In [1]:
```
import pandas as pd
from arch import arch_model
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import  mean_squared_error

R =pd.read_csv('../TD4/long_series_logret.csv',index_col=0)
R.index = pd.DatetimeIndex(R.index)

year = 252
r = (R[R.index.year>2005].values*year).T[0] #Just for convinience I annualized the returns
```

In [2]:
```
#In-sample and out-of-sample window size
tin,tout = 1500,5
```

In [209…
```
t = 0

#fitting and GARCH(1,1) on the in-sample window
Model = arch_model(r[t:t+tin], vol='GARCH', p=1,q=1,dist='t')
fit = Model.fit(disp='off')

#get the parameters
parms = fit.params
```

In [210…
```
parms
```

Out[210…
```
mu          0.243070
omega       0.687305
alpha[1]    0.074552
beta[1]     0.918357
nu          5.472628
Name: params, dtype: float64
```

In [43]:
```
#Convergence value of the conditional variance
conv = parms.loc['omega']/(1-parms.loc['alpha[1]']-parms.loc['beta[1]'])

conv
```

Out[43]: 96.92857292526473

In [211…
```
# Forecast of  500 time step analytically

forecast = fit.forecast(horizon=500,method='analytic')
```

In [213…
```
# Only the last row is the forecast, the columns are the lags
# The first element is the last in-sample estimate

forecast.variance
```

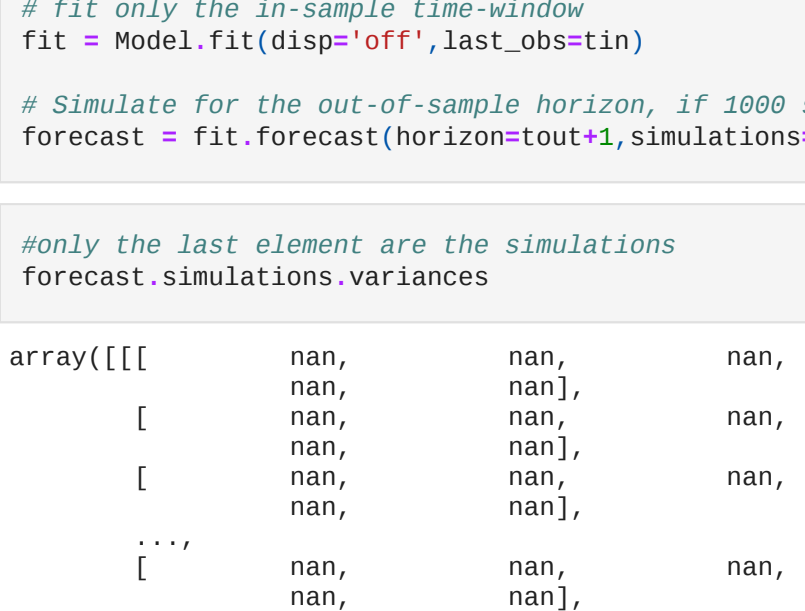
Out[213…

	h.001	h.002	h.003	h.004	h.005	h.006	h.007	h.008	h.009	h.010	...	h.491	h.492	h.493	h.494	h.495	h.496	h.497	h.498	h.499	h.50
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1495	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1496	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1497	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1498	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
1499	65.03922	65.265342	65.489861	65.712788	65.934135	66.153911	66.37213	66.588801	66.803935	67.017544	...	95.952891	95.959809	95.966678	95.973499	95.980271	95.986995	95.993672	96.000301	96.006884	96.01341

1500 rows × 500 columns

In [ ]:

In [50]:
```
plt.plot( forecast.variance.iloc[-1].values )
plt.hlines(conv,0,500)
plt.xlabel('h lags [future]',fontsize=16)
plt.ylabel(r'$E[\hat{\sigma}_t]$',fontsize=16)
plt.tick_params(labelsize=14)
```



In [214…
```
t = 0

# define a GARCH(1,1) for in-sample and out-of-sample
Model = arch_model(r[t:t+tin+tout], vol='GARCH', p=1,q=1,dist='t')
# fit only the in-sample time-window
fit = Model.fit(disp='off',last_obs=tin)

# Simulate for the out-of-sample horizon, if 1000 sims are too much, just reduce it
forecast = fit.forecast(horizon=tout+1,simulations=1000,method='simulation')
```

In [216…
```
#only the last element are the simulations
forecast.simulations.variances
```

Out[216…
```
array([[[         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        ...,
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan]],

       [[         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        ...,
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan]],

       [[         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        ...,
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan],
        [         nan,          nan,          nan,          nan,
                  nan,          nan]],

       ...,

       [[ 59.90080683,  55.83242943,  59.29639951,  56.06729228,
          78.53303679,  73.06901395],
        [ 59.90080683,  55.70422727,  53.82909931,  52.47537962,
          50.01678766,  46.73737283],
        [ 59.90080683,  60.2625186 ,  61.30432123,  69.21085278,
          67.78789017,  71.95312089],
        ...,
        [ 59.90080683,  70.19976384,  66.92239433,  66.21181172,
          74.27424808,  69.80369167],
        [ 59.90080683,  55.69830558,  69.43378959,  64.45431218,
          67.37258392,  69.18726256],
        [ 59.90080683,  56.93410479,  57.42484494,  53.61031569,
          53.67785308,  65.11927283]],

       [[ 73.8256421 ,  72.11833981,  67.08710483,  63.3374214 ,
          59.69094618,  69.56522187],
        [ 73.8256421 ,  85.04890194,  78.81454531,  73.08292843,
          69.79234339,  66.96111251],
        [ 73.8256421 ,  76.14194848,  80.7400202 ,  84.07265823,
          78.28824589,  72.59488951],
        ...,
        [ 73.8256421 ,  70.77583182,  72.74055002,  70.17267126,
          71.04095695,  65.96697462],
        [ 73.8256421 ,  76.35831954,  73.47161018,  74.64042417,
          71.12567497,  66.16121358],
        [ 73.8256421 ,  69.22226258,  65.2299207 ,  74.56366399,
          70.54698026,  65.73668804]],

       [[ 82.29317293,  85.00935342,  82.17510467,  90.42537749,
          83.79857876, 104.74467301],
        [ 82.29317293,  76.73806301, 108.58691336, 102.21599645,
          94.98306572,  89.88517306],
        [ 82.29317293,  78.13861882,  73.38603358,  68.56443911,
          64.8804909 ,  65.22732603],
        ...,
        [ 82.29317293,  94.80763988,  92.07752095,  85.94331938,
          79.61624682,  83.96663   ],
        [ 82.29317293,  81.68604017,  82.12364151,  78.100994  ,
          73.8855487 ,  71.62775071],
        [ 82.29317293,  79.79306394,  75.33373709,  76.94997602,
          80.07696205,  93.18642748]]])
```

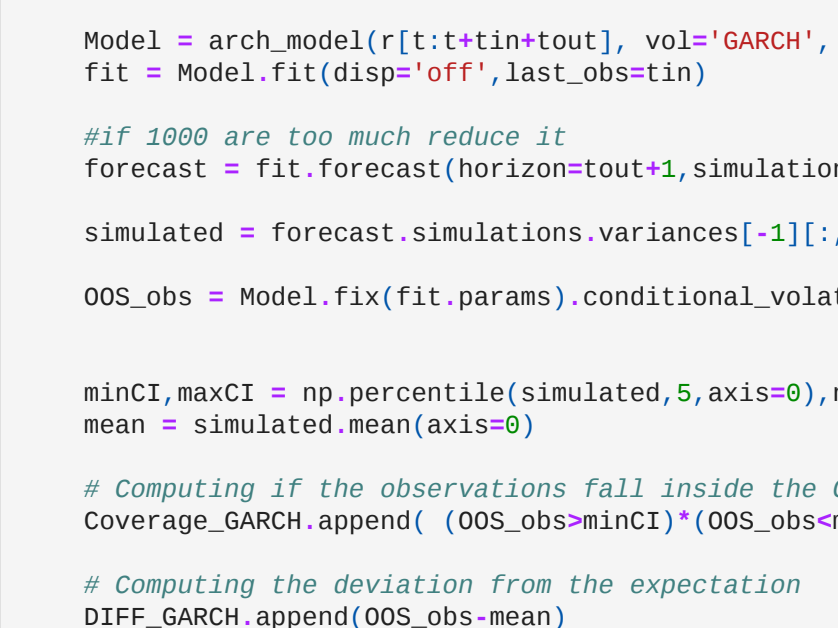
In [218…
```
# Simulations are by rows, times is by column. The first column is the last in-sample observation
forecast.simulations.variances[-1].shape
```

Out[218… (1000, 6)

In [219…
```
simulated = forecast.simulations.variances[-1][:,1:]

# Estimat the conditional variance by using the in-sample calibrated parameters
OOS_obs = Model.fix(fit.params).conditional_volatility[-tout:]**2
## fix computing overall the timeseries

# Get a 90% confidence interval
minCI,maxCI = np.percentile(simulated,5,axis=0),np.percentile(simulated,95,axis=0)

# Get the mean (it can be computed also analytically)
mean = simulated.mean(axis=0)
```

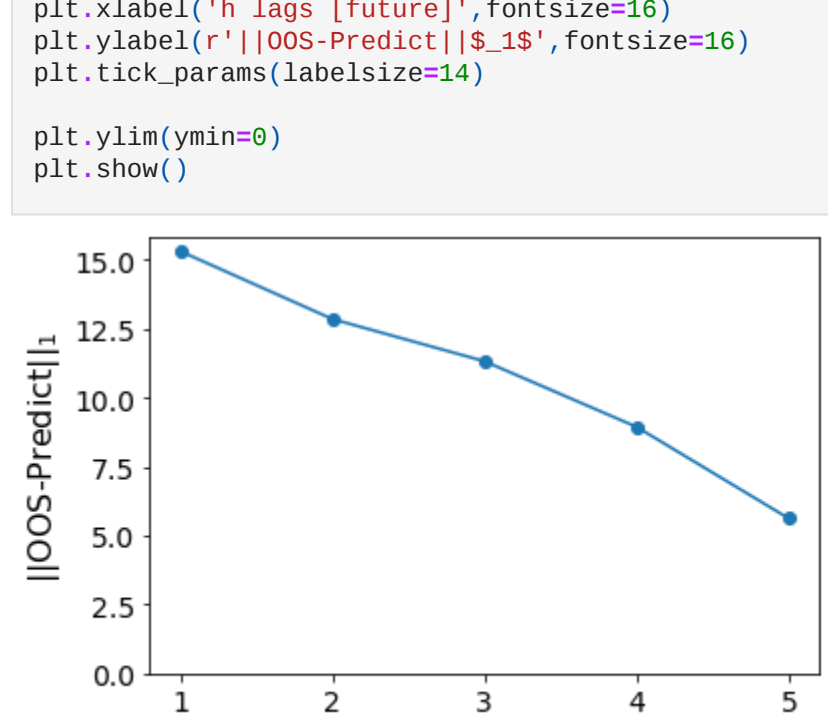
In [220…
```
dx = np.arange(1,tout+1)
plt.plot(dx,OOS_obs,'-o',label='OOS --> variance prédites')
plt.fill_between(dx,minCI,maxCI,alpha=0.4,label='90% Confidance Interval')
plt.plot(dx,mean,label='expected --> variance moyenne sur les simulations Monte Carlo')
plt.xlabel('h lags [future]',fontsize=16)
plt.ylabel(r'$σ_t^2$',fontsize=16)
plt.tick_params(labelsize=14)
plt.legend(loc='upper left',fontsize=13)
```

Out[220… <matplotlib.legend.Legend at 0x7efe71d85d30>

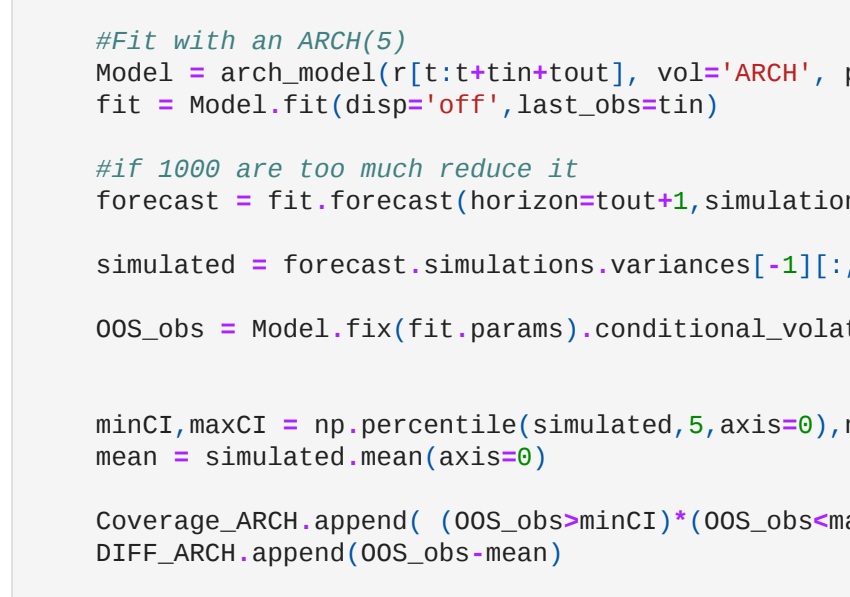


In [221…
```
Coverage_GARCH,DIFF_GARCH = [],[]

# Sliding windows in step of 5 days
for t in range(0,1000,5):

    Model = arch_model(r[t:t+tin+tout], vol='GARCH', p=1,q=1,dist='t')
    fit = Model.fit(disp='off',last_obs=tin)

    #if 1000 are too much reduce it
    forecast = fit.forecast(horizon=tout+1,simulations=1000,method='simulation')

    simulated = forecast.simulations.variances[-1][:,1:]

    OOS_obs = Model.fix(fit.params).conditional_volatility[-tout:]**2


    minCI,maxCI = np.percentile(simulated,5,axis=0),np.percentile(simulated,95,axis=0)
    mean = simulated.mean(axis=0)

    # Computing if the observations fall inside the CI
    Coverage_GARCH.append( (OOS_obs>minCI)*(OOS_obs<maxCI) )

    # Computing the deviation from the expectation
    DIFF_GARCH.append(OOS_obs-mean)
```

In [222…
```
dx = np.arange(1,1+tout)
plt.plot(dx, np.mean(Coverage_GARCH,axis=0),'-o' )
plt.hlines(0.9,0,7,linestyles='--')
plt.xlim([0.8,5.2])
plt.xlabel('h lags [future]',fontsize=16)
plt.ylabel('Fraction of True Positive',fontsize=16)
plt.tick_params(labelsize=14)

plt.title('Expected Coverage 90%',fontsize=16)
plt.show()
```



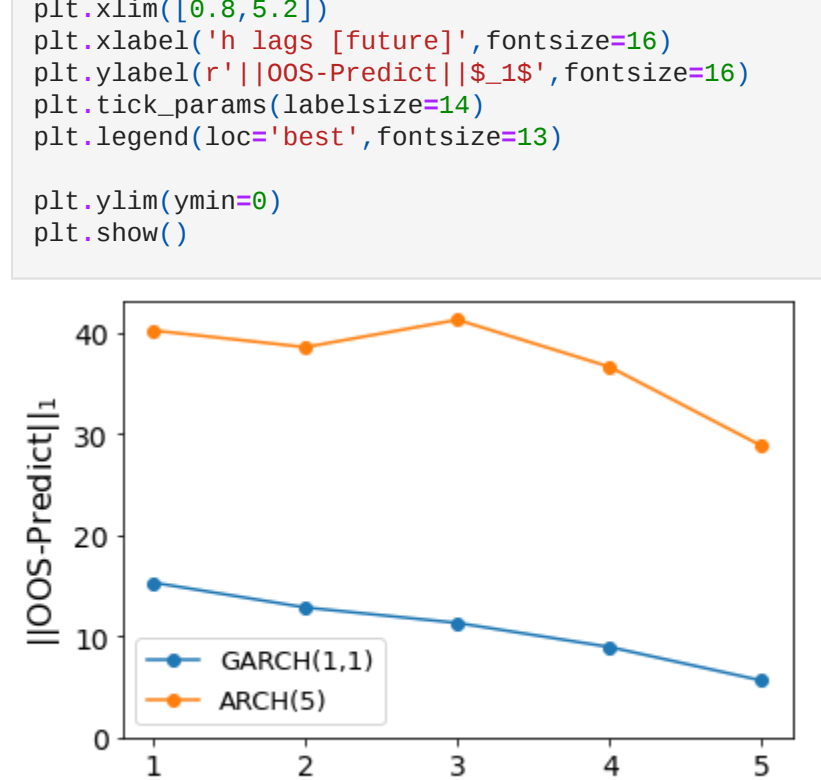
In [223…
```
plt.plot( dx,abs(np.array(DIFF_GARCH)).mean(axis=0),'-o' ,label='GARCH(1,1)' )

plt.xlim([0.8,5.2])
plt.xlabel('h lags [future]',fontsize=16)
plt.ylabel(r'||OOS-Predict||$_1$',fontsize=16)
plt.tick_params(labelsize=14)

plt.ylim(ymin=0)
plt.show()
```



In [226…
```
# The long-range predictions seems better
```

In [224…
```
Coverage_ARCH,DIFF_ARCH = [],[]
for t in range(0,1000,5):

    #Fit with an ARCH(5)
    Model = arch_model(r[t:t+tin+tout], vol='ARCH', p=5,dist='t')
    fit = Model.fit(disp='off',last_obs=tin)

    #if 1000 are too much reduce it
    forecast = fit.forecast(horizon=tout+1,simulations=1000,method='simulation')

    simulated = forecast.simulations.variances[-1][:,1:]

    OOS_obs = Model.fix(fit.params).conditional_volatility[-tout:]**2


    minCI,maxCI = np.percentile(simulated,5,axis=0),np.percentile(simulated,95,axis=0)
    mean = simulated.mean(axis=0)

    Coverage_ARCH.append( (OOS_obs>minCI)*(OOS_obs<maxCI) )
    DIFF_ARCH.append(OOS_obs-mean)
```

In [227…
```
dx = np.arange(1,1+tout)
plt.plot(dx, np.mean(Coverage_GARCH,axis=0),'-o',label='GARCH(1,1)' )
plt.plot(dx, np.mean(Coverage_ARCH,axis=0),'-o',label='ARCH(5)' )
plt.hlines(0.9,0,7,linestyles='--')
plt.xlim([0.8,5.2])
plt.xlabel('h lags [future]',fontsize=16)
plt.ylabel('Fraction of True Positive',fontsize=16)
plt.tick_params(labelsize=14)
plt.legend(loc='lower right',fontsize=13)
plt.title('Expected Coverage 90%',fontsize=16)
plt.show()
```



In [228…
```
plt.plot( dx,abs(np.array(DIFF_GARCH)).mean(axis=0),'-o' ,label='GARCH(1,1)' )
plt.plot( dx,abs(np.array(DIFF_ARCH)).mean(axis=0),'-o' ,label='ARCH(5)' )

plt.xlim([0.8,5.2])
plt.xlabel('h lags [future]',fontsize=16)
plt.ylabel(r'||OOS-Predict||$_1$',fontsize=16)
plt.tick_params(labelsize=14)
plt.legend(loc='best',fontsize=13)

plt.ylim(ymin=0)
plt.show()
```


In [229…
```
# The GARCH(1,1) is clearly better then the ARCH(5)
```

In [ ]:
```
# Please note that this is only an anecdotal example, you should not never assume it as a general rule.
```

In [ ]: