KEMENTERIAN PENDIDIKAN, KEBUDAYAAN,
RISET, DAN TEKNOLOGI
UNIVERSITAS BENGKULU – FAKULTAS TEKNIK
JURUSAN TEKNIK ELEKTRO

BUKU PETUNJUK PRAKTIKUM

ANALISIS DERET WAKTU DAN PERAMALAN

Time Series Analysis and Forecasting in Electrical Engineering

Dosen Pengampu:
Ir. Novalio Daratha, S.T., M.Sc., Ph.D.
NIP. 197811102003121002

Laboratorium Sistem Tenaga Listrik & Komputasi Cerdas

BENGKULU
2026

Daftar Isi

Petunjuk Umum & Lingkungan Komputasi

Tujuan Praktikum

Praktikum Komputasi Analisis Deret Waktu bertujuan untuk membekali mahasiswa S1 Teknik Elektro dengan keterampilan teknis praktis dalam:

1. Mengolah dan memanipulasi dataset temporal skala besar sistem tenaga listrik, sensor IoT, serta pembangkit EBT.
2. Membangun *pipeline* pemodelan deret waktu dari metode klasik hingga *Deep Learning*.
3. Menginterpretasikan performa algoritma secara kuantitatif sesuai kaidah saintifik rekayasa elektro.

Kebutuhan Perangkat Lunak

Setiap mahasiswa diharapkan menggunakan lingkungan Python 3.10+ (atau GNU Octave / MATLAB) dengan pustaka standar berikut:

```
# Instalasi pustaka utama via terminal/command prompt
pip install numpy pandas scipy matplotlib statsmodels scikit-learn xgboost torch
    filterpy pywavelets
```

Bab 1

Dekomposisi Sinyal Deret Waktu Kelistrikan

Informasi Modul 1

Sub-CPMK 1: Mampu mengidentifikasi komponen tren, musiman, siklis, dan residu pada data kelistrikan serta memvisualisasikannya secara komputasional.
Alokasi Waktu: 170 Menit (Praktikum Mandiri & Asistensi Lab)
Dataset: Data beban listrik historis per-jam (`load_grid_hourly.csv`)

1.1 Dasar Teori

Deret waktu keteknikan y_t dapat didekomposisi menjadi tiga komponen deterministik dan satu komponen stokastik:

$$\text{Model Aditif: } y_t = T_t + S_t + R_t \qquad \text{Model Multiplikatif: } y_t = T_t \times S_t \times R_t \tag{1.1}$$

di mana T_t adalah tren, S_t adalah musiman dengan periode m (misal $m = 24$ untuk data jam), dan R_t adalah residual (*irregular/noise*).

1.2 Langkah Praktikum & Kode Implementasi

```python
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# 1. Membaca dataset beban gardu induk
df = pd.read_csv('load_grid_hourly.csv', parse_dates=['timestamp'], index_col='timestamp')

# 2. Dekomposisi Klasik (Periode harian: 24 jam)
decomp = seasonal_decompose(df['load_mw'], model='additive', period=24)

# 3. Plot 4-Panel (Observed, Trend, Seasonal, Residual)
fig, axes = plt.subplots(4, 1, figsize=(12, 8), sharex=True)
decomp.observed.plot(ax=axes[0], color='navy', title='Observed Load (MW)')
decomp.trend.plot(ax=axes[1], color='darkred', title='Trend Component')
decomp.seasonal.plot(ax=axes[2], color='darkgreen', title='Daily Seasonality')
decomp.resid.plot(ax=axes[3], color='black', title='Residual / Noise')
plt.tight_layout()
plt.show()
```

1.3 Tugas & Laporan Praktikum 1

1. Hitunglah nilai variansi residual $\text{Var}(R_t)$ dan periksa apakah residual memiliki nilai rata-rata mendekati nol!
2. Lakukan dekomposisi multiplikatif pada data yang sama. Bandingkan nilai residual antara kedua model dan tentukan model mana yang paling cocok!

Bab 2

Uji Stasioneritas, Diferensiasi, dan Correlogram

Informasi Modul 2

Sub-CPMK 2: Mampu menghitung ACF/PACF, menguji stasioneritas (ADF, KPSS), dan melakukan transformasi diferensiasi data.
Dataset: Profil daya aktif transmisi (`active_power.csv`)

2.1 Dasar Teori

Sebagian besar model statistik linier mengasumsikan sinyal berada dalam keadaan stasioner kovarians:

$$\mathbb{E}[y_t] = \mu, \quad \text{Var}(y_t) = \sigma^2, \quad \text{Cov}(y_t, y_{t-k}) = \gamma_k \tag{2.1}$$

Uji *Augmented Dickey-Fuller* (ADF) menguji hipotesis nol (H_0) bahwa data memiliki *unit root* (non-stasioner), sedangkan uji *KPSS* menguji hipotesis nol (H_0) bahwa data stasioner di sekitar tren. Kombinasi keduanya memastikan apakah sinyal bersifat *Difference-Stationary* (DSP) atau *Trend-Stationary* (TSP).

2.2 Kode Implementasi Python

```python
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import matplotlib.pyplot as plt

def check_stationarity_full(series, label="Sinyal"):
    s = series.dropna()
    adf_res = adfuller(s, autolag='AIC')
    kpss_res = kpss(s, regression='c', nlags='auto')

    print(f"=== Uji Stasioneritas: {label} ===")
    print(f"ADF Stat: {adf_res[0]:.4f}, p-value: {adf_res[1]:.4e}")
    print(f"KPSS Stat: {kpss_res[0]:.4f}, p-value: {kpss_res[1]:.4f}")

    if adf_res[1] < 0.05 and kpss_res[1] > 0.05:
        print("Status: STASIONER MURNI (Kuadran I)")
    elif adf_res[1] > 0.05 and kpss_res[1] < 0.05:
        print("Status: NON-STASIONER (DSP) -> Wajib Diferensiasi")
    elif adf_res[1] > 0.05 and kpss_res[1] > 0.05:
        print("Status: TREND-STATIONARY (TSP) -> Butuh Detrending")
    else:
```

```
        print("Status: PERINGATAN (Structural Break Terdeteksi)")

# Uji data asli vs diferensiasi ordo-1
check_stationarity_full(df['power'], "Data Mentah")

df['diff_1'] = df['power'].diff()
check_stationarity_full(df['diff_1'], "Setelah Diferensiasi (d=1)")

# Plot ACF dan PACF
fig, axes = plt.subplots(1, 2, figsize=(12, 4))
plot_acf(df['diff_1'].dropna(), lags=40, ax=axes[0], title='Autocorrelation (ACF)'
    )
plot_pacf(df['diff_1'].dropna(), lags=40, ax=axes[1], title='Partial ACF (PACF)')
plt.tight_layout()
plt.show()
```

Bab 3

Metode Pemulusan Data (Holt-Winters)

Informasi Modul 3

Sub-CPMK 3: Mampu menerapkan metode pemulusan (*Moving Average*, Holt-Winters aditif/multiplikatif) untuk peramalan beban jangka pendek.
Dataset: Profil konsumsi listrik mingguan (`weekly_load.csv`)

3.1 Dasar Teori

Metode Holt-Winters mengakomodasi level (ℓ_t), tren (b_t), dan musiman (s_t):

$$\ell_t = \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \tag{3.1}$$

$$b_t = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1} \tag{3.2}$$

$$s_t = \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m} \tag{3.3}$$

$$\hat{y}_{t+h|t} = \ell_t + hb_t + s_{t+h-m} \tag{3.4}$$

3.2 Kode Implementasi Python

```python
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
import numpy as np

# Split Train dan Test (24 jam terakhir sebagai test set)
train, test = df['load'][:-24], df['load'][-24:]

# Fitting Model Holt-Winters Aditif
model = ExponentialSmoothing(train, seasonal_periods=24, trend='add', seasonal='
    add').fit()
forecast = model.forecast(24)

# Evaluasi Performa
rmse = np.sqrt(mean_squared_error(test, forecast))
mape = mean_absolute_percentage_error(test, forecast) * 100
print(f'RMSE: {rmse:.2f} MW | MAPE: {mape:.2f}%')
```

Bab 4

Estimasi Parameter ARMA dan Analisis Kestabilan Bidang-Z

Informasi Modul 4

Sub-CPMK 4: Mampu menyelesaikan matriks Yule-Walker, memplot akar karakteristik pada lingkaran satuan (Bidang Z), dan melakukan grid-search AIC.
Dataset: Sinyal getaran generator (`vibration_signal.csv`)

4.1 Dasar Teori

Estimasi parameter model AR(p) secara analitik menggunakan persamaan Yule-Walker: $\mathbf{R}\boldsymbol{\phi} = \mathbf{r}$. Kestabilan model ARMA dinilai dari lokasi akar polinomial karakteristik terhadap lingkaran satuan.

4.2 Kode Implementasi Python

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

# 1. Yule-Walker untuk AR(2)
lag_acf = acf(train_signal, nlags=2)
r1, r2 = lag_acf[1], lag_acf[2]
R = np.array([[1, r1], [r1, 1]])
r = np.array([r1, r2])
phi = np.linalg.inv(R).dot(r)
print(f"Estimasi phi_1: {phi[0]:.4f}, phi_2: {phi[1]:.4f}")

# 2. Plot Akar Kestabilan Polinomial AR
model = ARIMA(train_signal, order=(2,0,0)).fit()
ar_roots = model.arroots
plt.scatter(ar_roots.real, ar_roots.imag)
circle = plt.Circle((0,0), 1, color='red', fill=False)
plt.gca().add_patch(circle)
plt.title("Inverse AR Roots (Z-plane)")
plt.show()
```

Bab 5

Pemodelan SARIMA Metodologi Box-Jenkins

Informasi Modul 5

Sub-CPMK 5: Mampu menyusun ordo p, d, q, P, D, Q dan mengestimasi model SARIMA untuk data berkala musiman serta memvalidasi residual.
Dataset: Data time series gardu distribusi (`substation_load.csv`)

5.1 Dasar Teori

Model SARIMA disimbolkan dengan $(p, d, q) \times (P, D, Q)_s$:

$$\Phi_P(B^s)\phi_p(B)(1-B)^d(1-B^s)^D y_t = \Theta_Q(B^s)\theta_q(B)\epsilon_t \tag{5.1}$$

5.2 Kode Implementasi Python

```python
import pmdarima as pm
import pandas as pd
import matplotlib.pyplot as plt

# 1. Pencarian Ordo Otomatis (Auto-ARIMA)
print("Memulai Auto-ARIMA Grid Search...")
auto_model = pm.auto_arima(train_load, seasonal=True, m=24,
                           start_p=1, max_p=3, start_q=1, max_q=3,
                           d=1, D=1, stepwise=True, trace=True)

# Menampilkan hasil seleksi (termasuk Uji Ljung-Box secara otomatis)
print(auto_model.summary())

# 2. Prediksi 168 jam (1 Minggu)
n_periods = 168
fc, conf_int = auto_model.predict(n_periods=n_periods, return_conf_int=True, alpha
    =0.05)

# 3. Visualisasi
plt.figure(figsize=(10,5))
plt.plot(test_load.index[:n_periods], test_load[:n_periods], label='Aktual')
plt.plot(test_load.index[:n_periods], fc, color='red', label='Forecast SARIMA')
plt.fill_between(test_load.index[:n_periods], conf_int[:, 0], conf_int[:, 1],
                 color='red', alpha=0.2, label='95% CI')
plt.legend()
plt.title("Peramalan Beban 168 Jam dengan Auto-ARIMA")
```

```
26 plt.show()
```

Bab 6

Model Eksogen Cuaca: SARIMAX dan CCF

Informasi Modul 6

Sub-CPMK 6: Mampu membangun model SARIMAX dengan mengintegrasikan variabel eksogen cuaca dan mengimplementasikan sistem multivariat VAR.
Dataset: Beban gedung & data stasiun cuaca (`hvac_weather.csv`), serta Sinyal kualitas daya 3-kanal (`power_quality.csv`)

6.1 Dasar Teori

Variabel eksogen X_t (seperti temperatur) mempengaruhi konsumsi daya AC secara langsung:

$$y_t = \beta X_t + \frac{\Theta(B)}{\Phi(B)}\epsilon_t \tag{6.1}$$

6.2 Kode Implementasi Python

```python
# Integrasi variabel temperatur (X_train & X_test)
sarimax_model = SARIMAX(train['load'], exog=train[['temperature', 'humidity']],
                        order=(1, 1, 1), seasonal_order=(1, 1, 0, 24))
res_exog = sarimax_model.fit(disp=False)

# Prediksi menggunakan fitur eksogen masa depan
forecast_exog = res_exog.forecast(steps=24, exog=test[['temperature', 'humidity'
    ]])
```

6.3 Implementasi Vector AutoRegression (VAR)

```python
from statsmodels.tsa.api import VAR

# Memodelkan Tegangan, Arus, dan Beban secara bersamaan
model_var = VAR(train_multivariate)
results_var = model_var.fit(maxlags=15, ic='aic')

print(f"Ordo VAR terpilih berdasarkan AIC: {results_var.k_ar}")

# Plot Impulse Response Function (IRF)
irf = results_var.irf(10)
```

```
irf.plot(impulse='arus', response='tegangan')
plt.title("Respon Tegangan terhadap Impuls Arus")
plt.show()
```

Bab 7

Discrete Kalman Filter untuk Sinyal Sensor Bising

Informasi Modul 7

Sub-CPMK 7: Mampu memformulasikan model State-Space dan menerapkan Discrete Kalman Filter untuk memulihkan sinyal sensor bising.
Dataset: Sinyal telemetri arus DC dan tegangan (`noisy_sensor.csv`)

7.1 Dasar Teori

Persamaan Ruang Keadaan Diskrit:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1}, \quad w \sim \mathcal{N}(0, Q) \tag{7.1}$$

$$z_k = Hx_k + v_k, \quad v \sim \mathcal{N}(0, R) \tag{7.2}$$

Algoritma Kalman Filter bekerja dalam 2 fase: **Time Update (Prediksi)** dan **Measurement Update (Koreksi)**.

7.2 Kode Implementasi Python Berorientasi Objek (Matriks)

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

class KalmanFilter:
    def __init__(self, A, B, H, Q, R, x0, P0):
        self.A, self.B, self.H = A, B, H
        self.Q, self.R = Q, R
        self.x, self.P = x0, P0

    def predict(self, u=np.zeros((1,1))):
        # Time Update (Prediksi)
        self.x = self.A @ self.x + self.B @ u
        self.P = self.A @ self.P @ self.A.T + self.Q
        return self.x

    def update(self, z):
        # Measurement Update (Koreksi)
        S = self.H @ self.P @ self.H.T + self.R
        K = self.P @ self.H.T @ np.linalg.inv(S)
        y = z - self.H @ self.x # Inovasi (Error)
        self.x = self.x + K @ y
```

```
        I = np.eye(self.P.shape[0])
        self.P = (I - K @ self.H) @ self.P
        return self.x

# Contoh Penggunaan untuk Sistem 2-Dimensi (Posisi & Kecepatan)
# x = [Posisi, Kecepatan]^T
dt = 1.0
A = np.array([[1.0, dt], [0.0, 1.0]])
B = np.array([[0.5 * dt**2], [dt]])
H = np.array([[1.0, 0.0]]) # Hanya posisi yang disensor
Q = np.eye(2) * 1e-4
R = np.array([[0.1]]) # Variansi sensor
x0 = np.array([[0.0], [0.0]])
P0 = np.eye(2) * 1.0

kf = KalmanFilter(A, B, H, Q, R, x0, P0)

estimates = []
for z_meas in df['sensor_posisi'].values:
    kf.predict()
    x_est = kf.update(np.array([[z_meas]]))
    estimates.append(x_est[0, 0]) # Simpan estimasi posisi
```

Bab 8

Analisis Spektral FFT dan Dekomposisi Wavelet

Informasi Modul 8

Sub-CPMK 8: Mampu menganalisis spektrum frekuensi sinyal deret waktu menggunakan FFT dan dekomposisi Wavelet untuk deteksi transien/harmonisa.
Dataset: Sinyal getaran motor induksi & arus transien (`motor_vibration.csv`)

8.1 Dasar Teori

Transformasi Wavelet Diskrit (*Discrete Wavelet Transform* - DWT) menguraikan sinyal ke dalam koefisien aproksimasi (cA) dan detail (cD):

$$y(t) = \sum_{k} cA_{J,k}\phi_{J,k}(t) + \sum_{j=1}^{J}\sum_{k} cD_{j,k}\psi_{j,k}(t) \tag{8.1}$$

8.2 Kode Implementasi Python

```python
import pywt
import numpy as np

signal = df['current_transient'].values

# Dekomposisi Wavelet 3 Level menggunakan wavelet Daubechies (db4)
coeffs = pywt.wavedec(signal, 'db4', level=3)
cA3, cD3, cD2, cD1 = coeffs

# Denoising dengan Soft Thresholding pada koefisien detail
threshold = np.median(np.abs(cD1)) / 0.6745 * np.sqrt(2 * np.log(len(signal)))
coeffs_thresh = [cA3] + [pywt.threshold(c, threshold, mode='soft') for c in [cD3,
    cD2, cD1]]

# Rekonstruksi Sinyal Bersih
clean_signal = pywt.waverec(coeffs_thresh, 'db4')
```

Bab 9

Feature Engineering dan Machine Learning (XGBoost)

Informasi Modul 9

Sub-CPMK 9: Mampu merancang rekayasa fitur temporal dan menerapkan algoritma XGBoost/LightGBM untuk peramalan beban listrik jangka pendek.
Dataset: Dataset beban listrik multivariat (`power_features.csv`)

9.1 Dasar Teori

Mengubah masalah peramalan sekuensial menjadi regresi terawasi (*Supervised Learning*) melalui pembuatan fitur:

$$y_t = f(y_{t-1}, y_{t-2}, y_{t-24}, \bar{y}_{\text{roll\_24}}, \sin(2\pi \cdot \text{hour}/24), \cos(2\pi \cdot \text{hour}/24), \dots) \tag{9.1}$$

9.2 Kode Implementasi Python

```
import xgboost as xgb

# 1. Feature Engineering
def create_features(df):
    df['hour'] = df.index.hour
    df['dayofweek'] = df.index.dayofweek
    df['lag_1'] = df['load'].shift(1)
    df['lag_24'] = df['load'].shift(24)
    df['lag_168'] = df['load'].shift(168)
    df['rolling_mean_24'] = df['load'].shift(1).rolling(window=24).mean()
    return df.dropna()

df_feat = create_features(df)
X = df_feat[['hour', 'dayofweek', 'lag_1', 'lag_24', 'lag_168', 'rolling_mean_24']]
y = df_feat['load']

# 2. Training XGBoost Regressor
X_train, X_test = X[:-168], X[-168:]
y_train, y_test = y[:-168], y[-168:]

model_xgb = xgb.XGBRegressor(n_estimators=500, learning_rate=0.03, max_depth=6)
model_xgb.fit(X_train, y_train, eval_set=[(X_test, y_test)], early_stopping_rounds=30, verbose=False)
```

Bab 10

Deep Learning: Arsitektur LSTM dan Seq2Seq

Informasi Modul 10 & 11

Sub-CPMK 10 & 11: Mampu merancang tensor 3D sekuensial dan melatih arsitektur LSTM / Seq2Seq untuk peramalan daya pembangkit surya (PV).
Dataset: Data radiasi & daya PLTS (`solar_pv_generation.csv`)

10.1 Dasar Teori

Jaringan *Long Short-Term Memory* (LSTM) mengatur aliran memori dengan tiga gerbang utama: *Forget Gate* (f_t), *Input Gate* (i_t), dan *Output Gate* (o_t).

10.2 Kode Implementasi PyTorch

```
import torch
import torch.nn as nn

class LSTMForecaster(nn.Module):
    def __init__(self, input_dim=4, hidden_dim=64, num_layers=2, output_dim=24):
        super(LSTMForecaster, self).__init__()
        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers, batch_first=True,
            dropout=0.2)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        # x shape: (batch_size, seq_length, input_dim)
        out, (hn, cn) = self.lstm(x)
        # Ambil hidden state terakhir
        out = self.fc(out[:, -1, :])
        return out

# Inisialisasi Model, Loss, dan Optimizer
model = LSTMForecaster()
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

Bab 11

Deteksi Anomali Konsumsi Energi Berbasis Autoencoder

Informasi Modul 12

Sub-CPMK 12: Mampu membangun model rekonstruksi LSTM Autoencoder untuk deteksi pencurian listrik dan anomali sensor meter pintar.
Dataset: Profil konsumsi Smart Meter (`smartmeter_anomaly.csv`)

11.1 Dasar Teori

Autoencoder dilatih hanya pada data operasi normal. Ketika anomali terjadi, galat rekonstruksi (*Reconstruction Error* $\mathcal{L}$) akan melonjak melampaui ambang batas τ:

$$\mathcal{L}(x_t, \hat{x}_t) = \|x_t - \hat{x}_t\|^2 > \tau \tag{11.1}$$

11.2 Kode Implementasi Python

```
# Menghitung Reconstruction Error (MSE per sampel deret waktu)
reconstructed = autoencoder.predict(X_test)
mse_errors = np.mean(np.power(X_test - reconstructed, 2), axis=(1, 2))

# Menentukan ambang batas anomali (Threshold = Mean + 3*Std)
threshold = np.mean(mse_train_errors) + 3 * np.std(mse_train_errors)
anomalies = mse_errors > threshold

print(f'Jumlah titik anomali terdeteksi: {np.sum(anomalies)} dari {len(mse_errors)} jendela waktu')
```

Bab 12

Integrasi Peramalan pada Operasi Sistem Tenaga

Informasi Modul 13 & 14

Sub-CPMK 13: Mampu mengintegrasikan peramalan beban & EBT ke dalam simulasi penjadwalan baterai (BESS) pada microgrid berbasis ketidakpastian.
Dataset: Profil gabungan PV, Beban, dan Tarif Listrik (`microgrid_dispatch.csv`)

12.1 Dasar Teori

Keseimbangan daya pada mikrogrid dengan sistem penyimpanan baterai (BESS):

$$P_{\text{grid}}(t) = \hat{P}_{\text{load}}(t) - \hat{P}_{\text{PV}}(t) + P_{\text{charge}}(t) - P_{\text{discharge}}(t) \tag{12.1}$$

Tujuan optimasi adalah meminimalkan biaya pembelian energi listrik dari jaringan utama dengan memanfaatkan perkiraan daya surya saat jam puncak.

12.2 Tugas Simulasi Mandiri

1. Bangun model peramalan probabilistik (Quantile Regression 10%, 50%, dan 90%) untuk membatasi ketidakpastian daya fotovoltaik.
2. Simulasikan strategi operasi BESS (*Peak Shaving*) selama 24 jam dan hitung penghematan biaya listrik yang diperoleh!