Laboratory Manual: 11. Signal Modeling – AR and MA Models

Digital Signal Processing Laboratory

March 28, 2025

1 Signal Modeling: Autoregressive and Moving Average Models

Theoretical Background

In signal processing, stochastic models such as autoregressive (AR) and moving average (MA) models are used to represent random signals in a compact mathematical form. These models are especially useful for spectral analysis, prediction, compression, and simulation.

Autoregressive (AR) Model

An AR(p) model expresses a signal as a linear combination of its past p samples and a white noise term:

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + w[n], \tag{1}$$

where $w[n]$ is white Gaussian noise with zero mean and variance σ_w^2, and a_k are AR coefficients.

Moving Average (MA) Model

An MA(q) model represents a signal as a linear combination of current and past q noise terms:

$$x[n] = \sum_{k=0}^{q} b_k w[n-k], \tag{2}$$

where b_k are the MA coefficients, and $w[n]$ is white noise.

ARMA Model

A general ARMA(p, q) model combines both AR and MA components:

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + \sum_{k=0}^{q} b_k w[n-k] \tag{3}$$

ARIMA Model

An ARIMA(p, d, q) model is an ARMA model applied to the d-th order differenced signal:

$$\nabla^d x[n] = -\sum_{k=1}^{p} a_k \nabla^d x[n-k] + \sum_{k=0}^{q} b_k w[n-k], \tag{4}$$

where $\nabla^d x[n]$ denotes differencing:

$$\nabla x[n] = x[n] - x[n-1], \quad \nabla^2 x[n] = \nabla(\nabla x[n]) = x[n] - 2x[n-1] + x[n-2], \text{ etc.} \tag{5}$$

Use Cases

- Modeling signals with inherent structure or memory.
- Forecasting in time series (e.g., financial or biomedical signals).
- Spectral estimation and system identification.

Python Examples

1. Simulating an AR(2) Process

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfilter

N = 1000
w = np.random.normal(0, 1, N)
a = [1, -0.75, 0.5]  # AR coefficients
ar_signal = lfilter([1], a, w)

plt.plot(ar_signal)
plt.title("AR(2) Signal")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()
```

Listing 1: Generate and plot AR(2) signal

2. Simulating an MA(2) Process

```python
b = [1.0, 0.5, 0.25]  # MA coefficients
ma_signal = lfilter(b, [1], w)

plt.plot(ma_signal)
plt.title("MA(2) Signal")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()
```

Listing 2: Generate and plot MA(2) signal

3. Simulating an ARMA(2,2) Process

```
a = [1, -0.75, 0.5]
b = [1.0, 0.5, 0.25]
arma_signal = lfilter(b, a, w)

plt.plot(arma_signal)
plt.title("ARMA(2,2) Signal")
plt.xlabel("Sample")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()
```

Listing 3: Generate and plot ARMA(2

4. Estimating AR Model Parameters (Yule-Walker)

```
from statsmodels.regression.linear_model import yule_walker

rho, sigma = yule_walker(ar_signal, order=2)
print("Estimated AR Coefficients:", -rho)
print("Estimated Noise Variance:", sigma)
```

Listing 4: Estimate AR coefficients using Yule-Walker

5. ARIMA Modeling

```
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# Generate non-stationary signal: a trend + noise
N = 1000
np.random.seed(0)
trend = np.cumsum(np.random.normal(0, 1, N))

