Università Commerciale
Luigi Bocconi

MSc. Finance/CLEFIN
2016/2017 Edition

Financial Econometrics and Empirical Finance II

Daniel Toppo, Pictet Asset Management

1 Dynamic Models

```
"""
@author: dtoppo
Dynamic models
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

# Import dataserie from Excel
dataFile = "data_python_HW_2017.xlsx"
xslx = pd.ExcelFile(dataFile)

sheetName = "Dynamic Models"
dataSeries = xslx.parse(sheetName)

#===========================================================================
# Regression result
# dir(model) to display model attributes
#===========================================================================
model = ols("USVW ~ Treasury", dataSeries).fit()
print(model.summary()) # table summary
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.046
Model:                            OLS   Adj. R-squared:                  0.040
Method:                 Least Squares   F-statistic:                     7.409
Date:                Thu, 25 May 2017   Prob (F-statistic):           7.42e-05
Time:                        11:01:50   Log-Likelihood:                 755.82
No. Observations:                 465   AIC:                            -1504.
Df Residuals:                     461   BIC:                            -1487.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0093      0.002      3.952      0.000       0.005       0.014
x1             0.2155      0.047      4.630      0.000       0.124       0.307
x2            -0.0317      0.048     -0.667      0.505      -0.125       0.062
x3            -0.0318      0.046     -0.688      0.492      -0.123       0.059
==============================================================================
Omnibus:                       49.632   Durbin-Watson:                   1.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              238.733
Skew:                          -0.287   Prob(JB):                     1.44e-52
Kurtosis:                       6.463   Cond. No.                         24.4
==============================================================================
```

Figure 1: OLS Regression Results

```python
#==============================================================================
# Autocorrelation analysis
#==============================================================================
residuals = model.resid
lagged_residuals = model.resid.shift()
corr_matrix = np.corrcoef(residuals[1:], lagged_residuals[1:])
print(corr_matrix)
```

$$\begin{bmatrix} 1 & 0.075 \\ 0.075 & 1 \end{bmatrix}$$

```python
#==============================================================================
# Plot residuals
#==============================================================================
x_range = np.arange(0, np.size(residuals))
plt.plot(x_range, residuals, 'bD')

hlines_range = np.arange(-0.3, 0.2, 0.05)
for hline in hlines_range:
    plt.axhline(y=hline, color='#aaaaaa', linestyle=":")

min_value = -0.3
max_value = 0.15

plt.ylim(min_value, max_value)
```

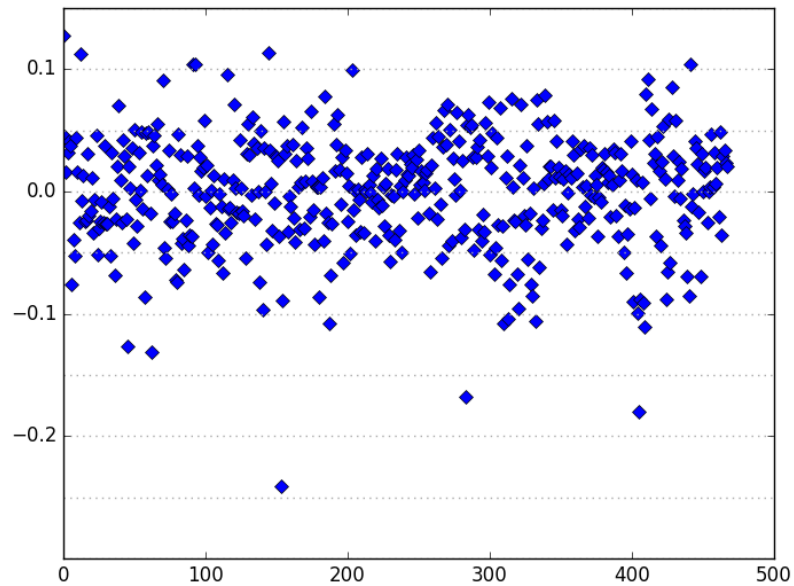

Figure 2: Residuals

Looking at the values of the residuals and at the graph above, there seems to be a tendency for negative values to follow negative values, and positive values to follow positive values. This is consistent with positive correlation between successive terms.
We can also check numerically the correlation between the residuals and the lagged values. The output indicates about 0.075 correlation between the errors one period apart.

2 ARMA Models

```python
"""
@author: dtoppo
ARMA models
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from statsmodels.formula.api import ols
from scipy import optimize

# Import dataserie from Excel
dataFile = "data_python_HW_2017.xlsx"
xslx = pd.ExcelFile(dataFile)

sheetName = "ARMA"
dataSeries = xslx.parse(sheetName)

ret_lag_1 = dataSeries.shift()[3:]
ret_lag_2 = dataSeries.shift(2)[3:]
ret_lag_3 = dataSeries.shift(3)[3:]

#==============================================================================
# Multiple regression
# dir(model) to display model attributes
#==============================================================================
serie = "Belgium"
y = dataSeries[serie][3:]
data = pd.concat([y, ret_lag_1[serie], ret_lag_2[serie], ret_lag_3[serie]],
                 axis=1, keys=["y", "x1", "x2", "x3"])
model = ols("y ~ x1 + x2 + x3", data).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.046
Model:                            OLS   Adj. R-squared:                  0.040
Method:                 Least Squares   F-statistic:                     7.409
Date:                Thu, 25 May 2017   Prob (F-statistic):           7.42e-05
Time:                        14:31:20   Log-Likelihood:                 755.82
No. Observations:                 465   AIC:                            -1504.
Df Residuals:                     461   BIC:                            -1487.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0093      0.002      3.952      0.000       0.005       0.014
x1             0.2155      0.047      4.630      0.000       0.124       0.307
x2            -0.0317      0.048     -0.667      0.505      -0.125       0.062
x3            -0.0318      0.046     -0.688      0.492      -0.123       0.059
==============================================================================
Omnibus:                       49.632   Durbin-Watson:                   1.986
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              238.733
Skew:                          -0.287   Prob(JB):                     1.44e-52
Kurtosis:                       6.463   Cond. No.                         24.4
==============================================================================
```

Figure 3: OLS Regression Results

```
#==========================================================================
# Plots
#==========================================================================
n = np.size(y)
x_range = np.arange(0, n)

predictedY = model.predict() # predicted values
residuals = model.wresid # residuals
returns = predictedY + residuals

# Plot data
plt.plot(x_range, predictedY)
plt.plot(x_range, returns)
```

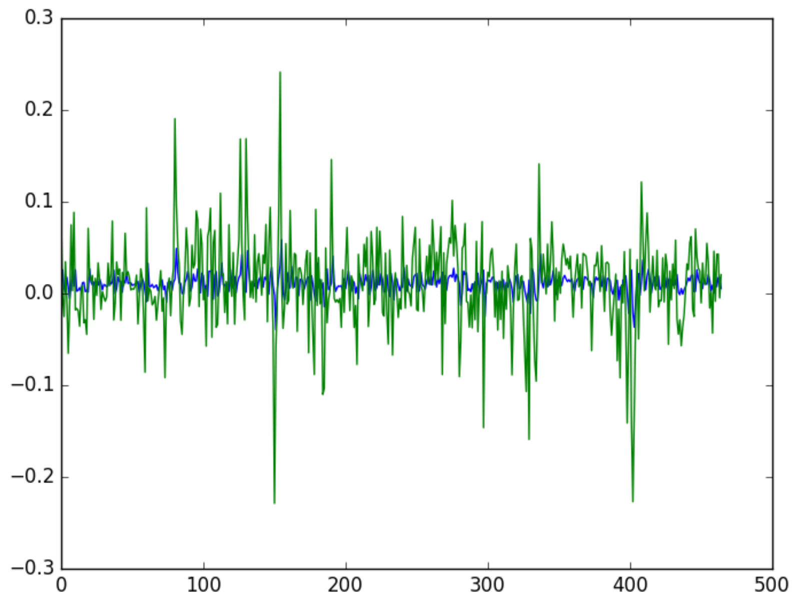

Figure 4: Actual and Fitted Values

```
# Residual plots
plt.figure()
plt.subplot(311)
plt.plot(ret_lag_1[serie], residuals, "p")
plt.axhline(y=0, color='#aaaaaa', linestyle=":")
plt.axvline(x=0, color='#aaaaaa', linestyle=":")

plt.subplot(312)
plt.plot(ret_lag_2[serie], residuals, "p")
plt.axhline(y=0, color='#aaaaaa', linestyle=":")
plt.axvline(x=0, color='#aaaaaa', linestyle=":")

plt.subplot(313)
plt.plot(ret_lag_3[serie], residuals, "p")
plt.axhline(y=0, color='#aaaaaa', linestyle=":")
plt.axvline(x=0, color='#aaaaaa', linestyle=":")
```

