DS-GA 3001.009 Modeling Time Series Data

Lab 1: ACF, CCF and ARMA

functions and packages needed

In [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
```

In [2]:
```
import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)
```

In [3]:
```
# statsmodels.graphics.tsaplots doesn't have plotting function for CCF s
o I have to write my own.
def plot_ccf(x, y, ax=None, lags=None, alpha=.05, use_vlines=True, unbia
sed=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, confi
nt, 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
```

Part I: Autocorrelation Function

A) implement ACF

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

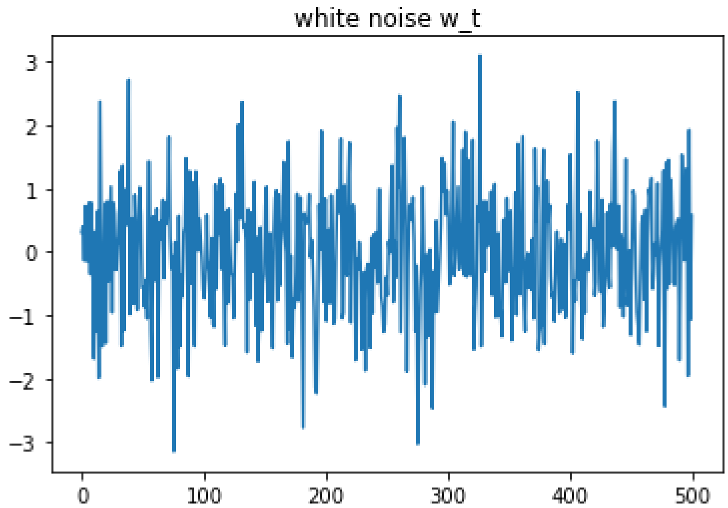
In [4]:
```
def acf_impl(x, nlags):
    """
    TODO
    @param x: a 1-d numpy array (data)
    @param nlags: an integer indicating how far back to compute the ACF
    @return a 1-d numpy array with (nlags+1) elements.
            Where the first element denotes the acf at lag = 0 (1.0 by d
efinition).
    """
    #TODO: replace the template code with your code here. This part will
be graded.

    mean_x = x.mean()
    acfs = []

    acf_0 = 0
    for i in range(0, len(x)):
        acf_0 = acf_0 + ((x[i] - mean_x) * (x[i] - mean_x))
    acf_0 = acf_0 / len(x)

    for j in range(nlags+1):
        total = 0
        for i in range(0, len(x)-j):
            total = total + ((x[i+j] - mean_x) * (x[i] - mean_x))
        total = total / len(x)
        total = total / acf_0
        acfs.append(total)

    return acfs
```

B) ACF of White Noise

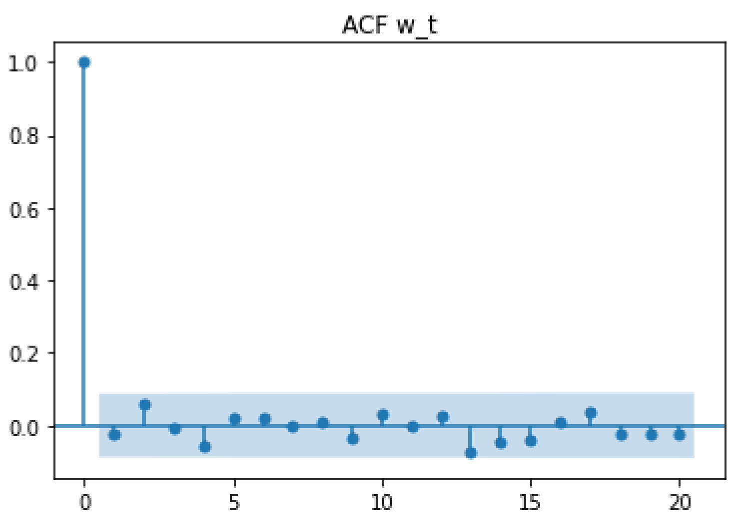
$$w_t \sim N(0, \sigma^2)$$

- Set σ = 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?

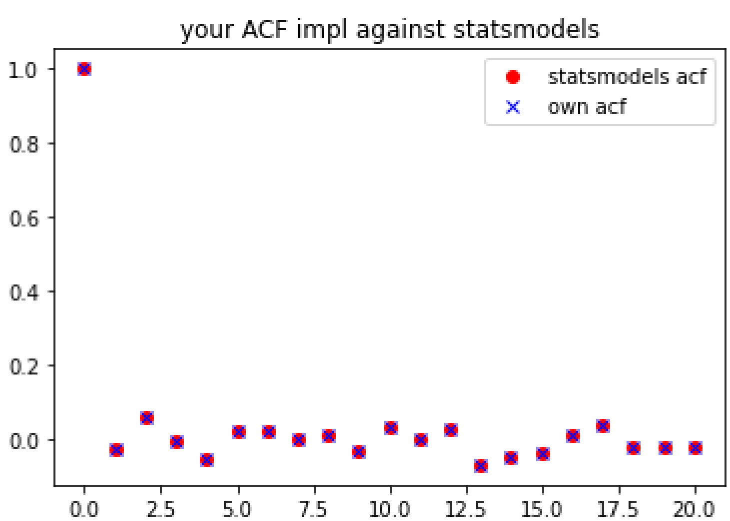
In [5]:
```
n = 500
mean = 0
std = 1
lag = 20

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

# 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')
```




```
/usr/local/anaconda3/envs/pTSA/lib/python3.8/site-packages/statsmodels/
tsa/stattools.py:662: FutureWarning: fft=True will become the default a
fter the release of the 0.12 release of statsmodels. To suppress this w
arning, explicitly set fft=False.
  warnings.warn(
```



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



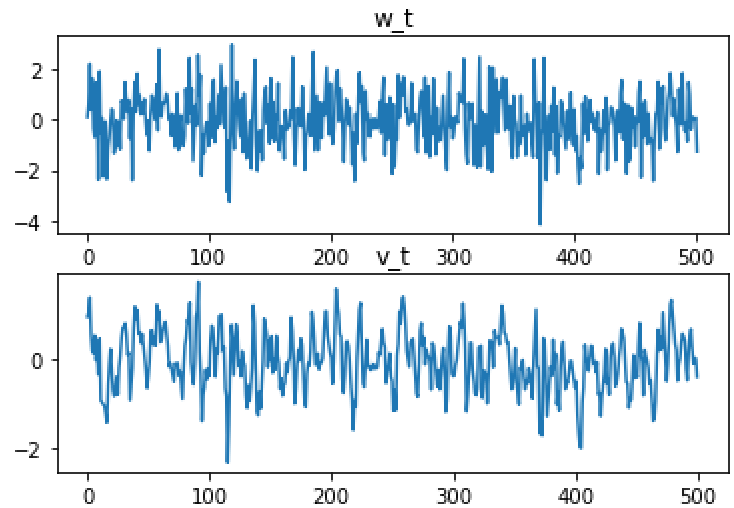
C) ACF of Moving Average

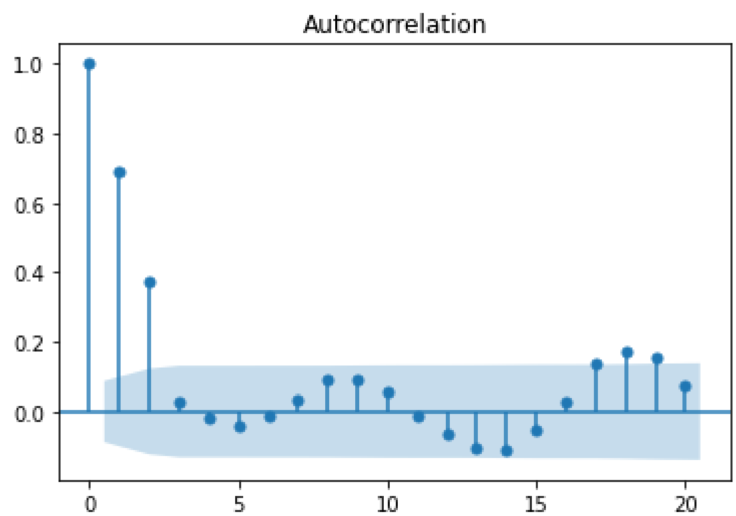
$$v_t = \frac{1}{3}(w_t + w_{t+1} + w_{t+2})$$

- 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.

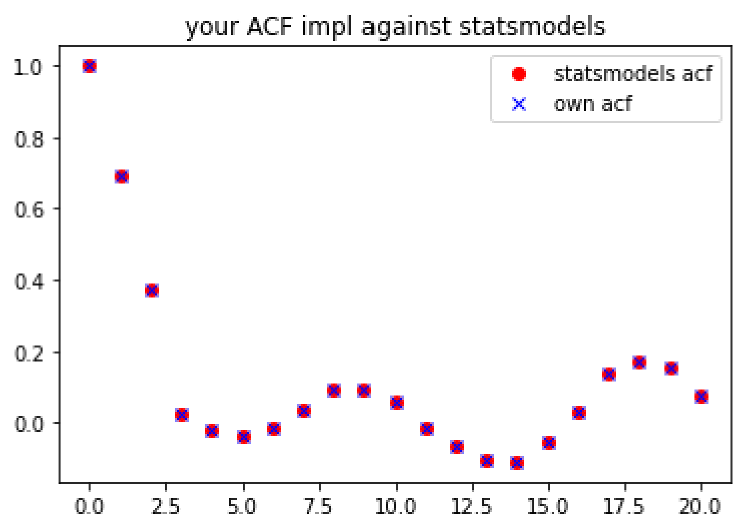
In [6]:
```
n = 500
mean = 0
std = 1
lag = 20

# 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.zeros(len(w_t))
v_t = np.array([(w_t[i] + w_t[i+1] + w_t[i+2])/3 for i in range(n)])

# 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')
```

```
/usr/local/anaconda3/envs/pTSA/lib/python3.8/site-packages/statsmodels/
tsa/stattools.py:662: FutureWarning: fft=True will become the default a
fter the release of the 0.12 release of statsmodels. To suppress this w
arning, explicitly set fft=False.
  warnings.warn(
```





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



D) ACF of signal in noise

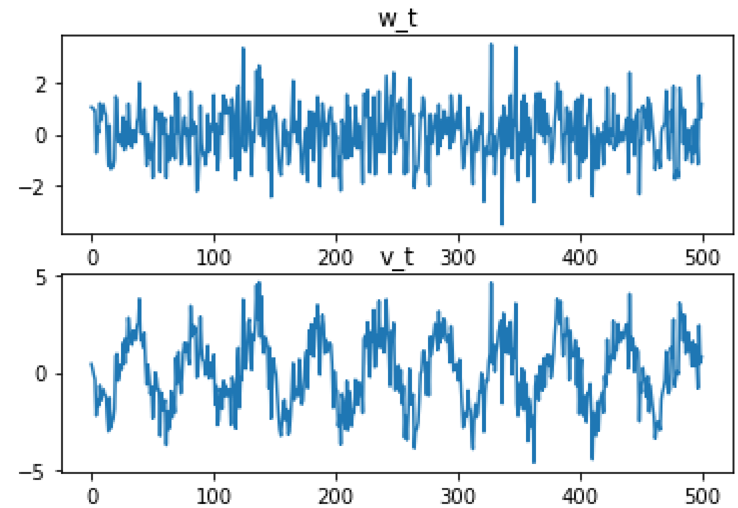
$$v_t = 2cos(\frac{2\pi t}{50} + 0.6\pi) + w_t$$

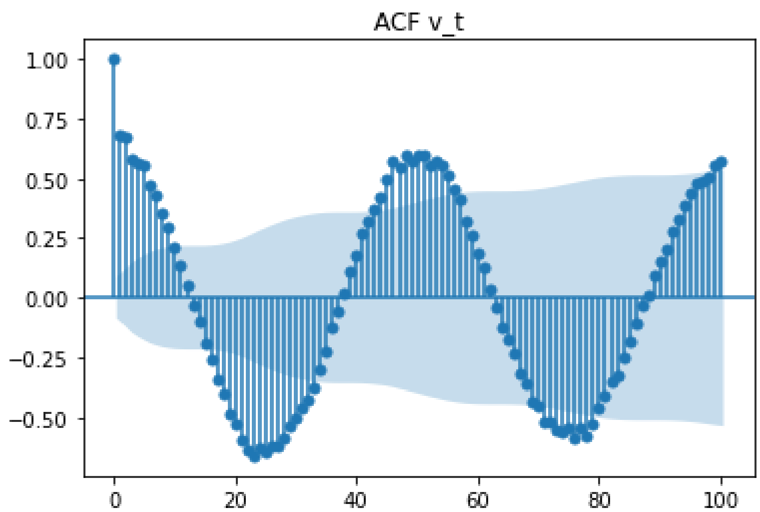
- Sample white noise of length n from $N(0, 1)$
- Add code to compute v_t.
- Plot both w_t and v_t. Compare the two plots.
- Plot the sample ACF of v_t. What's the pattern? What causes the observed pattern?

In [7]:

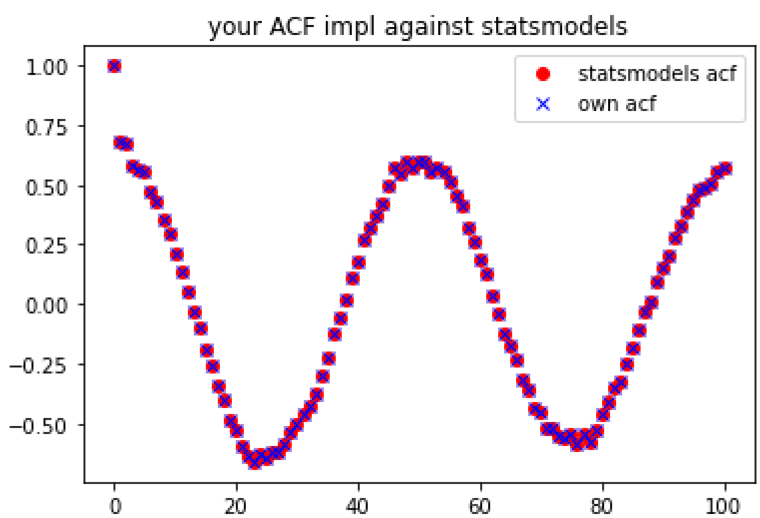
```
import math

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.
v_t = np.array([(2 * math.cos((2 * math.pi * i / 50) + (0.6 * math.pi))
+ w_t[i]) for i in range(n)])


