Big Data Report : Effect of Asynchronous arrival time on realized correlation

Audric Dongfack

March 2020

1 Introduction

In this paper, we propose to observe the effect of asynchronous arrival time of log-returns on the realized covariance. This will highlight the existence of the Epps effect[1].

The paper follows specific order of questions from the Big Data and Finance project given by Yoann Bourgeois. Codes written in $C++$ can be found at the end of the document.

Assumptions:
Let's consider the two following processes defined on a probability space (Ω , F , P).

$$dlog(X_t) = \mu_1 dt + \sigma_1 dW_t^1$$
$$dlog(Y_t) = \mu_2 dt + \sigma_2 dW_t^2$$

with W_t^i is a standard brownian motion, $d < W_t^1, W_t^1 >= \rho dt$, ρ is the constant instantaneous correlation between the two brownians. We consider a maturity $T = \frac{1}{365}$ (ie. one day); we note $M > 1$ an integer. for the discretization size such as $h = \frac{T}{M}$ is the discretization length. The realized covariance $V_h(w)$ is defined as:

$$V_h(w) = \sum_{i=1}^{M} (log(X_{t_i}^h(w)) - log(X_{t_{i-1}}^h(w))(log(Y_{t_i}^h(w)) - log(Y_{t_{i-1}}^h(w)))$$

for $w \in \Omega$

[1] In econometrics and time series analysis, the Epps effect, named after T. W. Epps, is the phenomenon that the empirical correlation between the returns of two different stocks decreases with the length of the interval for which the price changes are measured.

2 Convergence of the Monte Carlo estimator of the $\mathbb{E}[V_h]$

In this part we will show the convergence of the Monte Carlo estimator of the expectation $\mathbb{E}[V_h]$ thanks to a Monte Carlo simulation. We assume that we have synchronous arrival times for X and Y. We begin by recalling some conceptual results on the Monte Carlo estimator that we detail for the given expectation. Then we present the simulation results of this part in the table . In the table you will find the values of the Monte Carlo estimator that we have constructed, in the first row there is the values of N and in the first column the values of M.

2.1 Monte Carlo estimator

Monte Carlo methods allow quantities to be estimated using the simulation of random variables. Problems that may be encountered include calculating integrals, optimization problems and the resolution of linear systems. The method's simplicity, flexibility and efficiency for large scale problems make it an interesting tool that can serve as an alternative or reference for other numerical methods.

Let's consider a random variable $X = (X_1, ...X_d)$ which follows the law ν on $\mathbb{R}^d$ ($X \sim \nu$). And h a defined function from $\mathbb{R}^d$ to $\mathbb{R}$. we want to estimate :

$$\mathbb{E}_\nu[h(X)] = \int h(x)\nu(dx)$$

The standard solution to this problem is to simulate a sequence $(X_n)_{n>1} = (X_{1,n}, ...X_{d,n})_{n>1}$ of independent random variables identically distributed according to a ν distribution. Then, we estimate the expectation by the empirical mean.

$$Empirical - mean = \frac{1}{N}\sum_{n=1}^{N} h(X_n)$$

This is **Monte-Carlo estimator** of $\mathbb{E}_\nu[h(X)]$
In the rest of the section, we consider d = 1 and h is the identity function on $\mathbb{R}$. The convergence of expectation $\mathbb{E}_\nu[X]$ is ensured by the law of large numbers.

Theorem: Convergence of the Monte Carlo estimator - law of large numbers
For X an integrable random variable, we have with probability 1 :

$$\lim_{N-\infty} \frac{1}{N}\sum_{n=1}^{N} X_n = \mathbb{E}_\nu[X]$$

Sketch of proof : The evidence is multiple, notably by assuming X of finite variance and performing second moment calculations on the empirical mean, or without additional variance conditions using martingale convergences. Here, we

propose to follow Kolmogoroff's historical approach by assuming only integrable X.