# Fit ARIMA(p=2, d=1, q=2)
model = ARIMA(trend, order=(2, 1, 2))
model_fit = model.fit()

# Forecast future values
forecast = model_fit.predict(start=0, end=N-1)

plt.plot(trend, label="Original")
plt.plot(forecast, label="ARIMA(2,1,2) Fit")
plt.legend()
plt.title("ARIMA Model Fitting")
plt.xlabel("Sample")
plt.ylabel("Value")
plt.grid(True)
plt.show()
```

Listing 5: Fit and simulate ARIMA(2

Comparison Example: ARMA vs ARIMA

To demonstrate the difference between ARMA and ARIMA modeling approaches, consider a signal with a clear trend component. An ARMA model assumes stationarity and may not perform well without preprocessing (e.g., detrending), while an ARIMA model can inherently handle such trends through differencing.

ARMA(2,2) Model Expression

$$x[n] = -a_1 x[n-1] - a_2 x[n-2] + b_0 w[n] + b_1 w[n-1] + b_2 w[n-2], \tag{6}$$

where a_1, a_2 are AR coefficients, b_0, b_1, b_2 are MA coefficients, and $w[n]$ is white noise.

ARIMA(2,1,2) Model Expression

$$\nabla x[n] = -a_1 \nabla x[n-1] - a_2 \nabla x[n-2] + b_0 w[n] + b_1 w[n-1] + b_2 w[n-2], \tag{7}$$

which is equivalent to applying the ARMA(2,2) model to the first-differenced series $\nabla x[n] = x[n] - x[n-1]$.

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.arima_process import ArmaProcess

# Create a non-stationary signal with a linear trend
N = 500
np.random.seed(42)
trend = np.cumsum(np.random.normal(0, 1, N))

# Fit ARMA(2,2) model to non-stationary signal
arma_model = ARIMA(trend, order=(2, 0, 2))
arma_result = arma_model.fit()
arma_pred = arma_result.predict(start=0, end=N-1)

# Fit ARIMA(2,1,2) model
arima_model = ARIMA(trend, order=(2, 1, 2))
arima_result = arima_model.fit()
arima_pred = arima_result.predict(start=0, end=N-1)

# Plotting
plt.plot(trend, label="Original Signal")
plt.plot(arma_pred, label="ARMA(2,2)", linestyle='--')
plt.plot(arima_pred, label="ARIMA(2,1,2)", linestyle=':')
plt.title("Comparison: ARMA vs ARIMA")
plt.xlabel("Sample")
plt.ylabel("Value")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 6: Compare ARMA and ARIMA models

This example illustrates that ARIMA models better capture trends by incorporating differencing, while ARMA models may yield biased or poor fits unless the signal is preprocessed to be stationary.

Parameter Estimation: Yule-Walker and Burg Methods

Yule-Walker Method

The Yule-Walker method estimates AR model parameters by solving the Yule-Walker equations derived from the autocorrelation function. For an AR(p) process:

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + w[n], \tag{8}$$

we compute the biased autocorrelation estimate $r(k)$ and solve:

$$\mathbf{Ra} = \mathbf{r}, \tag{9}$$

where $\mathbf{R}$ is the autocorrelation matrix and $\mathbf{a}$ is the vector of AR coefficients.

Python Implementation:

```
from statsmodels.regression.linear_model import yule_walker

# Example signal: ar_signal
rho, sigma = yule_walker(ar_signal, order=2)
print("Estimated AR coefficients:", -rho)
print("Estimated noise variance:", sigma)
```

Listing 7: AR coefficient estimation using Yule-Walker

Burg Method

The Burg method is an alternative technique for AR estimation that minimizes forward and backward prediction errors while enforcing the Levinson-Durbin recursion. It typically provides better spectral resolution for short data records.

Key Properties:

- Based on minimizing prediction errors.
- Recursive and stable.
- Often used in high-resolution spectral estimation.

Python Implementation:

```
from spectrum import pburg

# Estimate AR model using Burg method
ar_model = pburg(ar_signal, order=2, NFFT=512)
ar_model.plot()
print("AR Coefficients (Burg):", ar_model.ar)
```

Listing 8: AR coefficient estimation using Burg method

Student Task Variants

Each task consists of generating a synthetic signal, constructing at least two different models, estimating their parameters, and comparing model quality.

1. Generate a white Gaussian noise signal. Fit AR(1), AR(2), and MA(1) models. Compare AIC and residual variance.

2. Simulate an AR(2) process. Fit AR(2) using Yule-Walker and Burg methods. Compare parameter estimates and residuals.

3. Generate a stationary signal from an MA(2) process. Fit MA(2) and AR(2) models. Discuss model adequacy.

4. Simulate a signal using ARMA(1,1). Fit AR(2), MA(2), and ARMA(1,1) models. Compare their fit visually and statistically.

5. Create a random walk signal (non-stationary). Fit AR(1), ARIMA(1,1,0), and ARIMA(2,1,2). Analyze forecast accuracy.

6. Simulate an AR(3) signal. Fit and compare AR(3) using Yule-Walker and Burg. Compare frequency-domain PSDs.

7. Generate a trend + white noise signal. Fit ARMA and ARIMA models. Compare AIC, BIC, and forecast results.

8. Generate MA(3) data. Fit MA(2), MA(3), and ARMA(1,1) models. Analyze residual autocorrelations.

9. Simulate ARMA(2,1) signal. Fit AR(3), MA(3), ARMA(2,1) models. Compare residuals and AIC.

10. Generate a sinusoidal signal with added noise. Fit AR and ARMA models. Examine how model order affects fitting.

11. Simulate a cumulative sum of white noise. Fit ARIMA(1,1,0) and ARIMA(1,1,1). Compare forecasts.

12. Generate heteroscedastic signal. Fit AR(1), ARIMA(1,0,1) and discuss why variance modeling may be needed.

13. Simulate AR(2) signal. Compare AR(2) estimated using Burg and Yule-Walker on short and long signals.

14. Generate signal composed of two segments (stationary and non-stationary). Fit AR, ARMA, and ARIMA. Analyze segmentation effects.

15. Generate synthetic stock-like signal. Fit ARIMA(1,1,1), ARIMA(2,1,2), compare forecasts and confidence intervals.