# 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')
```

```
/usr/local/anaconda3/envs/pTSA/lib/python3.8/site-packages/statsmodels/
tsa/stattools.py:662: FutureWarning: fft=True will become the default a
fter the release of the 0.12 release of statsmodels. To suppress this w
arning, explicitly set fft=False.
  warnings.warn(
```





Out[7]: `Text(0.5, 1.0, 'your ACF impl against statsmodels')`



Part II: Cross-correlation Function

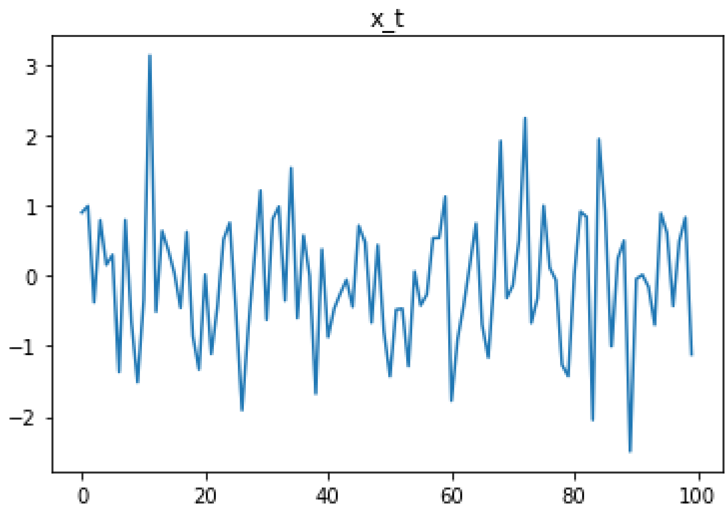
A) CCF of signal with noise

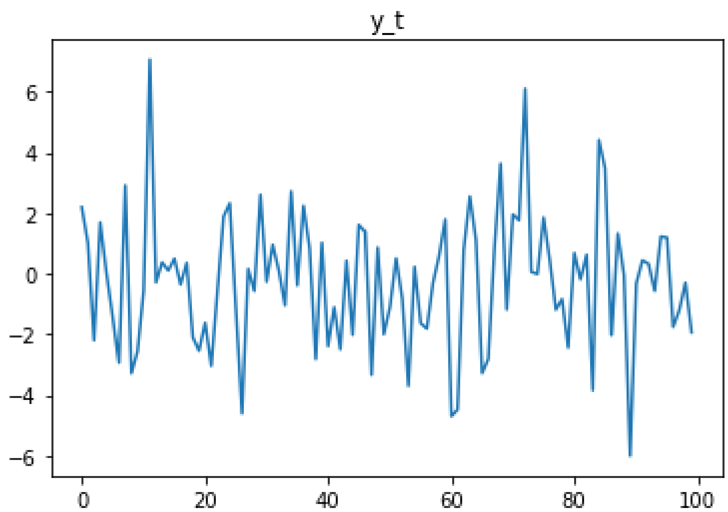
Synthetic Data

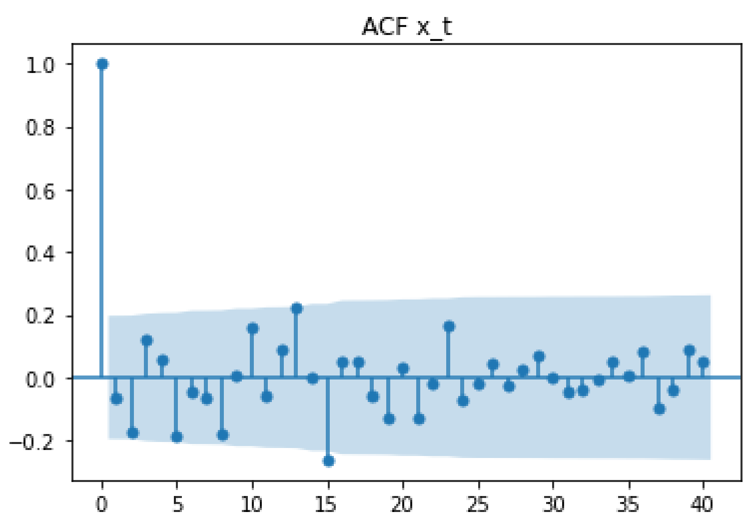
$$x_t \sim N(0, \sigma_x^2)$$
$$y_t = 2x_{t-5} + w_t$$
$$w_t \sim N(0, \sigma_x^2)$$

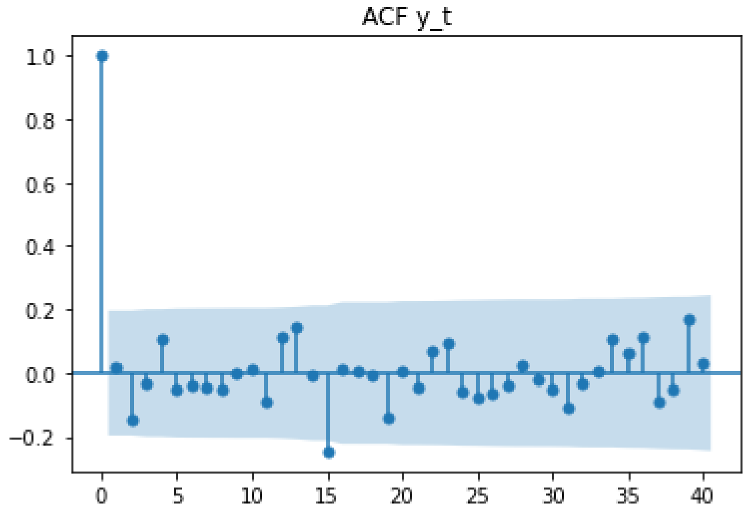
- In this example, we created two processes with a lag of 5.
- Plot both samples and verify the lag.
- Plot the empirical ACF for both samples.
- Plot the empirical CCF. What information can you conclude from the CCF plot?

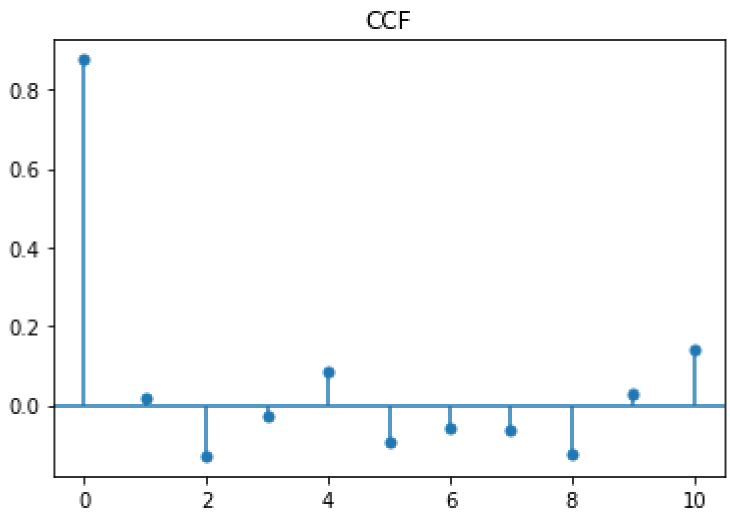
In [8]:
```
# Cross-correlation synthetic Example
n = 100
mean = 0
std = 1
lag=40
true_h = 5

