Assignment V– Individual Take Home Assignment

In [1]:

```
# import libraries
%matplotlib inline

import numpy as np
from scipy import stats
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import pyplot

import statsmodels.api as sm

from math import sqrt
from sklearn.metrics import mean_squared_error

from statsmodels.graphics.api import qqplot
from statsmodels.tsa.arima_process import arma_generate_sample
np.random.seed(12345)
```

1. Simulate time series (TS1)

In [2]:

```
# ARMA(2,2) with AR coefficients 0.75 and -0.25, and MA coefficients 0.65 and 0.35

arparams = np.array([.75, -.25])
maparams = np.array([.65, .35])
arparams = np.r_[1, -arparams] # add zero-lag and negate
maparams = np.r_[1, maparams] # add zero-lag
nobs = 500
TS1 = arma_generate_sample(arparams, maparams, nobs)
```

2. Simulate time series (TS2)

Introduce 25 extreme observations to the above data series (TS1) and save the new series as TS2.

In [3]:

```
# Introducing 25 extreme observations to a copy of TS1 and saving as TS2
TS2 = np.copy(TS1)
TS2[np.random.choice(501, 25)]=1*np.random.randn(25)+10
```

3. Model fitting procedure for TS1

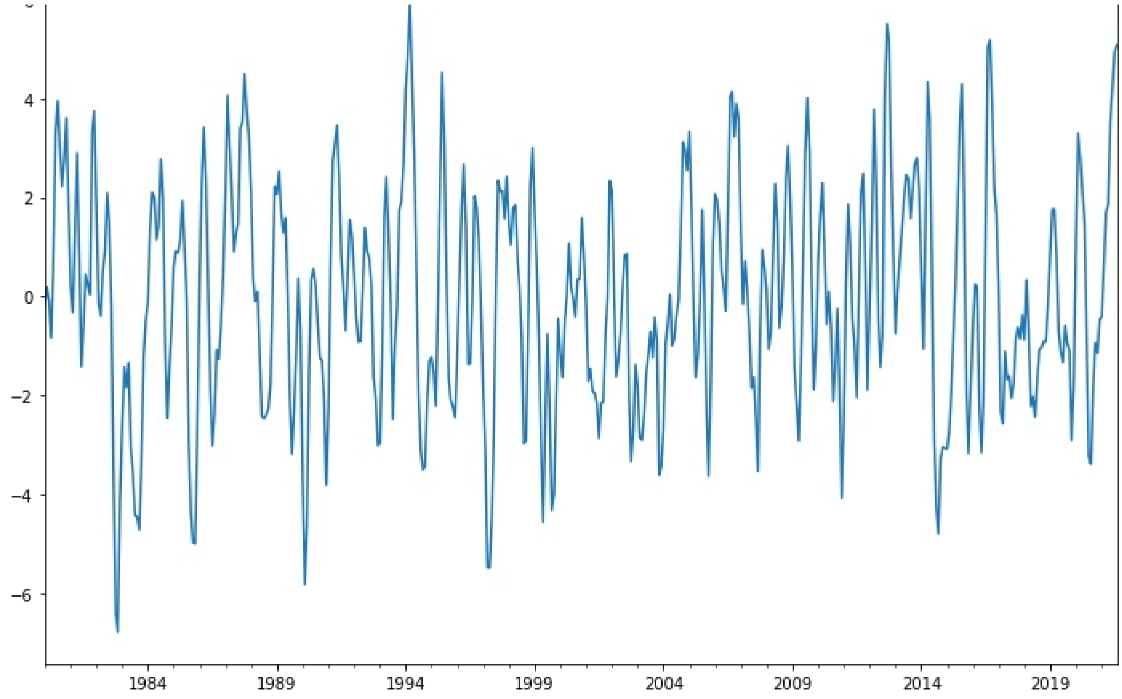
Classical time series model fitting for known artificial data

In [4]:

```
# add dates information
dates = sm.tsa.datetools.dates_from_range('1980m1', length=nobs)
TS1 = pd.Series(TS1, index=dates)
```

