Time Series Note

DongDong

2024-02-23

Table of contents

python python python
`plotly` `d3.js` `plotly`

1

```python
import plotly.graph_objects as go
import numpy as np
```

1.1 Time Series v.s. Cross Sectional

(time series) (cross sectional)

Table 1.1: Time Series v.s. Cross Sectional

	Time Series	Cross Sectional
	2000/01/01 2020/01/01	2000/01/01 50

1.2

2000/01/01 2020/01/01

1.3

- $U_t, t \in \{1, \cdots, T\}$

U_t U_t

- $U \rightarrow$
- $t \rightarrow$

$U_t = T_t + S_t + C_t + R_t$

- T_t : (Trend component)
- S_t : (Seasonal component)
- C_t : (Cyclical component)
- R_t : (Random component)

1.3.1

- Downward Trend: Television Newspaper
- No Trend: Radio
- Upward Trand: Internet

```python
title = 'Main Source for News'
labels = ['Television', 'Newspaper', 'Internet', 'Radio']
colors = ['rgb(67,67,67)', 'rgb(115,115,115)', 'rgb(49,130,189)', 'rgb(189,189,189)']

mode_size = [8, 8, 12, 8]
line_size = [2, 2, 4, 2]

x_data = np.vstack((np.arange(2001, 2014),)*4)

y_data = np.array([
    [74, 82, 80, 74, 73, 72, 74, 70, 70, 66, 66, 69],
    [45, 42, 50, 46, 36, 36, 34, 35, 32, 31, 31, 28],
    [13, 14, 20, 24, 20, 24, 24, 40, 35, 41, 43, 50],
    [18, 21, 18, 21, 16, 14, 13, 18, 17, 16, 19, 23],
])

fig = go.Figure()

for i in range(0, 4):
    fig.add_trace(go.Scatter(x=x_data[i], y=y_data[i], mode='lines',
        name=labels[i],
        line=dict(color=colors[i], width=line_size[i]),
        connectgaps=True,
    ))

    # endpoints
    fig.add_trace(go.Scatter(
        x=[x_data[i][0], x_data[i][-1]],
        y=[y_data[i][0], y_data[i][-1]],
        mode='markers',
        marker=dict(color=colors[i], size=mode_size[i])
    ))

fig.update_layout(
    xaxis=dict(
        showline=True,
        showgrid=False,
        showticklabels=True,
        linecolor='rgb(204, 204, 204)',
        linewidth=2,
        ticks='outside',
        tickfont=dict(
            family='Arial',
            size=12,
            color='rgb(82, 82, 82)',
        ),
    ),
    yaxis=dict(
        showgrid=False,
        zeroline=False,
        showline=False,
        showticklabels=False,
    ),
    autosize=False,
    margin=dict(
        autoexpand=False,
        l=100,
        r=20,
        t=110,
    ),
    showlegend=False,
    plot_bgcolor='white'
```

```python
)

annotations = []

# Adding labels
for y_trace, label, color in zip(y_data, labels, colors):
    # labeling the left_side of the plot
    annotations.append(dict(xref='paper', x=0.05, y=y_trace[0],
                              xanchor='right', yanchor='middle',
                              text=label + ' {}%'.format(y_trace[0]),
                              font=dict(family='Arial',
                                        size=16),
                              showarrow=False))
    # labeling the right_side of the plot
    annotations.append(dict(xref='paper', x=0.95, y=y_trace[11],
                              xanchor='left', yanchor='middle',
                              text='{}%'.format(y_trace[11]),
                              font=dict(family='Arial',
                                        size=16),
                              showarrow=False))
# Title
annotations.append(dict(xref='paper', yref='paper', x=0.0, y=1.05,
                              xanchor='left', yanchor='bottom',
                              text='Main Source for News',
                              font=dict(family='Arial',
                                        size=30,
                                        color='rgb(37,37,37)'),
                              showarrow=False))
# Source
annotations.append(dict(xref='paper', yref='paper', x=0.5, y=-0.1,
                              xanchor='center', yanchor='top',
                              text='Source: PewResearch Center & ' +
                                   'Storytelling with data',
                              font=dict(family='Arial',
                                        size=12,
                                        color='rgb(150,150,150)'),
                              showarrow=False))

fig.update_layout(annotations=annotations)

fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

1.3.2