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

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()
```


x_t
3
2
1
0
−1
−2
0
20
40
60
80
100


y_t
6
4
2
0
−2
−4
−6
0
20
40
60
80
100


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



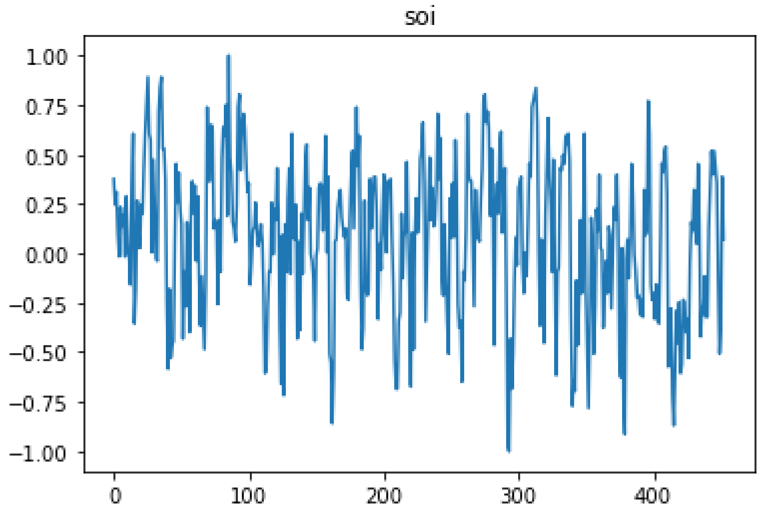


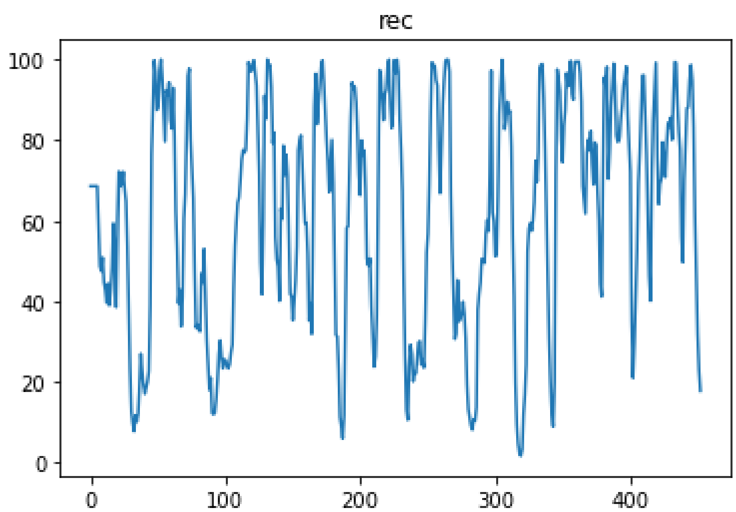
B) CCF of data

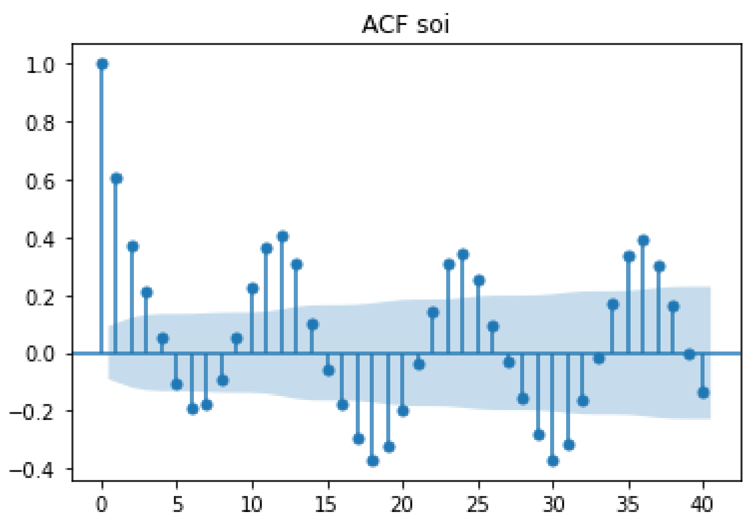
Southern Oscillation Index (SOI) v.s. Recruitment (Rec)

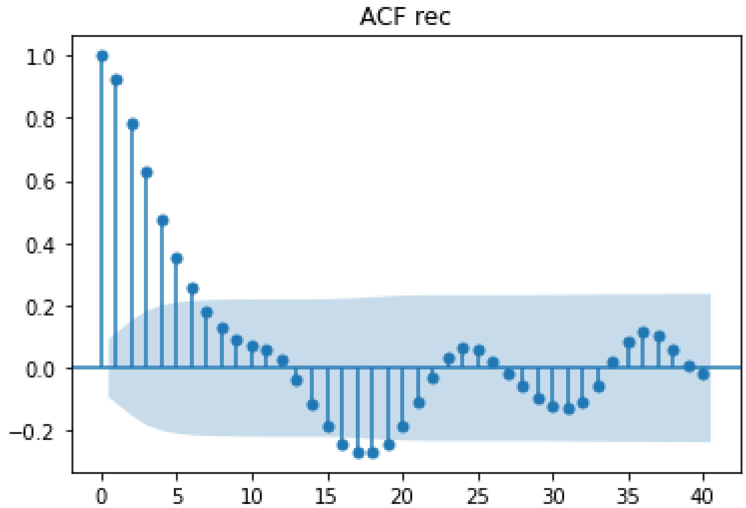
- Replicate the procedure in the previous section.
- What information can you tell from the CCF plot.
- In this example, our procedure is actually flawed. Unlike the previous example, we can not tell if the cross-correlation estimate is significantly different from zero by looking at the CCF. Why is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

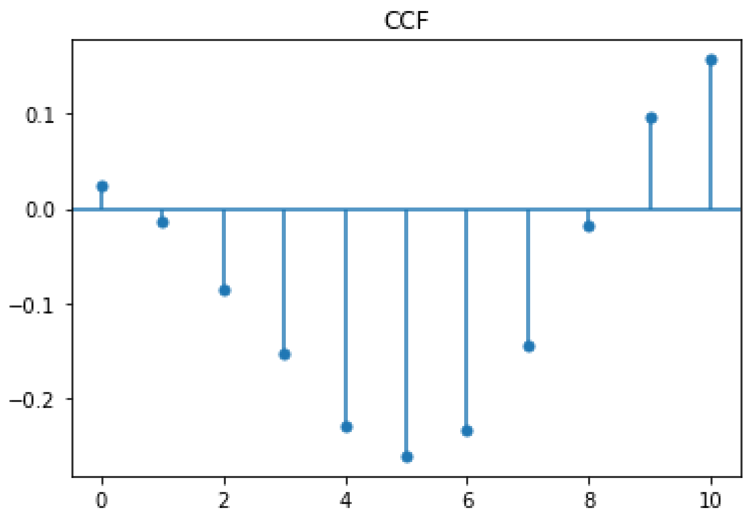
In [9]:
```
soi = np.array(pd.read_csv("../../data/soi.csv")["x"])
rec = np.array(pd.read_csv("../../data/rec.csv")["x"])
#TODO: This part will be graded.

