Vrije Universiteit Amsterdam

Time Series Models: Assignment 2

Authors:
Lukas Ekkelkamp(2627569):
l.ekkelkamp@outlook.com
Jacob Brüller(2590577):
j.bruller@student.vu.nl
Tristan de Keyzer(2623853):
t.k.de.keyzer@student.vu.nl
Tom Korenwinder(2612943):
t.korenwinder@student.vu.nl

Supervisor:
dr. S.J. Koopman

March 7, 2022

Introduction

In this assignment we will make use of the Stochastic Volatility (SV) model to analyze two datasets. We will state the used models with underlying assumptions and will return at Kalman filter methods to do our analysis. Our research is mainly based on Durbin and Koopman (2012). The first dataset that we therefore analyze is one that is also used in Durbin and Koopman (2012), Section 14.5: a dataset of exchange rates between Pounds and Dollars. We will make sure that this financial series is presented in returns, before modelling a linear SV model. This model is obtained after a transformation of the data. After this, we will perform the QML-method using the Kalman filter that is discussed in the previous assignment and estimate unknown coefficients. Based on these estimates, we will compute smoothed values in the model and present those. We present differences between smoothed and filtered values.
We extend the SV model with two extensions and compare these extended models to the original one. Within these extensions, we make use of the daily (closed) returns for the S&P500 index during the latest 5 years, which can be found at `https://realized.oxford-man.ox.ac.uk/`. We also make use of a Realized Volatility measure called the Parzen kernel function to improve the performance of the model. This volatility measure is obtained from the same website.

Question A

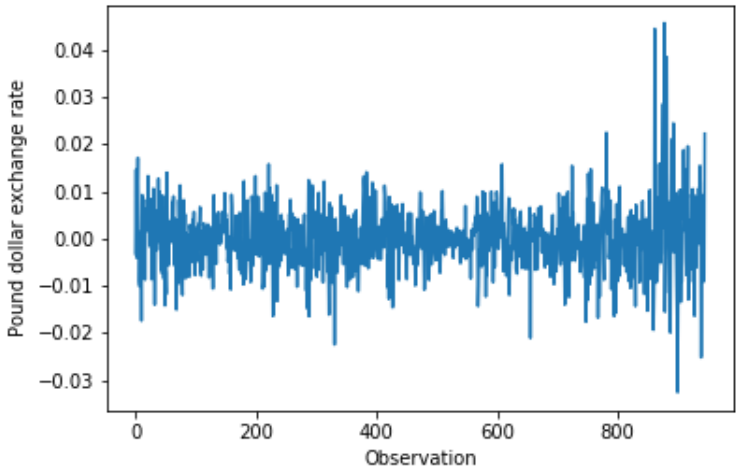

Figure 1: y_t: Daily logreturns of exchange rate between pound sterling and US dollar

In the figure above we have transformed the first data set that we use to returns. This data set contains daily exchange rates between pounds and dollars. The daily demeaned returns are presented in the figure above. We see that the volatility increases suddenly at the end of the sample. To avoid taking logs of the value zero, we have demeaned the sample. The resulting mean-corrected logreturns are then analysed by the QML method.

We obtain the SV model as follows:

$$y_t = \mu + \sigma_t \epsilon_t, \qquad \log \sigma_t^2 = \xi + H_t, \qquad H_{t+1} = \phi H_t + \sigma_\eta \eta_t \tag{1}$$

Since both σ_t and η_t are stochastic processes, this is a nonlinear time series model. In this model y_t is the first difference of the logged exchange rates between pound sterling and US dollar, so we can see y_t in the figure above.

Question B, C & D

As we have seen in the previous section, we obtained a nonlinear time series model. To make this model linear, we apply data transformation $x_t = \log(y_t - \mu)^2$ and some redefinitions to obtain:

$$x_t = h_t + u_t, \qquad h_{t+1} = \omega + \phi h_t + \sigma_\eta \eta_t, \tag{2}$$

where $u_t = \log \epsilon_t^2, \omega = (1 - \phi)\xi$ and $h_t = H_t + \xi$.
We have now obtained the linear AR(1) + noise model. However, the disturbance u_t is not necessarily Gaussian distributed. This forms the basis of the quasi maximum likelihood (QML) method of this SV model. This method uses the Kalman filter to compute the likelihood. That means that we do as if u_t is Gaussian with mean and variance corresponding to that of the of the $\log \chi^2$ distribution.
So we continue to assume that the disturbances in the model for x_t will be Gaussian distributed with mean and variance corresponding to that of the $\log \chi^2(1)$ distribution. We adopt the QML method to estimate the unknown coefficients. In this method, the Kalman filter is used and the mean and the variance of the $\log \chi^2(1)$ are used, respectively -1.27 and 4.93. The results of this estimation can be found in the table below.

Table 1: Parameter values SV-model

ω	ϕ	σ_η
-0.144	0.989	0.093

We proceed by using the estimates obtained in the previous section as our final estimates. We compute smoothed values of h_t based on the model for x_t in equation 2. This is done by using the Kalman filter and smoother. We present the smoothed values alongside the transformed data x_t in the figure below.

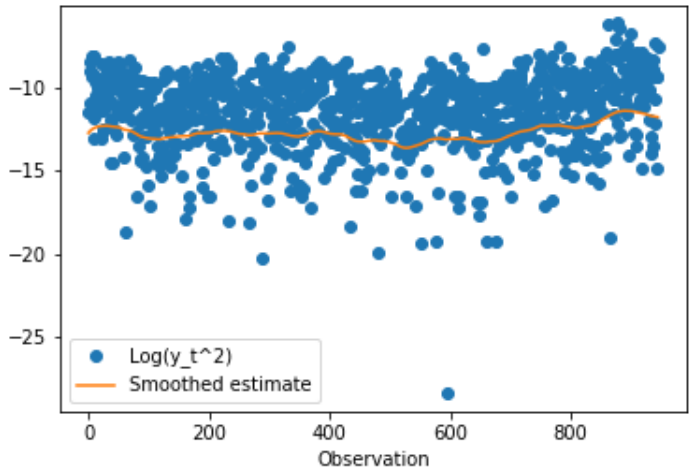

Figure 2: The x_t series with the smoothed estimate of h_t

Moreover, we present both the filtered and smoothed estimates of H_t in the figure below. These are computed by subtracting $x = \frac{\omega}{1-\phi}$ from the smoothed and filtered estimates of h_t. Since this is equal to $E[h_t]$, we expect the values of H_t to be around 0.

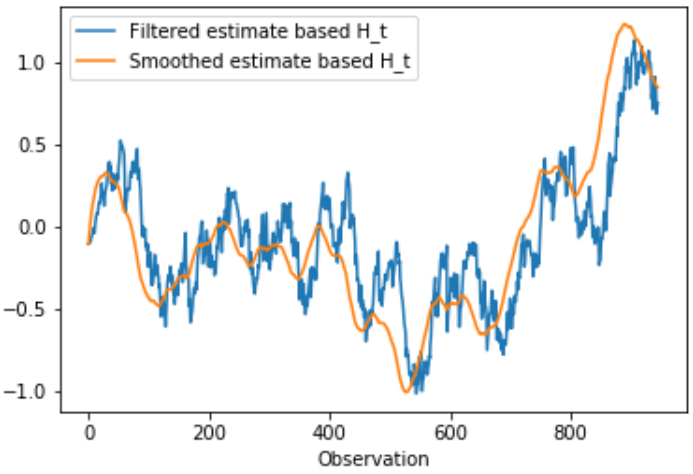

Figure 3: Filtered and smoothed estimates of H_t

We see that the filtered and smoothed estimates of H_t follow more or the less the same pattern and thus

take on more or less the same values. The values fluctuate around 0, which is in line with our expectations. Also, the filtered estimates change much more rapidly whereas the smoothed estimates form a smooth graph. The smoothed estimate is for most of the observations lower than the filtered estimate, except for the final observations. Between observations 400 and 600 we see some interesting movement: the smoothed estimate decreases at some points while the filtered estimate increases at the same observations.

Question E

We will now consider a different dataset. We consider the daily returns for the S&P500 index for the duration of the latest 5 years. We apply the same framework as described in the previous sections and we extend it. To improve the performance of the SV model, we extend our analysis with a Realized Volatility measure from Oxford-Man Institute: the Parzen kernel function. This means that we now consider the following extended model:

$$x_t = \beta \cdot \log RV_t + h_t + u_t, \qquad h_{t+1} = \omega + \phi h_t + \sigma_\eta \eta_t, \tag{3}$$

in which β is the regression coefficient and RV_t is the realized volatility measure: the Parzen kernel. To incorporate this into the Kalman filter, we subtract $\beta \cdot \log RV_t$ from every value of h_t. We chose for a constant beta instead of a diffuse beta, as we are interested in the effect of the realized volatility across the entire period. Potential future investigation might want to opt for a diffuse beta, if one is interested in studying changes of beta across different periods in time. We perform the same analysis as in the previous sections and obtain the new estimates for the unknown coefficients. In the table below, we provide both the estimates from the extended model and the previous model. The original model did not make use of a β coefficient so this model will not have an estimate for β.

Table 2: Parameter values

Model	ω	ϕ	σ_η	β
SV model	-0.445	0.965	0.354	-
SV model with RV	0.059	0.476	0.0006	-0.629

The RV model has a higher maximum likelihood than the model without: -2775.10 over -2869.40. This is according to expectations as we could always set β equal to 0 to get the original model. We see that the estimate for ω has increased whereas the value of ϕ has decreased. The estimate for σ_η has also become considerably lower, now being very close to 0. The coefficient for β tells us that the Realized Volatility measure has a negative impact on x_t. A higher realized volatility seems to imply a decrease in the transformed value of the data and therefore also a decrease in the S&P500 data. The lower coefficient of ϕ implies that the previous value of h_t has less power over the next value in the model that includes the realized volatility.

In figure 4 we plot the smoothed values of h_t for both models alongside the data.

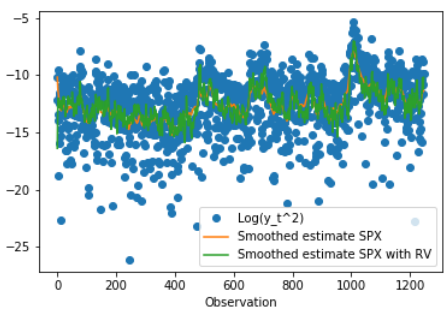

Figure 4: The SPX series with the smoothed estimates of h_t

Both models seem to capture the volatility quite well and get somewhat similar results. The RV model tends to estimate slightly higher values of h_t. Moreover, we present both the filtered and smoothed estimates of H_t for both models in figure 5 and 6 respectively.

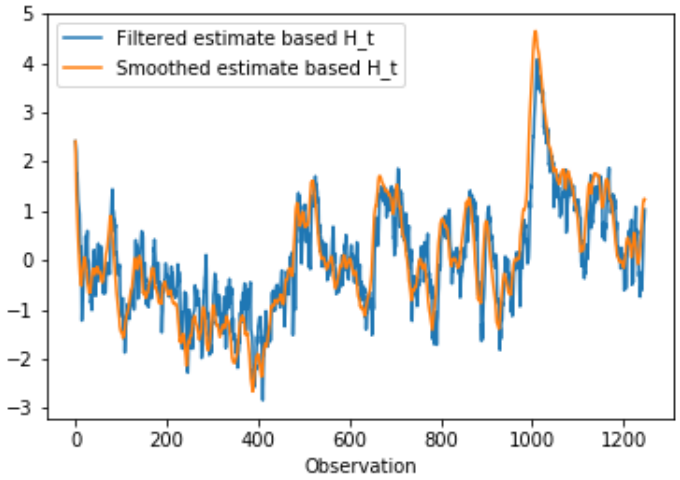

Figure 5: SPX Filtered and smoothed estimates of H_t

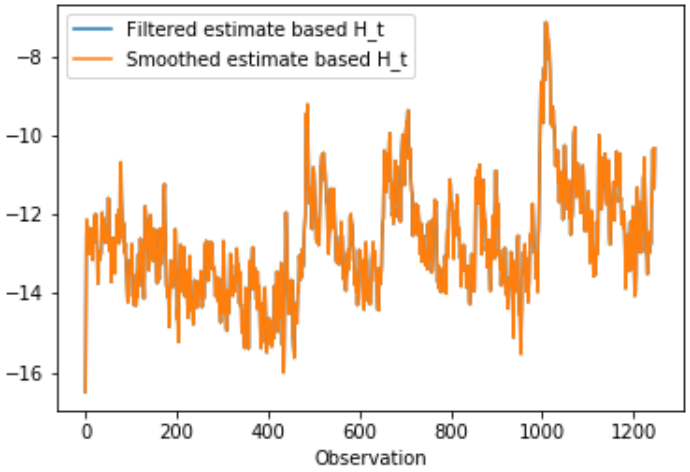

Figure 6: SPX Filtered and smoothed estimates of H_t with RV

The regular SV model provides values of H_t close to and around 0, according to expectations. The model still seems to perform as expected. The model which includes realized volatility has values of H_t around the mean of the data, as we no longer have $E[h_t] = \frac{\omega}{1-\phi}$.

References

Durbin, J. and S. J. Koopman (2012). *Time Series Analysis by State Space Methods* (Second ed.). Oxford University Press.

1 Appendix

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar  3 17:26:11 2021

@author: jmbru
"""

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



def max_likelihood(parameters,data,sigma_u_2):
    omega=parameters[0]
    phi=np.exp(parameters[1])/(1+np.exp(parameters[1]))
    sigma_eta=np.exp(parameters[2])
    v = np.zeros(len(data))
    a = np.zeros(len(data))
    P = np.zeros(len(data))
    F = np.zeros(len(data))
    v,a,P,F,k=kalman_filter(data,sigma_u_2,sigma_eta,omega,phi)
    value=-(len(data))/2 * np.log(2*np.pi) - 0.5*np.sum(np.log(F)+((v**2)/F))
    return -1*value

def max_likelihood2(parameters,data,sigma_u_2,RV):
    omega=parameters[0]
    phi=np.exp(parameters[1])/(1+np.exp(parameters[1]))
    sigma_eta=np.exp(parameters[2])
    beta=parameters[3]
    v,a,P,F,k=kalman_filter_RV(data,sigma_u_2,sigma_eta,omega,phi,beta,RV)
    value=-(len(data))/2 * np.log(2*np.pi) - 0.5*np.sum(np.log(F)+((v**2)/F))
    return -1*value

def optimize_variances(data):
    result=scipy.optimize.minimize(max_likelihood,[0,1,1],args=(data,(np.pi**2)/2),method='
    BFGS')
    omega,phi,sigma=result.x
    print(result)
    print(omega,np.exp(phi)/(1+np.exp(phi)),np.exp(sigma))
    return omega,np.exp(phi)/(1+np.exp(phi)),np.exp(sigma)

def optimize_variances2(data,RV):
    result=scipy.optimize.minimize(max_likelihood2,[1, 1, 1,1],args=(data,(np.pi**2)/2,RV),
    method='BFGS')
    omega,phi,sigma,beta=result.x
    print(result)
    print(omega,np.exp(phi)/(1+np.exp(phi)),np.exp(sigma),beta)
    return omega,np.exp(phi)/(1+np.exp(phi)),np.exp(sigma),beta

def kalman_filter(data,sigma_u_2,sigma_eta,omega,phi):
    v = np.zeros(len(data))
    a = np.zeros(len(data))
    P = np.zeros(len(data))
    F = np.zeros(len(data))
    a[0] =data[0]
    P[0] = 0
    v[0]=data[0]-a[0]
    F[0]=P[0]+sigma_u_2
    for t in range(1,len(data)):
        att=a[t-1]+P[t-1]*v[t-1]/F[t-1]
        a[t]=phi*att + omega
        Ptt=P[t-1]-(P[t-1]**2)/F[t-1]
```

```
        P[t]=phi**2 * (Ptt) + sigma_eta**2
        F[t]=P[t]+sigma_u_2
        if np.isnan(data[t]):
            v[t]=0
        else:
            v[t]=data[t]-a[t]
    return v,a,P,F,P/F

def smoothing_filter(data,k,a,F,v,P):
    L = 1- k
    r = np.zeros(len(data))
    alpha = np.zeros(len(data))
    V = np.zeros(len(data))
    N= np.zeros(len(data))
    for t in range(len(data)-1,-1, -1):
        if np.isnan(data[t]):
            r[t-1]=r[t]
            N[t-1]=N[t]
        else:
            r[t-1] = v[t]/F[t] + L[t]*r[t]
            N[t-1] = 1/F[t] +L[t]**2 * N[t]
        alpha[t]=a[t]+P[t]*r[t-1]
        V[t] =P[t]- P[t]**2 *N[t-1]

    return alpha,V,r,N

def kalman_filter_RV(data,sigma_u_2,sigma_eta,omega,phi,beta,RV):
    v = np.zeros(len(data))
    a = np.zeros(len(data))
    P = np.zeros(len(data))
    F = np.zeros(len(data))
    data=data
    a[0] =data[0]-beta*np.log(RV[0])
    P[0] = 0
    v[0]=data[0]-a[0]
    F[0]=P[0]+sigma_u_2
    for t in range(1,len(data)):
        att=a[t-1]+P[t-1]*v[t-1]/F[t-1]
        a[t]=phi*att + omega-beta*np.log(RV[t])
        Ptt=P[t-1]-(P[t-1]**2)/F[t-1]
        P[t]=phi**2 * (Ptt) + sigma_eta**2
        F[t]=P[t]+sigma_u_2
        if np.isnan(data[t]):
            v[t]=0
        else:
            v[t]=data[t]-a[t]
    return v,a,P,F,P/F

def smoothing_filter_RV(data,k,a,F,v,P,beta,RV):
    data=data-beta*np.log(RV)
    L = 1- k
    r = np.zeros(len(data))
    alpha = np.zeros(len(data))
    V = np.zeros(len(data))
    N= np.zeros(len(data))
    for t in range(len(data)-1,-1, -1):
        if np.isnan(data[t]):
            r[t-1]=r[t]
            N[t-1]=N[t]
        else:
            r[t-1] = v[t]/F[t] + L[t]*r[t]
            N[t-1] = 1/F[t] +L[t]**2 * N[t]
        alpha[t]=a[t]+P[t]*r[t-1]
        V[t] =P[t]- P[t]**2 *N[t-1]

    return alpha,V,r,N

def calculate_returns(data):
```

```
    returns=[]
    for i in range(1,len(data)):
        returns.append(np.log(data[i]/data[i-1]))
    return returns

def ImportData():
   data = pd.read_table('sv.dat')
   data =data.rename(columns= {'// Pound/Dollar daily exchange rates \sections 9.6 and 14.4'
     : 'Level ' })
   array = np.array(data)
   data =array[:,0]
   return(data)

def TransformData(data):
    return np.log((data-np.mean(data))**2)-1.27

def main():
   sigma_u_2=(np.pi**2)/2
   data = ImportData().astype(np.float)
   data=data/100-np.mean(data/100)


   plt.plot(data)
   plt.xlabel("Observation")
   plt.ylabel("Pound dollar exchange rate")
   plt.savefig("data")
   plt.show()

   transformed_data=np.log(data**2)
   plt.plot(transformed_data,'o')
   plt.xlabel("Observation")
   plt.ylabel("Log(y_t^2)")
   plt.savefig("Transformed data")
   plt.show()

   SPX=np.genfromtxt("SPX.csv", delimiter=',', dtype=float, skip_header=True)
   prices=SPX[:,17]
   RV=SPX[:,4][1:]
   prices=np.array(calculate_returns(prices))
   shift_transformed_prices=TransformData(prices)


   omega,phi,sigma_eta=optimize_variances((transformed_data-1.27))
   shift_transformed_data=transformed_data-1.27
   v,a,P,F,K=kalman_filter(shift_transformed_data,sigma_u_2,sigma_eta,omega,phi)
   alpha,V,r,N=smoothing_filter(shift_transformed_data,K,a,F,v,P)


   plt.plot(transformed_data,'o',label="Log(y_t^2)")
   plt.plot(alpha,label="Smoothed estimate")
   plt.legend()
   plt.xlabel("Observation")
   plt.savefig("Transformed data")
   plt.show()

   plt.plot(a-omega/(1-phi),label="Filtered estimate based H_t")
   plt.plot(alpha-omega/(1-phi),label="Smoothed estimate based H_t")
   plt.legend()
   plt.xlabel("Observation")
   plt.savefig("H_t estimates")
   plt.show()

   omega_SPX,phi_SPX,sigma_eta_SPX=optimize_variances((shift_transformed_prices))
   omega_SPX_RV,phi_SPX_RV,sigma_eta_SPX_RV,beta_SPX_RV=optimize_variances2(
    shift_transformed_prices,RV)

   v_SPX,a_SPX,P_SPX,F_SPX,K_SPX=kalman_filter(shift_transformed_prices,sigma_u_2,
    sigma_eta_SPX,omega_SPX,phi_SPX)
```

```
    alpha_SPX,V_SPX,r_SPX,N_SPX=smoothing_filter(shift_transformed_prices,K_SPX,a_SPX,F_SPX,
     v_SPX,P_SPX)
    v_SPX_RV,a_SPX_RV,P_SPX_RV,F_SPX_RV,K_SPX_RV=kalman_filter_RV(shift_transformed_prices,
     sigma_u_2,sigma_eta_SPX_RV,omega_SPX_RV,phi_SPX_RV,beta_SPX_RV,RV)
    alpha_SPX_RV,V_SPX_RV,r_SPX_RV,N_SPX_RV=smoothing_filter_RV(shift_transformed_prices,
     K_SPX_RV,a_SPX_RV,F_SPX_RV,v_SPX_RV,P_SPX_RV,beta_SPX_RV,RV)


    plt.plot(shift_transformed_prices,'o',label="Log(y_t^2)")
    plt.plot(alpha_SPX,label="Smoothed estimate SPX")
    plt.plot(alpha_SPX_RV,label="Smoothed estimate SPX with RV")
    plt.legend()
    plt.xlabel("Observation")
    plt.savefig("Transformed data SPX")
    plt.show()


    plt.plot(a_SPX-omega_SPX/(1-phi_SPX),label="Filtered estimate based H_t")
    plt.plot(alpha_SPX-omega_SPX/(1-phi_SPX),label="Smoothed estimate based H_t")
    plt.xlabel("Observation")
    plt.savefig("H_t estimates SPX")
    plt.legend()
    plt.show()

    plt.plot(a_SPX_RV-(omega_SPX_RV)/(1-phi_SPX_RV),label="Filtered estimate based H_t")
    plt.plot(alpha_SPX_RV-(omega_SPX_RV)/(1-phi_SPX_RV),label="Smoothed estimate based H_t")
    plt.xlabel("Observation")
    plt.savefig("H_t estimates SPX with RV")
    plt.legend()
    plt.show()

main()
```

Listing 1: Python code