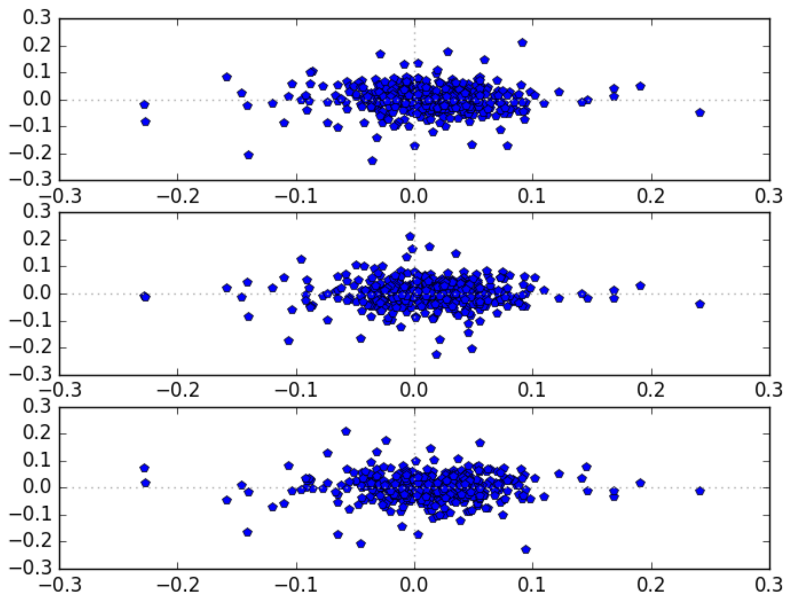

Figure 5: Residual Plots

```
# Fit plots
plt.figure()
plt.subplot(311)
plt.plot(ret_lag_1[serie], predictedY, "p")
plt.plot(ret_lag_1[serie], y, "p")
plt.axhline(y=0, color='#aaaaaa', linestyle=":")
plt.axvline(x=0, color='#aaaaaa', linestyle=":")

plt.subplot(312)
plt.plot(ret_lag_2[serie], predictedY, "p")
plt.plot(ret_lag_2[serie], y, "p")
plt.axhline(y=0, color='#aaaaaa', linestyle=":")
plt.axvline(x=0, color='#aaaaaa', linestyle=":")

plt.subplot(313)
plt.plot(ret_lag_3[serie], predictedY, "p")
plt.plot(ret_lag_3[serie], y, "p")
plt.axhline(y=0, color='#aaaaaa', linestyle=":")
plt.axvline(x=0, color='#aaaaaa', linestyle=":")
```

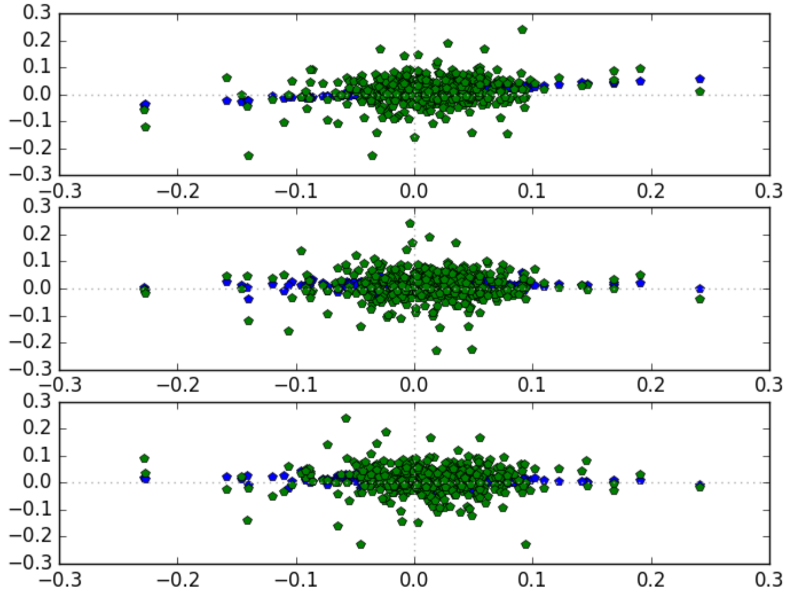

Figure 6: Line Fit Plots

```
#==================================================================
# ARIMA(1, 1, 0)
#==================================================================
serie = "France"
returns = dataSeries[serie]

# ARIMA(1, 1, 0)
def arima1(params):
    rho, theta = params

    n = np.size(returns)
    ma = np.zeros(n)
    ma[0] = 0

    for i in range(1, n):
        ma[i] = returns[i] - (rho * returns[i-1] + theta * ma[i-1])

    return ma

# Squared errors function to be minimized
def squaredError(params):
    return (arima1(params)**2).sum()

initParams = [0, 0]

results = optimize.minimize(squaredError, initParams, method='SLSQP')
print(results.x)
```

$$\begin{bmatrix} -0.206 & 0.337 \end{bmatrix}$$

```python
# Stationarity & invertibility
bnds =  ((-0.9999999999, 0.9999999999), (-0.9999999999, 0.9999999999))
results = optimize.minimize(squaredError, initParams, bounds=bnds,
                            method='SLSQP')
print(results.x)
```

$$\begin{bmatrix} -0.206 & 0.337 \end{bmatrix}$$

```python
#==============================================================================
# MA(2) using MLE
#==============================================================================
serie = "Spain"
returns = dataSeries[serie]

def epsilon(params):
    q, theta1, theta2 = params

    n = np.size(returns)
    epsilons = np.zeros(n)

    for i in range(0, n):
        epsilons[i] = returns[i] - q - epsilons[i - 1] * theta1 - \
            epsilons[i - 2] * theta2

    return epsilons

def logLikelihood(params):
    epsilons = epsilon(params)
    return -((-1/2*np.log(2*np.pi*np.var(epsilons)) \
        - 1/2*epsilons**2/np.var(epsilons)).sum())

initParams = [np.average(returns), 0, 0]

results = optimize.minimize(logLikelihood, initParams, method='SLSQP')
print(results.x)
```

$$\begin{bmatrix} 0.011 & 0.127 & -0.021 \end{bmatrix}$$

3 Simultaneous Equations

```python
"""
@author: dtoppo
Simultaneous Equations
"""

import scipy as sp
import pandas as pd
from statsmodels.formula.api import ols

# Import dataserie from Excel
dataFile = "data_python_HW_2017.xlsx"
xslx = pd.ExcelFile(dataFile)

sheetName = "Simult Equ"
dataSeries = xslx.parse(sheetName)

#==============================================================================
# Descriptive statistics
#==============================================================================
def descriptiveStats(serieName):

    serie = dataSeries[serieName]

    mean = sp.mean(serie)
    stdErr = sp.stats.sem(serie)
    median = sp.median(serie)
    mode = sp.stats.mode(serie)
    stdDev = sp.std(serie, ddof=1)
    var = sp.var(serie, ddof=1)
    kurtosis = sp.stats.kurtosis(serie, fisher=True)
    skewness = sp.stats.skew(serie)
    minVal = sp.amin(serie)
    maxVal = sp.amax(serie)
    rangeVal = maxVal - minVal
    sumVal = sp.sum(serie)
    count = sp.count_nonzero(serie)
    confInterval = sp.stats.norm.interval(0.05, loc=mean, scale=stdDev)

    print("*****************************************")
    print("--- %s ---" %serieName)
    print("Mean: %.2f" %mean)
    print("Standard Error: %.2f" %stdErr)
    print("Median: %.2f" %median)
    print("Mode: %.2f" %mode[0])
    print("Standard Deviation: %.2f" %stdDev)
    print("Sample Variance: %.2f" %var)
    print("Kurtosis: %.2f" %kurtosis)
    print("Skewness: %.2f" %skewness)
    print("Range: %.2f" %rangeVal)
    print("Minimum: %.2f" %minVal)
    print("Maximum: %.2f" %maxVal)
    print("Sum: %.2f" %sumVal)
    print("Count: %.2f" %count)
    print("Confidence Level (95%%): %.2f" %(confInterval[1] - confInterval[0]))
    print("*****************************************")
```

```python
for serieName in dataSeries:
    descriptiveStats(serieName)
```

```
*****************************************
--- Returns ---
Mean: 0.01
Standard Error: 0.00
Median: 0.02
Mode: 0.02
Standard Deviation: 0.04
Sample Variance: 0.00
Kurtosis: 3.40
Skewness: -1.00
Range: 0.36
Minimum: -0.23
Maximum: 0.13
Sum: 2.52
Count: 253.00
Confidence Level (95%): 0.01
*****************************************
```

