Risk that an observed cluster of price jumps has not yet exhausted: performance of an estimate on simulated data

Cecilia Mancini, 6/8/2021

Abstract

This software is designed to support the research reported in *Warnings about future jumps: properties of the Exponential Hawkes model,* by Rachele Foschi, Francesca Lilla, and Cecilia Mancini, where it is assumed that the log-prices of a financial asset evolve following a jump diffusion semimartingale, as in (21) within the paper, and the process N counting the jumps is an Exponential Hawkes model. Formula (7) quantifies the probability that an observed cluster of price jumps is not yet finished, while feasible approximations are given by formulas (5) and (6).

The software allows to verify, on simulated discrete time data, the reliability of the results obtained with the practical implementation of (5) and (6). The analysis is mentioned in Appendix B.3.4 and produces the results shown in Table 8.

Description

This software is designed by Cecilia Mancini to support the research reported in *Warnings about future jumps: properties of the Exponential Hawkes model* by Rachele Foschi, University of Pisa; Francesca Lilla, Bank of Italy; and Cecilia Mancini, University of Verona. The last revision of the work is the draft of 5/8/2021.

The software aims to verify the reliability of the results obtained with the practical implementation of formulas (5) and (6) in estimating the quantity in (7), through their use on simulated prices of a financial asset. The analysis is mentioned in Appendix B.3.4 and produces the results shown in Table 8.

Namely, the simulation study assumes that the log-prices of a financial asset evolve following a jump diffusion semimartingale, as in (21) within the paper, where the process N counting the jumps is an Exponential Hawkes model. Formula (7) quantifies the probability that an observed cluster of price jumps is not yet finished, while (5) and (6) provide its feasible counterparts. The latter require the knowledge of an estimate of the number of jumps observed within a time interval [U,S], without necessarily knowing their exact time locations; an estimate of the jump intensity lambda_U at time U; and an estimate of the parameters lambda_0, alpha and beta of the Hawkes process kernel.

Since in practice very often we only can use prices recorded each 5 minutes, the estimates of the mentioned [illegible] estimating the parameters of the jump intensity. The first step is done using the Threshold estimation method of Mancini (2009), while the second one through maximizing the likelihood of observing the estimated jump times rather than the true jump times. These two steps lead to a sum of measurement errors and may deliver a relevant bias in the estimate of probability (7).

This simulation study shows that the probability estimation error is indeed low and makes the practical implementation of formulas (5) and (6) a reliable measure of the jump risk that we have studied.

The bibliographic references are in the paper.

The Matlab routine nests 4 other routines (names in red). One of them is reported below and in turn nests a further routine which is reported. Spot sigma is estimated as in Mancini, Mattiussi and Renò (2015), and the routines LogLikelHawMP1 and conHawkes have been written by Francesca Lilla.

The input needed to run the main routine MainGlobEstimErrSuPeBounds is the number n of the observed asset prices.

Software %%

```
function [output1,output2,output2b,output3,output3b]=MainGlobEstimErrOnPandBounds(n)
        %%inputs
        %we used n=150000, the number of JPM asset prices a tour disposal
delta= 1/(252*80); %T=n*delta=7.44 years
        %for the Brownian semimartingale part of (21) the parameters are the same as in Cont and Mancini
        (2011) and are simiar to the ones in Huang and Tauchen (2005)
mu=0; rho=-0.7; LgSigZ=log(0.3);  K=0.09; barLgSig=log(0.25); omega=0.05;
        %parameters of the Exponentila Hawkes process: [lambda0, alpha, beta]:
        the kernel has the form alpha* e^{-beta x}
parr=[53.27, 4.72, 9.14]; ep=0.01;
        %%simulation of the lof returns
[DX,sigmat,meansigmat,NT,tau]=DXStochVolHawkesJs(n,delta,mu,rho,LgSigZ,K,barLgSig,omega,parr);
        %% estimation of sigma and jump times
ri=DX';
ti=delta*[1:n];
fn0= 1/(200*delta);
t=delta*[1:n];
hatsigmaq = KernelSpotSigma2Thresh(ti,ri,t,fn0,delta);     %row
thr=2*hatsigmaq*delta*log(1/delta);     %row; log(1/delta)=9.9115
hatJTimes= ti'.*(ri.^2>thr');
        % estimation of sigma and jump times: second round
IndNoSalti=ones(n,1).*(hatJTimes==0);        %gives 1 where there is no jump, 0 where a jump was detected
riSecRound=ri.*IndNoSalti;                    %only the returns not containing jumps
hatsigmaqSecRound =  sigma2_SecRound(ti,riSecRound,t,fn0,delta, hatsigmaq);     %row
        % in the second round the estimate of sigma was lowered
thrSecRound=2*hatsigmaqSecRound*delta*log(1/delta);       %row
hatJTimesSecRound= ti'.*(riSecRound.^2>thrSecRound');
        %% Estimation of the Hawkes parameters
LB=[1e-9,1e-9,1e-9];        % constraints for fmincon: lambda_0, alpha, beta >0;
T=n*delta; M=1; P=1;
[MLE,f_opt] = fmincon(@(x) -LogLikelHawMP1(x,tau,T),theta0,[],[],[],[],LB,[],@(x)conHawkes(x))
        % conHakes imposes alpha<beta
        %% OUTPUT1
ISalti=ones(n,1).*(hatJTimes>0)+ ones(n,1).*(hatJTimesSecRound>0);
hatJTimes=ti'.*ISalti;
lambda0=parr(1); alpha=parr(2); beta=parr(3);
hLambda0=MLE(1); halpha=MLE(2); hbeta=MLE(3);
```

```
        %
EMeanSig=abs(mean(sqrt(hatsigmaqSecRound))-meansigmat)/meansigmat;
ENJ=abs(length(ISalti(ISalti>0))-NT)/NT;
ELambda0=abs(parr(1)-hLambda0)/lambda0; Ealpha=abs(parr(2)-halpha)/alpha; Ebeta=abs(parr(3)-
hbeta)/beta;
output1=[EMeanSig, ENJ, ELambda0, Ealpha, Ebeta];
        %% computation of: P(T_{i+k}<S + t_ep(S)), formula (7), and estimation error; of the bounds and their
        estimates, S=0.215; U=0.18;
        % S, S2 have to differ form delta*i, for any i, so that sum((hatJTimes==S2))==0
        % Note that P>0 iff lambdaS+alpha*I_{jump in S}>lambda0*(1+ep) !!!!!!
        % that is, ep ha to be sufficiently small, in both the cases where
        % S= the time of a true jump and S is not a jump time
        % Note that the bounds in Proposiition 1 are give only in the case where S is not a jump time
S=0.215; U=0.18;
[P,hP,lambdaS,hatLambdaS,lambdaU,hatLambdaU,LowerB,UpperB,hLowerB,hUpperB]=...
        ErrOnPandBds(U,S,tau,hatJTimes,lambda0,alpha, beta,hLambda0, halpha,hbeta, ep);
        %not meaningful to compute the average value of: lambdaS, lambdaU, LowerB, P, UpperB,
        %hLowerB,hP, hUpperB
        %
        %% OUTPUT  S; U;
ElambdaS=(hatLambdaS-lambdaS)/lambdaS; ElambdaU=(hatLambdaU-lambdaU)/lambdaU;
EP=abs((P-hP)/P);
distPeLB=(P-LowerB)/P;  distPehLB=abs(P-hLowerB)/P;
distPeUB=(UpperB-P)/P; distPehUB=abs(hUpperB-P)/P;
rangeBs=(UpperB-LowerB)/P; rangehBs=(hUpperB-hLowerB)/P;
ELB=abs(LowerB-hLowerB)/LowerB;
EUB=abs(UpperB-hUpperB)/UpperB;
if  lambdaS+alpha*sum((tau==S))>lambda0*(1+ep)...
   & hatLambdaS+halpha*sum((hatJTimes==S))>hLambda0*(1+ep)...
   & sum((tau==S))==0 & sum((hatJTimes==S))==0
   output2=[P, hP, EP, LowerB, hLowerB,ELB, UpperB, hUpperB, EUB];
   output2b=[distPeLB, distPehLB, distPeUB, distPehUB, rangeBs,rangehBs,ElambdaS, ElambdaU];
else  output2=2*ones(1,9); output2b=2*ones(1,8);
end;
        %% computation of: P(T_{i+k}<S + t_ep(S)) and error of its estimate; bounds  and their estimates,
        % with S2=0.07 U2=0.02 lower than before
S2=0.07; U2=0.02;
[P2,hP2,lambdaS2,hatLambdaS2,lambdaU2,hatLambdaU2,LowerB2,UpperB2,hLowerB2,hUpperB2]=...
ErrOnPandBds(U2,S2,tau,hatJTimes,lambda0,alpha, beta,hLambda0, halpha,hbeta, ep);
        %% OUTPUT  %S2; U2;      %~=
ElambdaS2=(hatLambdaS2-lambdaS2)/lambdaS2; ElambdaU2=(hatLambdaU2-lambdaU2)/lambdaU2;
EP2=abs((P2-hP2)/P2);
distPeLB2=(P2-LowerB2)/P2;  distPehLB2=abs(P2-hLowerB2)/P2;
distPeUB2=(UpperB2-P2)/P2; distPehUB2=abs(hUpperB2-P2)/P2;
rangeBs2=(UpperB2-LowerB2)/P2;   rangehBs2=(hUpperB2-hLowerB2)/P2;
ELB2=abs(LowerB2-hLowerB2)/LowerB2;
EUB2=abs(UpperB2-hUpperB2)/UpperB2;
if   lambdaS2+alpha*sum((tau==S2))>lambda0*(1+ep)...
    & hatLambdaS2+halpha*sum((hatJTimes==S2))>hLambda0*(1+ep)...
```

```
    & sum((tau==S2))==0 & sum((hatJTimes==S2))==0
    output3=[P2, hP2, EP2, LowerB2, hLowerB2,ELB2, UpperB2, hUpperB2, EUB2];
    output3b=[distPeLB2,distPehLB2,distPeUB2,distPehUB2,rangeBs2,rangehBs2,ElambdaS2,ElambdaU2];
else output3=2*ones(1,9); output3b=2*ones(1,8);
end;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [DX,sigmat,meansigmat,NT,tau]= DXStochVolHawkesJs(n,delta,mu,rho,LgSigZ,K,barLgSig,omega,pars)

        % Simulation of one path of a process having stochastic volatility as in Huang and Tauchen (2005) and
        % Exponential Hawkes jumps. Step delta, n= n di oss.
        % J sizes iid N(0, 0.03) independent on sigma: 0.03 stresses the difficulty in identifying the jumps and
        %renders the path of X similare to the one of JPM observed each 5 minutes on [2006, 2013]
        %% continuous part
DW=randn(1,n);
DW2=rho*DW+sqrt(1-rho^2)*randn(1,n);
LgSig(1)=LgSigZ;
for i=2:n u=LgSig(i-1); % definiz di u
   LgSig(i)=u-K*(u-barLgSig)*delta+omega*sqrt(delta)*DW2(i);
end;
sigmat=exp(LgSig);
DXc=(mu-sigmat.^2/2)*delta+ sqrt(delta)*sigmat.*DW;
        %% DN
T=n*delta; Tau= simulHawkCec(lambda0, alpha, beta, T)
DN=zeros(1, n); clear i;     %row
if floor(tau(1)/delta)==0
    DN(1)=1;
end;
for k=2:length(tau),
  if floor(tau(k)/delta)== floor(tau(k-1)/delta)
    DN(floor(tau(k)/delta)+1)=DN(floor(tau(k-1)/delta)+1)+1;
    else DN(floor(tau(k)/delta)+1)=1;
  end;
end;
        % DN(i)=m if we have m components k of tau within ((i-1)*delta, i*delta]
NT=sum(DN);
        %% J sizes
        %J sizes are Gauss and independent on sigma
clear i;
for i=1:n,  DJ(i)=sum(randn(1,DN(i))*0.03);  end;
        %% price path
DX=(mu-sigmat.^2/2)*delta+ sqrt(delta)*sigmat.*DW + DJ;
meansigmat=mean(sigmat);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function tau = simulHawkCec(mu, alpha, beta, T)
            %I simulate one path of a univariate Exponential Hawkes process (Rasmussen, 2011)
            % I take m(t)= lambda(t)= mu + alpha \sum_{t_i<t} e^{-beta(t-t_i)}, ell(t)=+\infty
            % the distribution function of the exponential law with par. lambda is 1-e^{-lambda x}
            % For example: mu=0.5; alpha=0.9; beta=1, as in e.g. in Rasmussen p. 6
            %n=152559; delta=1/(252*80);  are only needed to define T=n*delta;
t=0; c=0; lambdat=mu;
while t<=T
   V=rand(1); s=-log(1-V)/lambdat; U= rand(1);
   lambdatpius=mu+ (lambdat-mu+alpha)* exp(-beta*s);
   thresh=lambdatpius/lambdat;
   if (t+s>T)+(U>thresh)>0, t=t+s;
   else c=c+1, tau(c)=t+s, lambdat=lambdatpius; t=t+s;
   end;
end
            %figure(1), plot(tau, zeros(1, length(tau)), '.')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```