lag=40

# plot data
plt.plot(soi)
plt.title('soi')
plt.show()
plt.plot(rec)
plt.title('rec')
plt.show()

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

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


soi
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
100
200
300
400


rec
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





Part III: AR models

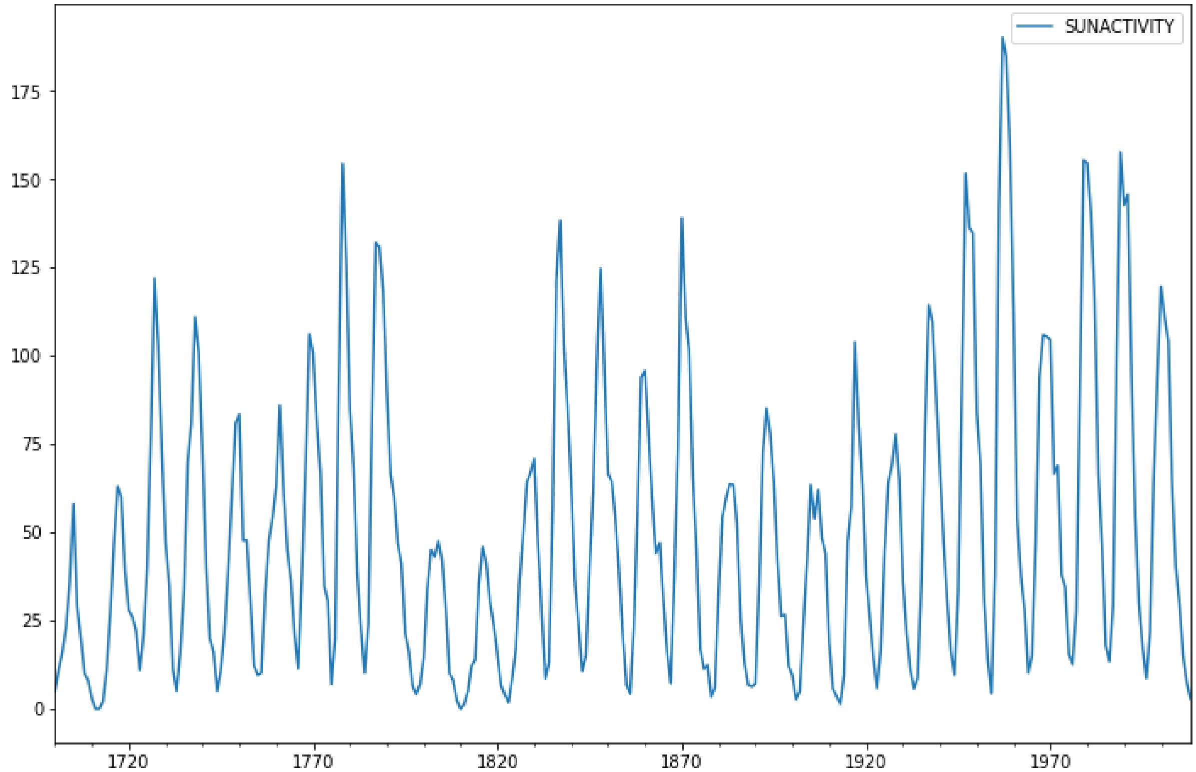
In this example, we will fit an AR(p) model to the SunActivity data, which denotes the number of sunspots for each year.

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 (https://en.wikipedia.org/wiki/Sunspot)](https://en.wikipedia.org/wiki/Sunspot)

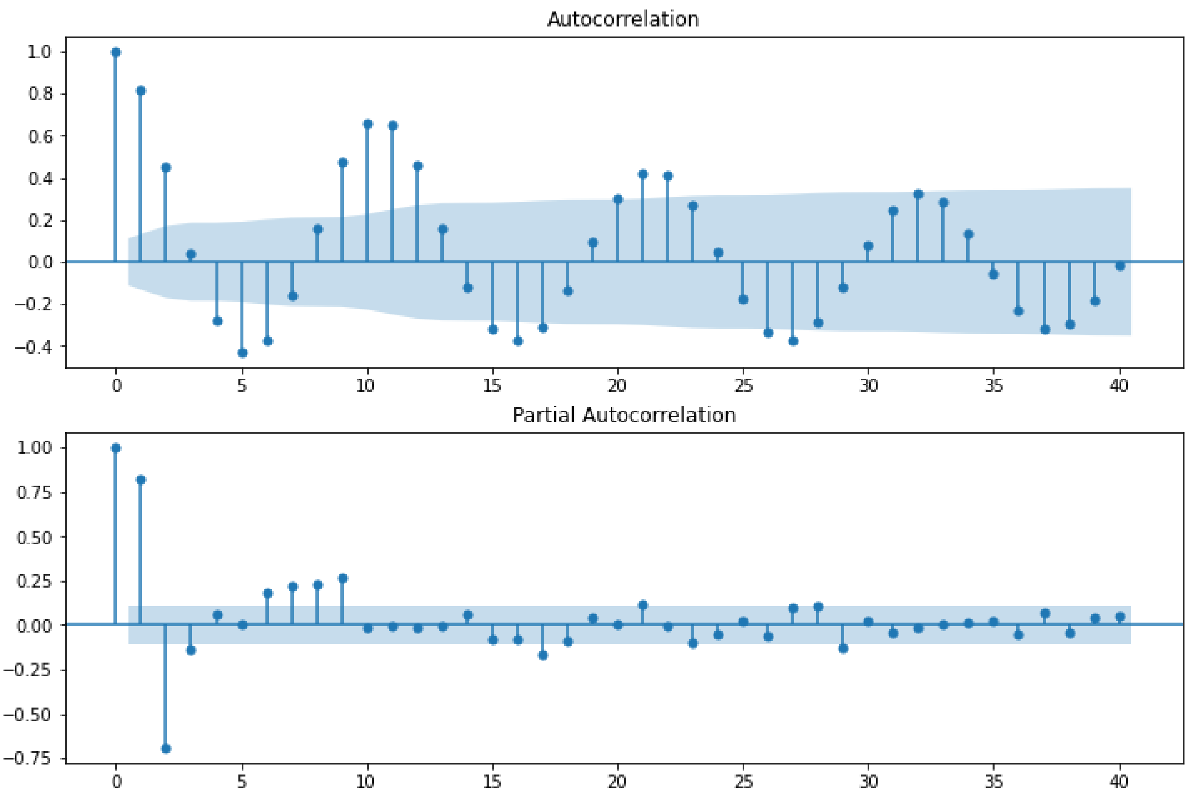
The code in this section is selected from the tutorial specified in the reference section.

In [10]:
```
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

In [11]:
```
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

In [12]:
```
# 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

# Analytical Calculation for rho0, rho1, rho2, rho3 using paramters

phi_1 = arma_mod.params['ar.L1.SUNACTIVITY']
phi_2 = arma_mod.params['ar.L2.SUNACTIVITY']
phi_3 = arma_mod.params['ar.L3.SUNACTIVITY']

rho = []
rho_0 = 1
rho_1 = ((- phi_2 * phi_3) - phi_1) / ((phi_2 - 1) + (phi_3 * (phi_1 + p
hi_3)))
rho_2 = -((phi_2 - 1) * rho_1 + phi_1) / phi_3
rho_3 = phi_1 * rho_2 + phi_2 * rho_1 + phi_3

rho.append(rho_0)
rho.append(rho_1)
rho.append(rho_2)
rho.append(rho_3)

rho = np.array(rho)

'''

# Estimated sample predicted rho

predict_sunspots = arma_mod.predict('2008', '2100', dynamic=True)
rho = acf_impl(x=predict_sunspots, nlags=p)
predict_sunspots = arma_mod.predict('1720', '2008', dynamic=True)
rho_2 = acf(x=dta['SUNACTIVITY'], nlags=p)
# print("ACF's at lag 0, 1, ..., p : \t", rho)

'''

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

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

```
const                  49.749891
ar.L1.SUNACTIVITY       1.300810
ar.L2.SUNACTIVITY      -0.508093
ar.L3.SUNACTIVITY      -0.129650
dtype: float64
roots:  [0. 0.]