General equivalence:

$$\mathbb{E}|X| < \infty \iff \sum_{n=1}^{N} \mathbb{P}(|X| > n) < \infty$$

This relation came from the fact that : $\mathbb{E}|X| = \int_0^\infty \mathbb{P}(|X| > n)dx$. Let's pose now :

$$Y_n = X_n \mathbb{I}_{|X_n| \leq n}, Z_n = Y_n - \mathbb{E}(Y_n)$$

From the general equivalence, we remark that :

$$\sum_{n=1}^{\infty} \mathbb{P}(X_n \neq Y_n) = \sum_{n=1}^{\infty} \mathbb{P}(|X_n| > n) = \sum_{n=1}^{\infty} \mathbb{P}(|X| > n) < \infty$$

which implies by Borel-Cantelli's lemma (almost surely), the sequences (X_n) and (Y_n) have only a finite number of different elements. Thus, with probability 1, if one of the limits $\lim_{N-\infty} \frac{1}{N} \sum_{n=1}^{N} X_n$ or $\lim_{N-\infty} \frac{1}{N} \sum_{n=1}^{N} Y_n$ exists and is finite, then the other limit also exists and both limits are equal. Using the dominated convergence theorem, we justify $\lim_{N-\infty} \mathbb{E}(Y_n) = \mathbb{E}(X)$ and thus $\lim_{N-\infty} \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}(Y_n) = \mathbb{E}(X)$. It is thus sufficient to prove that $\lim_{N-\infty} \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}(Z_n) = 0$ P.S . One admits at this level that $\sum_{n=1}^{N} \frac{Z_n}{n}$ converges for n tends to ∞ (The proof is very complex). Kronecker's lemma (with $a_n = n$) allows us to conclude the proof of the theorem.

Lemma (Kronecker) : If (a_n) is an increasing sequence towards infinity and if $\sum x_n$ is a converging series , then $\frac{1}{a_N} \sum_{n=1}^{N} a_n x_n$ tends to 0 when n tends to ∞

So we do have the theoretical convergence of the **Monter carlo estimator**.

2.2 Convergence of the Monte Carlo Estimator of $\mathbb{E}[V_h]$

compared to our previous theoretical analysis, Monte Carlo estimator of $\mathbb{E}[V_h]$ is given by :

$$MSE(\mathbb{E}[V_h]) = \frac{1}{N} \sum_{n=1}^{N} V_h^n$$

For different values of N and M we find the following table:

Convergence of the Monte Carlo Estimator						
(M,N)	2	3	4	5	6	7
3	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

We can notice that the Monte Carlo estimator converges well. This matrix contains only null values which shows that our estimator converges in all cases towards 0.The values of N that have been chosen remain quite consistent, especially as the maturity set is small.

3 Monte Carlo estimators for $\mathbb{E}[V_h]$ with respect to several Poisson intensities and the effect of asynchronicity

In this section, we will now assume that the arrival times are different for X and Y. We will also assume in this part that the variables X and Y are two independent Poisson processes. Considering these assumptions, we will also calculate the Monte Carlo estimator. In this part as in the previous one, we will first explain a theoretical part which allows us to show how to simulate a Poisson law.

3.1 Definition of a Poisson process

The Poisson process is an extremely used tool in counting phenomena. Indeed, it appears naturally in these situations, as we will see below.

We look at events that occur at random dates, and we're interested, for any $t > 0$ to the number of events that occurred during the interval $N(t)$. The Poisson process of intensity $\lambda > 0$ is then defined as a process $N(t)$ satisfying the following conditions:

1. C_1 : the process is incrementally independent, i.e. for all $0 < t_1 < ..... < t_n$ the random variables $N(t_i) - N(t_{i-1})$ are globally independent
2. C_2 : the process is stationary increments, i.e. for all t, h the law of $N(t_i) - N(t_{i-1})$ depends only on h
3. C_3 : $P(N(t+h) - N(t) > 1) = \lambda h + o(h)$ and $P(N(t+h) - N(t) > 1) = o(h)$. This Condition means that over a small interval of time, 0 or 1 event is observed, and the probability of observing an event is proportional to the spent time. Note that $N(0) = 0$. The trajectories are by definition increasing, continuous on the right with a limit on the left. They increase by jumps of one unit.

Link with Poisson's Law : for all $t > 0$, the random variable follows the Poisson's law of parameter λt . i.e

$$P(N(t) = n) = e^{-\lambda t} \frac{\lambda t^n}{n!}$$

3.2 The emergence of exponential law

When observing a Poisson process, it is natural to be interested in the waiting time between jumps; the basic result is as follows:
Property: If T_n designe the instant of the n-th jump, then the variables $T_n - T_{n-1}$ are iid and follow the exponential law of parameter λ.

This result justifies the particular place of the exponential law in the study of duration models. We deduce that the law ofT_n is the Erlang's law of parameters (n, λ).

3.3 Non-homogeneous Poisson process

In the condition C_3 characterizing the Poisson process, a time-dependent intensity can be introduced, $\lambda(t)$. We obtain a process which is no longer stationary, the law of increase $N(t+h) - N(t)$ is then a Poisson's law of parameter $\int_t^{t+h} \lambda(u)du$.
Then $N(t)$ follows a Poisson law of parameter:

$$\int_0^t \lambda(u)du$$

We also have the following property : $P(N(t) > 0) = e^{-\int_0^t \lambda(u)du}$
Relationship with the Uniform Law
Let's consider F_τ the distribution function of τ, we have the following result.

$$F_\tau(t) = P(0 < \tau < t) = 1 - e^{-\int_0^t \lambda(u)du}$$

If U designe a uniform law on $[0,1]$, we have for all $x\ in [0,1]$: $F_U(t) = x$.
Thus,

$$P(U < F_\tau(t)) = P(0 < \tau < t) = P(N(t) > 0)$$

3.4 Simulation of Poisson's law

The algorithm for simulating a Poisson's law is defined as follows.

Algorithm 1: Poisson's law simulation algorithm

```
Result: We return X
X = 0, P = 1;
while P > e^{-λ} do
    the variable u is pull uniformly over [0,1];
    P = P*u;
    X = X+1;
end
```

The results of this Part are presented in the table below. λ_1, λ_2 and the values of the Monte Carlo estimator are observed. In the first column we have

the values of λ_2 and in the first row the values of λ_1. We have simulated with values of M =12 and N=3.

Convergence of the Monte Carlo Estimator			
(λ_1,λ_2)	(0,1)	(0,2)	(0,3)
(0,1)	1.14	0.97	1.06
(0,2)	0.74	0.55	0.60
(0,3)	1.14	0.69	0.64

The colour code of the table is as follows:

1. Black values were obtained for M =12 and N=3
2. The value in red was obtained for M =12 and N=2
3. Values in blue were obtained for M =12 and N=1
4. The value in green was obtained for M =5 and N=1

It can be seen that the higher the values of λ_1 and λ_2 increase, the weaker the convergence becomes. This makes it easier to see because one is obliged to decrease M and N to make the estimator converge from certain values of λ_1 and λ_2. We also notice that the convergence of the estimator depends on the sampling frequency of the model. The smaller h is, the harder the estimator converges.
We still find the convergence of the estimator but with constraints on λ_1 and λ_2, and also on M and N. We can see that the estimator converges to values that have an average of 0.88.

4 Hayashi-Yoshida estimator

The Hayashi-Yoshida estimator is defined as follow :

$$HY_n = \sum_{i,j} \Delta X(I_i^x)\Delta Y(I_j^y)\mathbb{I}_{I_i^x \cap I_j^y \neq \oslash}$$

Where I_i^x is the time interval between t_i^x and t_{i+1}^x (see figure). n is the numbers of intervals.
The algorithm is define as follow:

Algorithm 2: Hayashi-Yoshida estimator

Result: We return C
$C = 0$, $i, j = 1$;
while $i < n$ *or* $j < n$ **do**
  C= C + $\Delta X(I_i^*)\Delta Y(I_j^*)$;
  if $t_j^y < t_i^x$ **then**
    $j++$;
  else
    $i++$;
  end
end

Codes

The following codes were written in C++, the codeblock compiler was used. However the reader can use online compilers such as OnlineGDB.

For Question-1

```
#include <iostream>
#include <math.h>
#include <random>
#include <time.h>
#include <chrono>
#include <fstream>
using namespace std;
// We first define the covariance estimator
double covariance (double X0, double sigma1,double sigma2,double
    mu1,double mu2,double rho,int M)
{ int T=1/365
int h;
h=(T/M);
double Cov=0;
    // we generate a two correlate gaussian variable
    unsigned seed=chrono::system_clock::now().time_since_epoch().
    count();
    default_random_engine generator (seed);
    normal_distribution <double > gaussian;
    double W1 = gaussian(generator);
    double Z = gaussian (generator);
    double W2 = rho * W1 + sqrt (1 - rho * rho)*Z;
    // we compute the covariance
    double Xt, Xt1, Yt, Yt1;
    for (int t=1;t<=M;t++){
    // process
    Xt = X0 * exp (mu1*t*h + sigma1*W1*sqrt(t*h));
    Xt1= X0 * exp (mu1*(t-1)*h + sigma1*W1*sqrt(t-1)*h);
    Yt = X0 * exp (mu2*t*h + sigma2*W2*sqrt(t)*h);
    Yt1= X0 * exp (mu2*(t-1)*h + sigma2*W2*sqrt(t-1)*h);
    Cov=Cov+ (log10(Xt)-log10(Xt1))*(log10(Yt)-log10(Yt1));
    }
    return Cov;
```

```
}
    // The following function define the monte carlo estimator
double expectation(int N, int M){
    double R;
    For (int t=1;t<=N;t++) {
        R+=covariance (1, 0.15,0.15 , 0, 0,0.9, M);
    }
    double MCE=(R/N);
    return MCE;
}
    // main function
int main ()
{
printf ("The expectation is :\%f", expectation(6,3));
return 0 ;
}
```

For Question-2

```
#include <iostream>
#include <math.h>
#include <random>
#include <time.h>
#include <chrono>
#include <fstream>
using namespace std;
using namespace std;
double cov (float lamda1,float lamda2,int M){
    int T=1/365;
    int h;
    h=T/M;
    double C =0;
    // unifor random variable
    unsigned seed = chrono:system_clock: now ().time_since_epoch ()
    .count ();
    default_random_engine generator (seed);
    uniform_real_distribution < double >distribution (0.0, 1.0);
    double U = distribution (generator);
    // poisson algorithm
    for (int t=0;t<=M;t++){
        int b = 0;
        double s = 0;
        float n = 2;
        // first loop on xt process
        while (b == 0){
                n = n + 1 ;
                s = (s - (log (U) / lamda1));
                if (s > t) {
                    b = 1;
                }
        }
        double d1=(log (n)-log(n-1));
    // second loop on yt process
        int k=0;
        int p=0;
        int m=2;
```

```
37          while (k == 0){
38              m = m + 1 ;
39              p = (p - (log (U) / lamda2));
40              if (p > t) {
41                  k = 1;
42
43              }
44
45          }
46          double d2=(log(m)-log(m-1));
47          C= C+(d1*d2);
48
49      }
50  return C;
51  }
52  // Monte carlo estimator
53  double expectation (int N , int M){
54      double R;
55      for (int x=1;x<=N;x++){
56          R+=cov(0.1,0.1,M);
57
58      }
59      double MCE=(R/N);
60      return MCE;
61  }
62      // main function
63  int main ()
64  {
65      printf ( "the expectation is  \%f",expectation(5, 6));
66      return 0;
67  }
```

For Question-3

```
1  #include <iostream>
2  #include <math.h>
3  #include <random>
4  #include <time.h>
5  #include <chrono>
6  #include <fstream>
7  using namespace std;
8  // We first define the covariance estimator
9  double covariance (double ti,double ti1,double tj,double tj1,double
       X0, double sigma1,double sigma2,double mu1,double mu2,double
      rho)
10 {
11     // we generate a two correlate gaussian variable
12     unsigned seed=chrono::system_clock::now().time_since_epoch().
      count();
13     default_random_engine generator (seed);
14     normal_distribution <double > gaussian;
15     double W1 = gaussian(generator);
16     double Z = gaussian (generator);
17     double W2 = rho * W1 + sqrt (1 - rho * rho)*Z;
18     // we compute the covariance
19     double Xt, Xt1, Yt, Yt1;
```

```
20      // process
21      Xt = X0 * exp (mu1*ti*h + sigma1*W1*sqrt(ti*h));
22      Xt1= X0 * exp (mu1*(ti1)*h + sigma1*W1*sqrt(ti1-1)*h);
23      Yt = X0 * exp (mu2*tj*h + sigma2*W2*sqrt(tj)*h);
24      Yt1= X0 * exp (mu2*(t1)*h + sigma2*W2*sqrt(t1)*h);
25      Cov=(Xt-Xt1)*(Yt-Yt1);
26      return Cov;
27  }
28
29  // main function
30  int main ()
31  {
32      Cov = 0;
33      int i = 1;
34      int j = 1;
35      //Ty[] ,T[] we initialize this two array as we want
36      // we define time discretization of i and j
37      while(i <n or j <n){
38
39          Cov = Cov + covariance(Tx[i],Tx[i-1],Ty[j],Ty[j-1]1,
        0.15,0.15 , 0, 0,0.9, M);
40
41          if (Ty[j] < Tx[i] ){
42              j++;
43          }else{
44              i++;
45          }
46  }
47
48  printf ("The Hayashi-Yoshida estimator is given by :\%f", Cov );
49  return 0 ;
50  }
```

References

[1] Yacine Aït et al. High frequence covariance estimates with noisy and asynchronous financial data

[2] TAkaki Hayashi and Nakahiro Yoshida[2005]. On covariance estimation of non-synchronously observed diffusion processes

[3] Julien STOEHR. Méthodes de Monte Carlo, Université de paris Dauphine