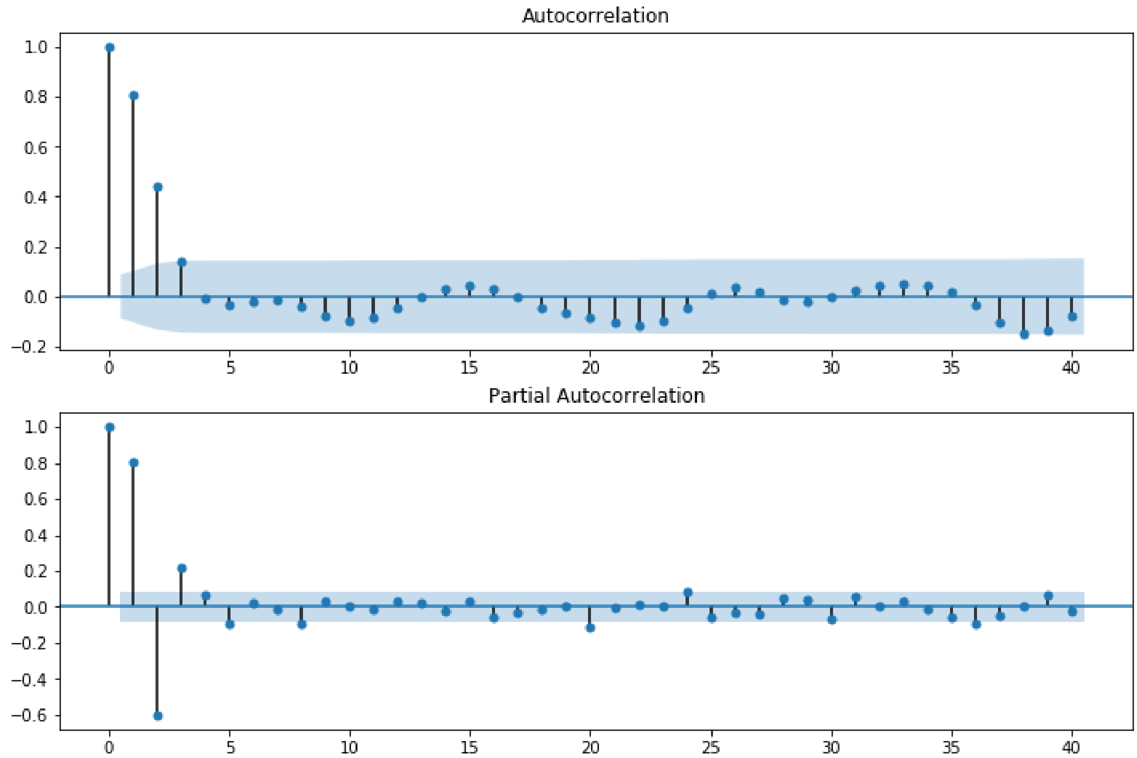
In [5]:

```
# plot TS1
TS1.plot(figsize=(12,8));
```



In [6]:

```
# plot ACF & PACF for TS1
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(TS1.values.squeeze(), lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(TS1, lags=40, ax=ax2)
```



In [7]:

```
# split data into train and test-sets
TS1_train, TS1_test = TS1[0:-12], TS1[-12:]

# model ARMA (2,2) for TS1 and print summary
arma_mod_TS1 = sm.tsa.ARMA(TS1_train, order=(2,2))
arma_res_TS1 = arma_mod_TS1.fit(trend='nc', disp=-1)
print(arma_res_TS1.summary())
```

```
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: Va
lueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
```

```
                              ARMA Model Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  488
Model:                     ARMA(2, 2)   Log Likelihood                -699.765
Method:                       css-mle   S.D. of innovations              1.013
Date:                Sun, 01 Mar 2020   AIC                           1409.530
Time:                        00:20:28   BIC                           1430.481
Sample:                    01-31-1980   HQIC                          1417.760
                         - 08-31-2020
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1.y        0.7352      0.110      6.702      0.000       0.520       0.950
ar.L2.y       -0.2357      0.088     -2.683      0.008      -0.408      -0.063
ma.L1.y        0.7038      0.106      6.653      0.000       0.496       0.911
ma.L2.y        0.3942      0.077      5.132      0.000       0.244       0.545
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            1.5599           -1.3454j            2.0599           -0.1133
AR.2            1.5599           +1.3454j            2.0599            0.1133
MA.1           -0.8926           -1.3191j            1.5927           -0.3447
MA.2           -0.8926           +1.3191j            1.5927            0.3447
-----------------------------------------------------------------------------
```

In [8]:

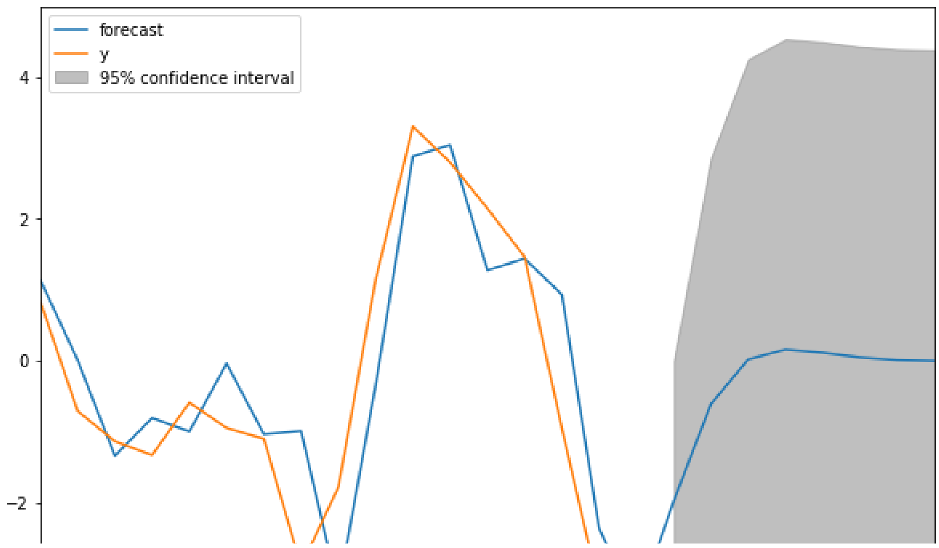
```
# Tail of training set of TS1
TS1_train.tail()
```

