lab1

September 18, 2020

1 DS-GA 3001.009 Modeling Time Series Data

2 Lab 1: ACF, CCF and ARMA

functions and packages needed

[1]:
```
# Install statsmodels
# conda install -c conda-forge statsmodels
import statsmodels
from statsmodels.tsa.stattools import acf, ccf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.graphics import utils
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from statsmodels.graphics.api import qqplot
```

[2]:
```
# statsmodels.graphics.tsaplots doesn't have plotting function for CCF so I␣
↪have to write my own.
def plot_ccf(x, y, ax=None, lags=None, alpha=.05, use_vlines=True,␣
↪unbiased=False,
             fft=False, title='Cross-correlation', zero=True, **kwargs):
    fig, ax = utils.create_mpl_ax(ax)
    lags, nlags, irregular = statsmodels.graphics.tsaplots.
↪_prepare_data_corr_plot(x, lags, zero)
    confint = None
    ccf_val = ccf(x, y)
    if lags is not None:
        ccf_val = ccf_val[:nlags+1]
    # statsmodels.graphics.tsaplots._plot_corr(ax, title, ccf_val, confint,␣
↪lags, irregular, use_vlines, **kwargs)
    # Depending on your version of statsmodels, you may have to use the␣
↪following instead:
    statsmodels.graphics.tsaplots._plot_corr(ax, title, ccf_val, confint, lags,␣
↪irregular, use_vlines, vlines_kwargs=kwargs)
    return fig
```

2.1 Part I: Autocorrelation Function

2.1.1 A) implement ACF

Do your own implementation of the ACF function. Your implementation will be checked against statsmodels.tsa.stattools.acf.

[3]:
```
def acf_impl(x, nlags):
    """
    TODO
    @param x: a 1-d numpy array (data)
    @param nlags: an integer indicating how far back to compute the ACF
    @return a 1-d numpy array with (nlags+1) elements.
            Where the first element denotes the acf at lag = 0 (1.0 by␣
↪definition).
    """
    #TODO: replace the template code with your code here. This part will be␣
↪graded.

    ans = []
    n = len(x)
    x_mean = np.mean(x)
    x_var = np.var(x)
    for i in range(1, nlags + 1):
        x1, x2 = x[i:], x[:-i]
        ans.append(np.sum((x1 - x_mean) * (x2 - x_mean) / (n * x_var)))

    return np.array([1] + ans)
```

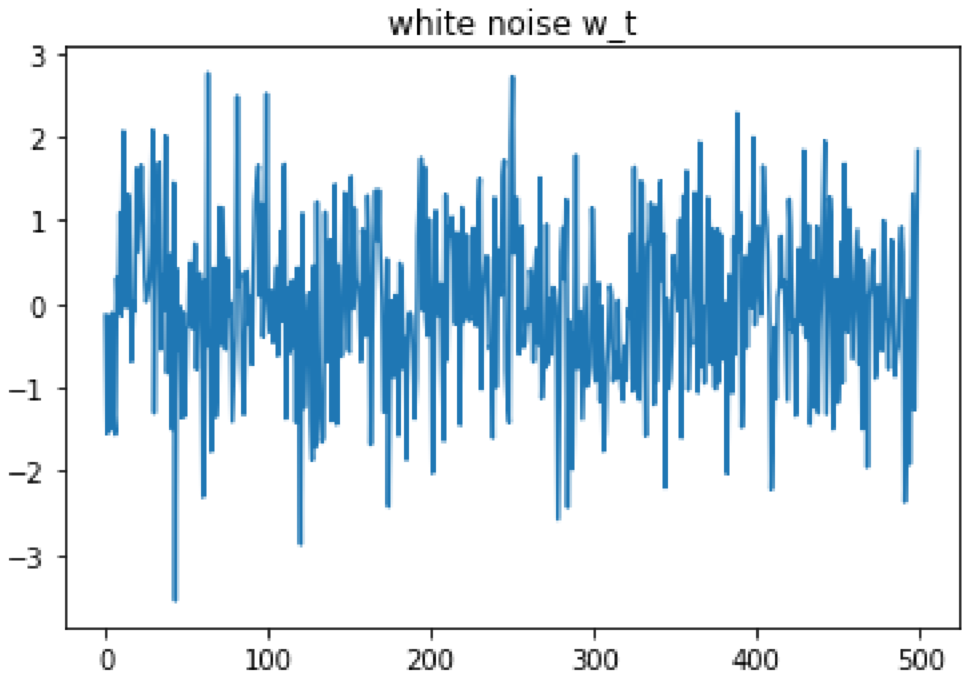
2.1.2 B) ACF of White Noise

$$w_t \sim N(0, \sigma^2)$$

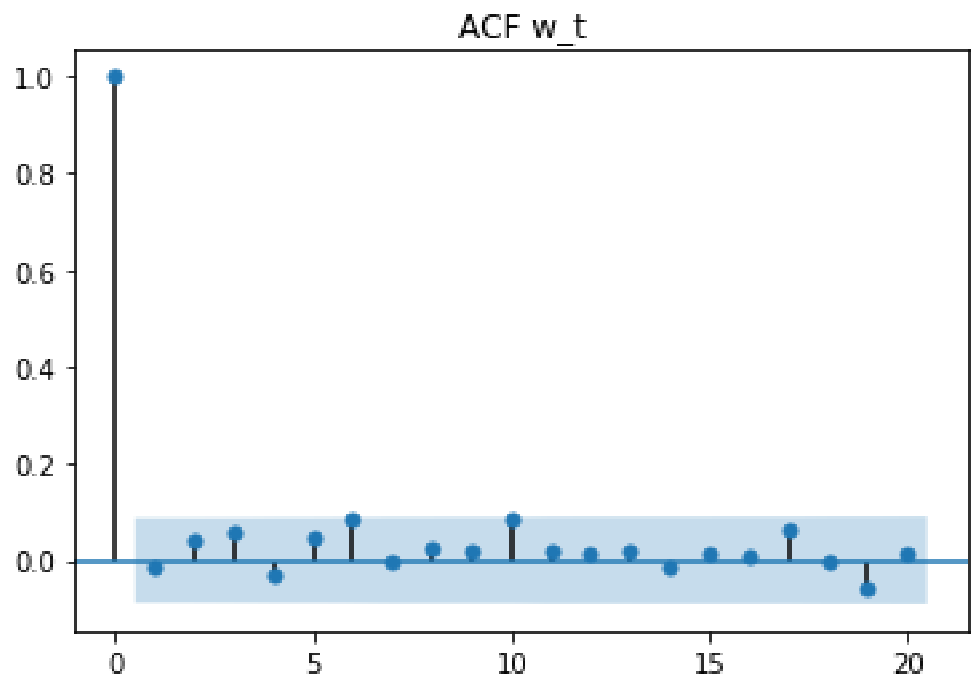
- Set $\sigma = 1$, sample n = 500 points from the process above
- Plot the white noise
- Plot the sample ACF up to lag = 20.
- Calculate the analytical ACF and compare it with the sample ACF.
- What trend/observation can you find in the ACF plot?
- Change n to 50, compare the new ACF plot (n=50) to the old ACF plot (n=500). What causes the difference?

[4]:
```
n = 500
mean = 0
std = 1
lag = 20

# create white noise
w_t = np.random.normal(mean, std, size=n)
```

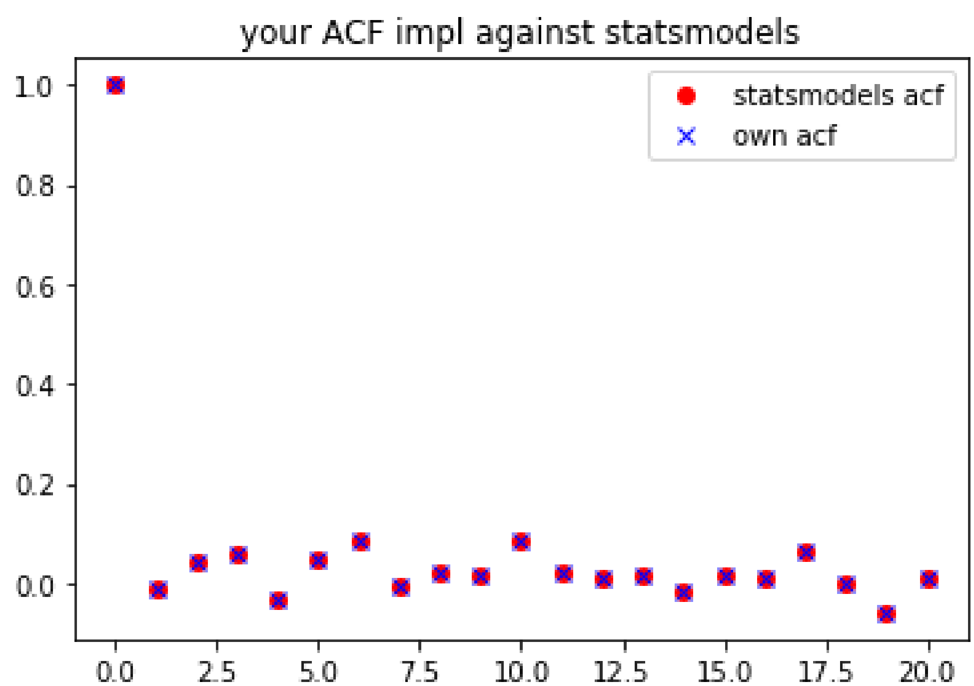
```python
# plot white noise
plt.plot(w_t)
plt.title("white noise w_t")
plt.show()

# calculate acf
acf_val = acf(x=w_t, nlags=lag)
plot_acf(x=w_t, lags=lag, title="ACF w_t")
plt.show()

# your implementation:
acf_val_impl = acf_impl(x=w_t, nlags=lag)
plt.figure()
plt.plot(acf_val, 'or', label='statsmodels acf')
plt.plot(acf_val_impl, 'xb', label='own acf')
plt.legend();
plt.title('your ACF impl against statsmodels')
```




```
C:\Users\zhouy\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:572:
FutureWarning: fft=True will become the default in a future version of
statsmodels. To suppress this warning, explicitly set fft=False.
  FutureWarning
```



[4]: Text(0.5, 1.0, 'your ACF impl against statsmodels')



