Stationary Processes and ARIMA Models

Building upon foundational concepts in time series analysis

Time Series Components and Fundamental Concepts

Time series analysis begins with understanding the fundamental components that constitute any temporal dataset. The classical decomposition model represents a time series as the combination of trend, seasonal, and irregular components.

The **trend** component captures the long-term directional movement in data, representing underlying growth or decline patterns over extended periods.

Seasonality refers to predictable fluctuations that occur at regular intervals, such as quarterly business cycles or annual temperature variations.

Periodicity encompasses broader cyclical patterns that may not follow calendar-based intervals, while **irregular components** represent random fluctuations or noise that cannot be attributed to systematic patterns.

The decomposition framework provides the mathematical foundation:

$$Y_t = T_t + S_t + I_t$$

for additive models or

$$Y_t = T_t \times S_t \times I_t$$

for multiplicative models, where Tt, St, and It represent trend, seasonal, and irregular components respectively.

Smoothing Techniques and Moving Averages

Moving averages serve as fundamental smoothing techniques that estimate local trends by averaging observations within sliding windows. Simple moving averages compute

$$MA_t = k_1 \sum_{i=0}^{k-1} Y_{t-i}$$

where k represents the window size. Exponential smoothing extends this concept by applying exponentially decreasing weights to historical observations:

$$S_t = \alpha Y_t + (1 - \alpha) S_{t-1}$$

, where α controls the smoothing parameter

Adaptive smoothing techniques adjust smoothing parameters based on local data characteristics, providing more responsive estimates during periods of change while maintaining stability during stationary periods. These methods form the foundation for understanding more sophisticated forecasting approaches and highlight the trade-off between responsiveness and stability that characterizes many time series methods.

Autocorrelation Function and Statistical Properties

The autocorrelation function (ACF) measures linear dependence between observations separated by various time lags:

$$\rho_k = \frac{Cov(Y_t, Y_{t-k})}{Var(Y_t)}$$

This function reveals the persistence of correlations across time and provides essential diagnostic information about underlying processes. The ACF properties include symmetry (ρk=ρ−k), bounded values ($-1 \leq \rho_k \leq 1$), and the relationship to spectral density through Fourier transforms.

Portmanteau tests, including the Ljung-Box test, evaluate whether residuals from fitted models exhibit significant autocorrelation[5]. These tests aggregate autocorrelations across multiple lags:

$$Q = n(n+2) \sum_{k=1}^{h} \frac{\rho_k^2}{n-k}$$

where h represents the maximum lag considered. Understanding these statistical tests is essential for model validation and ensuring that fitted models adequately capture temporal dependencies.

Gaussian Process Transformations

Transformations to achieve Gaussian processes involve mathematical operations that convert non-normal data into approximately normal distributions. Common transformations include logarithmic

$$(ln(Y_t)), \text{Box-Cox } (\frac{Y_t^{\lambda} - 1}{\lambda})$$

, and power transformations. These transformations serve multiple purposes: stabilizing variance, linearizing relationships, and satisfying distributional assumptions required by many time series models.

The Box-Cox transformation generalizes many common transformations through a single parameter λ, allowing data-driven selection of optimal transformations.

When $\lambda = 0$, the transformation reduces to the natural logarithm, while $\lambda = 1$ represents no transformation. Proper transformation selection enhances model performance and ensures statistical assumptions are satisfied.

Stationary Processes

Mathematical Foundation of Stationarity

Stationarity represents one of the most fundamental concepts in time series analysis, providing the mathematical framework that enables consistent statistical inference and reliable forecasting. A time series process $\{Y_t\}$ is **strictly stationary** if the joint distribution of any collection of observations $(Y_{t1},Y_{t2},\dots,Y_{tk})$ is identical to $(Y_{t1+h},Y_{t2+h},\dots,Y_{tk+h})$ for any lag h and any finite set of time points.

This definition requires that all statistical properties remain unchanged under time translations.

Weak stationarity (or covariance stationarity) imposes less stringent requirements, requiring only that the mean function $E[Y_t]=\mu$ remains constant over time, the variance $Var(Y_t)=\sigma^2$ stays finite and constant, and the autocovariance function $Cov(Y_t,Y_{t-k})=\gamma_k$ depends only on the lag k, not on the absolute time t. Mathematically, these conditions are expressed as:

$E[Y_t]=\mu$ for all t

$Var(Y_t)=\gamma \quad 0<\infty$ for all t

$Cov(Y_t,Y_{t-k})=\gamma_k$ for all t,k

The importance of stationarity extends beyond mathematical convenience. Stationary processes exhibit predictable statistical behavior that allows for reliable parameter estimation and forecast generation.

Non-stationary processes, by contrast, may exhibit changing statistical properties over time, making traditional inference procedures unreliable and potentially leading to spurious relationships in econometric modeling.

Testing for Stationarity

Several statistical tests have been developed to assess stationarity empirically.

The Augmented Dickey-Fuller (ADF) test examines the null hypothesis of a unit root (non-stationarity) against the alternative of stationarity. The test is based on the regression:

$$\Delta Y_t = \alpha + \beta t + \gamma Y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta Y_{t-i} + \epsilon_t$$

where the null hypothesis H0:γ=0 indicates the presence of a unit root (non-stationarity), while H1:γ<0 suggests stationarity.

The KPSS test reverses this framework, testing stationarity as the null hypothesis against the alternative of non-stationarity.

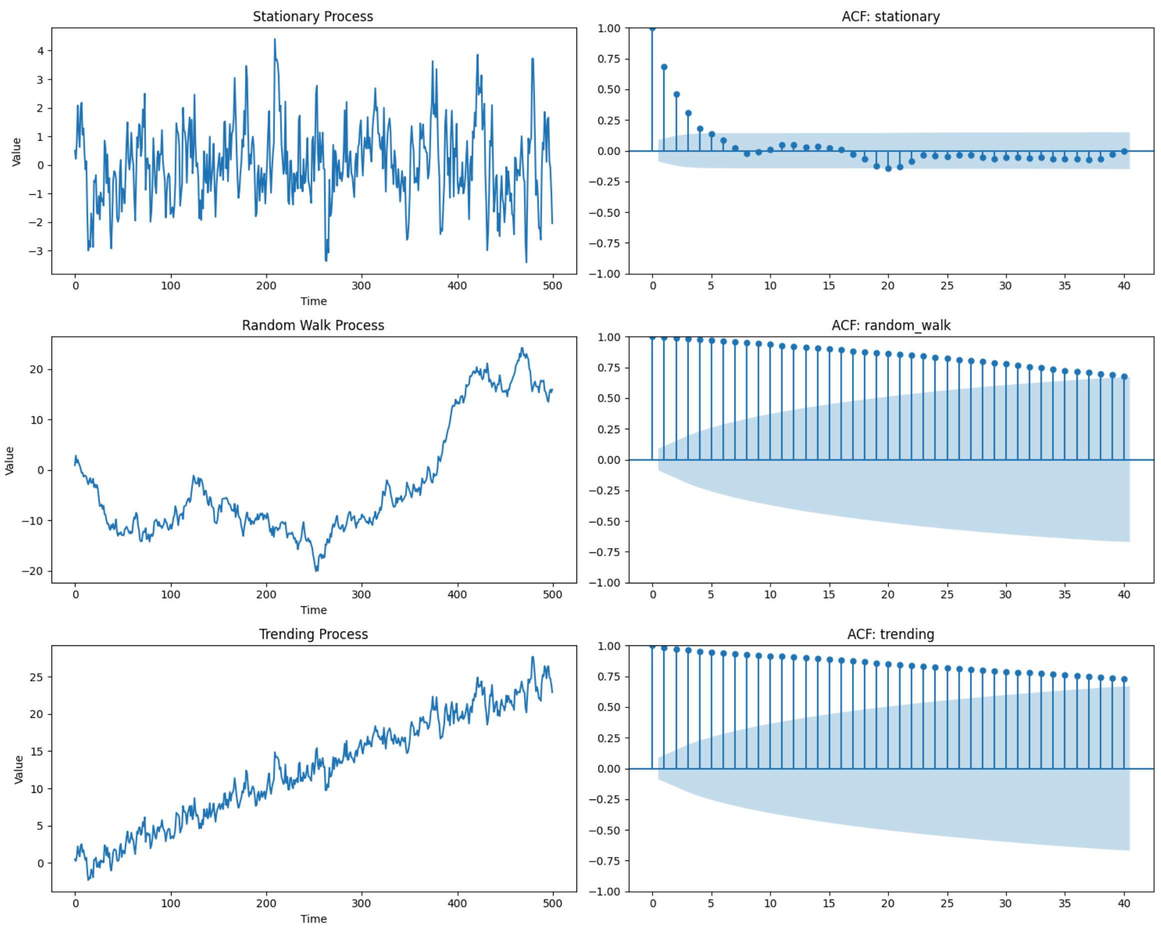
More sophisticated approaches examine stationarity across different aspects of the distribution.

Second-order stationarity tests focus specifically on the stability of variance and covariance structures. These tests are particularly valuable when analyzing financial time series or other applications where volatility clustering may indicate non-stationary variance behavior.

Python Implementation and Practical Example

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.tsa.arima_process import ArmaProcess
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import seaborn as sns

# Generate synthetic stationary and non-stationary processes
np.random.seed(42)
n = 500

# Stationary process: AR(1) with phi < 1
phi = 0.7
```

```
stationary_process = ArmaProcess([1, -phi], [1]).generate_sample(n)

# Non-stationary process: Random walk
random_walk = np.cumsum(np.random.normal(0, 1, n))

# Create trending non-stationary process
trend = 0.05 * np.arange(n)
trending_process = stationary_process + trend

# Combine into DataFrame
data = pd.DataFrame({
    'time': range(n),
    'stationary': stationary_process,
    'random_walk': random_walk,
    'trending': trending_process
})

# Visualization
fig, axes = plt.subplots(3, 2, figsize=(15, 12))

# Plot time series
for i, col in enumerate(['stationary', 'random_walk', 'trending']):
    axes[i, 0].plot(data['time'], data[col])
    axes[i, 0].set_title(f'{col.replace("_", " ").title()} Process')
    axes[i, 0].set_xlabel('Time')
    axes[i, 0].set_ylabel('Value')

    # Plot ACF
    plot_acf(data[col], ax=axes[i, 1], lags=40, title=f'ACF: {col}')

plt.tight_layout()
plt.show()

# Statistical Tests Function
def stationarity_tests(series, series_name):
    """Perform comprehensive stationarity tests"""
    print(f"\n=== Stationarity Tests for {series_name} ===")

    # ADF Test
    adf_result = adfuller(series, autolag='AIC')
    print(f"\nAugmented Dickey-Fuller Test:")
    print(f"Test Statistic: {adf_result[0]:.6f}")
    print(f"p-value: {adf_result[1]:.6f}")
    print(f"Critical Values: {adf_result[4]}")
    print(f"Conclusion: {'Stationary' if adf_result[1] < 0.05 else 'Non-stationary'}")

    # KPSS Test
    kpss_result = kpss(series, regression='c')
    print(f"\nKPSS Test:")
    print(f"Test Statistic: {kpss_result[0]:.6f}")
    print(f"p-value: {kpss_result[1]:.6f}")
    print(f"Critical Values: {kpss_result[3]}")
    print(f"Conclusion: {'Stationary' if kpss_result[1] > 0.05 else 'Non-stationary'}'

    return adf_result, kpss_result

# Apply tests to all series
for col in ['stationary', 'random_walk', 'trending']:
    stationarity_tests(data[col], col)

# Demonstrate differencing for non-stationary series
print("\n=== After First Differencing ===")
diff_random_walk = np.diff(data['random_walk'])
diff_trending = np.diff(data['trending'])

stationarity_tests(diff_random_walk, 'Differenced Random Walk')
stationarity_tests(diff_trending, 'Differenced Trending Process')
```



Let's understand the code

First, the necessary libraries are imported:

```
import numpy as npimport pandas as pdimport matplotlib.pyplot as pltfrom stats
```

- `numpy` is used for numerical operations.
- `pandas` is used for creating and manipulating DataFrames.
- `matplotlib.pyplot` is used for creating visualizations.
- `adfuller` and `kpss` from `statsmodels.tsa.stattools` are used to perform stationarity tests.

- `ArmaProcess` from `statsmodels.tsa.arima_process` is used to generate a synthetic stationary process.
- `plot_acf` and `plot_pacf` from `statsmodels.graphics.tsaplots` are used to plot the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF). Note that `plot_pacf` is imported but not used in this specific code block.
- `seaborn` is used for enhancing the plot aesthetics.

Next, synthetic time series data is generated to represent different types of processes:

```
# Generate synthetic stationary and non-stationary processesnp.random.seed(4
```

- `np.random.seed(42)` ensures that the random data generated is the same each time the code is run.
- `n` sets the number of data points for each time series.
- `stationary_process` creates an AutoRegressive (AR) process of order 1 using `ArmaProcess`. The parameter `phi` is set to 0.7, which is less than 1, ensuring stationarity.
- `random_walk` creates a non-stationary random walk by cumulatively summing random normal values using `np.cumsum`.
- `trending_process` creates another non-stationary series by adding a linear `trend` to the `stationary_process`.
- These three series are then combined into a pandas DataFrame called `data`.

The code then visualizes these generated time series and their Autocorrelation Functions (ACF):

```
# Visualizationfig, axes = plt.subplots(3, 2, figsize=(15, 12))# Plot time seriesfor i,
```

- `plt.subplots` creates a figure and a grid of subplots (3 rows, 2 columns).

- A `for` loop iterates through the 'stationary', 'random_walk', and 'trending' columns of the `data` DataFrame.
- For each column, the time series itself is plotted in the left column of the subplots (`axes[i, 0]`).
- The Autocorrelation Function (ACF) is plotted in the right column of the subplots (`axes[i, 1]`) using `plot_acf` . The ACF helps visualize the correlation of a series with its lagged values. For stationary series, the ACF usually drops off quickly, while for non-stationary series, it tends to decay slowly.
- `plt.tight_layout()` adjusts the spacing between subplots to prevent overlap.
- `plt.show()` displays the plots.

A function `stationarity_tests` is defined to perform formal statistical tests for stationarity:

```
# Statistical Tests Functiondef stationarity_tests(series, series_name):    """Perfo
```

- This function takes a time `series` and its `series_name` as input.
- It performs two common stationarity tests: the Augmented Dickey-Fuller (ADF) test and the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test.
- The **ADF test** checks if a unit root is present in the time series. The null hypothesis is that a unit root is present (non-stationary), and the alternative hypothesis is that there is no unit root (stationary). A small p-value (typically < 0.05) leads to rejecting the null hypothesis and concluding the series is stationary.
- The **KPSS test** checks for trend stationarity. The null hypothesis is that the series is trend stationary, and the alternative hypothesis is that it is not stationary (has a unit root). A large p-value (typically > 0.05) leads to failing to reject the null hypothesis and concluding the series is stationary around a trend or mean [1].
- The function prints the test statistics, p-values, critical values, and a conclusion based on the p-value for both tests.

The `stationarity_tests` function is then applied to each of the generated time series:

```
# Apply tests to all seriesfor col in ['stationary', 'random_walk', 'trending']:    stati
```

- This loop calls the `stationarity_tests` function for the 'stationary', 'random_walk', and 'trending' columns of the `data` DataFrame.

Finally, the code demonstrates how differencing can be used to make non-stationary series stationary:

```
# Demonstrate differencing for non-stationary seriesprint("\n=== After First Diffe
```

- `np.diff()` calculates the difference between consecutive elements in the 'random_walk' and 'trending' series. This is known as first-order differencing.
- The `stationarity_tests` function is then called on these differenced series to show that differencing can often remove non-stationarity.

Results

```
=== Stationarity Tests for stationary ===

Augmented Dickey-Fuller Test:
Test Statistic: -9.569757
p-value: 0.000000
Critical Values: {'1%': np.float64(-3.4435228622952065), '5%': np.float64(-
2.867349510566146), '10%': np.float64(-2.569864247011056)}
Conclusion: Stationary

KPSS Test:
Test Statistic: 0.154522
p-value: 0.100000
```

```
Critical Values: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Conclusion: Stationary

=== Stationarity Tests for random_walk ===

Augmented Dickey-Fuller Test:
Test Statistic: -0.282015
p-value: 0.928013
Critical Values: {'1%': np.float64(-3.4435228622952065), '5%': np.float64(-
2.867349510566146), '10%': np.float64(-2.569864247011056)}
Conclusion: Non-stationary

KPSS Test:
Test Statistic: 2.327090
p-value: 0.010000
Critical Values: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Conclusion: Non-stationary

=== Stationarity Tests for trending ===

Augmented Dickey-Fuller Test:
Test Statistic: -0.442416
p-value: 0.902772
Critical Values: {'1%': np.float64(-3.4437936797256317), '5%': np.float64(-2.
867468682890213), '10%': np.float64(-2.5699277594606915)}
Conclusion: Non-stationary

KPSS Test:
Test Statistic: 3.894911
p-value: 0.010000
Critical Values: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Conclusion: Non-stationary

=== After First Differencing ===

=== Stationarity Tests for Differenced Random Walk ===
Augmented Dickey-Fuller Test:
Test Statistic: -22.620213
p-value: 0.000000
Critical Values: {'1%': np.float64(-3.4435494520411605), '5%': np.float64(-2.
8673612117611267), '10%': np.float64(-2.5698704830567247)}
Conclusion: Stationary

KPSS Test:
Test Statistic: 0.432797
p-value: 0.063019
Critical Values: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Conclusion: Stationary

=== Stationarity Tests for Differenced Trending Process ===

Augmented Dickey-Fuller Test:
Test Statistic: -11.035010
p-value: 0.000000
Critical Values: {'1%': np.float64(-3.4437936797256317), '5%': np.float64(-2.
867468682890213), '10%': np.float64(-2.5699277594606915)}
Conclusion: Stationary

KPSS Test:
Test Statistic: 0.027923
p-value: 0.100000
Critical Values: {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}
Conclusion: Stationary
```

```
<ipython-input-1-3cad6cd70e4f>:62: InterpolationWarning: The test statistic i
s outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_result = kpss(series, regression='c')
<ipython-input-1-3cad6cd70e4f>:62: InterpolationWarning: The test statistic i
```

```
s outside of the range of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_result = kpss(series, regression='c')
<ipython-input-1-3cad6cd70e4f>:62: InterpolationWarning: The test statistic i
s outside of the range of p-values available in the
look-up table. The actual p-value is smaller than the p-value returned.

  kpss_result = kpss(series, regression='c')
<ipython-input-1-3cad6cd70e4f>:62: InterpolationWarning: The test statistic i
s outside of the range of p-values available in the
look-up table. The actual p-value is greater than the p-value returned.

  kpss_result = kpss(series, regression='c')
  ((np.float64(-11.035010085083881),
  np.float64(5.545429875799353e-20),
  9,
  489,
  {'1%': np.float64(-3.4437936797256317),
   '5%': np.float64(-2.867468682890213),
   '10%': np.float64(-2.5699277594606915)},
  np.float64(1385.0112612772027)),
 (np.float64(0.027922960541109355),
  np.float64(0.1),
  16,
  {'10%': 0.347, '5%': 0.463, '2.5%': 0.574, '1%': 0.739}))
```

We observe the practical identification of stationary and non-stationary processes through both visual inspection and formal statistical testing.

The code generates three distinct processes: a stationary AR(1) process, a non-stationary random walk, and a trending non-stationary process, observe how different types of non-stationarity manifest in both time plots and autocorrelation functions.