Figure 7: Summary Statistics for Returns

```python
#==============================================================================
# Regressions
#==============================================================================
data = pd.concat([dataSeries["Returns"], dataSeries["dprod"], dataSeries["dspread"],
    dataSeries["rterm"], dataSeries["dcredit"], dataSeries["dmoney"]], axis=1)
returns_model = ols("Returns ~ dprod + dspread + rterm + dcredit + dmoney", data).fit()
print(returns_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Returns   R-squared:                       0.010
Model:                            OLS   Adj. R-squared:                 -0.011
Method:                 Least Squares   F-statistic:                    0.4745
Date:                Thu, 25 May 2017   Prob (F-statistic):              0.795
Time:                        14:52:11   Log-Likelihood:                 433.79
No. Observations:                 253   AIC:                            -855.6
Df Residuals:                     247   BIC:                            -834.4
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0113      0.005      2.474      0.014       0.002       0.020
dprod         -0.0015      0.002     -0.994      0.321      -0.005       0.002
dspread       -0.0396      0.042     -0.943      0.346      -0.122       0.043
rterm          0.0034      0.010      0.320      0.749      -0.017       0.024
dcredit    -1.706e-07   5.06e-07     -0.337      0.736   -1.17e-06    8.25e-07
dmoney        -0.0012      0.002     -0.546      0.586      -0.006       0.003
==============================================================================
Omnibus:                       57.083   Durbin-Watson:                   1.952
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              151.874
Skew:                          -1.006   Prob(JB):                     1.05e-33
Kurtosis:                       6.219   Cond. No.                     1.37e+05
==============================================================================
```

Figure 8: Reduced-form equation for Returns

```python
data = pd.concat([dataSeries["Inflation"], dataSeries["dprod"], dataSeries["dspread"],
    dataSeries["rterm"], dataSeries["dcredit"], dataSeries["dmoney"]], axis=1)
inflation_model = ols("Inflation ~ dprod + dspread + rterm + dcredit + dmoney", data).fit()
print(inflation_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Inflation   R-squared:                       0.219
Model:                            OLS   Adj. R-squared:                  0.203
Method:                 Least Squares   F-statistic:                     13.88
Date:                Thu, 25 May 2017   Prob (F-statistic):           5.93e-12
Time:                        14:53:31   Log-Likelihood:                -841.69
No. Observations:                 253   AIC:                             1695.
Df Residuals:                     247   BIC:                             1717.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    214.3211      0.709    302.171      0.000     212.924     215.718
dprod          0.0109      0.241      0.045      0.964      -0.463       0.485
dspread        7.1308      6.495      1.098      0.273      -5.662      19.924
rterm         -1.0572      1.624     -0.651      0.516      -4.255       2.141
dcredit        0.0006   7.82e-05      8.113      0.000       0.000       0.001
dmoney         0.1552      0.346      0.448      0.654      -0.527       0.837
==============================================================================
Omnibus:                        1.397   Durbin-Watson:                   0.242
Prob(Omnibus):                  0.497   Jarque-Bera (JB):                1.281
Skew:                          -0.025   Prob(JB):                        0.527
Kurtosis:                       2.655   Cond. No.                     1.37e+05
==============================================================================
```

Figure 9: Reduced-form equation for Inflation

The importance of using 2SLS:

If we estimate a SSSE by OLS, OLS estimators will suffer from the so-called Simultaneous Equation Bias, arising from the presence of correlation between independent variables and regressors in some equations of the SSSE. The errors will be correlated with the regressors and this violates an assumption of the regression framework. Applying standard ordinary least squares (OLS) under these circumstances results in inconsistent estimates.
To remedy this problem we can apply 2SLS: 2SLS imply replacing endogenous variables on the RHS with fitted OLS values. Even if 2SLS estimators are still biased, the advantage of 2SLS estimators over OLS is that they are consistent.

```
#==========================================================================
# 2SLS Model with predicted returns and predicted inflation
#==========================================================================
predicted_returns = pd.DataFrame({"PredictedReturns" : returns_model.predict()},
    index=dataSeries.index)
preidcted_inflation = pd.DataFrame({"PredictedInflation": inflation_model.predict()},
    index=dataSeries.index)

data = pd.concat([dataSeries["Inflation"], predicted_returns, dataSeries["dcredit"],
    dataSeries["dprod"], dataSeries["dmoney"]], axis=1)
inflation_model = ols("Inflation ~ PredictedReturns + dcredit + dprod + dmoney", data).fit()
print(inflation_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Inflation   R-squared:                       0.219
Model:                            OLS   Adj. R-squared:                  0.206
Method:                 Least Squares   F-statistic:                     17.39
Date:                Thu, 25 May 2017   Prob (F-statistic):           1.36e-12
Time:                        14:54:29   Log-Likelihood:                -841.73
No. Observations:                 253   AIC:                             1693.
Df Residuals:                     248   BIC:                             1711.
Df Model:                           4
Covariance Type:            nonrobust
====================================================================================
                       coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------
Intercept          216.5030      1.946    111.274      0.000     212.671     220.335
PredictedReturns  -192.9712    156.389     -1.234      0.218    -500.991     115.048
dcredit              0.0006   8.36e-05      7.209      0.000       0.000       0.001
dprod               -0.2824      0.333     -0.847      0.398      -0.939       0.374
dmoney              -0.0948      0.372     -0.255      0.799      -0.827       0.638
==============================================================================
Omnibus:                        1.328   Durbin-Watson:                   0.242
Prob(Omnibus):                  0.515   Jarque-Bera (JB):                1.233
Skew:                          -0.018   Prob(JB):                        0.540
Kurtosis:                       2.660   Cond. No.                     3.30e+06
==============================================================================
```

Figure 10: Inflation

```
data = pd.concat([dataSeries["Returns"], preidcted_inflation, dataSeries["dprod"],
    dataSeries["dspread"], dataSeries["rterm"]], axis=1)
inflation_model = ols("Returns ~ PredictedInflation + dprod + dspread + rterm", data).fit()
print(inflation_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Returns   R-squared:                       0.008
Model:                            OLS   Adj. R-squared:                 -0.008
Method:                 Least Squares   F-statistic:                    0.5258
Date:                Thu, 25 May 2017   Prob (F-statistic):              0.717
Time:                        15:25:47   Log-Likelihood:                 433.65
No. Observations:                 253   AIC:                            -857.3
Df Residuals:                     248   BIC:                            -839.6
Df Model:                           4
Covariance Type:            nonrobust
======================================================================================
                         coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------
Intercept              0.0787      0.173      0.454      0.650      -0.262       0.420
PredictedInflation    -0.0003      0.001     -0.395      0.693      -0.002       0.001
dprod                 -0.0016      0.002     -1.069      0.286      -0.005       0.001
dspread               -0.0360      0.042     -0.850      0.396      -0.119       0.047
rterm                  0.0020      0.010      0.193      0.847      -0.018       0.022
==============================================================================
Omnibus:                       57.420   Durbin-Watson:                   1.954
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              157.719
Skew:                          -0.999   Prob(JB):                     5.65e-35
Kurtosis:                       6.312   Cond. No.                     1.37e+04
==============================================================================
```

Figure 11: Returns

Except for dcredit in the Inflation equation and intercept in Returns Equation, none of the parameters is even close to statistical significance. The conclusion is that the inflation fitted value term is not significant in the stock return equation, therefore inflation can be considered exogenous for stock returns. The same happens in inflation equation: fitted stock return term is not significant in the inflation equation, suggesting that stock returns are exogenous.