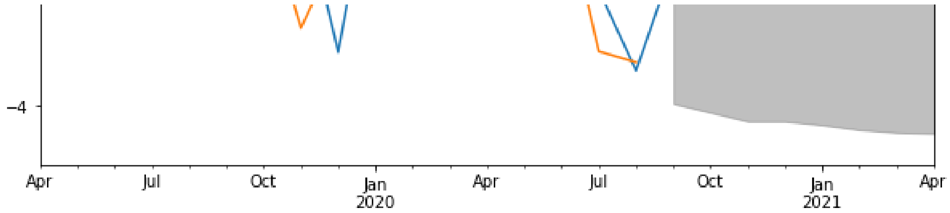
Out[8]:

```
2020-04-30    2.144944
2020-05-31    1.460525
2020-06-30   -0.956910
2020-07-31   -3.233294
2020-08-31   -3.386809
dtype: float64
```

In [9]:

```
# plot forecasted values for TS1
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(10,8))
fig = arma_res_TS1.plot_predict(start='2019-04-30', end='2021-04-30', ax=ax)
legend = ax.legend(loc='upper left')
```

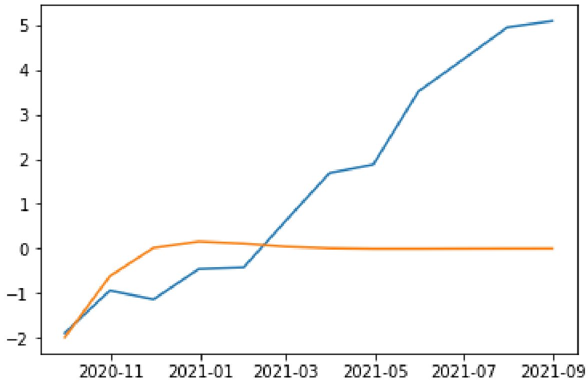




In [10]:

```
# plot forecasted values vs observed values for validation data in TS1 and print root mean square
error
predictions = arma_res_TS1.predict(start='2020-09-30', end='2021-08-31', dynamic=True)
# report performance
rmse = sqrt(mean_squared_error(TS1_test, predictions))
print('Test RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(TS1_test)
pyplot.plot(predictions)
pyplot.show()
```

```
Test RMSE: 2.727
```



3. Model fitting procedure for TS2

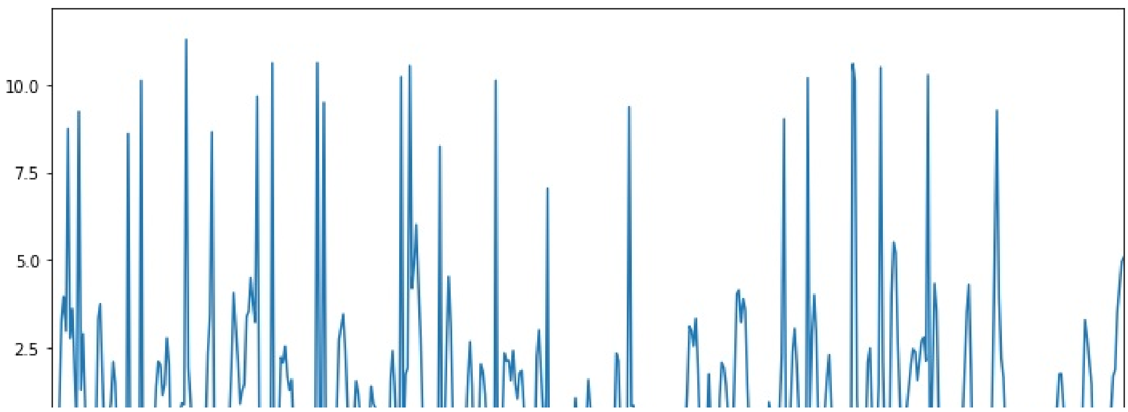
Classical time series model fitting for unknown artificial data

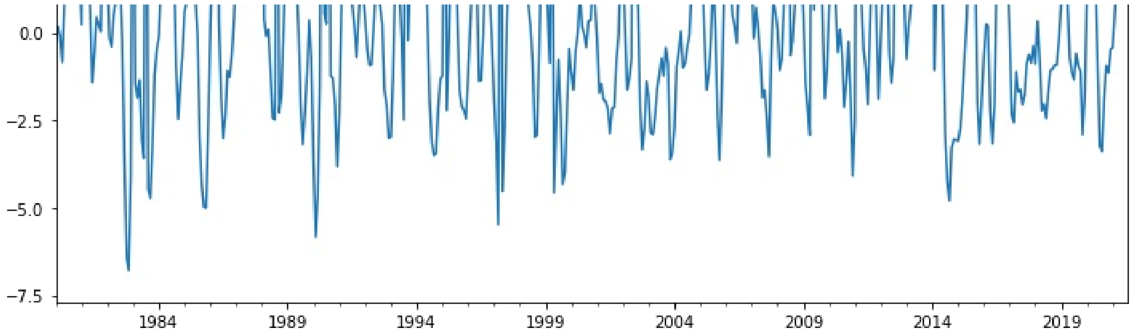
In [11]:

```
# add dates information to TS2
dates = sm.tsa.datetools.dates_from_range('1980m1', length=nobs)
TS2 = pd.Series(TS2, index=dates)
```

In [12]:

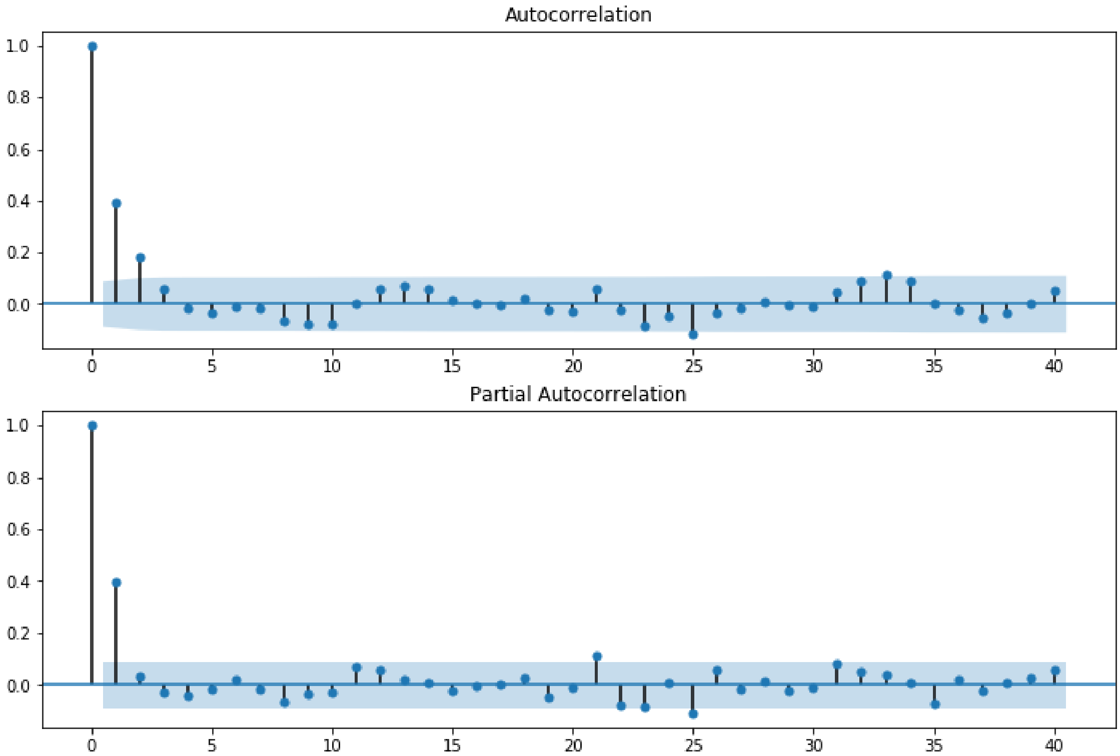
```
# plot TS2
TS2.plot(figsize=(12,8));
```





In [13]:

```
# ACF and PACF for TS2
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(TS2.values.squeeze(), lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(TS2, lags=40, ax=ax2)
```



In [14]:

```
# Loop to test multiple combinations of AR, MA values
for x in range(3):
    for y in range(3):
        arma_mod = sm.tsa.ARMA(TS2, (x,y)).fit(disp=False)
        print("X:",x)
        print("Y:",y)
        print(arma_mod.aic, arma_mod.bic, arma_mod.hqic)
```

```
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: Va
lueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: Va
lueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: Va
lueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: Va
lueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
```

```
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
```

```
X: 0
Y: 0
2521.0481516739233 2529.4773678707675 2524.3557623363263
X: 0
Y: 1
2454.0395122099117 2466.6833365051784 2459.000928203516
X: 0
Y: 2
2442.580677154195 2459.439109547884 2449.1958984790012
X: 1
Y: 0
2438.8140296086217 2451.4578539038885 2443.7754456022262
X: 1
Y: 1
2440.214438390107 2457.0728707837957 2446.829659714913
X: 1
Y: 2
2441.712164211557 2462.785204703668 2449.9811908675642
X: 2
Y: 0
2440.117844984379 2456.976277378068 2446.7330663091852
```

```
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
```

```
X: 2
Y: 1
2441.993075527876 2463.066116019987 2450.262102183883
```

```
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
```

```
X: 2
Y: 2
2442.7334801162733 2468.0211287068064 2452.656312103482
```

In [15]:

```
# split data into train and test-sets
TS2_train, TS2_test = TS2[0:-12], TS2[-12:]

# fit ARMA (2,2) to TS2
arma_mod_TS2 = sm.tsa.ARMA(TS2_train, (1,1)).fit(disp=False)
print(arma_mod_TS2.params)
print(arma_mod_TS2.aic, arma_mod_TS2.bic, arma_mod_TS2.hqic)
```

```
const      0.361525
ar.L1.y    0.446071
ma.L1.y   -0.072266
dtype: float64
2389.1571357425864 2405.918397365999 2395.74102405854
```

```
C:\ProgramData\Miniconda3\envs\sample1\lib\site-packages\statsmodels\tsa\base\tsa_model.py:165: ValueWarning: No frequency information was provided, so inferred frequency M will be used.
  % freq, ValueWarning)
```

In [16]:

```
# Durbin Watson Test
```

```
# Durbin Watson Test
print(sm.stats.durbin_watson(arma_mod_TS2.resid.values))
```

1.9993581572867287

In [17]:

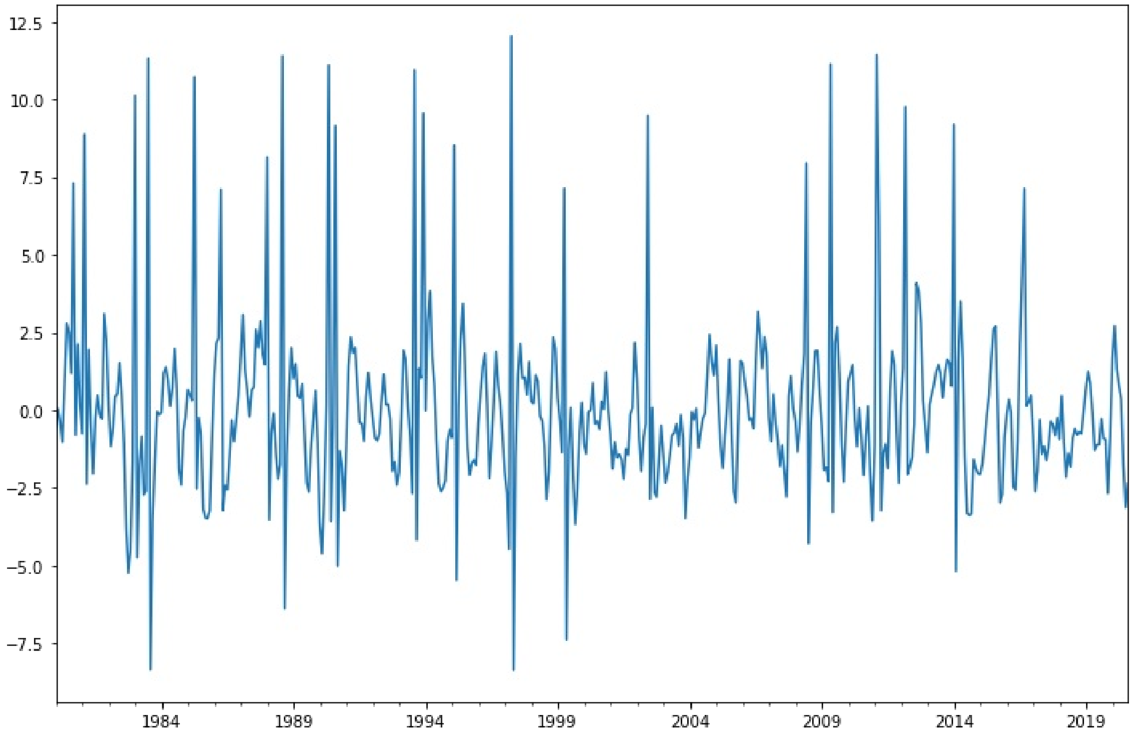
```
# Ljung-Box test
sm.stats.acorr_ljungbox(arma_mod_TS2.resid.values, lags=[20])
```

Out[17]:

(array([11.49613543]), array([0.93232825]))

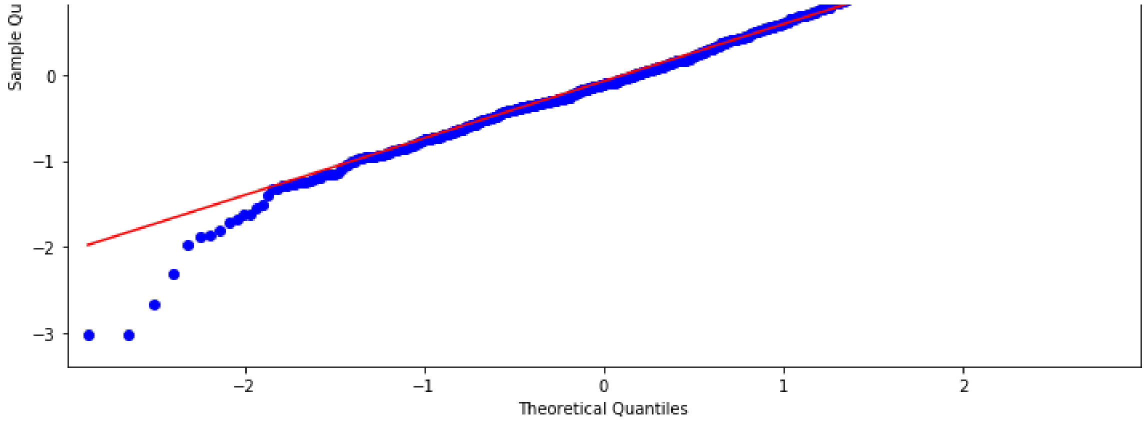
In [18]:

```
# residual plot for TS2 on ARMA (2,2)
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111)
ax = arma_mod_TS2.resid.plot(ax=ax);
```



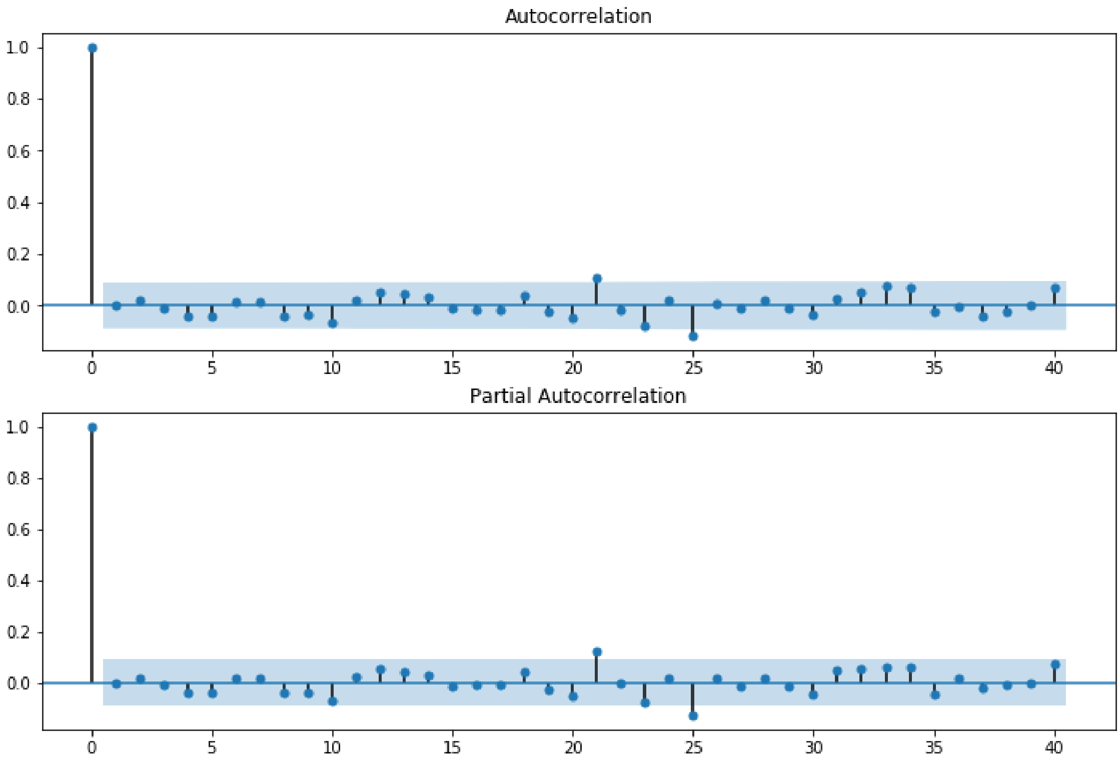
In [19]:

```
# Q-Q Plot

resid = arma_mod_TS2.resid
stats.normaltest(resid)
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111)
fig = qqplot(resid, line='q', ax=ax, fit=True)
```



In [20]:

```
# Residual ACF & PACF for TS2

fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = sm.graphics.tsa.plot_acf(resid.values.squeeze(), lags=40, ax=ax1)
ax2 = fig.add_subplot(212)
fig = sm.graphics.tsa.plot_pacf(resid, lags=40, ax=ax2)
```



In [21]:

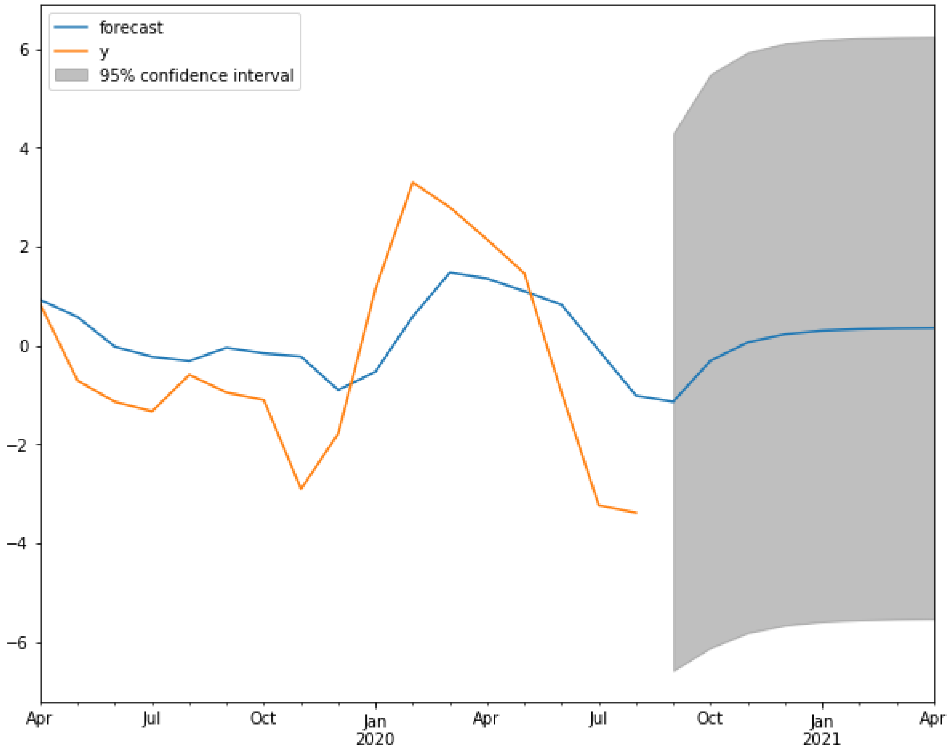
```
# tail of TS2 training test

TS2_train.tail()
```

Out[21]:

```
2020-04-30    2.144944
2020-05-31    1.460525
2020-06-30   -0.956910
2020-07-31   -3.233294
2020-08-31   -3.386809
dtype: float64
```

In [22]:

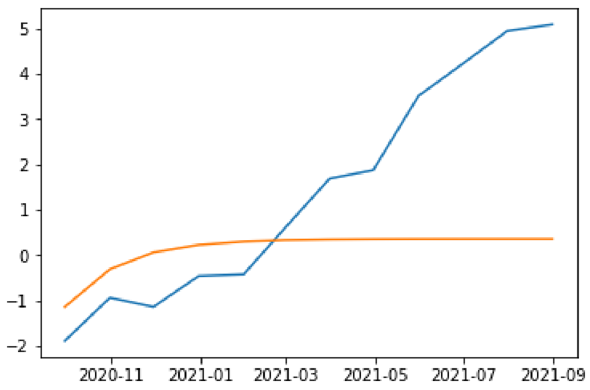
```
# forecest for TS2 with ARMA (2,2) model

import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(10,8))
fig = arma_mod_TS2.plot_predict(start='2019-04-30', end='2021-04-30', ax=ax)
legend = ax.legend(loc='upper left')
```



In [23]:

```
# plot forecasted values vs observed values for validation data in TS2 and print root mean square
error
predictions = arma_mod_TS2.predict(start='2020-09-30', end='2021-08-31', dynamic=True)
# report performance
rmse = sqrt(mean_squared_error(TS2_test, predictions))
print('Test RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(TS2_test)
pyplot.plot(predictions)
pyplot.show()
```

```
Test RMSE: 2.511
```



4. ANN model fitting procedure

In [24]:

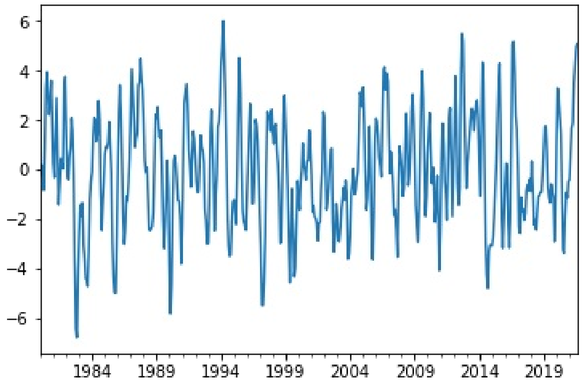
```
# import libraries
```

```
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Dense
from tensorflow.python.keras.layers import LSTM
from math import sqrt
from matplotlib import pyplot
import numpy
```

In [25]:

```
print(TS1.head())
# line plot
TS1.plot()
pyplot.show()
```

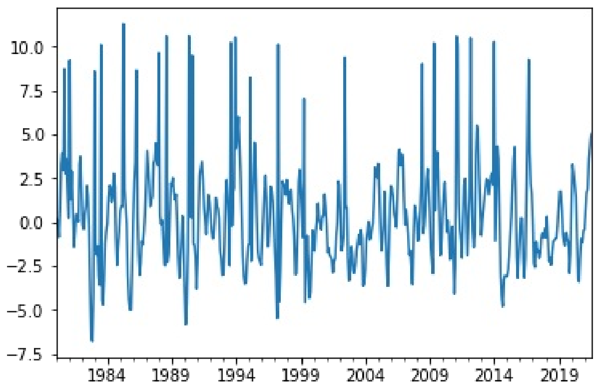
```
1980-01-31   -0.204708
1980-02-29    0.192353
1980-03-31   -0.084332
1980-04-30   -0.837072
1980-05-31    0.816031
dtype: float64
```



In [38]:

```
print(TS2.head())
# line plot
TS2.plot()
pyplot.show()
```

```
1980-01-31   -0.204708
1980-02-29    0.192353
1980-03-31   -0.084332
1980-04-30   -0.837072
1980-05-31    0.816031
dtype: float64
```



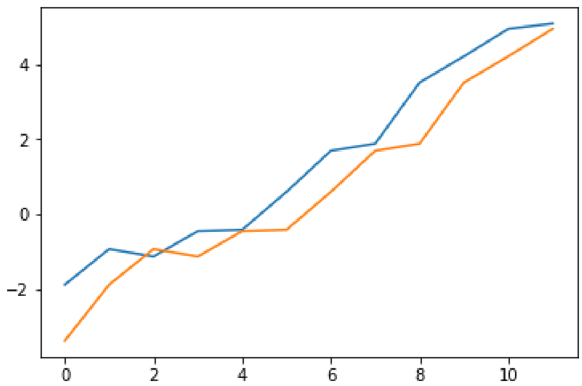
Persistence Model Forecast

Baseline Model

In [26]:

```
# split data into train and test
X = TS1.values
train, test = X[0:-12], X[-12:]
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    # make prediction
    predictions.append(history[-1])
    # observation
    history.append(test[i])
# report performance
rmse = sqrt(mean_squared_error(test, predictions))
print('RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(test)
pyplot.plot(predictions)
pyplot.show()
```

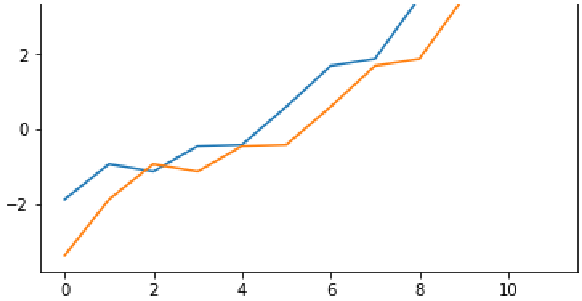
```
RMSE: 0.896
```



In [40]:

```
# split data into train and test
X = TS2.values
train, test = X[0:-12], X[-12:]
# walk-forward validation
history = [x for x in train]
predictions = list()
for i in range(len(test)):
    # make prediction
    predictions.append(history[-1])
    # observation
    history.append(test[i])
# report performance
rmse = sqrt(mean_squared_error(test, predictions))
print('RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(test)
pyplot.plot(predictions)
pyplot.show()
```

```
RMSE: 0.896
```



In [27]:

```
# frame a sequence as a supervised learning problem
def timeseries_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)
    df.fillna(0, inplace=True)
    return df
```

In [28]:

```
# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled
```

In [29]:

```
# inverse scaling for a forecasted value
def invert_scale(scaler, X, value):
    new_row = [x for x in X] + [value]
    array = numpy.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]
```

In [30]:

```
# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], 1, X.shape[1])
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=True))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
        model.reset_states()
    return model
```

In [31]:

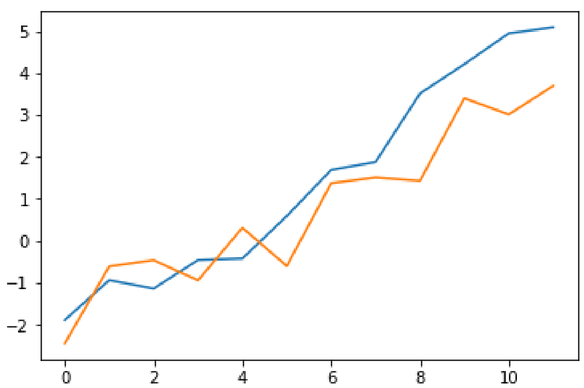
```
# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, 1, len(X))
    yhat = model.predict(X, batch_size=batch_size)
    return yhat[0,0]
```

Long Short-Term Memory recurrent neural network model fitting procedure for TS1

In [32]:

```
# transform data to be supervised learning
supervised = timeseries_to_supervised(TS1, 1)
supervised_values = supervised.values

# split data into train and test-sets
train, test = supervised_values[0:-12], supervised_values[-12:]

# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)

# fit the model
lstm_model = fit_lstm(train_scaled, 1, 20, 4)
# forecast the entire training dataset to build up state for forecasting
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
lstm_model.predict(train_reshaped, batch_size=1)

# walk-forward validation on the test data
predictions = list()
for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
    yhat = forecast_lstm(lstm_model, 1, X)
    # invert scaling
    yhat = invert_scale(scaler, X, yhat)
    # store forecast
    predictions.append(yhat)
    raw_values = TS1.values
    expected = raw_values[len(train) + i]
    print('Month=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

# report performance
rmse = sqrt(mean_squared_error(raw_values[-12:], predictions))
print('Test RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(raw_values[-12:])
pyplot.plot(predictions)
pyplot.show()
```

```
Month=1, Predicted=-2.445661, Expected=-1.887990
Month=2, Predicted=-0.607034, Expected=-0.936715
Month=3, Predicted=-0.460770, Expected=-1.137193
Month=4, Predicted=-0.941245, Expected=-0.456617
Month=5, Predicted=0.311034, Expected=-0.419927
Month=6, Predicted=-0.599351, Expected=0.591934
Month=7, Predicted=1.372153, Expected=1.689647
Month=8, Predicted=1.510329, Expected=1.877750
Month=9, Predicted=1.430099, Expected=3.512217
Month=10, Predicted=3.399665, Expected=4.209811
Month=11, Predicted=3.012022, Expected=4.944123
Month=12, Predicted=3.693534, Expected=5.087330
Test RMSE: 1.079
```



Long Short-Term Memory recurrent neural network model fitting procedure for TS2

In [33]:

```
# transform data to be supervised learning
supervised = timeseries_to_supervised(TS2, 1)
supervised_values = supervised.values

# split data into train and test-sets
train, test = supervised_values[0:-12], supervised_values[-12:]

# transform the scale of the data
scaler, train_scaled, test_scaled = scale(train, test)

# fit the model
lstm_model = fit_lstm(train_scaled, 1, 20, 4)
# forecast the entire training dataset to build up state for forecasting
train_reshaped = train_scaled[:, 0].reshape(len(train_scaled), 1, 1)
lstm_model.predict(train_reshaped, batch_size=1)

# walk-forward validation on the test data
predictions = list()
for i in range(len(test_scaled)):
    # make one-step forecast
    X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
    yhat = forecast_lstm(lstm_model, 1, X)
    # invert scaling
    yhat = invert_scale(scaler, X, yhat)
    # store forecast
    predictions.append(yhat)
    raw_values = TS1.values
    expected = raw_values[len(train) + i]
    print('Month=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

# report performance
rmse = sqrt(mean_squared_error(raw_values[-12:], predictions))
print('Test RMSE: %.3f' % rmse)
# line plot of observed vs predicted
pyplot.plot(raw_values[-12:])
pyplot.plot(predictions)
pyplot.show()
```

```
Month=1, Predicted=-1.712033, Expected=-1.887990
Month=2, Predicted=-0.600970, Expected=-0.936715
Month=3, Predicted=-0.247514, Expected=-1.137193
Month=4, Predicted=-0.574822, Expected=-0.456617
Month=5, Predicted=0.140141, Expected=-0.419927
Month=6, Predicted=-0.106984, Expected=0.591934
Month=7, Predicted=0.766190, Expected=1.689647
Month=8, Predicted=1.290435, Expected=1.877750
Month=9, Predicted=1.297783, Expected=3.512217
Month=10, Predicted=2.203537, Expected=4.209811
Month=11, Predicted=2.529807, Expected=4.944123
Month=12, Predicted=2.836607, Expected=5.087330
Test RMSE: 1.378
```