/usr/local/anaconda3/envs/pTSA/lib/python3.8/site-packages/statsmodels/
tsa/arima_model.py:472: FutureWarning:
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.ARIMA
have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (note the
.
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 release.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace framework
and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until they ar
e
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARMA',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARIMA',
                        FutureWarning)

  warnings.warn(ARIMA_DEPRECATION_WARN, FutureWarning)
/usr/local/anaconda3/envs/pTSA/lib/python3.8/site-packages/statsmodels/
tsa/base/tsa_model.py:524: ValueWarning: No frequency information was p
rovided, so inferred frequency A-DEC will be used.
  warnings.warn('No frequency information was'
```

In [13]:

```
# predicted ACF of model at lag 0, 1, 2, 3
rho
```

Out[13]:

```
array([1.        , 0.82333673, 0.45616662, 0.04540509])
```

In [14]:

```
# sample ACF of model at lag 0, 1, 2, 3
sample_rho = acf_impl(x=dta['SUNACTIVITY'], nlags=3)
sample_rho
```

Out[14]:

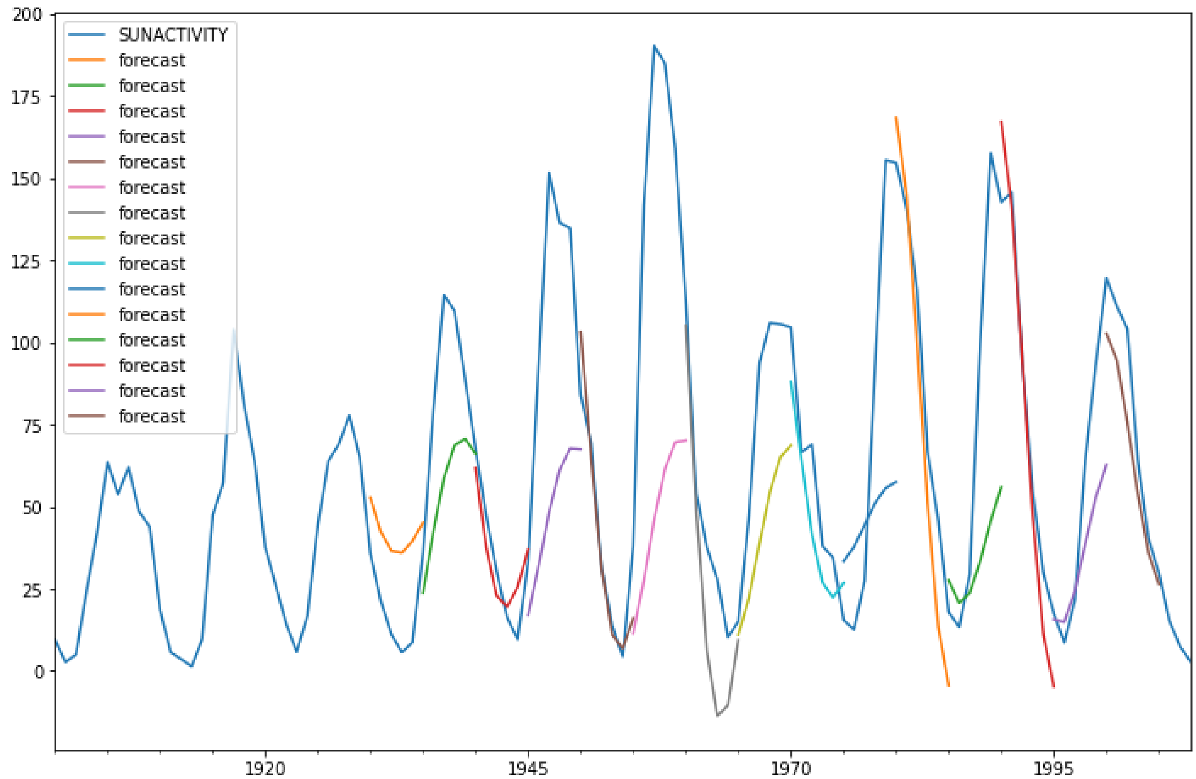
```
[1.0, 0.8202012944200218, 0.45126849200956737, 0.03957655157031837]
```

prediction

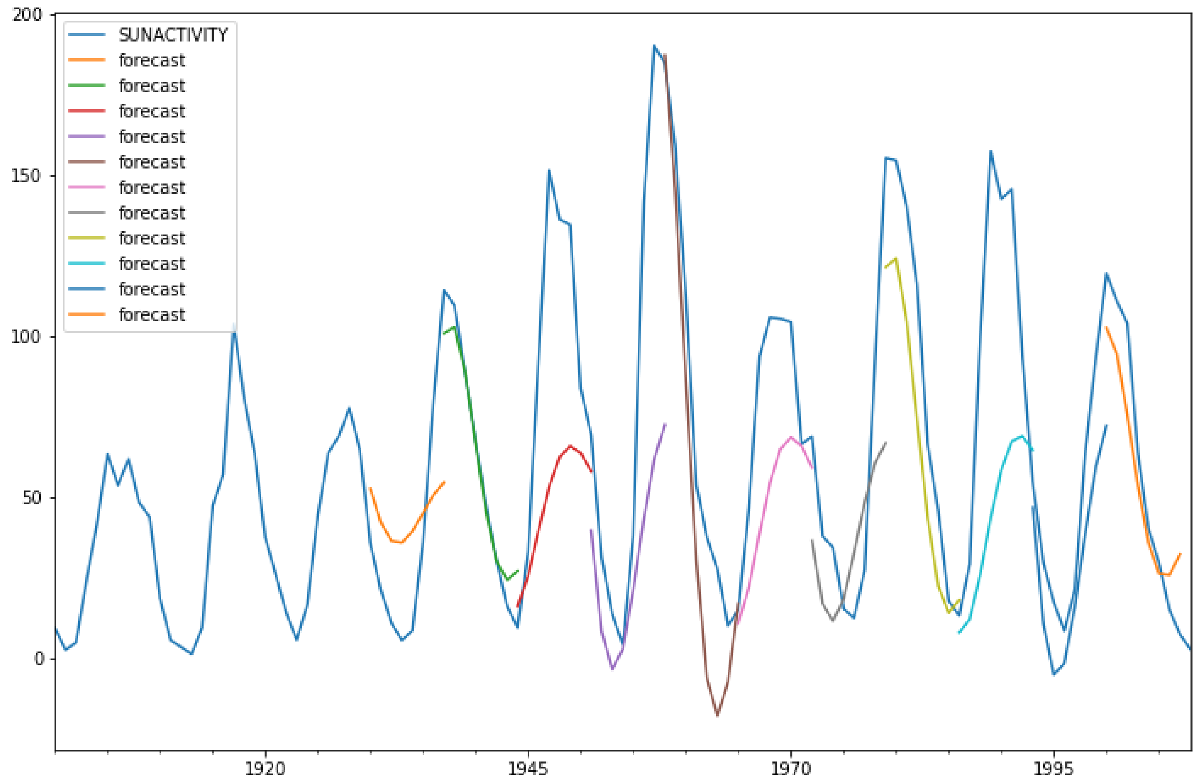
In [15]:
```
predict_sunspots = arma_mod.predict('1990', '2012', dynamic=True)
print(predict_sunspots)
```

```
1990-12-31    167.047408
1991-12-31    140.992976
1992-12-31     94.859066
1993-12-31     46.860832
1994-12-31     11.242503
1995-12-31     -4.721378
1996-12-31     -1.166989
1997-12-31     16.185629
1998-12-31     39.021839
1999-12-31     59.449844
2000-12-31     72.170124
2001-12-31     75.376765
2002-12-31     70.436431
2003-12-31     60.731546
2004-12-31     50.201743
2005-12-31     42.075964
2006-12-31     38.114221
2007-12-31     38.454579
2008-12-31     41.963758
2009-12-31     46.869237
2010-12-31     51.423218
2011-12-31     54.399679
2012-12-31     55.321652
Freq: A-DEC, dtype: float64
```

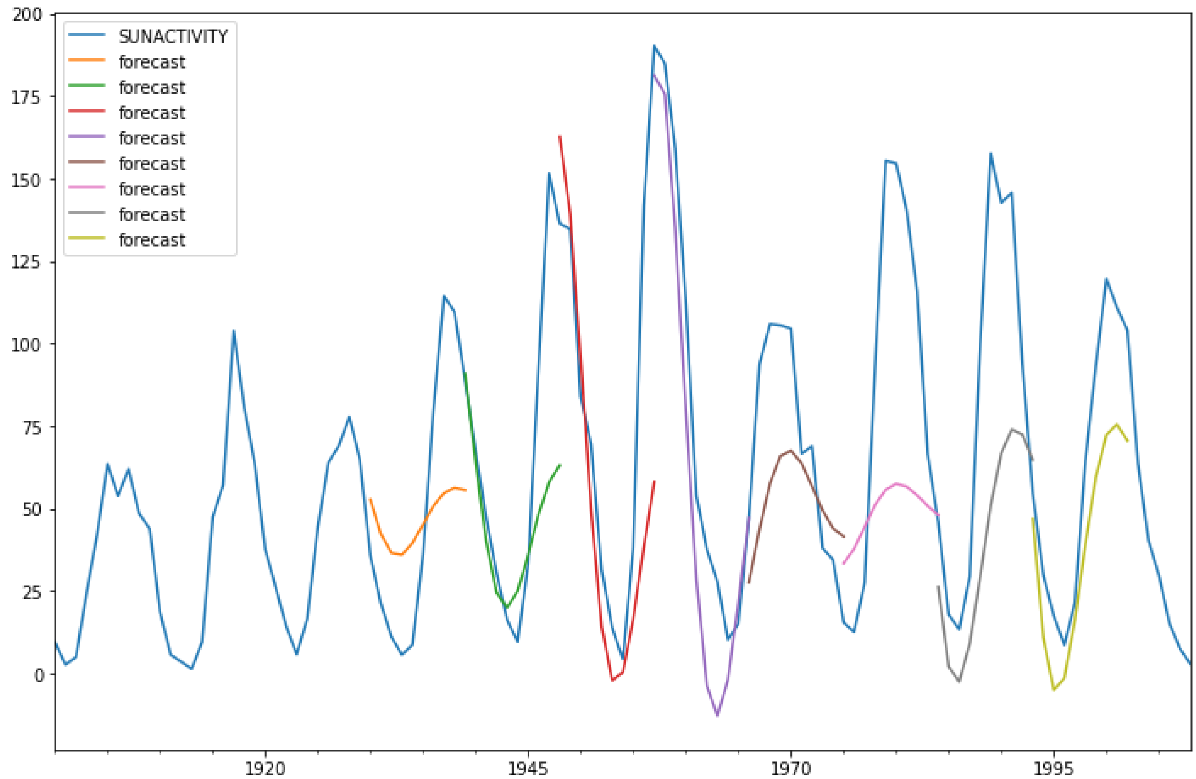
In [16]:
```
# TODO: try to predict further into the future by increasing tsteps
tsteps=5

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()
```



In [17]:
```
# TODO: try to predict further into the future by increasing tsteps
tsteps=7

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()
```

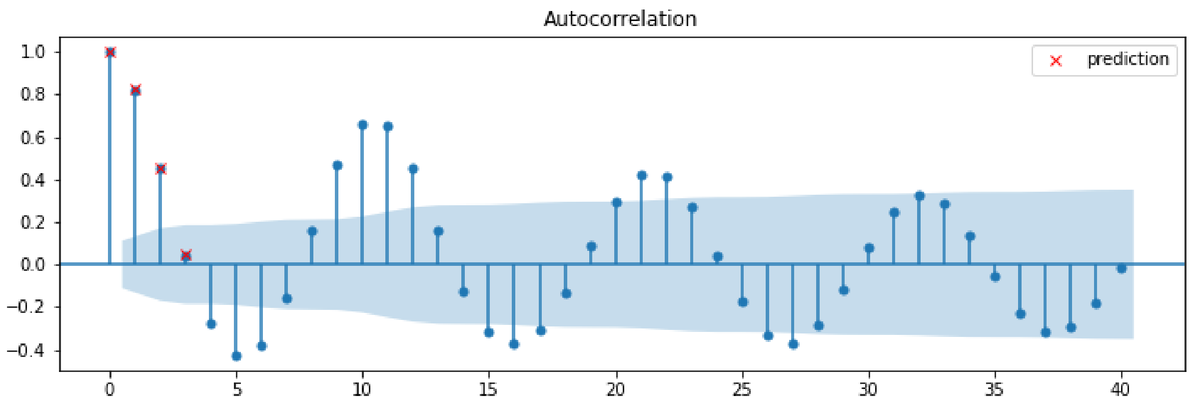


In [18]:
```
# TODO: try to predict further into the future by increasing tsteps
tsteps=9

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

In [19]:
```
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()
```



Part IV

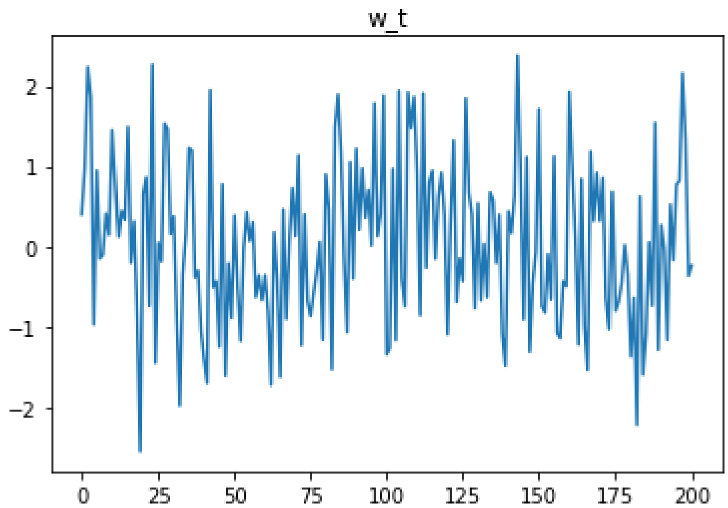
Moving Average

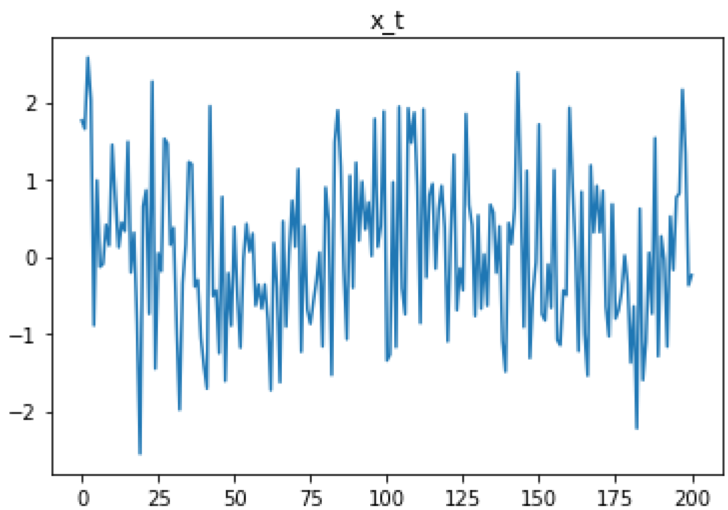
$$x_t = 0.5x_{t-1} - 0.5w_{t-1} + w_t$$
$$w_t \sim N(0, \sigma^2)$$

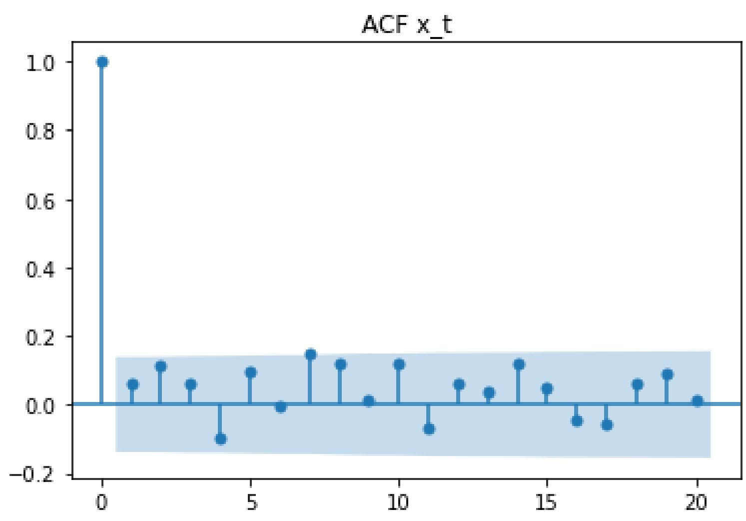
Is x_t same as white noise w_t? Think about ACF.