4 VAR & Cointegration with Trend

```python
"""
@author: dtoppo
VAR & Cointegration with Trend
"""

import pandas as pd
import numpy as np
from statsmodels.formula.api import ols
from statsmodels.tsa.vector_ar import var_model

# Import dataserie from Excel
dataFile = "data_python_HW_2017.xlsx"
xslx = pd.ExcelFile(dataFile)

sheetName = "VAR & Cointegration trend"
dataSeries = xslx.parse(sheetName)

#==============================================================================
# Stationarity testing
#==============================================================================
diff = dataSeries.diff()
lag = dataSeries.shift()
difflag = diff.shift()

dependentVar = "WTI"

trend = pd.DataFrame(np.arange(0, np.size(dataSeries[dependentVar]), 1),
  index=dataSeries.index)
trend = trend.shift(1)

data = pd.concat([diff[dependentVar], lag[dependentVar],
  difflag[dependentVar], trend], axis=1)
data.columns=["diff", "lag", "difflag", "trend"]
model = ols("diff ~ lag + difflag + trend", data[2:]).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   diff   R-squared:                       0.080
Model:                            OLS   Adj. R-squared:                  0.073
Method:                 Least Squares   F-statistic:                     11.28
Date:                Thu, 25 May 2017   Prob (F-statistic):           4.15e-07
Time:                        15:31:15   Log-Likelihood:                -1139.9
No. Observations:                 393   AIC:                             2288.
Df Residuals:                     389   BIC:                             2304.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.1770      0.448      0.395      0.693      -0.703       1.057
lag           -0.0336      0.012     -2.801      0.005      -0.057      -0.010
difflag        0.2667      0.049      5.426      0.000       0.170       0.363
trend          0.0063      0.003      1.995      0.047    9.17e-05       0.012
==============================================================================
Omnibus:                       71.155   Durbin-Watson:                   2.039
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              556.260
Skew:                          -0.487   Prob(JB):                    1.62e-121
Kurtosis:                       8.747   Cond. No.                         466.
==============================================================================
```

Figure 12: Crude Oil WTI

```
dependentVar = "USDX"
data = pd.concat([diff[dependentVar], lag[dependentVar],
  difflag[dependentVar], trend], axis=1)
data.columns=["diff", "lag", "difflag", "trend"]
model = ols("diff ~ lag + difflag + trend", data[2:]).fit()
print(model.summary())
```

```
                           OLS Regression Results
==============================================================================
Dep. Variable:                   diff   R-squared:                       0.010
Model:                            OLS   Adj. R-squared:                  0.002
Method:                 Least Squares   F-statistic:                     1.281
Date:                Thu, 25 May 2017   Prob (F-statistic):              0.281
Time:                        15:32:25   Log-Likelihood:                -921.17
No. Observations:                 393   AIC:                             1850.
Df Residuals:                     389   BIC:                             1866.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      1.4798      1.121      1.320      0.188      -0.724       3.684
lag           -0.0158      0.010     -1.630      0.104      -0.035       0.003
difflag        0.0095      0.051      0.186      0.852      -0.090       0.109
trend         -0.0002      0.001     -0.116      0.908      -0.003       0.003
==============================================================================
Omnibus:                        9.892   Durbin-Watson:                   2.001
Prob(Omnibus):                  0.007   Jarque-Bera (JB):               17.972
Skew:                           0.007   Prob(JB):                     0.000125
Kurtosis:                       4.048   Cond. No.                     2.11e+03
==============================================================================
```

Figure 13: The U.S. Dollar Index(USDX)

Because for both prices and dividends, the ADF t Stat fails to exceed any ADF critical value (-3.42 for 5% confidence interval), we fail to reject the null hypothesis of non-stationarity of the two series.

Please note that the usual p-value (next to t Stat) doesn't make any sense here. It is the value if we havetdistribution. Under the unit root process, we haveTaudistribution. So, we have to compare computedtStat with the critical values (given before) which were computed using theTaudistribution.

```
#==========================================================================
# Stationarity of difference
#==========================================================================
diffdiff = diff.diff()
diffdifflag = diffdiff.shift()

dependentVar = "WTI"

trend = pd.DataFrame(np.arange(0, np.size(dataSeries[dependentVar]), 1),
  index=dataSeries.index)
trend = trend.shift(2)

data = pd.concat([diffdiff[dependentVar], difflag[dependentVar],
  diffdifflag[dependentVar], trend], axis=1)
data.columns=["diffdiff", "difflag", "diffdifflag", "trend"]
model = ols("diffdiff ~ difflag + diffdifflag + trend", data[3:]).fit()
```

```python
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               diffdiff   R-squared:                       0.378
Model:                            OLS   Adj. R-squared:                  0.373
Method:                 Least Squares   F-statistic:                     78.47
Date:                Thu, 25 May 2017   Prob (F-statistic):           1.08e-39
Time:                        15:33:16   Log-Likelihood:                -1140.9
No. Observations:                 392   AIC:                             2290.
Df Residuals:                     388   BIC:                             2306.
Df Model:                           3
Covariance Type:            nonrobust
===============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept      0.1219      0.452      0.270      0.788      -0.767       1.011
difflag       -0.7134      0.062    -11.449      0.000      -0.836      -0.591
diffdifflag   -0.0518      0.051     -1.020      0.308      -0.152       0.048
trend         -0.0006      0.002     -0.303      0.762      -0.005       0.003
==============================================================================
Omnibus:                      100.824   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              757.230
Skew:                          -0.861   Prob(JB):                   3.71e-165
Kurtosis:                       9.588   Cond. No.                         454.
==============================================================================
```

Figure 14: Crude Oil WTI

```python
dependentVar = "USDX"
data = pd.concat([diffdiff[dependentVar], difflag[dependentVar],
  diffdifflag[dependentVar], trend], axis=1)
data.columns=["diffdiff", "difflag", "diffdifflag", "trend"]
model = ols("diffdiff ~ difflag + diffdifflag + trend", data[3:]).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               diffdiff   R-squared:                       0.506
Model:                            OLS   Adj. R-squared:                  0.502
Method:                 Least Squares   F-statistic:                     132.6
Date:                Thu, 25 May 2017   Prob (F-statistic):           3.92e-59
Time:                        15:33:59   Log-Likelihood:                -918.02
No. Observations:                 392   AIC:                             1844.
Df Residuals:                     388   BIC:                             1860.
Df Model:                           3
Covariance Type:            nonrobust
===============================================================================
                  coef    std err          t      P>|t|      [0.025      0.975]
-------------------------------------------------------------------------------
Intercept      -0.2825      0.257     -1.100      0.272      -0.788       0.223
difflag        -0.8894      0.071    -12.473      0.000      -1.030      -0.749
diffdifflag    -0.1105      0.050     -2.191      0.029      -0.210      -0.011
trend           0.0012      0.001      1.020      0.308      -0.001       0.003
==============================================================================
Omnibus:                       13.637   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               28.248
Skew:                          -0.094   Prob(JB):                     7.34e-07
Kurtosis:                       4.302   Cond. No.                         456.
==============================================================================
```

Figure 15: The U.S. Dollar Index(USDX)

For both Oil and USDX, the ADF t Stat exceeds the ADF critical value (5%, -3.42), we reject the null hypothesis of non-stationarity; we then conclude that Oil and USDX are I(1).

```
#==============================================================================
# OLS Oil vs USDX
# The - sign can be used to remove columns/variables. For instance,
# we can remove the intercept from a model by adding "-1" to the formula
#==============================================================================
model = ols("WTI ~ USDX -1", dataSeries).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    WTI   R-squared:                       0.567
Model:                            OLS   Adj. R-squared:                  0.566
Method:                 Least Squares   F-statistic:                     516.3
Date:                Thu, 25 May 2017   Prob (F-statistic):           1.19e-73
Time:                        15:34:45   Log-Likelihood:                -1950.2
No. Observations:                 395   AIC:                             3902.
Df Residuals:                     394   BIC:                             3906.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
USDX           0.3991      0.018     22.723      0.000       0.365       0.434
==============================================================================
Omnibus:                       52.586   Durbin-Watson:                   0.021
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               71.087
Skew:                           1.030   Prob(JB):                     3.66e-16
Kurtosis:                       2.732   Cond. No.                         1.00
==============================================================================
```

Figure 16: OLS for Cointegration

```
#==============================================================================
# Lagged cointegration
#==============================================================================
residuals = model.wresid.to_frame()
diffResiduals = residuals.diff()
laggedResiduals = residuals.shift()

diffResiduals.columns = ["diffResiduals"]
laggedResiduals.columns = ["laggedResiduals"]

data = pd.concat([diffResiduals, laggedResiduals], axis=1)
model = ols("diffResiduals ~ laggedResiduals -1", data[1:]).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          diffResiduals   R-squared:                       0.006
Model:                            OLS   Adj. R-squared:                  0.003
Method:                 Least Squares   F-statistic:                     2.249
Date:                Thu, 25 May 2017   Prob (F-statistic):              0.134
Time:                        15:44:25   Log-Likelihood:                -1185.6
No. Observations:                 394   AIC:                             2373.
Df Residuals:                     393   BIC:                             2377.
Df Model:                           1
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
laggedResiduals    -0.0110      0.007     -1.500      0.134      -0.025       0.003
==============================================================================
Omnibus:                      106.472   Durbin-Watson:                   1.524
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1018.590
Skew:                          -0.839   Prob(JB):                    6.55e-222
Kurtosis:                      10.696   Cond. No.                         1.00
==============================================================================
```

Figure 17: Summary Output

The t Stat is 1.49. The 5% critical value for the model with no intercept is 3.37.

The t Stat falls within the non-rejection region and the null hypothesis of no cointegration is NOT rejected.

Of course this is not Johansen's method and it appears to be primitive when compared to it.

```
#==============================================================================
# VAR(2)
# First using regression
#==============================================================================
difflaglag = difflag.shift()

data = pd.concat([diff["WTI"], difflag["WTI"], difflaglag["WTI"],
  difflag["USDX"], difflaglag["USDX"]], axis=1)
data.columns=["WTI", "difflagWTI", "difflaglagWTI", "difflagUSDX", "difflaglagUSDX"]
model = ols("WTI ~ difflagWTI + difflaglagWTI + difflagUSDX + difflaglagUSDX",
  data[3:]).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    WTI   R-squared:                       0.069
Model:                            OLS   Adj. R-squared:                  0.060
Method:                 Least Squares   F-statistic:                     7.189
Date:                Thu, 25 May 2017   Prob (F-statistic):           1.36e-05
Time:                        15:45:54   Log-Likelihood:                -1139.8
No. Observations:                 392   AIC:                             2290.
Df Residuals:                     387   BIC:                             2309.
Df Model:                           4
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Intercept        0.0019      0.225      0.008      0.993      -0.441       0.445
difflagWTI       0.2475      0.052      4.761      0.000       0.145       0.350
difflaglagWTI    0.0377      0.052      0.722      0.471      -0.065       0.140
difflagUSDX      0.0786      0.091      0.860      0.390      -0.101       0.258
difflaglagUSDX  -0.1113      0.091     -1.217      0.224      -0.291       0.069
==============================================================================
Omnibus:                      102.077   Durbin-Watson:                   2.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              724.669
Skew:                          -0.894   Prob(JB):                    4.37e-158
Kurtosis:                       9.416   Cond. No.                         5.18
==============================================================================
```

Figure 18: Crude Oil WTI

```
data = pd.concat([diff["USDX"], difflag["WTI"], difflaglag["WTI"],
  difflag["USDX"], difflaglag["USDX"]], axis=1)
data.columns=["USDX", "difflagWTI", "difflaglagWTI", "difflagUSDX", "difflaglagUSDX"]
model = ols("USDX ~ difflagWTI + difflaglagWTI + difflagUSDX + difflaglagUSDX",
  data[3:]).fit()
print(model.summary())
```

OLS Regression Results

Dep. Variable:	USDX	R-squared:	0.021
Model:	OLS	Adj. R-squared:	0.011
Method:	Least Squares	F-statistic:	2.041
Date:	Thu, 25 May 2017	Prob (F-statistic):	0.0880
Time:	15:48:31	Log-Likelihood:	-916.99
No. Observations:	392	AIC:	1844.
Df Residuals:	387	BIC:	1864.
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.0557	0.128	-0.436	0.663	-0.307	0.195
difflagWTI	-0.0503	0.029	-1.708	0.088	-0.108	0.008
difflaglagWTI	0.0236	0.030	0.798	0.425	-0.035	0.082
difflagUSDX	-0.0152	0.052	-0.293	0.769	-0.117	0.087
difflaglagUSDX	0.1218	0.052	2.351	0.019	0.020	0.224

Omnibus:	15.691	Durbin-Watson:	2.004
Prob(Omnibus):	0.000	Jarque-Bera (JB):	31.726
Skew:	-0.175	Prob(JB):	1.29e-07
Kurtosis:	4.349	Cond. No.	5.18

Figure 19: The U.S. Dollar Index(USDX)

The series are not cointegrated, therefore we estimate a VAR(2) on their first differences (since both the series are non-stationary in levels).

Using a 5% significance level, DiffOil is significantly related to its own past values, while DiffUSDX is significantly related to Lag2DiffUDSX.

5 GARCH

```python
"""
@author: dtoppo
GARCH
"""

import numpy as np
from scipy import optimize
import time
import pandas as pd

# Import dataserie from Excel
dataFile = "data_python_HW_2017.xlsx"
xslx = pd.ExcelFile(dataFile)

sheetName = "GARCH"
dataSeries = xslx.parse(sheetName)

#==============================================================================
# MAX Likelihood function
#==============================================================================
def minLikelihood(params, data):
    s = GARCH(params, data)
    logL = -((-1/2 * np.log(2*np.pi) - 1/2 * np.log(s) - 1/2 * data**2/s).sum())
    return logL

#==============================================================================
# MAX Likelihood function for Variance Targeting
#==============================================================================
def minLikelihoodVarTarget(params, data):
    s = GARCHTargetVar(params, data)
    logL = -((-1/2 * np.log(2*np.pi) - 1/2 * np.log(s) - 1/2 * data**2/s).sum())
    return logL

#==============================================================================
# GARCH(1,1)
#==============================================================================
def GARCH(param, data):
    alpha, beta, omega = param
    s = np.zeros(len(data))
    s[0] = np.var(data, ddof=1)

    for i in range(1, len(data)):
        s[i] = omega + alpha*data[i-1]**2 + beta*(s[i-1])  # GARCH(1,1) model
    return s

#==============================================================================
# GARCH(1,1) with Variance Targeting
#==============================================================================
def GARCHTargetVar(param, data):
    alpha, beta, omega = param
    s = np.zeros(len(data))
    s[0] = np.var(data, ddof=1)
    omega=s[0]*(1-alpha-beta)

    for i in range(1, len(data)):
```

```
        s[i] = omega + alpha*data[i-1]**2 + beta*(s[i-1])  # GARCH(1,1) model
    return s

#==============================================================================
# Optimizer
#==============================================================================
def maximizeMLE(initParams, data):
    # Constraints
    # 1 - (alpha*(1 + theta^2)+beta) >= 0
    def persistenceIndexConstraint(params):
        omega, alpha, beta = params
        return 1 - (alpha + beta)

    cons = {'type' : 'ineq', 'fun' : persistenceIndexConstraint}

    # Could be also a lambda expression
    # cons = {'type' : 'ineq', 'fun' : lambda params : 1 - np.sum(params)}

    # Bounds
    # 0 <= parameters <= 1
    bnds = ((0, 1), (0, 1), (0, 1))

    # Run the minimizer
    results = optimize.minimize(minLikelihood, initParams, data, method='SLSQP',
        bounds=bnds, constraints=cons)

    return results.x


#==============================================================================
# Optimizer Variance Targeting
#==============================================================================
def maximizeMLEVarTarget(initParams, data):
    # Constraints
    # 1 - (alpha*(1 + theta^2)+beta) >= 0
    def persistenceIndexConstraint(params):
        omega, alpha, beta = params
        return 1 - (alpha + beta)

    cons = {'type' : 'ineq', 'fun' : persistenceIndexConstraint}

    # Bounds
    # 0 <= parameters <= 1
    bnds = ((0, 1), (0, 1), (0, 1))

    # Run the minimizer
    results = optimize.minimize(minLikelihoodVarTarget, initParams, data, method='SLSQP',
        bounds=bnds, constraints=cons)

    return results.x

#==============================================================================
# Main routine
#==============================================================================
def main() :

    data = dataSeries["Germany"]
```

```python
# Initial parameter guesses (alpha, beta, omega)
initParams = [0.0832, 0.8759, 0.0001]

# GARCH
start = time.time()
results = maximizeMLE(initParams, data)
end   = time.time()

# Print the results
print("Alpha: %.6f\nBeta: %.6f\nOmega: %.6f" % (results[0], results[1], results[2]))
print("Computed in %.2f secs" % (end - start))
print()
```

Alpha	0.1480
Beta	0.8040
Omega	0.0002

The persistence index $(0.148 + 0.804 = 0.952)$ is lower than 1. A high persistence implies that shocks, which may push variance away from its long-run average, will persist for a long time.

```python
# GARCH with Variance Targeting
start = time.time()
results = maximizeMLEVarTarget(initParams, data)
end   = time.time()

# Print the results
print("Alpha: %.6f\nBeta: %.6f\nOmega: %.6f" % (results[0], results[1], results[2]))
print("Computed in %.2f secs" % (end - start))
```

Alpha	0.1374
Beta	0.8062
Omega	0.0002

6 GARCH with Leverage

```
"""
@author: dtoppo
GARCH with Leverage
"""

import pandas as pd
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt
from statsmodels.formula.api import ols

# Import dataserie from Excel
dataFile = "data_python_HW_2017.xlsx"
xslx = pd.ExcelFile(dataFile)

sheetName = "GARCH with Leverage"
dataSeries = xslx.parse(sheetName)

#==============================================================================
# MAX Likelihood function
#==============================================================================
def minLikelihood(params, data):
    s = GARCHLeveraged(params, data)
    logL = -((-1/2 * np.log(2*np.pi) - 1/2 * np.log(s) - 1/2 * data**2/s).sum())
    return logL

#==============================================================================
# GARCH(1,1) with leverage model
#==============================================================================
def GARCHLeveraged(params, data):
    alpha, beta, omega, theta = params

    s = np.zeros(np.size(data))
    s[0] = np.var(data, ddof=1)

    for i in range(1, np.size(data)):
        s[i] = omega + alpha*((data[i-1] - theta * np.sqrt(s[i-1]))**2) + beta*(s[i-1])
    return s

#==============================================================================
# Main routine
#==============================================================================
def maximizeMLE(initParams, data):
    alpha, beta, omega, theta = initParams

    # Constraints
    # 1 - (alpha*(1 + theta^2)+beta) >= 0
    def persistenceIndexConstraint(params):
        return 1 - (alpha*(1 + theta**2) + beta)

    cons = {'type' : 'ineq', 'fun' : persistenceIndexConstraint}

    # Could be also a lambda expression
    # cons = {'type' : 'ineq',
    #   'fun' : lambda params : 1 - (params[0]*(1 + params[2]**2) + params[1])}
```

```python
    # Bounds
    # 0 <= parameters <= 1
    bnds = ((0, 1), (0, 1), (0, 1), (0, 1))

    # Run the minimizer - SLSQP
    # SLSQP uses Sequential Least SQuares Programming to minimize a
    # function of several variables
    # with any combination of bounds, equality and inequality constraints.
    results = optimize.minimize(minLikelihood, initParams, data,
        method='SLSQP', bounds=bnds, constraints=cons)
    return results.x

#==============================================================================
# Lagged autocorrelation function
#==============================================================================
def laggedSquaredAutoCorrelation(data, nbLags):

    rSquared = data**2

    autocorrel = np.zeros(nbLags)
    baseSquaredArray = pd.DataFrame(rSquared[0: np.size(rSquared) - 1])

    for i in np.arange(0, nbLags, 1):
        lagged = baseSquaredArray.shift(-i-1).fillna(0)
        autocorrel[i] = np.corrcoef(baseSquaredArray[0].values, lagged[0].values)[0,1]

    return autocorrel

#==============================================================================
# Squared Standard auto correlation
#==============================================================================
def laggedSquaredStandardAutoCorrelation(data, nbLags):

    rSquared = data**2

    autocorrel = np.zeros(nbLags)

    # Initial parameter guesses (alpha, beta, omega, theta)
    initParams = [0.2, 0.8, 0.0, 0.0]

    garchParams = maximizeMLE(initParams, data)
    garchVol = GARCHLeveraged(garchParams, data)

    squaredStandardizedRet = rSquared / garchVol

    baseSquaredArray = squaredStandardizedRet[0: np.size(squaredStandardizedRet) - 1]
    lagged = baseSquaredArray

    for i in np.arange(0, nbLags, 1):
        lagged = np.delete(lagged, 0)
        lagged = np.append(lagged, 0)
        autocorrel[i] = np.corrcoef(baseSquaredArray, lagged)[0,1]

    return autocorrel

#==============================================================================
```

```python
# Main routine
#==========================================================================
def main():

    seriesName = "Italy"

    # Initial parameter guesses (alpha, beta, omega, theta)
    initParams = [0.1, 0.85, 0.000005, 0.0]

    results = maximizeMLE(initParams, dataSeries[seriesName].values)

    # Print the results
    print("\nAlpha: %.6f\nBeta: %.6f\nOmega: %.6f\nTheta: %.6f" % (results[0], results[1],
        results[2], results[3]))
```

Alpha	0.2069
Beta	0.5978
Omega	0.0010
Theta	0.1129

Leverage Effect: a negative return on a stock implies a drop in the equity value, which implies that the company becomes more highly levered and thus riskier (assuming the level of debt stays constant).

```python
    #======================================================================
    # Plot sample autocorrelation coefficients at lags 1 through 100
    #======================================================================
    nbLags = 100

    squaredAutocorrel = laggedSquaredAutoCorrelation(dataSeries[seriesName].values, nbLags)
    squaredStandardAutoCorrel =
        laggedSquaredStandardAutoCorrelation(dataSeries[seriesName].values, nbLags)
    lag = np.arange(1, nbLags + 1, 1)

    barlettLowerBand = np.array(np.ones(100))
    barlettLowerBand = barlettLowerBand * (-1.96 / np.size(dataSeries[seriesName])**0.5)
    barlettUpperBand = -barlettLowerBand

    # Plot data
    plt.plot(lag, squaredAutocorrel)
    plt.plot(lag, squaredStandardAutoCorrel)

    # Plot bands
    plt.plot(barlettLowerBand, '--r')
    plt.plot(barlettUpperBand, '--r')
```

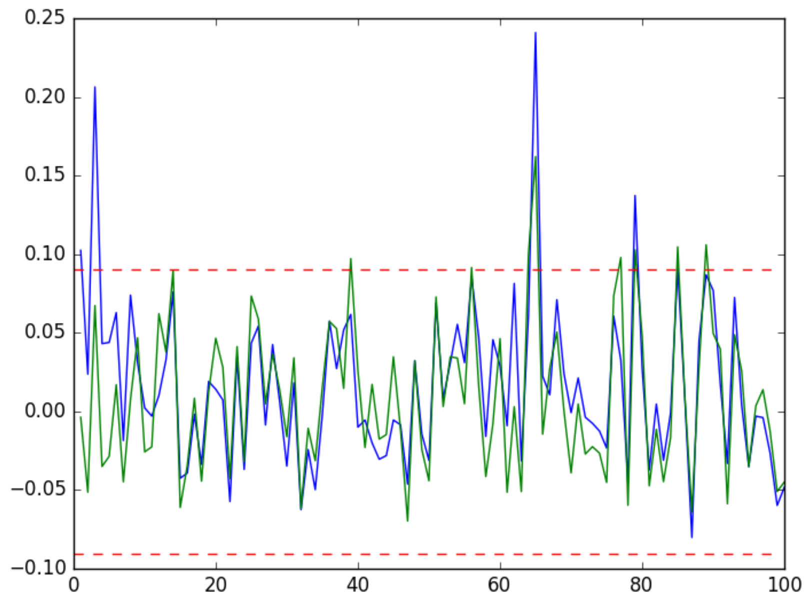

Figure 20: Sample Autocorrelation

```
#==================================================================
# Regression of daily squared returns on variance
#==================================================================
forcastedVar = GARCHLeveraged(results, dataSeries[seriesName].values)
squaredReturns = dataSeries[seriesName].values**2

sqretDataFrame = pd.DataFrame(data = squaredReturns,
    index = dataSeries.index, columns = ["sqRet"])
garchDataFrame = pd.DataFrame(data = forcastedVar,
    index = dataSeries.index, columns = ["garchVar"])

data = pd.concat([sqretDataFrame, garchDataFrame], axis=1)
model = ols("sqRet ~ garchVar", data).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  sqRet   R-squared:                       0.021
Model:                            OLS   Adj. R-squared:                  0.019
Method:                 Least Squares   F-statistic:                     10.17
Date:                Thu, 25 May 2017   Prob (F-statistic):            0.00152
Time:                        16:33:22   Log-Likelihood:                 1524.6
No. Observations:                 468   AIC:                            -3045.
Df Residuals:                     466   BIC:                            -3037.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0023      0.001      2.343      0.020       0.000       0.004
garchVar       0.5351      0.168      3.189      0.002       0.205       0.865
==============================================================================
Omnibus:                      458.729   Durbin-Watson:                   2.032
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            14357.875
Skew:                           4.393   Prob(JB):                         0.00
Kurtosis:                      28.673   Cond. No.                         389.
==============================================================================
```

Figure 21: Summary Output

The intercept is significantly different from zero at a 5% confidence level; the slope is around 0.54 and this estimate is significantly different from 1. Therefore the GARCH(1,1) model with leverage offers a poor variance model.

7 QQ Plots

```
"""
@author: dtoppo
QQ Plots
"""

import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
import pandas as pd

from GARCHLeverage import maximizeMLE, GARCHLeveraged

# Import dataserie from Excel
dataFile = "data_python_HW_2017.xlsx"
xslx = pd.ExcelFile(dataFile)

sheetName = "QQ Plot"
dataSeries = xslx.parse(sheetName)


#==============================================================================
# Get descriptive statistics
#==============================================================================
def descriptiveStats(data):
    nbObs = len(data)
    mean = np.mean(data)
    stdev = np.std(data, ddof=1)
    skewness = sp.stats.skew(data, bias=False)
    kurtosis = sp.stats.kurtosis(data, bias=False)

    result = {"nbObs" : nbObs, "mean" : mean, "stdev" : stdev,
        "skewness" : skewness, "kurtosis" : kurtosis}
    return result

#==============================================================================
# Descriptive statistics printing helper
#==============================================================================
def printDescriptiveStats(stats):
    print("Descriptive Statistics")
    print("Number of obs.\t%d" %stats["nbObs"])
    print("Mean\t\t%.4f%%" %(stats["mean"] * 100))
    print("Std. Deviation\t%.4f%%" %(stats["stdev"] * 100))
    print("Skewness\t%.4f" %stats["skewness"])
    print("Kurtosis\t%.4f" %stats["kurtosis"])

#==============================================================================
# QQ Plot function
#==============================================================================
def qqPlot(normalizedReturns):

    nbObs = len(normalizedReturns)

    sortedNormalizedReturns = np.sort(normalizedReturns)
    normalizedQuantiles = sp.stats.norm.ppf((np.arange(1, nbObs+1) - 0.5)/nbObs)

```

```python
    plt.figure()
    plt.plot(normalizedQuantiles, sortedNormalizedReturns, 'rD')

    # Establish min and max values for axes
    min_value = np.around(sortedNormalizedReturns[0])
    max_value = np.around(sortedNormalizedReturns[len(sortedNormalizedReturns) - 1])

    absolute_value = max((np.abs(min_value), max_value))

    plt.xlim(-absolute_value, absolute_value)
    plt.ylim(-absolute_value, absolute_value)

    # Bisectrix plot
    x = np.arange(-absolute_value, absolute_value + 1, 1)
    plt.plot(x, x)


def main():

    #=====================================================================
    # Simple QQ Plot
    #=====================================================================
    data = dataSeries["Italy"].values

    # Standard statistics
    stats = descriptiveStats(data)
    printDescriptiveStats(stats)
```

Number of obs.	468
Mean	1.0732%
Std. Deviation	6.9915%
Skewness	0.5249
Kurtosis	1.3336

```python
    # normalize returns
    normalizedReturns = data / stats["stdev"];

    # QQ Plot
    qqPlot(normalizedReturns)
```

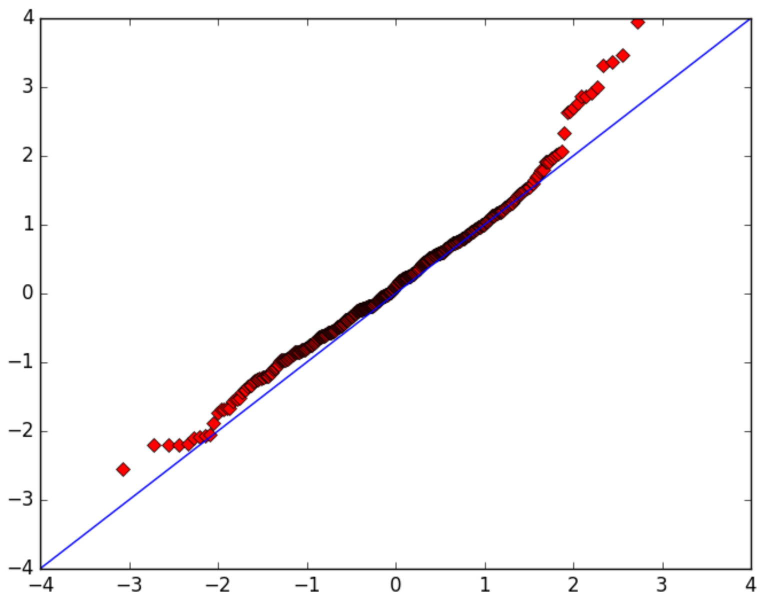

Figure 22: QQ Plot of Italian Equity Returns (scaled by unconditional variance)

```
#=====================================================================
# QQ Plot using GARCH vol
#=====================================================================
data = dataSeries["Italy"].values

# GARCH vol
initParams = [0.2, 0.8, 0.0, 0.0]
garchParams = maximizeMLE(initParams, data)
garchVol = GARCHLeveraged(garchParams, data)

# normalize returns
normalizedReturns = data / np.sqrt(garchVol)

# Standard statistics
stats = descriptiveStats(normalizedReturns)
printDescriptiveStats(stats)
```

Number of obs.	468
Mean	15.0207%
Std. Deviation	99.2437%
Skewness	0.3608
Kurtosis	0.8175

```
# QQ Plot
qqPlot(normalizedReturns)
```

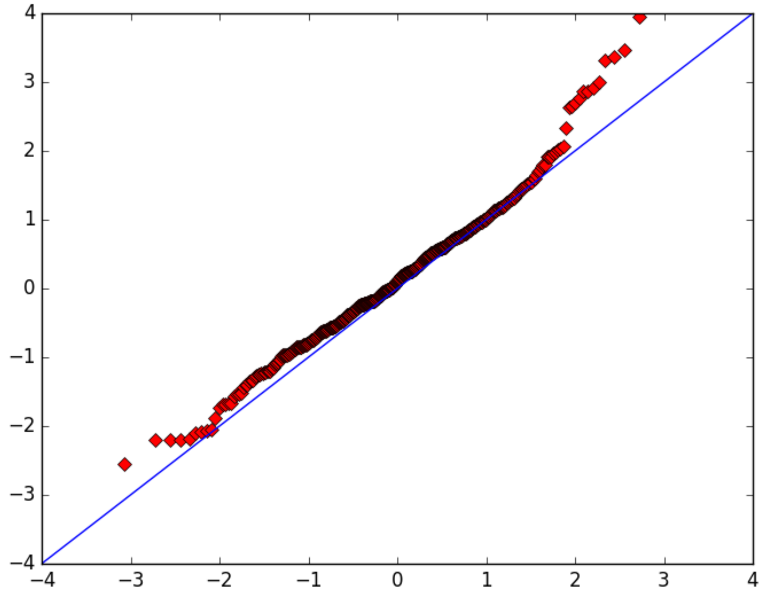

Figure 23: QQ Plot of Italian Equity Returns with GARCH(1,1) Shocks

8 Correlations

```python
"""
@author: dtoppo
Correlations
"""

import numpy as np
import pandas as pd

# Import dataserie from Excel
dataFile = "data_python_HW_2017.xlsx"
xslx = pd.ExcelFile(dataFile)

sheetName = "Correlation"

dataSeries = xslx.parse(sheetName)


# Covariance matrix
cov_matrix = np.cov(dataSeries.T)
print("Covariance matrix: ")
print(cov_matrix)
```

$$\begin{bmatrix} 0.00488812 & 0.00157657 & 0.00205688 & 0.00175714 & 0.00186691 \\ 0.00157657 & 0.00201506 & 0.00176448 & 0.0014482 & 0.00159337 \\ 0.00205688 & 0.00176448 & 0.00291552 & 0.00170511 & 0.00160591 \\ 0.00175714 & 0.0014482 & 0.00170511 & 0.00240827 & 0.00154439 \\ 0.00186691 & 0.00159337 & 0.00160591 & 0.00154439 & 0.00304086 \end{bmatrix}$$

```python
# Determinant
determinant = np.linalg.det(cov_matrix)
print("Determinant : %.6f" % determinant)
```

Determinant	0

```python
# Correlation matrix
corr_matrix = np.corrcoef(dataSeries.T);
print("\n\nCorrelation matrix: ")
print(corr_matrix)
```

$$\begin{bmatrix} 1. & 0.50234023 & 0.54485513 & 0.51213205 & 0.48423435 \\ 0.50234023 & 1. & 0.72797427 & 0.65740331 & 0.64368594 \\ 0.54485513 & 0.72797427 & 1. & 0.64349111 & 0.53934197 \\ 0.51213205 & 0.65740331 & 0.64349111 & 1. & 0.57069908 \\ 0.48423435 & 0.64368594 & 0.53934197 & 0.57069908 & 1. \end{bmatrix}$$

```python
# Determinant
determinant = np.linalg.det(corr_matrix)
print("Determinant : %.6f" % determinant)
```

Determinant	0.82543

9 Dynamic Conditional Correlations

```python
"""
@author: dtoppo
Dynamic Conditional Correlations
"""

import pandas as pd
import numpy as np
import scipy as sp
from scipy import optimize
from scipy.special import gammaln

from GARCHLeverage import GARCHLeveraged, maximizeMLE

# Import dataserie from Excel
dataFile = "data_python_HW_2017.xlsx"
xslx = pd.ExcelFile(dataFile)

sheetName = "DCC"
dataSeries = xslx.parse(sheetName)


#==============================================================================
# Unconditional sample covariance and correlation
#==============================================================================
sampleCov = np.cov(dataSeries[["Germany","Japan"]].T, ddof=0)[0,1]
sampleCorr = np.corrcoef(dataSeries[["Germany","Japan"]].T, ddof=0)[0,1]

#==============================================================================
# Portfolio unconditional covariance
#==============================================================================
GermanyWgt = 0.5;
JapanWgt = 0.5;

GermanyVar = np.var(dataSeries["Germany"], ddof=1)
JapanVar = np.var(dataSeries["Japan"], ddof=1)

# PPF = percent point function === quantile function = norminv
# q = prob(X<=x)
VaRGermany = -sp.stats.norm.ppf(q=0.01, loc=0, scale=1)*np.sqrt(GermanyVar)*GermanyWgt
VaRJapan = -sp.stats.norm.ppf(q=0.01, loc=0, scale=1)*np.sqrt(JapanVar)*JapanWgt
sumOfVaRs = VaRGermany + VaRJapan

pfCov = np.cov(dataSeries[["Germany","Japan"]].T, ddof=0)[0,1]
pfVar = GermanyWgt**2*GermanyVar + JapanWgt**2*JapanVar + 2*GermanyWgt*JapanWgt*pfCov

VaRPf = -sp.stats.norm.ppf(q=0.01, loc=0, scale=1)*np.sqrt(pfVar)

print(sumOfVaRs);
print(VaRPf);
```

Sum of VaRs	0.1239
Portfolio VaRs	0.1035

Due to diversification, the VaR of the portfolio is less than the sum of the individual VaRs.

```python
#==============================================================================
# NAGARCH(1,1)
#==============================================================================
deData = dataSeries["Germany"].values

deInitParams = [0.2, 0.5, 0.0008, 0.75]
deParams = maximizeMLE(deInitParams, deData)
deGarchVol = GARCHLeveraged(deParams, deData)
deStdRet = deData / np.sqrt(deGarchVol)


jpData = dataSeries["Japan"].values

jpInitParams = [0.2, 0.5, 0.0008, 0.75]
jpParams = maximizeMLE(jpInitParams, jpData)
jpGarchVol = GARCHLeveraged(jpParams, jpData)
jpStdRet = jpData / np.sqrt(jpGarchVol)

#==============================================================================
# Lamda estimation
#==============================================================================
def computeQ(lda):
    qDE = np.ones(np.size(deData))
    qDEJP = np.ones(np.size(deData))
    qJP = np.ones(np.size(deData))

    q = pd.DataFrame(data=np.array([qDE, qDEJP, qJP]).T,
                     columns=["DE-DE", "DE-JP", "JP-JP"], index=dataSeries.index)

    q["DE-DE"][0] = 1
    q["DE-JP"][0] = (deStdRet * jpStdRet).sum() / np.size(deStdRet)
    q["JP-JP"][0] = 1

    for t in range(1, np.size(deData)):
        q["DE-DE"][t] = (1-lda)*deStdRet[t-1]**2 + lda*q["DE-DE"][t-1]
        q["DE-JP"][t] = (1-lda)*deStdRet[t-1]*jpStdRet[t-1] + lda*q["DE-JP"][t-1]
        q["JP-JP"][t] = (1-lda)*jpStdRet[t-1]**2 + lda*q["JP-JP"][t-1]

    return q

def dccMinLikelihood(lda):
    q = computeQ(lda)
    r = q["DE-JP"] / np.sqrt(q["DE-DE"] * q["JP-JP"])
    logL = -1/2*((np.log(1-r**2)) + (deStdRet**2+jpStdRet**2-2*r*deStdRet*jpStdRet)/(1-r**2))
    return -(logL.sum())

bnds = [(0.00001, 0.99999)]
initParam = 0.94
resultDcc = optimize.minimize(dccMinLikelihood, initParam,
                              method='SLSQP', bounds=bnds)

print(resultDcc)
```

Optimal Lambda	0.9841

```python
#==============================================================================
# GARCH(1,1)-t(d) model
```

```python
#==============================================================================
data = dataSeries["Italy"]

# Initialize GARCH(1,1)
def garch(params):
    omega, alpha, beta, delta, d = params
    s = np.zeros(np.size(data))
    s[0] = np.var(data, ddof=1)
    for i in range(1, np.size(data)):
        s[i] = omega + alpha*(data[i-1] - delta*np.sqrt(s[i-1]))**2 + beta*s[i-1]
    return s

# MLE
def mle(params):
    s = garch(params)
    di= params[4]
    logL = (gammaln(di*0.5 + 0.5) - 0.5*np.log(np.pi) - gammaln(di*0.5) - \
            0.5*np.log(di-2)-(di*0.5 + 0.5)*np.log(1+(data**2/s)/(di - 2)) \
            - 0.5*np.log(s)).sum()
    return -logL

# Optimization
initParameters = [0.0001, 0.085, 0.876, 0.5, 3]
bnds = [(0.0001,0.9999), (0.0001,0.9999), (0.0001,0.9999), (0.0001,None),
        (2.0001,None)]

def persistence(params):
    omega, alpha, beta, delta, d = params
    return 1 - (alpha*(1 + delta**2) + beta)

cons = {"type":"ineq","fun":persistence}

result = optimize.minimize(mle, initParameters, method="SLSQP", bounds=bnds,
                           constraints=cons)
print(result)
```

Alpha	0.1373
Beta	0.7616
Omega	0.0005
Theta	0.2041
d	7.5876