0.5)
plt.legend()
plt.tight_layout()
plt.show()

# Print final steady-state MSD
print("\nFinal MSD values after {} iterations:".format(N))
print("LMS:           {:.4e}".format(msd_lms[-1]))
print("Sign-Error LMS: {:.4e}".format(msd_se[-1]))
print("Sign-Regressor: {:.4e}".format(msd_sr[-1]))
print("Sign-Sign LMS:  {:.4e}".format(msd_ss[-1]))

```

LMS: 3.206709227010673e-05
 Sign-Error LMS: 8.16115546662197e-04
 Sign-Regressor: 4.324394501941463e-05
 Sign-Sign LMS: 1.24199999999998e-03