2.1.3 C) ACF of Moving Average

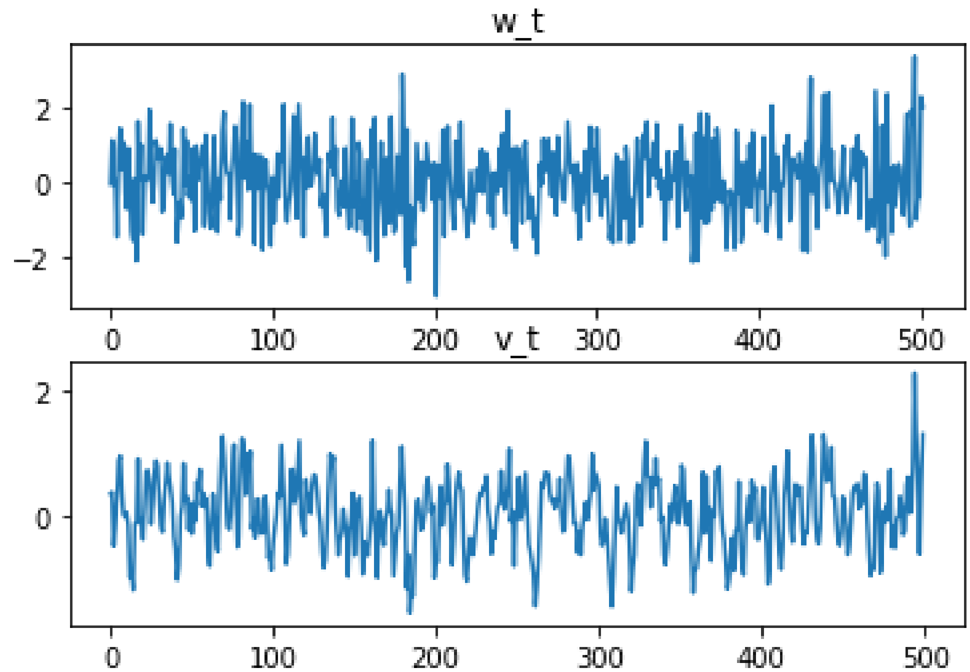
$$v_t = \frac{1}{3}(w_t + w_{t+1} + w_{t+2})$$

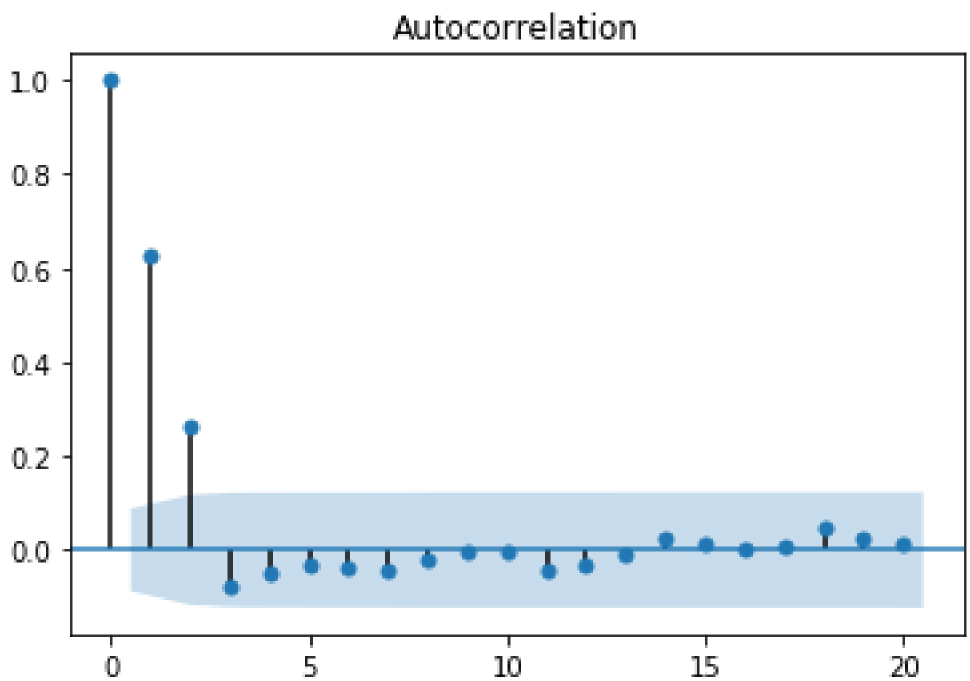
- Sample n+2 white noise from N(0,1)
- Add code to compute the moving average v_t.
- Plot both w_t and v_t and compare the two time series.
- Derive the analytical ACF
- Plot the sample/empirical ACF of v_t and compare it with the analytical ACF.

[5]:
```
n = 500
mean = 0
std = 1
lag = 20

# create white noise
w_t = np.random.normal(mean, std, size=n+2)
# create moving average
#TODO: replace the template code with your code here. This part will be graded.
v_t = np.convolve(w_t, np.ones(3) / 3, mode = 'valid')

# plot white noise
plt.figure(1)
plt.subplot(211)
plt.plot(w_t)
plt.title("w_t")

# plot moving average
plt.subplot(212)
plt.plot(v_t)
plt.title("v_t")

# calculate acf
acf_val = acf(x=v_t, nlags=lag)
plot_acf(x=v_t, lags=lag)
plt.show()

# your implementation:
acf_val_impl = acf_impl(x=v_t, nlags=lag)
plt.figure()
plt.plot(acf_val, 'or', label='statsmodels acf')
plt.plot(acf_val_impl, 'xb', label='own acf')
plt.legend();
plt.title('your ACF impl against statsmodels')
```

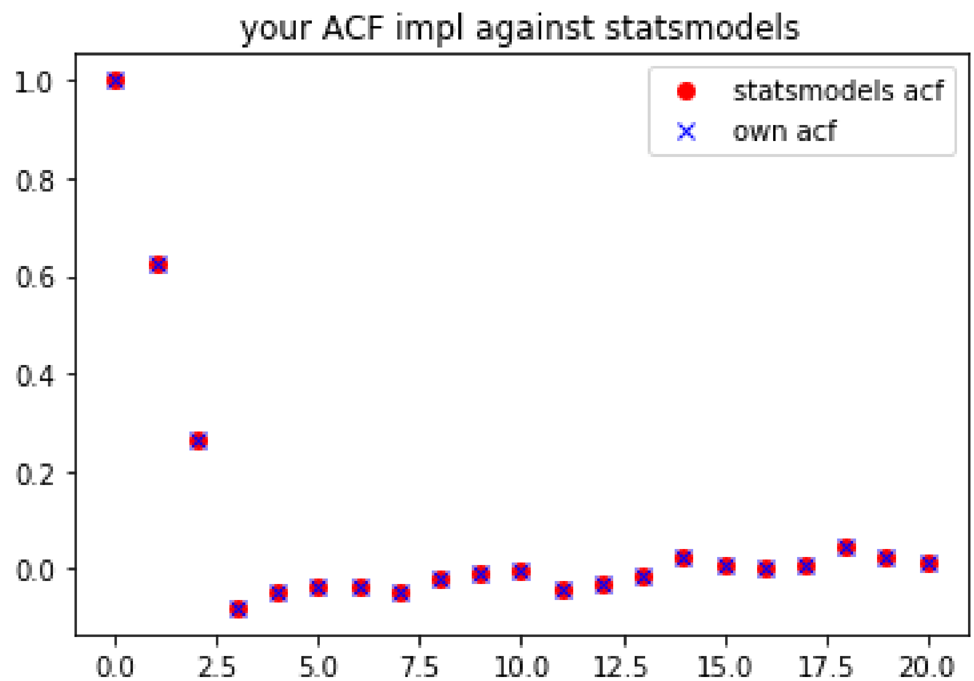
```
C:\Users\zhouy\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:572:
FutureWarning: fft=True will become the default in a future version of
statsmodels. To suppress this warning, explicitly set fft=False.
```

FutureWarning





[5]: Text(0.5, 1.0, 'your ACF impl against statsmodels')



2.1.4 D) ACF of signal in noise

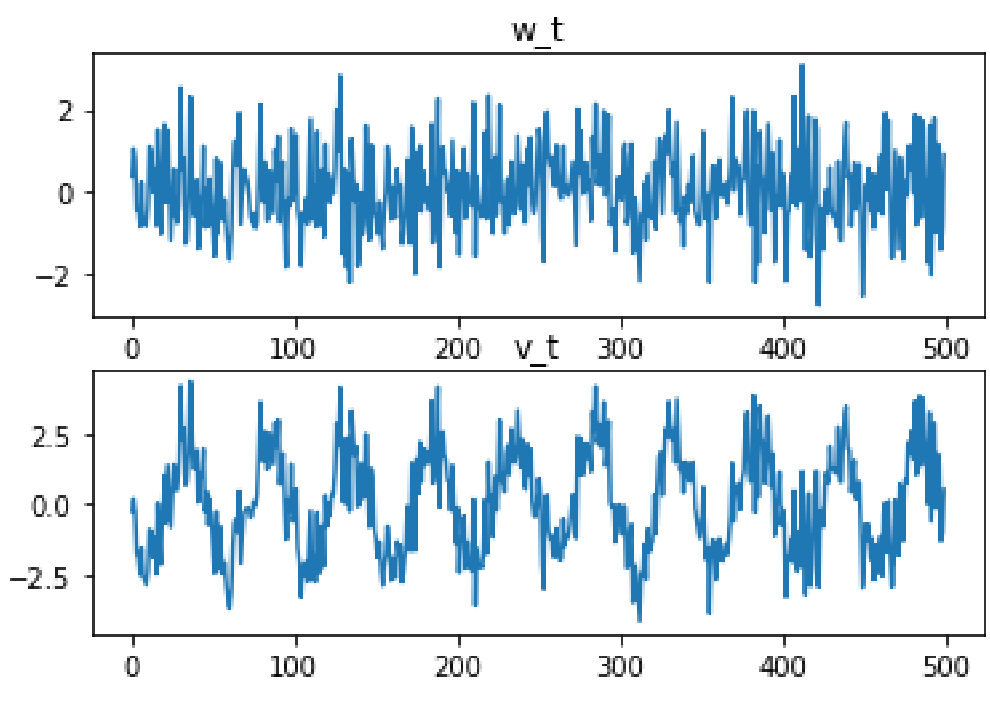
$$v_t = 2cos(\frac{2\pi t}{50} + 0.6\pi) + w_t$$

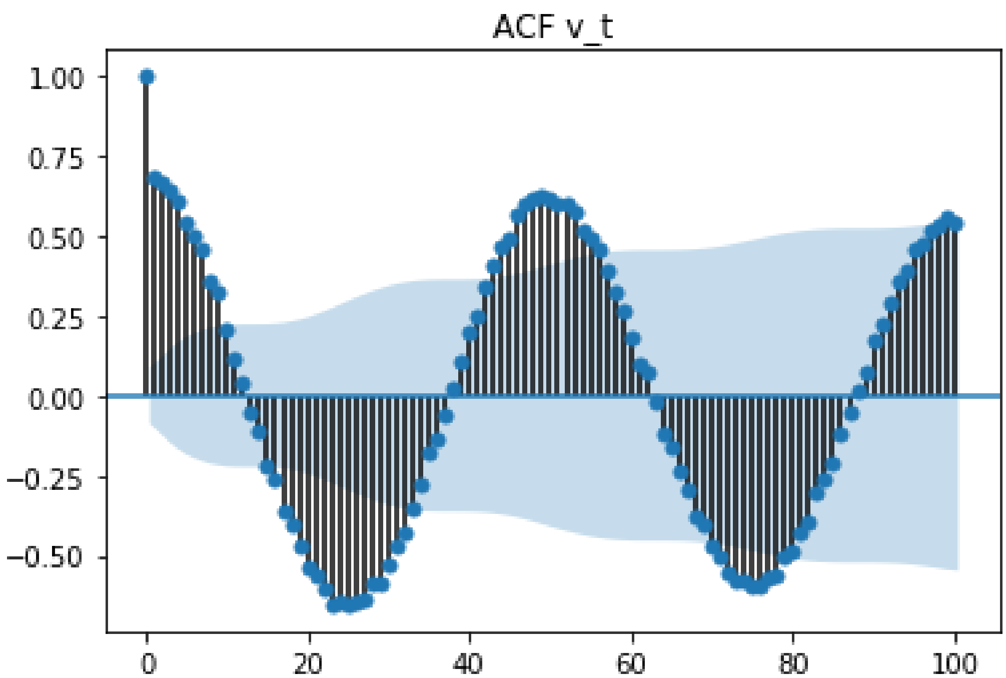
- Sample white noise of length n from $N(0, 1)$
- Add code to compute v_t.
- Plot both w_t and v_t. Compare the two plots.
- Plot the sample ACF of v_t. What's the pattern? What causes the observed pattern?

[6]:
```
n = 500
mean = 0
std = 1
lag = 100

# create white noise
w_t = np.random.normal(mean, std, size=n)
# create signal w. noise
#TODO: replace the template code with your code here. This part will be graded.
t = np.arange(n)
v_t = 2 * np.cos((2 * np.pi * t / 50) + 0.6 * np.pi) + w_t


# plot white noise
```

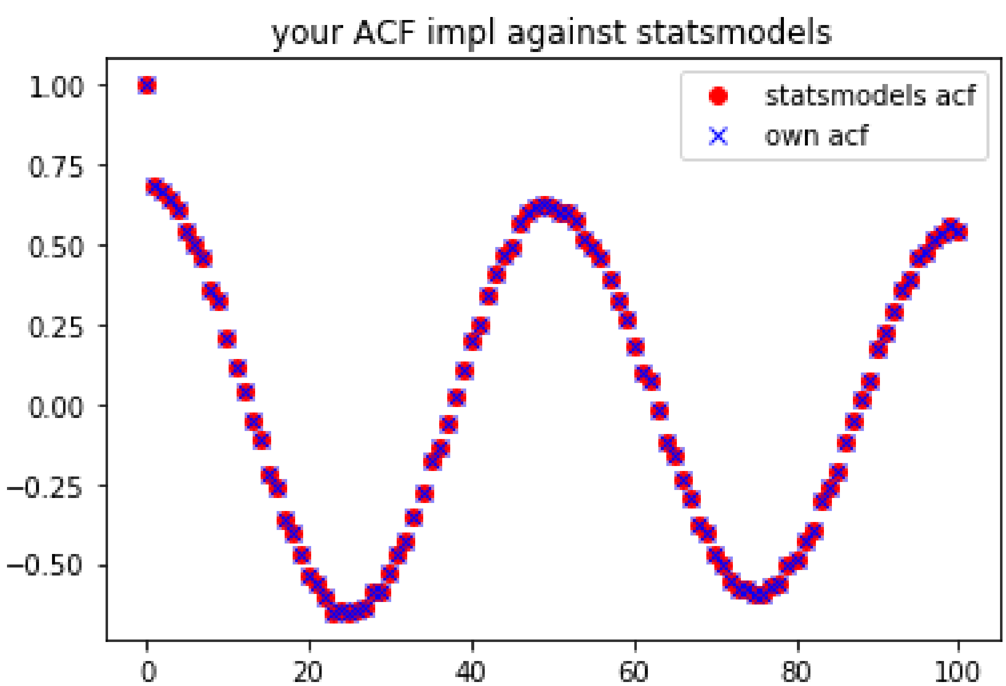
```
plt.figure(1)
plt.subplot(211)
plt.plot(w_t)
plt.title("w_t")
# plot signal with noise
plt.subplot(212)
plt.plot(v_t)
plt.title("v_t")

# plot acf
acf_val = acf(x=v_t, nlags=lag)
plot_acf(x=v_t, lags=lag, title="ACF v_t")
plt.show()

# your implementation:
acf_val_impl = acf_impl(x=v_t, nlags=lag)
plt.figure()
plt.plot(acf_val, 'or', label='statsmodels acf')
plt.plot(acf_val_impl, 'xb', label='own acf')
plt.legend();
plt.title('your ACF impl against statsmodels')
```

```
C:\Users\zhouy\anaconda3\lib\site-packages\statsmodels\tsa\stattools.py:572:
FutureWarning: fft=True will become the default in a future version of
statsmodels. To suppress this warning, explicitly set fft=False.
  FutureWarning
```





[6]: Text(0.5, 1.0, 'your ACF impl against statsmodels')



2.2 Part II: Cross-correlation Function

2.2.1 A) CCF of signal with noise

Synthetic Data

$$x_t \sim N(0, \sigma_x^2)$$

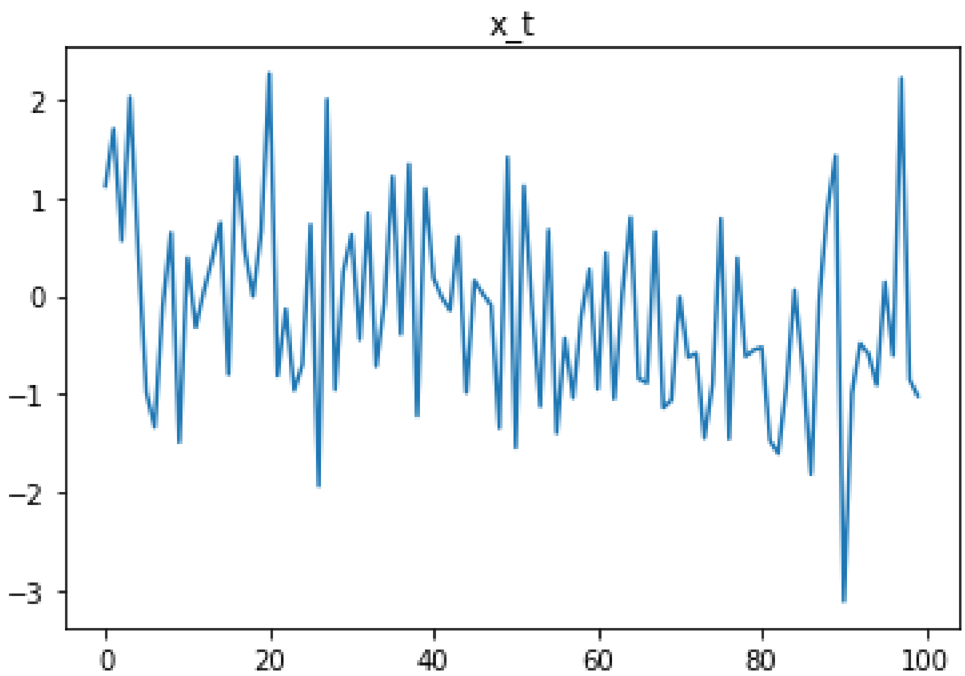
$$y_t = 2x_{t-5} + w_t$$

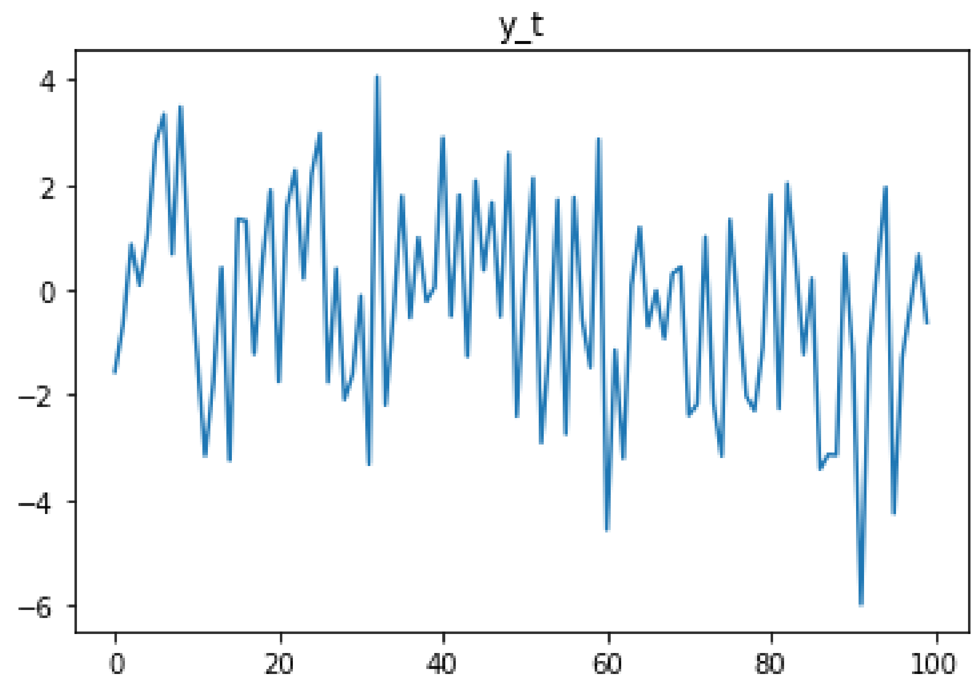
$$w_t \sim N(0, \sigma_x^2)$$

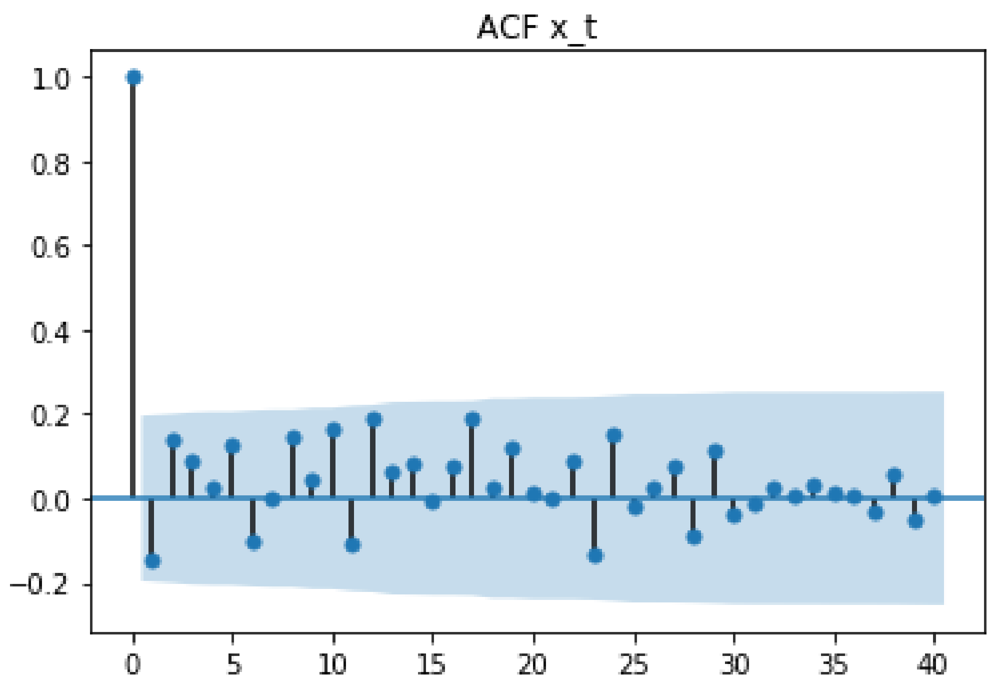
- In this example, we created two processes with a lag of 5.
- Plot both samples and verify the lag.
- Plot the empirical ACF for both samples.
- Plot the empirical CCF. What information can you conclude from the CCF plot?

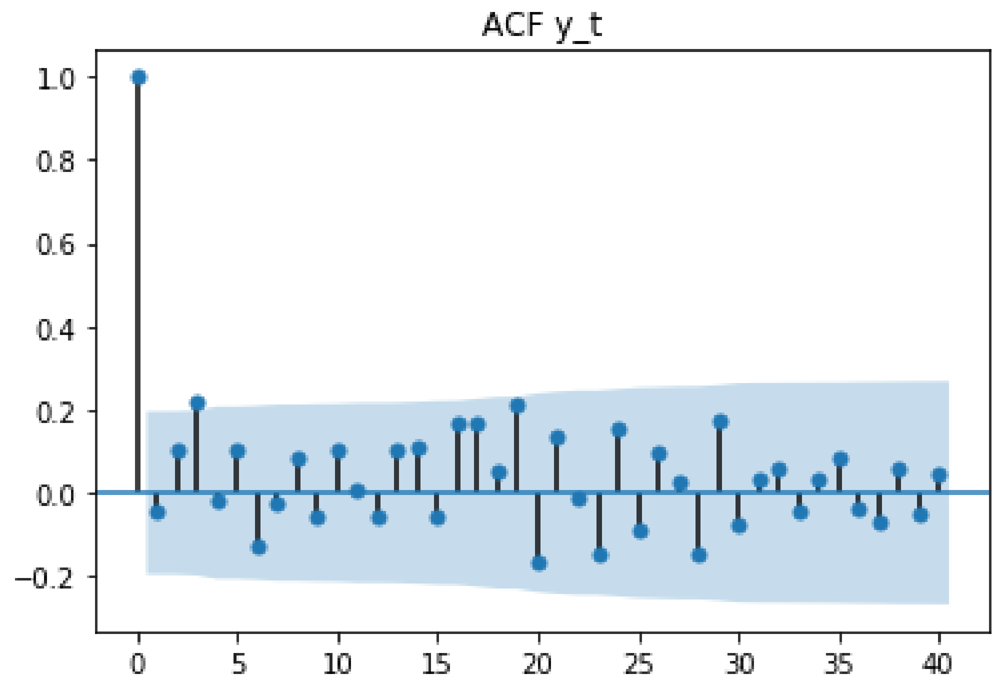
[7]:
```
# Cross-correlation synthetic Example
n = 100
mean = 0
std = 1
lag = 40
true_h = 5

x_t = np.random.normal(mean, std, size=n+5)
w_t = np.random.normal(mean, std, size=n+5)
#TODO: replace the template code with your code here. This part will be graded.
y_t = list(w_t[:5])
for i in range(5, n):
    y_t.append(2 * x_t[i - 5] + w_t[i])

y_t = np.array(y_t)
x_t = x_t[:n]

# plot the original data
plt.plot(x_t)
plt.title("x_t")
plt.show()
plt.plot(y_t)
plt.title("y_t")
plt.show()

# plot acf
plot_acf(x=x_t, lags=lag, title="ACF x_t")
plot_acf(x=y_t, lags=lag, title="ACF y_t")
```

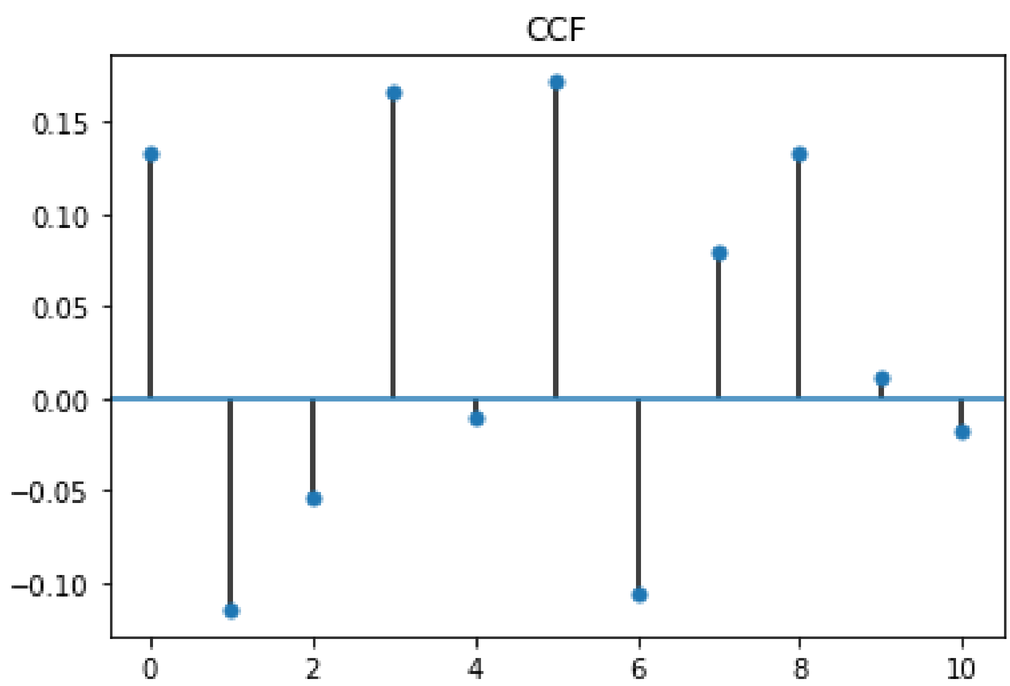
```
plt.show()

# plot ccf
ccf_val = ccf(y_t, x_t)
plot_ccf(x_t, y_t, title="CCF", lags=10)
plt.show()
```




y_t


ACF x_t


ACF y_t
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
25
30
35
40

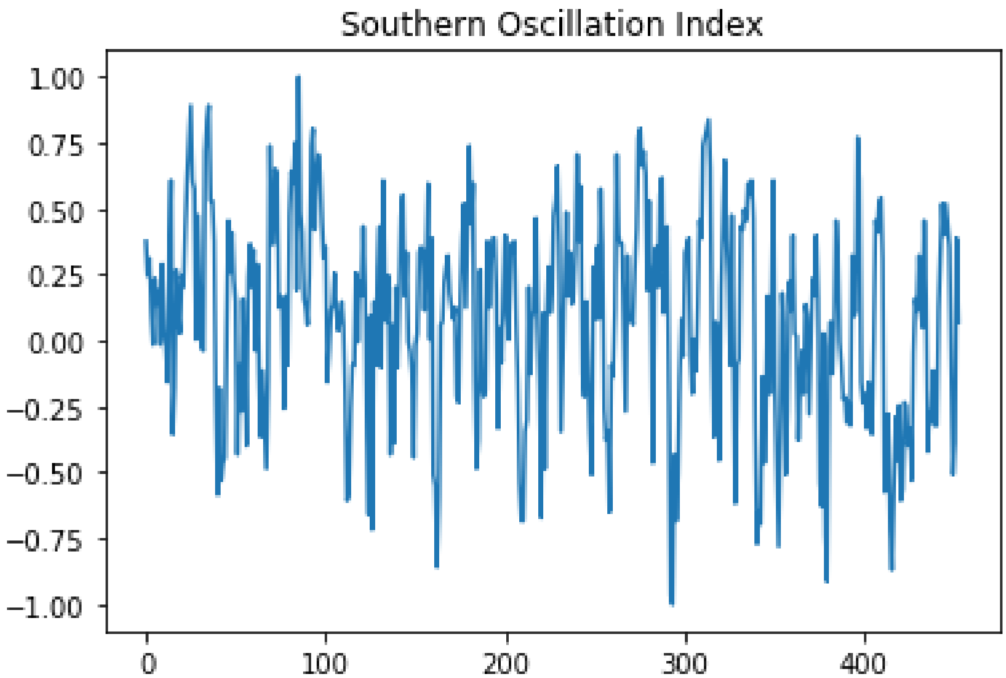

CCF
0.15
0.10
0.05
0.00
−0.05
−0.10
0
2
4
6
8
10

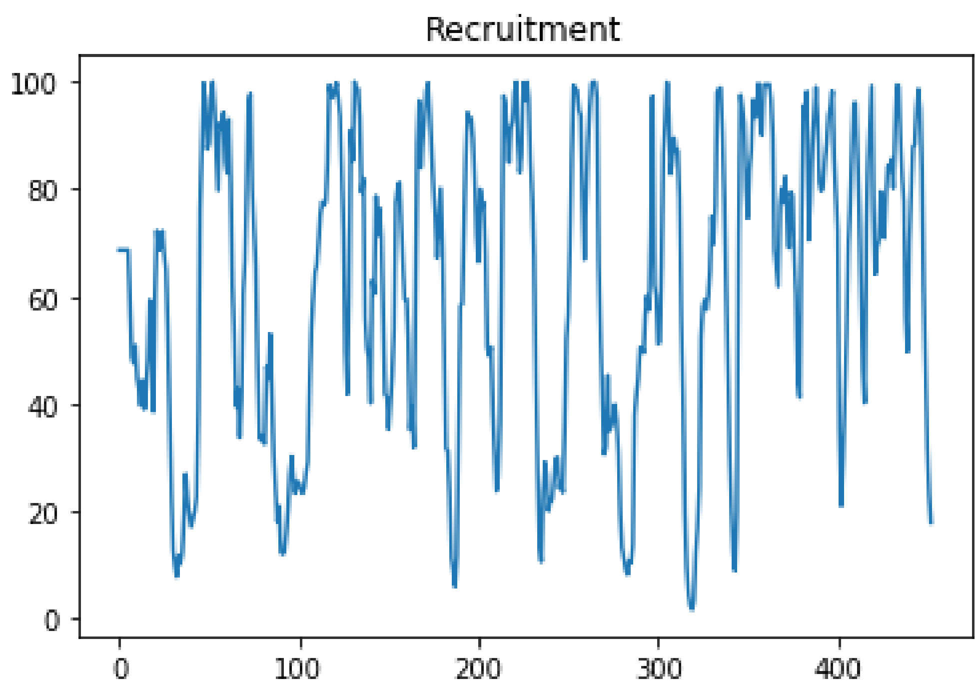
2.2.2 B) CCF of data

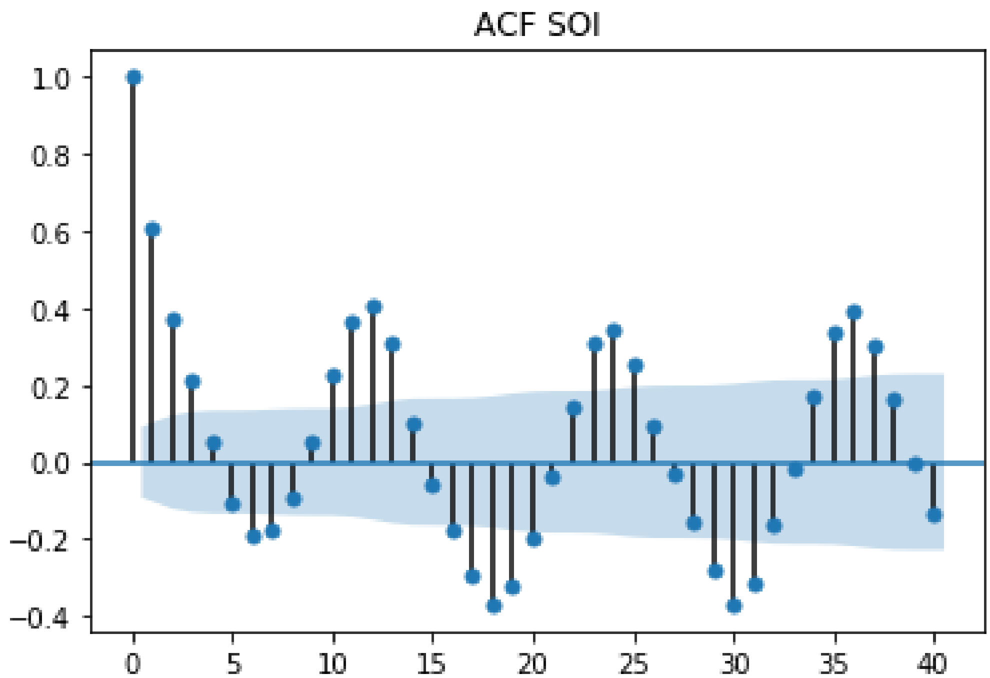
Southern Oscillation Index (SOI) v.s. Recruitment (Rec)

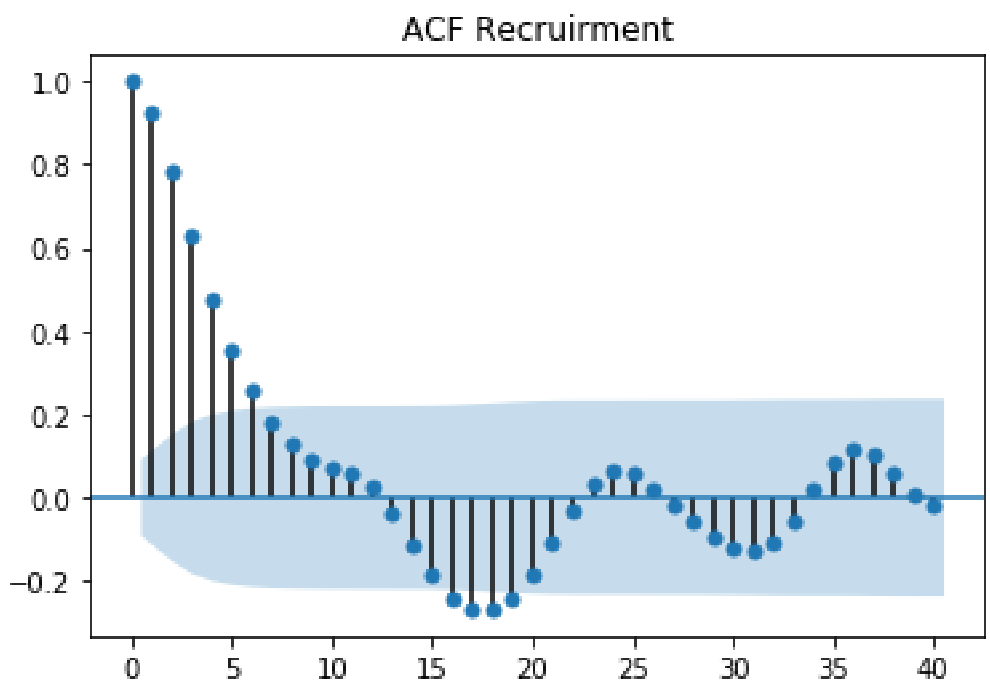
- Replicate the procedure in the previous section.
- What information can you tell from the CCF plot.
- In this example, our procedure is actually flawed. Unlike the previous example, we can not tell if the cross-correlation estimate is significantly different from zero by looking at the CCF. Why is that? What can we do to address this issue?

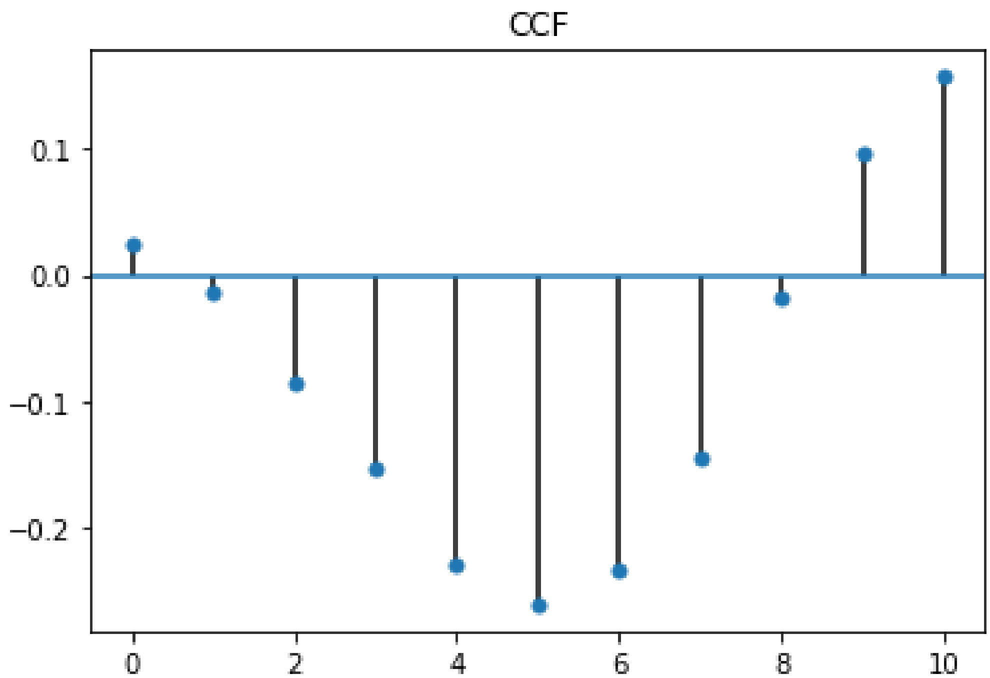
[8]:
```
soi = np.array(pd.read_csv("../../data/soi.csv")["x"])
rec = np.array(pd.read_csv("../../data/rec.csv")["x"])
#TODO: This part will be graded.
# plot data
plt.plot(soi)
plt.title("Southern Oscillation Index")
plt.show()
plt.plot(rec)
plt.title("Recruitment")
plt.show()

# plot acf
plot_acf(x=soi, lags=lag, title="ACF SOI")
plot_acf(x=rec, lags=lag, title="ACF Recruirment")
plt.show()

# plot ccf
ccf_val = ccf(rec, soi)
plot_ccf(soi, rec, title="CCF", lags=10)
plt.show()
```


Southern Oscillation Index
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
100
200
300
400


Recruitment
100
80
60
40
20
0
0
100
200
300
400


ACF SOI
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0
5
10
15
20
25
30
35
40


ACF Recruirment
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20
25
30
35
40



2.3 Part III: AR models

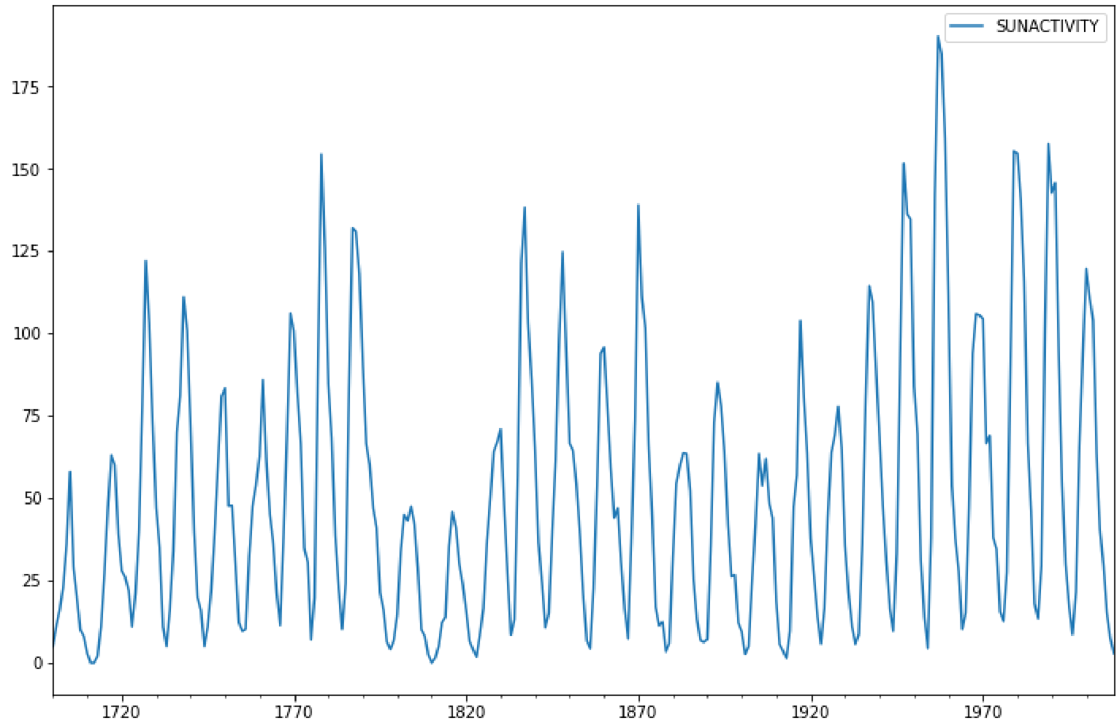
In this example, we will fit an AR(p) model to the SunActivity data, which denotes the number of sunspots for each year.

We will determine p, fit the model, compute the roots and the lag 0 to p components of the ACF.

Wikipedia for sunspots: https://en.wikipedia.org/wiki/Sunspot

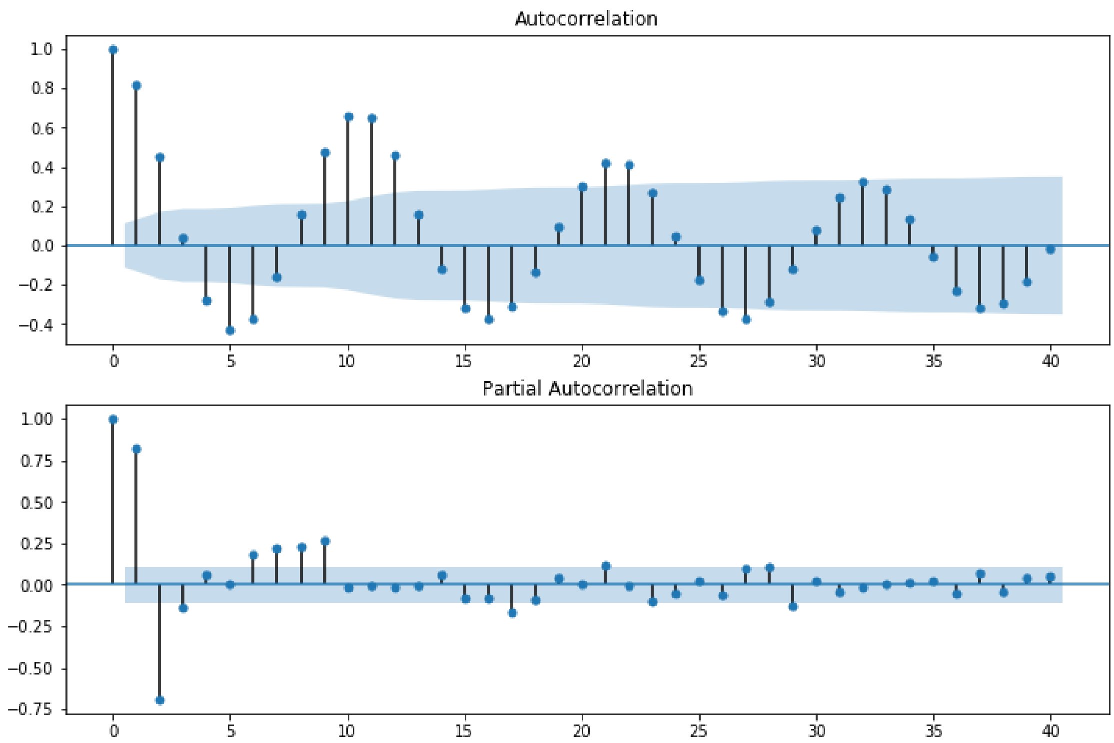
The code in this section is selected from the tutorial specified in the reference section.

[9]:
```
dta = sm.datasets.sunspots.load_pandas().data
dta.index = pd.Index(sm.tsa.datetools.dates_from_range('1700', '2008'))
del dta["YEAR"]
dta.plot(figsize=(12,8))
plt.show()
```



ACF & PACF

[10]:
```
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(dta.values.squeeze(), lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(dta, lags=40, ax=ax2)
plt.show()
```



Fit AR Model of order p

[11]:
```python
# TODO: chose p appropriately
p = 3

arma_mod = sm.tsa.ARMA(dta, (p,0)).fit(disp=False)
print(arma_mod.params)

# TODO: predict ACF of model at lag 0, 1, ..., p
phi_1, phi_2, phi_3 = arma_mod.params[1:]
rho_0 = 1
rho_1 = (phi_2 * phi_3 + phi_1) / (1 - phi_2 - phi_3 * (phi_1 + phi_3))
rho_2 = phi_2 + (phi_1 + phi_3) * rho_1
rho = [rho_0, rho_1, rho_2]

for i in range(3, p + 1):
    new_rho = phi_1 * rho[i - 1] + phi_2 * rho[i - 2] + phi_3 * rho[i - 3]
    rho.append(new_rho)
rho = np.array(rho)

# TODO: compute roots
roots = np.zeros(2)
```

```
print('roots: ', roots)
```

```
const                 49.749917
ar.L1.SUNACTIVITY      1.300810
ar.L2.SUNACTIVITY     -0.508093
ar.L3.SUNACTIVITY     -0.129649
dtype: float64
roots:  [0. 0.]
```

```
C:\Users\zhouy\anaconda3\lib\site-
packages\statsmodels\tsa\base\tsa_model.py:162: ValueWarning: No frequency
information was provided, so inferred frequency A-DEC will be used.
  % freq, ValueWarning)
```

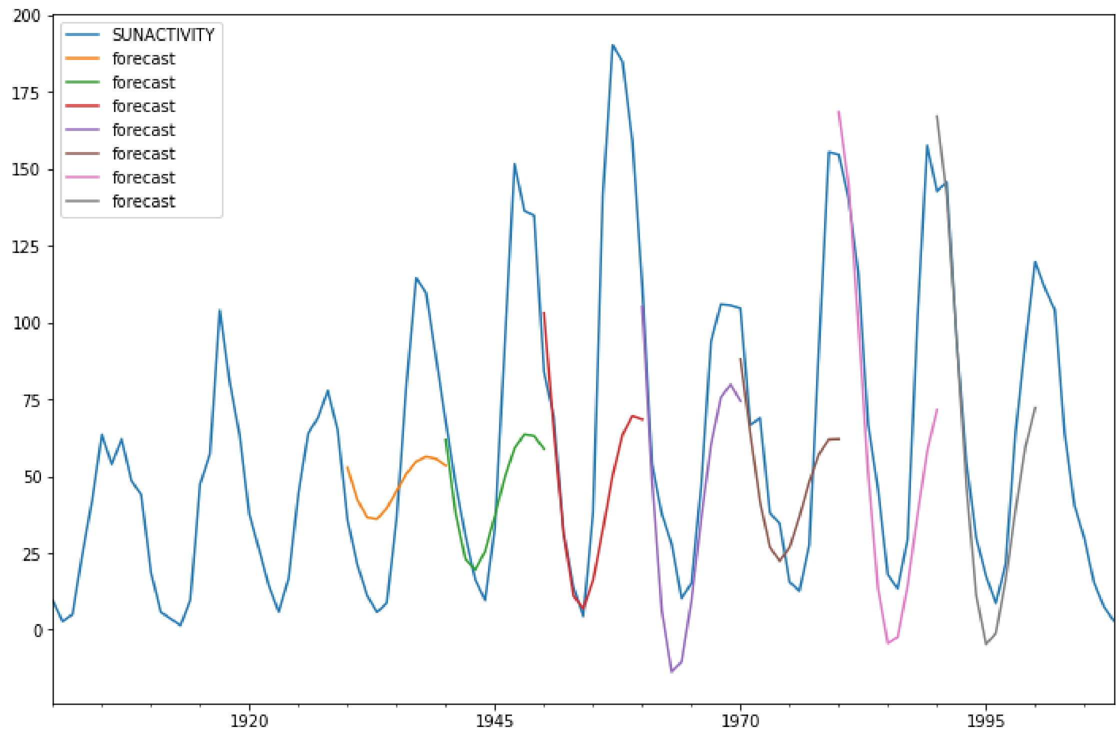
prediction

[12]:
```
predict_sunspots = arma_mod.predict('1990', '2012', dynamic=True)
print(predict_sunspots)
```

```
1990-12-31    167.047414
1991-12-31    140.992989
1992-12-31     94.859089
1993-12-31     46.860866
1994-12-31     11.242545
1995-12-31     -4.721333
1996-12-31     -1.166945
1997-12-31     16.185668
1998-12-31     39.021870
1999-12-31     59.449866
2000-12-31     72.170139
2001-12-31     75.376777
2002-12-31     70.436445
2003-12-31     60.731564
2004-12-31     50.201767
2005-12-31     42.075994
2006-12-31     38.114254
2007-12-31     38.454614
2008-12-31     41.963791
2009-12-31     46.869267
2010-12-31     51.423244
2011-12-31     54.399702
2012-12-31     55.321673
Freq: A-DEC, dtype: float64
```

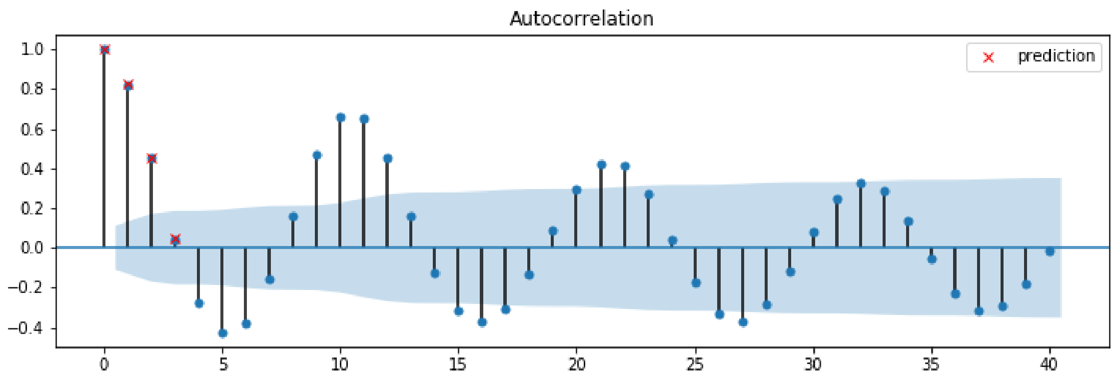
[13]:
```
# TODO: try to predict further into the future by increasing tsteps
tsteps=10

fig, ax = plt.subplots(figsize=(12, 8))
```

```
ax = dta.loc['1900':].plot(ax=ax)
T = np.arange(1930, 2010, tsteps)
for tt in range(len(T)-1):
    fig = arma_mod.plot_predict(np.str(T[tt]), np.str(T[tt+1]), dynamic=True,␣
 ↪ax=ax, plot_insample=False)
plt.show()
```



plot ACF and PACF

[14]:
```
predict_sunspots = arma_mod.predict('1950', '2012', dynamic=True)

fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(dta.values.squeeze(), lags=40, ax=ax1)
ax1.plot(np.arange(p+1), rho, 'xr', label='prediction')
ax1.legend()
plt.show()
```



2.4 Part IV

2.4.1 Moving Average

$$x_t = 0.5x_{t-1} - 0.5w_{t-1} + w_t$$

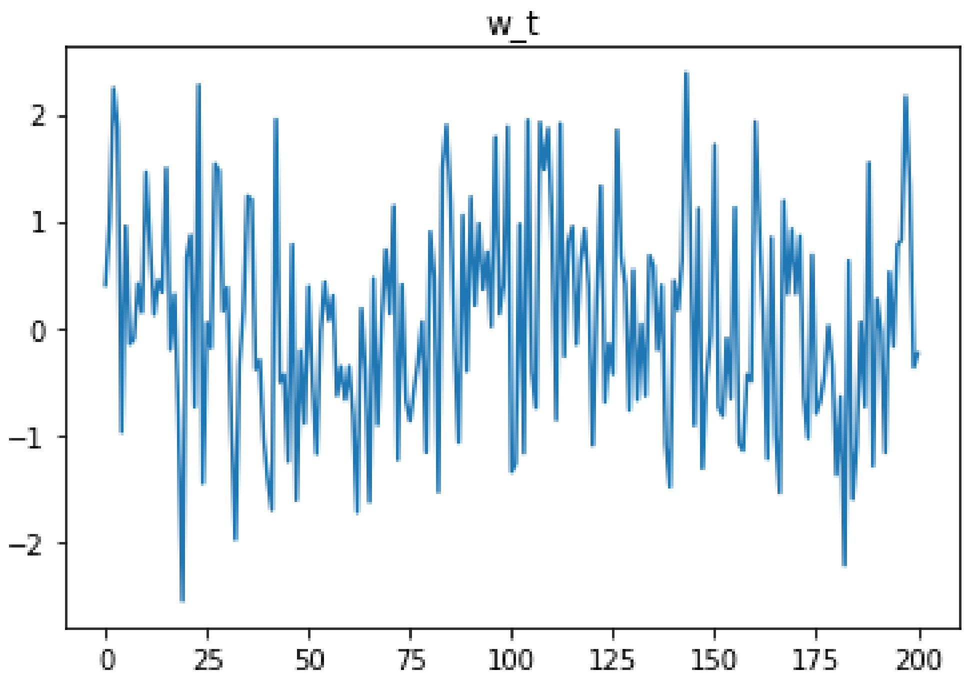
$$w_t \sim N(0, \sigma^2)$$

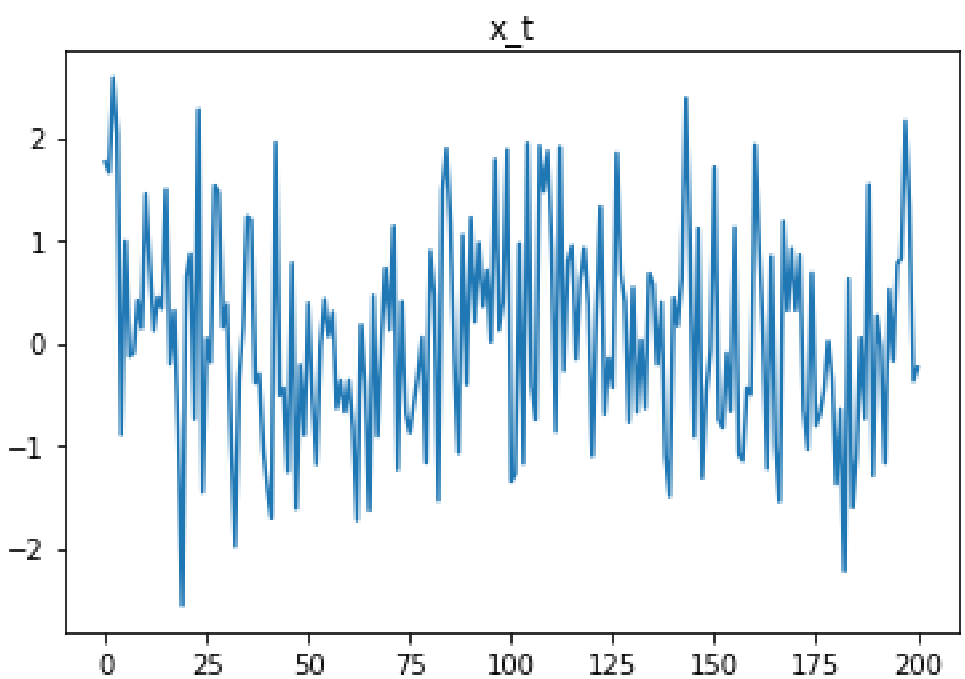
Is x_t same as white noise w_t? Think about ACF.

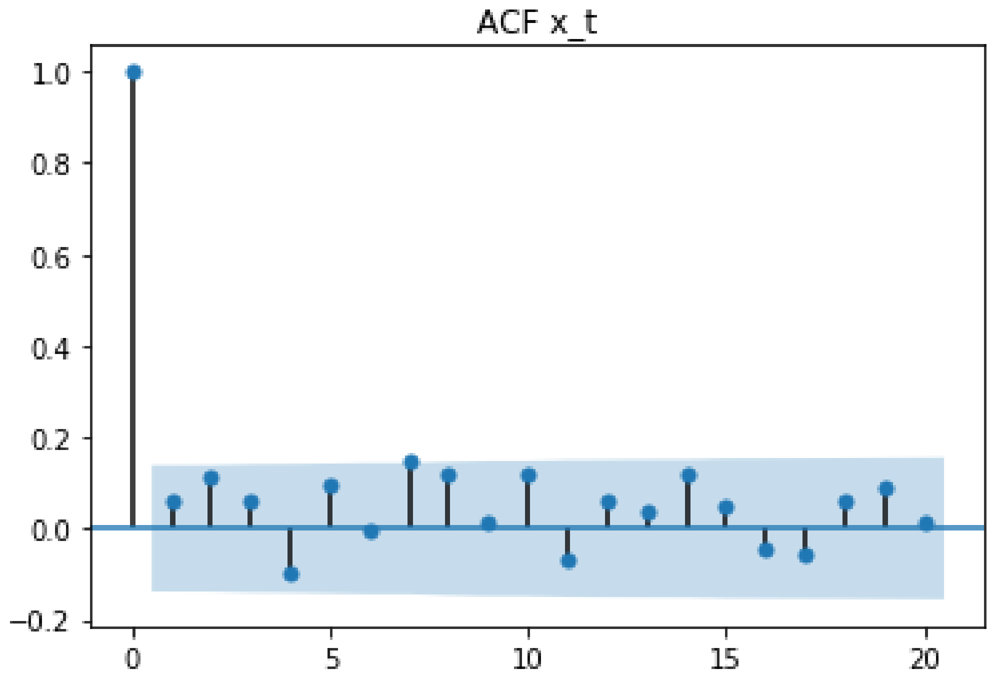
Then use code below to assess and verify your guess.

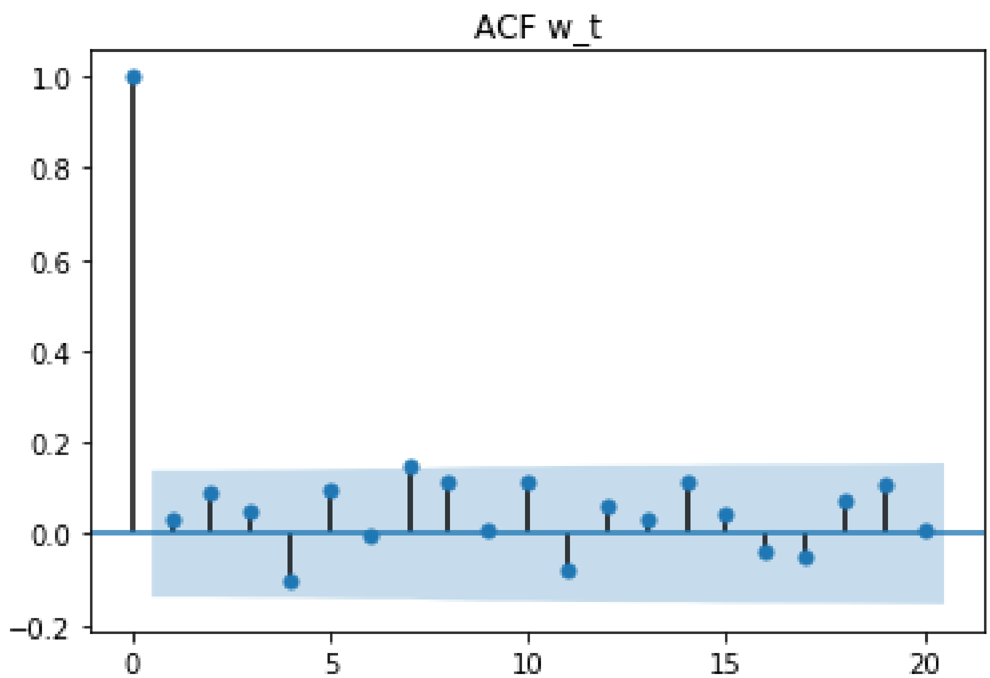
[15]:
```
n = 200
mean = 0
std = 1
lag = 20

# create white noise
np.random.seed(0)
x_t = list(np.random.normal(mean, std, size=1))
w_t = np.random.normal(mean, std, size=n+1)
for i in range(1, n+1):
    x_t.append(0.5 * x_t[i-1] - 0.5 * w_t[i-1] + w_t[i] )

# plot x_t & w_t
plt.plot(w_t)
plt.title("w_t")
plt.show()
plt.plot(x_t)
plt.title("x_t")
plt.show()

# acf & pacf
plot_acf(x=x_t, lags=lag, title="ACF x_t")
```

```
plot_acf(x=w_t, lags=lag, title="ACF w_t")
plt.show()
```









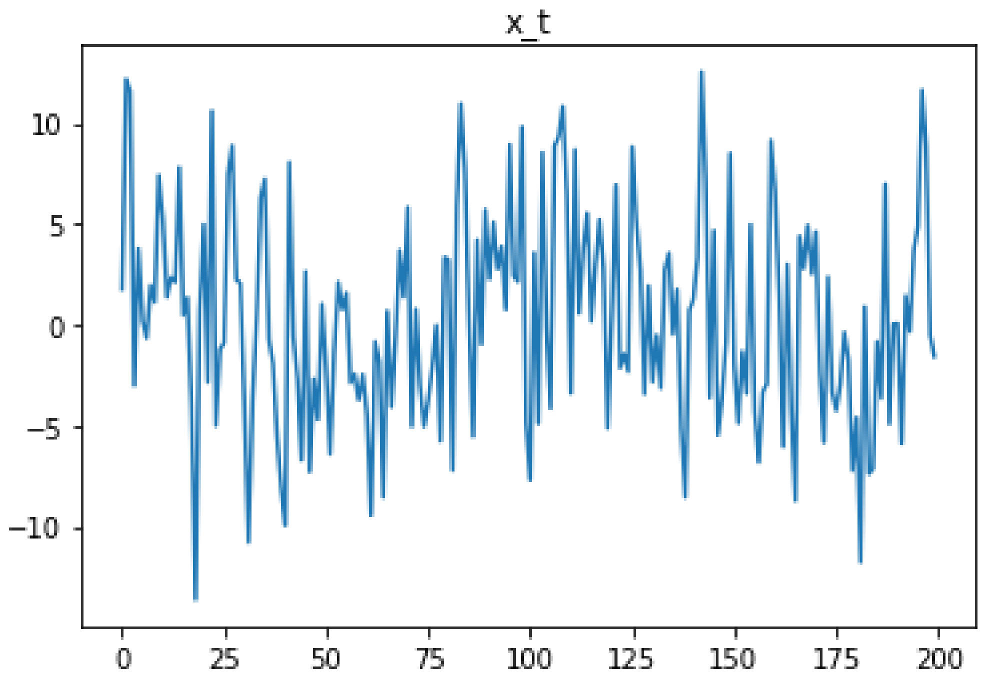
$$x_t = w_t + \frac{1}{5}w_{t-1}, w_t \sim N(0, 25)$$

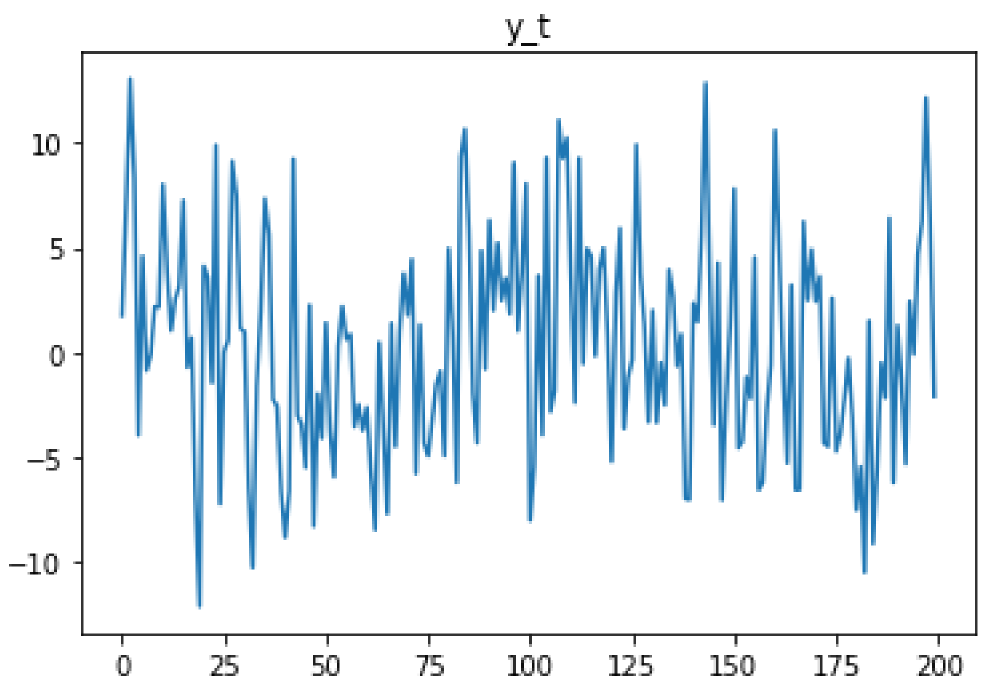
$$y_t = v_t + 5v_{t-1}, v_t \sim N(0, 1)$$

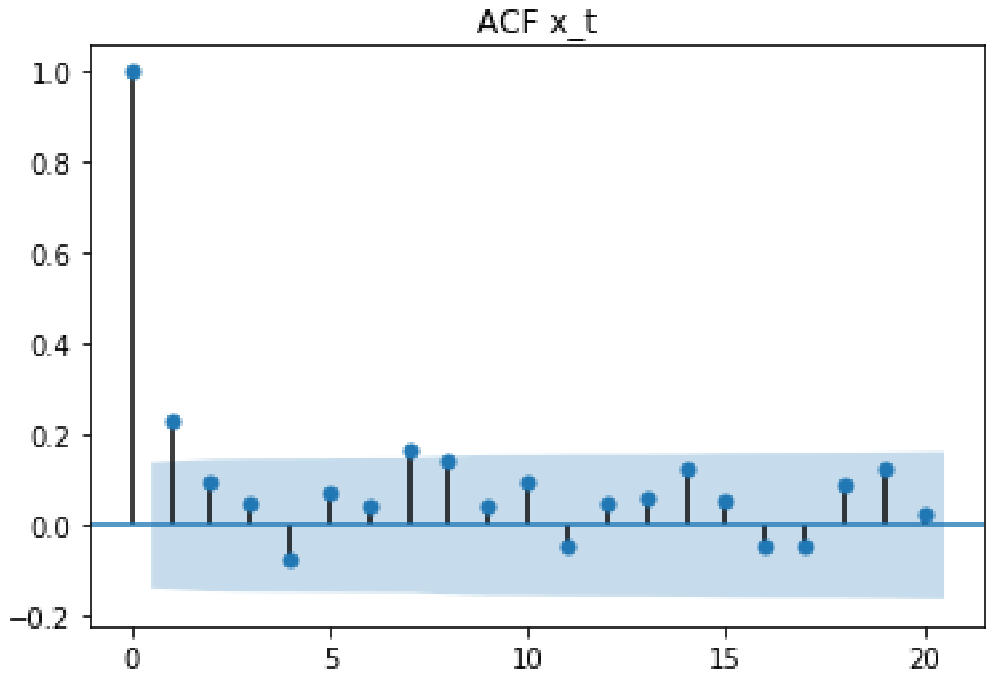
Are x_t and y_t the same? Think about ACF.

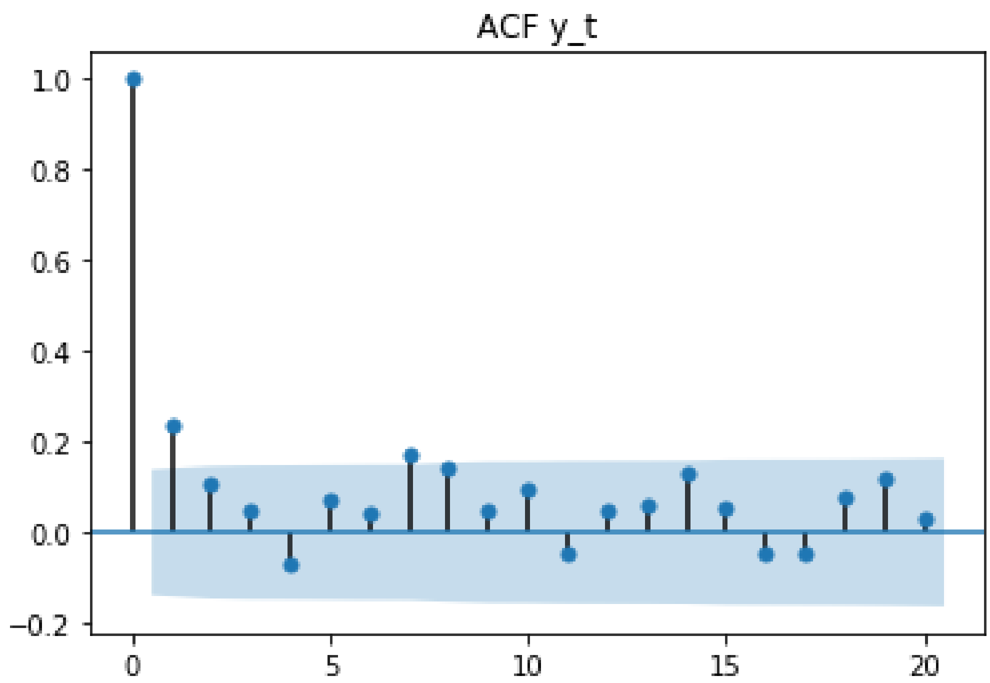
Then use code below to assess and verify your guess.

[16]:
```
n = 200
mean = 0
lag = 20

# create white noise
np.random.seed(0)
x_t = list(np.random.normal(mean, std, size=1))
w_t = np.random.normal(mean, 5, size=n+1)
np.random.seed(0)
y_t = list(np.random.normal(mean, std, size=1))
v_t = np.random.normal(mean, 1, size=n+2)
for i in range(2, n+1):
    x_t.append(w_t[i] + 0.2 * w_t[i-1])
    y_t.append(v_t[i] + 5 * v_t[i-1])

# plot x_t & y_t
plt.plot(x_t)
plt.title("x_t")
plt.show()
plt.plot(y_t)
plt.title("y_t")
plt.show()

# acf & pacf
plot_acf(x=x_t, lags=lag, title="ACF x_t")
plot_acf(x=y_t, lags=lag, title="ACF y_t")
plt.show()
```


x_t
10
5
0
−5
−10
0
25
50
75
100
125
150
175
200


y_t
10
5
0
−5
−10
0
25
50
75
100
125
150
175
200


ACF x_t
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20


ACF y_t
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
5
10
15
20

2.4.2 Please turn in the code before 09/22/2020 11:59 pm.

2.4.3 Your work will be evaluated based on the code and plots. You don't need to write down your answers to these questions in the text blocks. Everything that will be graded is indicated by the "TODO".