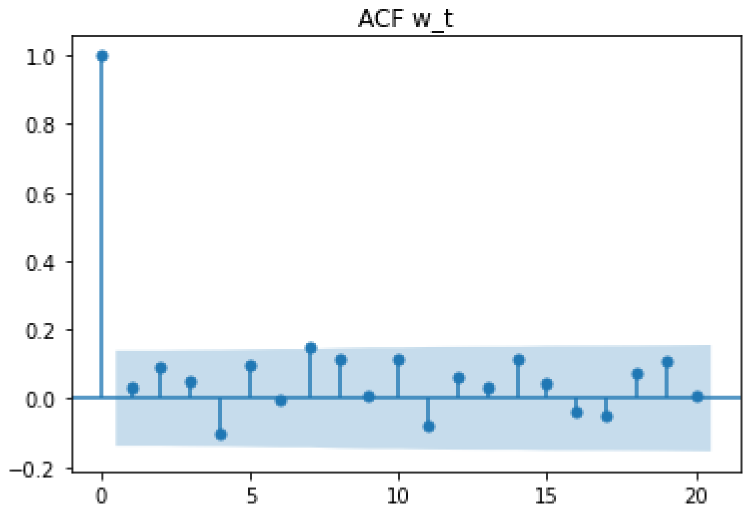
Then use code below to assess and verify your guess.

In [20]:
```
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()
```


w_t


x_t


ACF x_t

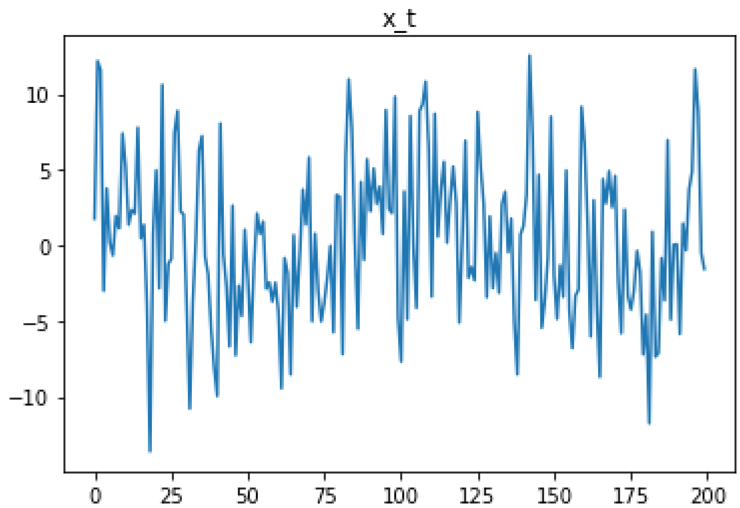


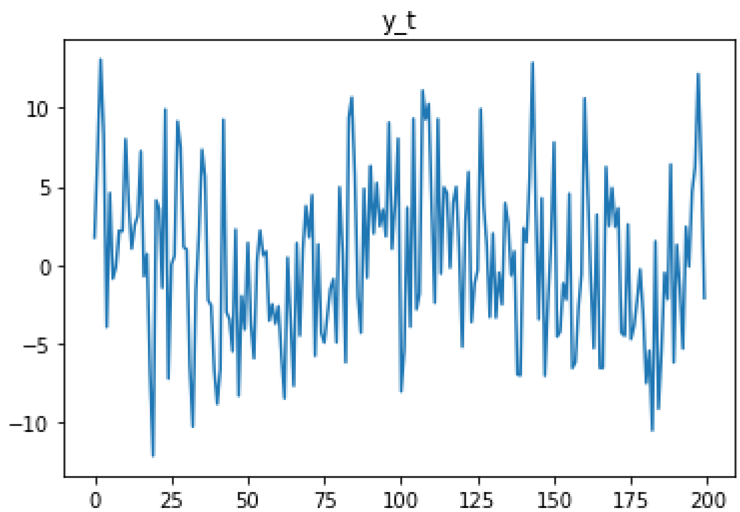
$$x_t = w_t + \frac{1}{5}w_{t-1}, w_t \sim N(0, 25)$$
$$y_t = v_t + 5v_{t-1}, v_t \sim N(0, 1)$$

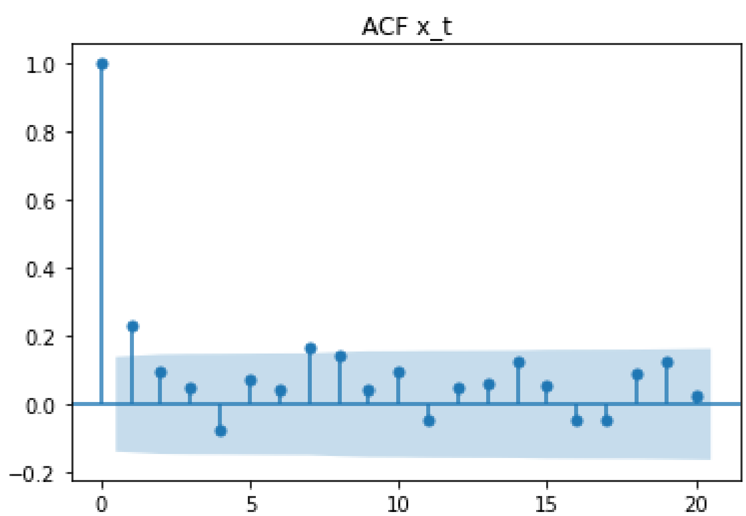
Are x_t and y_t the same? Think about ACF.

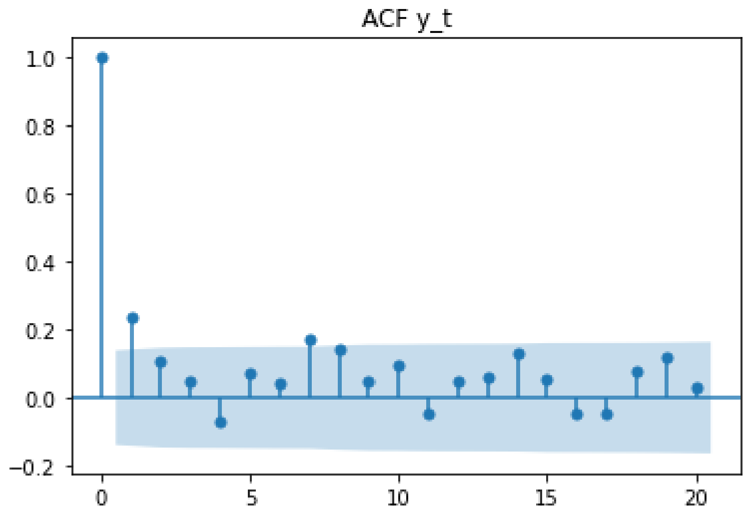
Then use code below to assess and verify your guess.

In [21]:
```
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



In [ ]:

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

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".

In [ ]: