Homework 5 Quantitative Risk Management

Group G03
Jakob Amaya Scott, Martin Hammerues, Albin Henriksson, Markus Trætli

October 2022

Question 1

Due to computational time, we do not use the entire dataset. Instead, we use the first 30% of the data when evaluating the two methods on each stock. Also, we used a value of $T = 10$ and $H = 0.15$. The goodness estimate for the two methods on each stock can be seen in table 1. As we can see, the GK-estimate outperforms the rough volatility model for all stocks, since small values of the goodness estimate are desired.

	AAP	ABC	ACWI	AES	AKAM	ALGN	AMP	AMZN	ANSS	ANTM —
GK-estimate	0.42	0.44	0.23	0.30	0.29	0.39	0.38	0.24	0.31	0.53
Rough volatility estimate	0.52	0.54	0.49	0.56	0.49	0.53	0.51	0.44	0.57	0.63

Code

```python
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt

#Replace data_path if you have a different relative path to the data
data_path = "./DataPS5"
file_names = [f for f in os.listdir(data_path) if (os.path.isfile(os.path.join(data_path, f))) and f.endswith(".csv")] #Get a list of all csv files in the folder

#Garman-Klass estimator
def GK_estimator(h, l, o, c):
    #T and t given implicitly as length of vectors and the items themselves
    T = len(h)
    assert T == len(l) and len(l) == len(o) and len(o) == len(c)
    return np.sqrt( (1/T)*np.sum( 0.5*(np.log(h/l)**2) - (2*np.log(2)-1)*(np.log(c/o)**2) ) )

#Garman-Klass-Yang-Zhang
def GKYZ_estimator(h, l, o, c, c_prev):
    #T and t given implicitly as length of vectors and the items themselves
    T = len(h)
    assert T == len(l) and len(l) == len(o) and len(o) == len(c) and len(c) == len(c_prev)
    return np.sqrt((1/T)*np.sum(0.5*(np.log(o/c))**2 + 0.5*(np.log(h/l))**2 - (2*np.log(2)-1)*(np.log(c/o))**2))

#Checks if all the elements in the given array of datetime objects are defined on the same day and that theres 10 minutes in between
def any_diff(data, a, b):
    if (data[b] - data[a]).seconds/60 > 10:
        return False
    for i in range(a + 1, b):
        if data[i].day != data[i-1].day:
            return False
    return True

#Load one of the datasets

stock_index = 0
data = pd.read_csv(os.path.join(data_path, file_names[stock_index]))
print("Stock used:",file_names[stock_index])
data = data.head(int(0.5*data.shape[0])) #Take a smaller subset of the data

#region manipulate data

#First we construct the mid prices. and filter out those that are 0
data["close"] = 0.5*(data["bid"] + data["ask"]) #Calculate c_t (2)
data = data[data["close"] > 0].reset_index(drop=True)

#Now calculate the rolling maximum and minimum with a window of 5
data["high"] = data["close"].rolling(window=5).max()
data["low"] = data["close"].rolling(window=5).min()

#Add open as the close from the previous day
data["open"] = pd.concat( [pd.Series(np.nan), data["close"][:-1]] , ignore_index=True)

#Convert the date to actual datetime datatype
data["trade_time"] = pd.to_datetime(data["trade_time"], infer_datetime_format=True)
```

```

#Temporary reset index
data = data.reset_index(drop=True)

#Now create a boolean mask to filter out the points where:
# 1: one of the previous 4 points were on a different day
# 2: the time difference between the first and the last is greater than 10 minutes
mask = np.zeros(len(data["ask"]))
for i in range(5, len(mask)):
    mask[i] = any_diff(data["trade_time"], i-5, i)
mask = pd.Series(mask, dtype=bool)

#Apply filter. Then get every 5th observation so we get no overlaps of our 10 minute windows
new_df = data[mask.values].iloc[::5].reset_index(drop=True)
print("Post-processed dataframe:")
print(new_df.head(10))

plt.plot(new_df["trade_time"], new_df["close"])
#plt.show()

#endregion


#region calculate volatility

T = 100 #How many points to consider in the rolling window
delta = 15 #The period in the future we want to predict

#Calculate the rolling window estimates of sigma using the Garman-Klass estimator on T values at a time
sigmas = np.zeros(new_df.shape[0])
for i in range(T + 1, len(sigmas)):
    sigmas[i] = GK_estimator(new_df["high"][(i-T):i], new_df["low"][(i-T):i], new_df["open"][(i-T):i], new_df["close"][(i-T):i])
sigmas = sigmas[(T+1):]
sigmas = sigmas**2

plt.plot(new_df["trade_time"][(T+1):], sigmas)
#plt.show()

#Obtain our forecast volatility estimate
forecasts = sigmas

H = 0.15
integrated = np.zeros(len(sigmas)) #Will hold the values of the integral for all t
log_sigmas = np.log(sigmas)
for t in range(1,len(integrated)): #The first element must be 0 since we integrate from t=0 to t=0. Thus start at index 1
    denom1 = np.flip(np.arange(1, t + 1))
    denom2 = denom1**(H + 0.5)
    denom = (denom1 + delta)*denom2
    integrated[t] = np.sum(log_sigmas[:t]/(denom)) + log_sigmas[t]/(delta**(H+0.5))

#Multiply by the factor to get rough volatility estimate
rough_volatility = (np.cos(H*np.pi)/np.pi)*(delta**(H+0.5))*integrated[(T+1):]
Q = pd.Series(log_sigmas).rolling(window=T).std()[(T+1):] #First T entries are nan
pred_volatility = np.exp(rough_volatility + 0.5*Q) #sigma hat

#endregion
```

```

#region quality check

#Lets create a df to store the results
df = pd.DataFrame()
df["actual_sigma"] = sigmas
df["forecast"] = pd.concat([pd.Series([np.nan for j in range(delta)]), pd.Series(forecasts[:-delta])], ignore_index=True) #the delta last ones are out of range for our data
df["rough"] = pd.concat([pd.Series([np.nan for j in range(T+1 + delta)]), pd.Series(pred_volatility[:-delta])], ignore_index=True)
print(df.head(T+delta + 10).tail(delta+10))
print(df.tail(30))
plt.show()
plt.plot(df["rough"])
plt.show()

#Calculate the rolling window standard deviation
unconditional_mean = np.sum((df["actual_sigma"] - np.mean(df["actual_sigma"]))**2) #Sum instead of mean since 1/N cancels
goodness_forecast = np.sqrt(np.sum((df["actual_sigma"][delta:] - df["forecast"][delta:])**2)/unconditional_mean) #P
goodness_rough = np.sqrt(np.sum((df["actual_sigma"][(T+1+delta):] - df["rough"][(T+1+delta):])**2)/unconditional_mean)
print("Forecast:", goodness_forecast)
print("rough:", goodness_rough)

#To explain the indexing: the forecast uses only the previous time period to predict, thus we cannot calculate a prediction for the first element
# For the rough volatility we use a rolling window of T samples and need the previous thus we cannot calculate a prediction for the first T+1 elements

#endregion
```