```python
title = 'Seasonal Effect'

x_data = np.linspace(-10, 10, 100)

y_data = np.array(
    np.sin(x_data) + np.random.normal(size=x_data.shape) * 0.25
)

fig = go.Figure()

fig.add_trace(go.Scatter(x=x_data, y=y_data, mode='lines+markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```python
title = 'Cyclical Effect'

x_data = np.linspace(-10, 10, 100)

y_data = np.array(
    np.sin(x_data)*np.exp(x_data/10) + np.random.normal(size=x_data.shape) * 0.25
)

fig = go.Figure()
```

```
fig.add_trace(go.Scatter(x=x_data, y=y_data, mode='lines+markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

1.3.3

sin

```
title = 'Seasonal Effect'

x_data = np.linspace(-10, 10, 100)

y_data = np.array(
    np.sin(x_data) + np.random.normal(size=x_data.shape) * 1
)

fig = go.Figure()

fig.add_trace(go.Scatter(x=x_data, y=y_data, mode='lines+markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

1.4

```
title = 'Cyclical Effect'

x_data = np.linspace(-10, 10, 1000)

y_data = np.array(
    np.sin(x_data)*np.exp(x_data/100)/10 + np.random.normal(size=x_data.shape)
)

fig = go.Figure()

fig.add_trace(go.Scatter(x=x_data, y=y_data, mode='lines',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
import plotly.figure_factory as ff
import numpy as np

hist_data = [y_data]

group_labels = ['Data (no T)']
colors = ['#A56CC1']

# Create distplot with curve_type set to 'normal'
fig = ff.create_distplot(hist_data, group_labels, colors=colors,
                         bin_size=.2, show_rug=False)

# Add title
fig.update_layout(title_text='Hist and Curve Plot')
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

1.5 Stationary

1.5.1 Weak stationary

$\{Z_t\}$ (second-order or covariance stationary)

- $\mu(t) = C : \mu$
- $\gamma(t, t-k) = \gamma(0, k) :$ (lag)

$$\sim \mathcal{N}(0, \sigma^2 I)$$

$$X_t = X_{t-1} + \epsilon$$

ϵ

- $\mathbb{E}(X_t) = \mathbb{E}(X_{t-1})$
- $\mathrm{Var}(X_t) = t\sigma^2$

```python
title = 'Seasonal Effect'

T = 1000
walks = []

loc = 0
for i in range(T):
    loc += + np.random.normal(0,1)
    walks.append(loc)
walks = np.array(walks)

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=walks, mode='lines',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

$$D_t = X_t - X_{t-1}$$

- $\mathbb{E}(D_t) = 0$
- $\mathrm{Var}(X_t) = \sigma^2$

```
title = 'Seasonal Effect'

D = walks[1:] - walks[:-1]

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T-1)), y=D, mode='lines',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

1.5.2 Strict stationary

$\{Z_t\}$

$Z_{t_1}, Z_{t_2}, \cdots, Z_{t_n}$ $\quad$ $Z_{t_1-k}, Z_{t_2-k}, \cdots, Z_{t_n-k}$

Note:

-
-

1.5.3 Sample AutoCorrelation Function (ACF)

? **** ** ACF

$$\rho_\ell = \frac{\mathrm{cov}(r_t, r_{t-\ell})}{\mathrm{var}(r_t)}$$

-
- $\gamma_k = \mathrm{Cov}(r_t, r_{t-k}) = E[(r_t - \mu)(r_{t-k} - \mu)]$: k
- $\rho_0 = 1$: 1
- $\rho_k = \rho_{-k}$:

Lag-ℓ μ ( )

$$\hat{\rho}_\ell = \frac{\Sigma_{t=1}^{T-\ell}(r_t - \bar{r})(r_{t+\ell} - \bar{r})}{\Sigma_{t=1}^{T}(r_t - \bar{r})^2}$$

- $\bar{r}$:
- T :

1.5.3.1

( ) T T
ACF

1.5.3.2 Python

python `numpy.correlate` `numpy.correlate(a, v, mode)` v filter
a v a ( ) `numpy.correlate`

$$c_k = \sum_n a_{n+k} * \bar{v}_n$$

- $\bar{v}_n$ complex conjugation
- `mode:`

– `valid`: v a a
– `same`: v a
– `full`: v a

```
def acf(series, lag=None):
    """
    Calculate the autocorrelation function (ACF) for a given time series.

    Parameters:
        series (array-like): The time series data.
        lag (int, optional): The maximum lag for which to calculate the ACF.
            If None (default), the ACF is calculated for all possible lags.

    Returns:
        acf_values (array): Autocorrelation values for the given time series.
            If `lag` is provided, returns a single value representing
            autocorrelation at that lag. Otherwise, returns an array of
            autocorrelation values for all lags.
    """
    series = np.asarray(series)
    n = len(series)

    mean = np.mean(series)
    centered_data = series - mean

    # Calculate autocovariance function
    acov = np.correlate(centered_data, centered_data, mode='full')
    acf_values = acov / sum(centered_data ** 2)

    if lag is not None:
        return acf_values[(n-1):(n-1+lag)]
    else:
        return acf_values[(n-1):]
```

ACF

```
rw_acf = acf(walks, lag=90)
title = 'ACF Random Walks'

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=rw_acf, mode='markers',
        name="sin",
        line=dict(color="blue", width=1),
        connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
d_acf = acf(D, lag=90)
title = 'ACF D'

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T-1)), y=d_acf, mode='markers',
        name="sin",
        line=dict(color="blue", width=1),
        connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

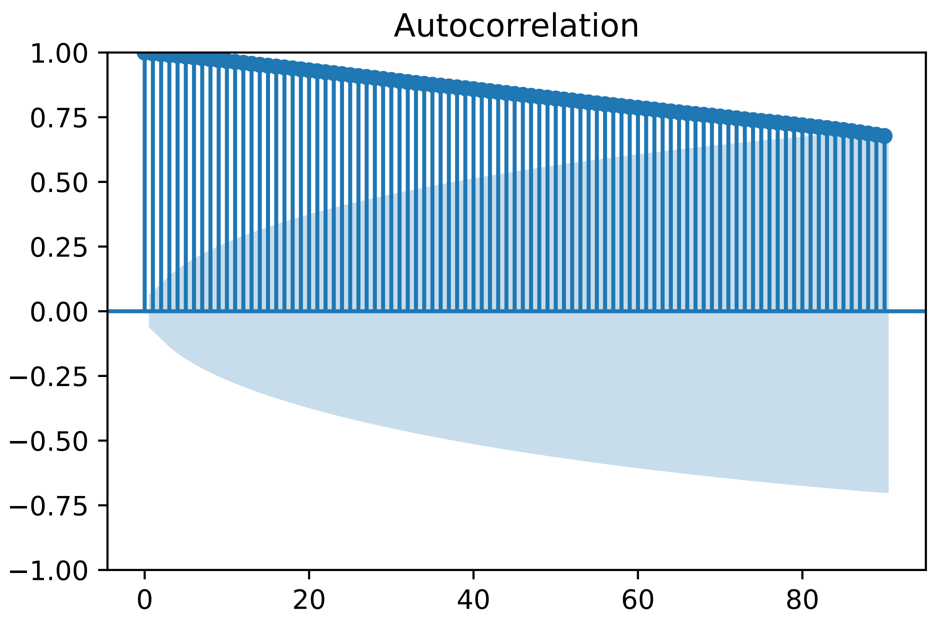
```
Unable to display output for mime type(s): text/html
```

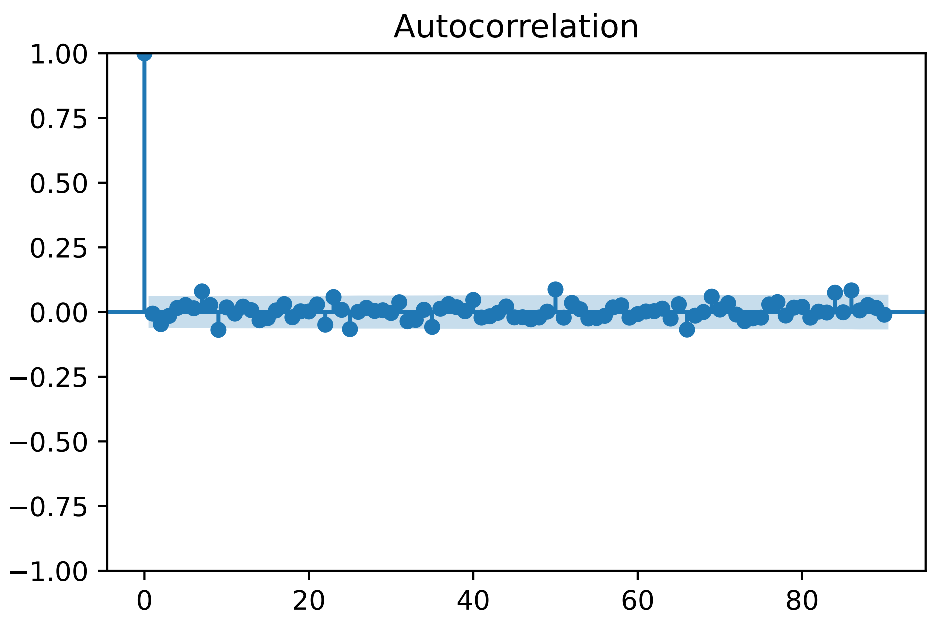
python `statsmodels.graphics.tsaplots.plot_acf`

```
from statsmodels.graphics.tsaplots import plot_acf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

plot_acf(walks, lags=90)
plt.show()

plot_acf(D, lags=90)
plt.show()
```


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
20
40
60
80

2

$$r_t = \mu + \sum_{i=0}^{\infty} \psi_i a_{t-i}$$

- μ : r_t
- $a(t)$: 0 t t innovation shock
- ψ : $\psi_0 = 1$

$$\mathbb{E}(r_t) = \mu, \mathrm{Var}(r_t) = \sigma_a^2 \sum_{i=0}^{\infty} \psi_i^2$$

$\mathrm{Var}(a_t) = \sigma_a^2$ $\{\psi^2\}$ $i \to \infty$ $\psi_i^2 \to 0$

- t a_{t-i}

? ℓ

$$\rho_\ell = \frac{\mathrm{Cov}(r_t, r_{t-\ell})}{\mathrm{Var}(r_t)} = \frac{\gamma_\ell}{\gamma_0} = \frac{\sum_{i=0}^{\infty} \psi_i \, \psi_{i+\ell}}{1 + \sum_{i=1}^{\infty} \psi_i^2}, \quad \ell \geq 0$$

r_t $\mu + \sum_{i=0}^{\infty} \psi_i a_{t-i}$ $i \to \infty$ $\psi_i \to 0$ 0

2.1 Auto Regressive (AR)

AR AR

$$r_t = \phi_0 + \phi_1 r_{t-1} + a_t$$

- ϕ_0 :
- ϕ_1 :

$$\mathbb{E}(r_t|r_{t-1}) = \phi_0 + \phi_1 r_{t-1}, \qquad \mathrm{Var}(r_t|r_{t-1}) = \mathrm{Var}(a_t) = \sigma_a^2$$

Note:

- Markov :

2.1.1 AR(p)

AR(1) AR(p)

$$r_t = \phi_0 + \phi_1 r_{t-1} + \cdots + \phi_p r_{t-p} + a_t$$

p

2.1.2 AR

AR ACF AR(p) ACF ?

```
import numpy as np
import plotly.graph_objects as go

def acf(series, lag=None):
    """
    Calculate the autocorrelation function (ACF) for a given time series.

    Parameters:
        series (array-like): The time series data.
        lag (int, optional): The maximum lag for which to calculate the ACF.
            If None (default), the ACF is calculated for all possible lags.

    Returns:
        acf_values (array): Autocorrelation values for the given time series.
            If `lag` is provided, returns a single value representing
            autocorrelation at that lag. Otherwise, returns an array of
            autocorrelation values for all lags.
    """
    series = np.asarray(series)
    n = len(series)

    mean = np.mean(series)
    centered_data = series - mean

    # Calculate autocovariance function
    acov = np.correlate(centered_data, centered_data, mode='full')
    acf_values = acov / sum(centered_data ** 2)

    if lag is not None:
        return acf_values[(n-1):(n-1+lag)]
    else:
        return acf_values[(n-1):]
```

ACF AR(1)

```
title = 'Random Walks'

T = 1000
walks = []

loc = 0
for i in range(T):
    loc = loc + np.random.normal(0,1)
    walks.append(loc)
walks = np.array(walks)

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=walks, mode='lines',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

ACF

```
rw_acf = acf(walks, lag=90)
title = 'ACF Random Walks'

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=rw_acf, mode='markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

? 0.3

$$r_t = 0.3 * r_{t-1} + a_t$$

```
title = 'Random Walks (coef=0.3)'

T = 1000
walks = []
```

```python
loc = 0
for i in range(T):
    loc = .7 * loc + np.random.normal(0,1)
    walks.append(loc)
walks = np.array(walks)

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=walks, mode='lines',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```python
rw_acf = acf(walks, lag=90)
title = 'ACF Random Walks (coef=0.3)'

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=rw_acf, mode='markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

ACF ? ACF

2.1.3 AR(1)

AR γ_j j

$$\begin{aligned}
r_t &= \phi_0 + \phi_1 r_{t-1} + a_t \\
\mathbb{E}(r_t) &= \phi_0 + \phi_1 \mathbb{E}(r_{t-1}) \\
\mu &= \phi_0 + \phi_1 \mu \\
\mu &= \frac{\phi_0}{1-\phi_1}
\end{aligned}$$

- $\mathbb{E}(r_t) = \mathbb{E}(r_{t-1}) = \mu$
- $\phi_0 = 0 \rightarrow E(r_t) = 0$ AR(1) ϕ_0

$$\begin{aligned}
r_t - \mu &= \phi_1(r_{t-1} - \mu) + a_t \\
r_t - \mu &= a_t + \phi_1 a_{t-1} + \phi_1^2 a_{t-2} + \cdots
\end{aligned}$$

$$= \sum_{i=0}^{\infty} \phi_1^i a_{t-i}$$

AR

2.1.4 AR(1)

$$\begin{aligned}
r_t &= \phi_0 + \phi_1 r_{t-1} + a_t \\
\mathrm{Var}(r_t) &= \phi_1^2 \mathrm{Var}(r_{t-1}) + \sigma_a^2 \\
\mathrm{Var}(r_t) &= \frac{\sigma_a^2}{1-\phi_1^2}
\end{aligned}$$

- $\mathrm{Cov}(r_{t-1}, a_t) = 0:$ a_t r_{t-1} $\phi_0 + \phi_1 r_{t-2} + a_{t-1}$
- $\phi_1^2 < 1:$

- **AR(1)** $|\phi_1| < 1$

2.1.5 AR(1)

$\phi_\ell = \gamma_\ell / \gamma_0$ γ_0 ( ) $\gamma_\ell = \mathbb{E}[(r_t - \mu)(r_{t-\ell} - \mu)]$

$$\gamma_\ell = \begin{cases} \phi_1 \gamma_1 + \sigma_a^2 & \text{if } \ell = 0, \\ \phi_1 \gamma_{\ell-1} & \text{if } \ell > 0. \end{cases}$$

- $\gamma_0 = \mathrm{Var}(r_t) = \frac{\sigma_a^2}{1-\phi_1^2}$
- $\gamma_\ell = \phi_1 \gamma_{\ell-1}$ $(\ell > 0)$

$$\rho_\ell = \phi_1 \rho_{\ell-1} = \cdots = \phi_1^\ell \rho_0, \quad \text{for} \quad \ell \geq 0$$

- **AR(1) ACF**

AR(2)

2.1.6 AR Order

AR ? p AIC BIC p BIC AIC AIC BIC

- AIC:
- BIC: BIC

AR(3) AIC 2~4 3 BIC 3

AIC BIC Partial Autocorrelation Function (PACF) PACF K lag-K K K 1 ? PACF

$$\begin{aligned} r_t &= \phi_{0,1} + \phi_{1,1} r_{t-1} + e_{1t}, \\ r_t &= \phi_{0,2} + \phi_{1,2} r_{t-1} + \phi_{2,2} r_{t-2} + e_{2t}, \\ r_t &= \phi_{0,3} + \phi_{1,3} r_{t-1} + \phi_{2,3} r_{t-2} + \phi_{3,3} r_{t-3} + e_{3t}, \\ &\vdots \quad \vdots \end{aligned}$$

- $\phi_{i,j}$ AR(j) r_{t-i}
- e

- ϕ_0

$\hat{\phi}_{j,j}$ (lag-i PACF) $\hat{\phi}_{2,2}$ r_{t-2} AR(1) PACF OLS, Yule-Walker, Burg"s method, Levinson-Durbin python

```
# source https://www.jianshu.com/p/811f9ea0b52d
import numpy as np
from scipy.linalg import toeplitz

def yule_walker(ts, order):
    '''
    Solve yule walker equation
    '''
    x = np.array(ts) - np.mean(ts)
    n = x.shape[0]

    r = np.zeros(order+1, np.float64) # to store acf
    r[0] = x.dot(x) / n # r(0)
    for k in range(1, order+1):
        r[k] = x[:-k].dot(x[k:]) / (n - k) # r(k)

    R = toeplitz(r[:-1])

    return np.linalg.solve(R, r[1:]) # solve `Rb = r` to get `b`


def pacf(ts, k):
    '''
    Compute partial autocorrelation coefficients for given time series unbiased
    '''
    res = [1.]
    for i in range(1, k+1):
        res.append(yule_walker(ts, i)[-1])
    return np.array(res)
```

lag-1 0

```
rw_acf = pacf(walks, k=90)
title = 'PACF Random Walks (coef=0.3)'

fig = go.Figure()
```

```
fig.add_trace(go.Scatter(x=list(range(T)), y=rw_acf, mode='markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

PACF

- $\hat{\phi}_{j,j} \to \phi_p \quad \text{as} \quad T \to \infty$
- $\hat{\phi}_{\ell,\ell} \to 0 \quad \forall \ell > j$
- $\hat{\phi}_{\ell,\ell}$ $1/T$ $\ell > j$

AR(p) PACF lag-p

2.1.7 AR

AR(p) AR(p)

$$r_t = \phi_0 + \phi_1 r_{t-1} + \cdots + \phi_p r_{t-p} + a_t, \quad t = p+1, \ldots, T$$

$T - p$ OLS

$$\hat{r}_t = \hat{\phi}_0 + \hat{\phi}_1 r_{t-1} + \cdots + \hat{\phi}_p r_{t-p}$$

( ) $\hat{a}_t = r_t - \hat{r}_t$

Note: AR(p)

2.1.8 AR

forecasting

$$E\{[r_{h+\ell} - \hat{r}_h(\ell)]^2 | F_h\} \le \min_g E[(r_{h+\ell} - g)^2 | F_h]$$

- F_h h

  h l MSE g

  AR(p) l

$$r_{h+\ell} = \phi_0 + \phi_1 r_{h+\ell-1} + \cdots + \phi_p r_{h+\ell-p} + a_{h+\ell}$$

$1, 2, \cdots, l{-}1$ AR(p) l

2.1.9 AR

1. AR(3)

$$r_t = 0.01 + 0.1 * r_{t-1} - 0.1 * r_{t-3} + a_t$$

```
np.random.seed(42)

def generate_ar_data(n, mu=0, sigma=1):
    """
    Generate AR data with the specified equation.

    Parameters:
        n (int): Number of data points to generate.
        mu (float): Mean of the random noise.
        sigma (float): Standard deviation of the random noise.

    Returns:
        np.array: Array of AR data points.
    """
    # Initialize arrays
    ar_data = np.zeros(n)
    epsilon = np.random.normal(mu, sigma, n)
    # Generate AR data
    for t in range(3, n):
        ar_data[t] = 0.01 + 0.1 * ar_data[t-1] - 0.1 * ar_data[t-3] + epsilon[t]

    return ar_data
data = generate_ar_data(500)

title = "Simulated AR(3) data"

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=data, mode='lines',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

2. ACF plot

```
data_acf = acf(data, lag=10)
title = 'ACF AR(3)'

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=data_acf, mode='markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

lag-3

3. PCAF plot

```
data_pacf = pacf(data, k=10)
title = 'PACF AR(3)'

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=data_pacf, mode='markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

ACF   lag-3

4. AIC   AR order

```
from statsmodels.tsa.ar_model import AutoReg

best = float("inf")
best_order = -1
for order in range(1, 10):
    ar_model = AutoReg(data, lags=order).fit()
    print(f"AR({order}) AIC: ", ar_model.aic)
    if ar_model.aic < best:
        best = ar_model.aic
        best_order = order
print("Selected best order: ", best_order)
```

```
AR(1) AIC:  1407.3539983067183
AR(2) AIC:  1407.2913499349963
AR(3) AIC:  1402.828223161296
AR(4) AIC:  1398.1094017600367
AR(5) AIC:  1397.0403557874909
AR(6) AIC:  1396.8567941625547
AR(7) AIC:  1393.7280505207527
AR(8) AIC:  1389.2987596911696
AR(9) AIC:  1389.2034251860591
Selected best order:  9
```

5. AIC order fit data

```
ar_model = AutoReg(data, lags=best_order).fit()
print(ar_model.summary())
```

```
                            AutoReg Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  500
Model:                     AutoReg(9)   Log Likelihood                -683.602
Method:               Conditional MLE   S.D. of innovations              0.974
Date:                Mon, 04 Mar 2024   AIC                           1389.203
Time:                        13:05:59   BIC                           1435.364
Sample:                             9   HQIC                          1407.331
                                  500
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0117      0.044      0.267      0.790      -0.075       0.098
y.L1           0.0928      0.045      2.053      0.040       0.004       0.181
y.L2           0.0003      0.045      0.007      0.994      -0.088       0.089
y.L3          -0.0903      0.045     -2.003      0.045      -0.179      -0.002
y.L4          -0.0882      0.045     -1.947      0.052      -0.177       0.001
y.L5           0.0406      0.045      0.894      0.372      -0.048       0.130
y.L6           0.0287      0.045      0.635      0.526      -0.060       0.117
y.L7          -0.0319      0.045     -0.708      0.479      -0.120       0.057
y.L8          -0.0871      0.045     -1.929      0.054      -0.176       0.001
y.L9          -0.0058      0.045     -0.127      0.899      -0.094       0.083
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.2163           -0.5437j            1.3323           -0.0669
AR.2            1.2163           +0.5437j            1.3323            0.0669
AR.3            0.5383           -1.1338j            1.2551           -0.1794
AR.4            0.5383           +1.1338j            1.2551            0.1794
AR.5           -1.3244           -0.5728j            1.4430           -0.4350
AR.6           -1.3244           +0.5728j            1.4430            0.4350
AR.7           -0.6288           -1.2763j            1.4228           -0.3229
AR.8           -0.6288           +1.2763j            1.4228            0.3229
AR.9          -14.7478           -0.0000j           14.7478           -0.5000
-----------------------------------------------------------------------------
```

6.

```
data_acf = acf(ar_model.resid, lag=10)
title = 'ACF AR(3)'

fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(T)), y=data_acf, mode='markers',
    name="sin",
    line=dict(color="blue", width=1),
    connectgaps=True,
))
fig.update_layout(title=title)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

fit AR ACF 0

2.2 Moving Average (MA)

AR order p order

$$r_t = \phi_0 + a_t + \sum_{i=1}^{\infty} \phi_i r_{t-i}$$

$$\phi_i = -\theta_1^i$$

$$r_t = \phi_0 + a_t + \sum_{i=1}^{\infty} -\theta_1^i r_{t-i}$$

$|\theta_i| < 1$ $r_t - \theta_1 r_{t-1}$

$$r_t = \phi_0(1 - \theta_1) + a_t - \theta_1 a_{t-1}$$

MA(1) shock a_t shock MA

$$r_t = c_0 + a_t + \sum_{i=1}^{q} \theta_i a_{t-i}$$

- c_0
- a_i shock ( )

MA shocks MA shocks r_t **shocks**

Note:

a_t

2.2.1 MA(q)

shocks 0 MA(q)

$$\mathbb{E}(r_t) = c_0$$

2.2.2 MA(q)

a_i a_j $(i \neq j)$

$$\text{Var}(r_t) = (1 + \theta_1^2 + \theta_2^2 + \cdots + \theta_q^2)\sigma_a^2$$

MA **MA**

2.2.3 MA(q)

MA(1) ACF MA(q) $c_0 = 0$

$$r_{t-\ell}r_t \ = \ r_{t-\ell}a_t \ + \ \theta_1 r_{t-\ell}a_{t-1}$$

$\text{Var}(r_t) = (1 + \theta_1^2)\sigma_a^2$

$$\rho_0 = 1, \quad \rho_1 = \frac{\theta_1}{1+\theta_1^2}, \quad \rho_\ell = 0, \quad \text{for} \quad \ell > 1$$

0 MA(1) Lag-1 0 MA(q) ACF $\rho_e ll = 0$ $\ell > q$
ACF lag-q MA(q)

Note:

MA(q) > q MA

References