Malliavin-Mancino estimators implemented with non-uniform fast Fourier transforms

Patrick Chang[a], Etienne Pienaar[a], Tim Gebbie[a]

[a]*Department of Statistical Science, University of Cape Town, Rondebosch 7701, South Africa*

Abstract

We implement and test kernel averaging Non-Uniform Fast-Fourier Transform (NUFFT) methods to enhance the performance of correlation and covariance estimation on asynchronously sampled event-data using the Malliavin-Mancino Fourier estimator. The methods are benchmarked for Dirichlet and Fejér Fourier basis kernels. We consider test cases formed from Geometric Brownian motions to replicate synchronous and asynchronous data for benchmarking purposes. We consider three standard averaging kernels to convolve the event-data for synchronisation via over-sampling for use with the Fast Fourier Transform (FFT): the Gaussian kernel, the Kaiser-Bessel kernel, and the exponential of semi-circle kernel. First, this allows us to demonstrate the performance of the estimator with different combinations of basis kernels and averaging kernels. Second, we investigate and compare the impact of the averaging scales explicit in each averaging kernel and its relationship between the time-scale averaging implicit in the Malliavin-Mancino estimator. Third, we demonstrate the relationship between time-scale averaging based on the number of Fourier coefficients used in the estimator to a theoretical model of the Epps effect. We briefly demonstrate the methods on Trade-and-Quote (TAQ) data from the Johannesburg Stock Exchange to make an initial visualisation of the correlation dynamics for various time-scales under market microstructure.

Keywords: Malliavin-Mancino estimator, non-uniform fast Fourier transform, Trade-and-Quote event-data, Epps effect

Contents

1. Introduction

Data-informed approaches to modelling the relationships between fast asynchronous streaming event-data features requires efficient algorithms to compute the dependency or similarity across data features. This can be useful to relate collections of similar features to similar but potentially useful information on the appropriate decision time-scale. When the dependency structure can be approximated by an averaged realised correlation or covariance matrix then the problem of estimation from asynchronous event data can be significantly simplified. Then the problem of correlation and covariance estimation over asynchronous event data can be addressed using the Malliavin-Mancino estimator [1–3].

This has several advantages over *ad-hoc* averaging and interpolation methods built on the underlying assumptions of continuity, such as the approach taken in the well understood Hayashi-Yoshida estimator [4]. However, the Malliavin-Mancino estimator is built on numerically evaluating Fourier transforms and their inverses. This has a computational cost. Quickly extracting realised correlations or covariances on a given time-scale for large feature sets of distinct asynchronous events without biased interpolation is key to avoiding spurious correlations that can lead to ineffective decision making under uncertainty.

Email addresses: CHNPAT005@myuct.ac.za (Patrick Chang), etienne.pienaar@uct.ac.za (Etienne Pienaar), tim.gebbie@uct.ac.za (Tim Gebbie)

This paper directly addresses two key issues: First, that of performance as measured by computational speed. Second, the implicit dependence of time-scale in the estimation of realised covariances and correlations on asynchronous event data using the Malliavin-Mancino estimator. The key contribution is to mitigate the first problem using non-uniform fast Fourier transforms to compute the Malliavin-Mancino estimator, and to provide clarity into the second idea using insights from the non-uniform fast Fourier transform.

Performance is a key requirement in two related use cases. That of being able to carry out large scale Monte-Carlo simulations over many features and many time-scales where one needs to iterate and recompute the correlation matrix over event data. In addition to the speed requirements for simulation, in a real-time environment where decisions are being made on streaming event-data, the use of fast methods can reduce the time-scales of effective data-sampling. For example, the minimum effective sampling rate of correlation based state detection is bounded by the compute time of the correlation matrix. A speed improvement on the compute time of the realised covariance, or realised correlation matrix, potentially allows more time for learning algorithm convergence and identification. This can be of particular importance for learning algorithms that require many updates to identify a reliable optimal relationship between actions and system states given an objective, such as Q-learning based implementations of reinforcement learning for trading [5–7].

Concretely, we extend an approach to performance enhancement based on the fast Fourier transform [8] in the context of the Malliavin-Mancino estimator [1, 2] by using non-uniform fast Fourier transform methods [9–11]. This combines the performance advantage of fast Fourier transforms while providing intuition into the time-scale averaging. This follows from the basic idea behind the non-uniform fast Fourier transform; by convolving the data onto a uniform grid (dependent on the number of Fourier coefficients required) through a choice of averaging kernel. Furthermore, the averaging kernel has an explicit averaging scale which provides avenues for controlling speed and accuracy.

We hope to follow Reno [12] and Precup and Iori [13] by using the choice of the number of Fourier coefficients, N, as the method of tuning the estimation to different time-scales. To implement this with confidence, given that we use NUFFT to improve the compute times, we need to understand the relative dependencies between kernel averaging (proxied by the tolerances) and time-scale averaging (proxied by the number of Fourier coefficients) under simulation; to evaluate their impact on the estimated correlations.

To explore this idea we consider three different averaging kernels: 1.) the Gaussian kernel [10] (See equations (5) and (6)), 2.) the Kaiser-Bessel kernel [14] (See equations (8) and (9)), and 3.) the exponential of semi-circle kernel [11] (See equations (12) and (13)). In conjunction with these choices of averaging kernels we consider two different choices of Fourier basis kernels: i.) the Dirichlet, and ii.) the Fejér basis kernels. Combinations of these are compared with different length and breadth data-sets and for different numbers of Fourier coefficients. This allows us to better understand the relative algorithm performance by comparing algorithm compute times, with data-size and various tolerance levels (See Figure sets 4 and 5).

These combinations of kernel choices are benchmarked against three vanilla algorithms that implement the Malliavin-Mancino estimator: 1.) the benchmark "for-loop" implementation first provided by Mancino, Rechioni and Sanfelici [3], 2.) a vectorised implementation with speed enhancements assuming real-valued data [5, 15, 16], and 3.) a zero-padded Fast Fourier implementation [16, 17] that allows the use of the fast Fourier transform on asynchronous data without the need to apply an averaging kernel, but using an underlying missing data approach to implement lossless interpolation.

Here an important observation is that using the zero-padded FFT to compute the MM estimator can only work for uniformly sampled data that has missing data points and fails for truly asynchronous data (See Figure 6 and Section 2.2.4). This is the key motivation for the necessary requirement of using a non-uniform FFT in the setting of speeding up the compute time of the Malliavin-Mancino estimator using the fast Fourier transform method for asynchronous event data. The zero-padding FFT biases the data; the non-uniform FFT does not if correctly used. It is for this reason that we promote the idea of using the NUFFT in conjunction with the MM estimator if the data is asynchronous, discrete and event driven.

The paper is organised as follows: Section 2, we outline the various implementation methods for the Malliavin-Mancino estimator. Section 3, we benchmark the various algorithms to understand the factors impacting speed and accuracy. Section 4, we demonstrate the link between the number of Fourier coefficients and the implicit time-scale investigated; along with its relation to a theoretical model of the Epps effect. We then carry-out EDA on real world TAQ data to investigate the correlation dynamics under market microstructure. We finally conclude in Section 5 to summarise our findings.

2. Algorithm Outline

2.1. Malliavin-Mancino estimators

Malliavin and Mancino [1, 2] proposed an estimator that is constructed in the frequency domain. It expresses the Fourier coefficients of the volatility process using the Fourier coefficients of the price process $p_i(t) = \ln(S_i(t))$ where $S_i(t)$ is a generic asset price at time t. By re-scaling the trading times from $[0, T]$ to $[0, 2\pi]$ (See Algorithm 2) and using the *Bohr* convolution product (See Theorem 2.1 [1]) we have that for all $k \in \mathbb{Z}$ and N samples:

$$\mathcal{F}(\Sigma^{ij})(k) = \lim_{N\to\infty} \frac{2\pi}{2N+1} \sum_{|s|\leq N} \mathcal{F}(dp_i)(s)\mathcal{F}(dp_j)(k-s). \quad (1)$$

Here $\mathcal{F}(*)(\star)$ is the $\star^{\text{th}}$ Fourier coefficient of the $*$ process. Now using previous tick interpolation to avoid a downward bias in the estimator [18] and a simple function approximation for the Fourier coefficients (See [1, 4, 15]), we obtain the Dirichlet representation of the integrated volatility/co-volatility estimator:

$$\hat{\Sigma}^{ij}_{n,N} = \frac{1}{2N+1} \sum_{\substack{|s|\leq N \\ j=1,i=1}}^{n-1,n-1} e^{is(t_i - t_j)} \delta_1(I_i)\delta_2(I_j), \tag{2}$$

where the trade intervals are $I_i := [t_i, t_{i+1})$ and $I_j := [t_j, t_{j+1})$ while the price fluctuation are $\delta_1(I_i) := p_1(t_{i+1}) - p_1(t_i)$ and $\delta_2(I_j) := p_2(t_{j+1}) - p_2(t_j)$ for the i^{th} and j^{th} asset respectively.

An alternate version of the Fourier estimator is the Fejér representation:

$$\hat{\Sigma}^{ij}_{n,N} = \frac{1}{N+1} \sum_{\substack{|s|\leq N \\ j=1,i=1}}^{n-1,n-1} \left(1 - \frac{|s|}{N}\right) e^{is(t_i - t_j)} \delta_1(I_i)\delta_2(I_j), \tag{3}$$

which is more stable under the presence of market microstructure noise [1].

The various implementation methods follow the same general structure (Outlined in Algorithm 1). First we re-scale the trading times from $[0, T]$ to $[0, 2\pi]$ (See Algorithm 2) and compute the Nyquist frequency[1] (See Algorithm 4). Second we compute the non-normalised Fourier coefficients $\mathcal{F}(dp_i)(k)$ $k \in \{-N, ..., N\}$ for all assets, and finally, we compute either the Dirichlet or Fejér representation of the estimator. The implementation methods only differ in the computation for the Fourier coefficients.

2.2. Implementation methods

We outline the various methods to evaluate the Fourier coefficients $\mathcal{F}(dp_i)(k)$ along with their use-case [2], benefits, pitfalls and general algorithm complexity [3].

2.2.1. Benchmark for-loop implementation

The Mancino *et al.* implementation (See Algorithm 5) is from the appendix of [3] and uses a for-loop construction. The evaluation relies on looping through $\{-N, ..., N\}$ to compute the k^{th} Fourier mode. The implementation does not rely on any techniques to improve performance and will act as a benchmark to compare against other methods. The method can be used for all synchronous and asynchronous cases and the complexity is the same as Discrete Fourier Transforms (DFTs) complexity of $O(n^2)$.

2.2.2. Vectorised implementation

The legacy code implementation (See Algorithm 6) is based on a MATLAB implementation [16, 17]. The difference compared to the Mancino *et al.* implementation is that all the Fourier modes are evaluated in parallel by vectorising the computation. Here we further improve upon the legacy code by exploiting techniques found in [3]. Concretely we exploit the Hermitian symmetry $\mathcal{F}(dp_i)(k) = \overline{\mathcal{F}(dp_i)(-k)}$ where $\overline{\mathcal{F}}$ denotes the conjugate function of $\mathcal{F}$. This is possible because the source strengths δ_j are all real-valued. Therefore, we only need to evaluate $k \in \{1, ..., N\}$ and obtain the conjugates for these Fourier modes. Finally, $\mathcal{F}(dp_i)(0) = \sum_{j-1}^{n-1} \delta_j / 2\pi$ must be computed to complete the range of Fourier modes required for the convolution. The method can be used for all synchronous and asynchronous cases and the complexity is the same as DFTs $O(n^2)$. The key concern with this method is the memory usage constraints that it can face. After inspecting Algorithm 6 we see that a large matrix of size (nxN) is required for the vectorisation and this can adversely affect performance by either: i.) pre-maturely ending the computation due to either insufficient memory or heap-size constraints, or ii.) slow down performance due to an over-reliance on virtual-memory management. Hence care with regards to memory management is crucial for effective performance enhancement for large data-sets.

2.2.3. The fast Fourier transform

The FFT implementation we use here is the current state-of-the-art FFTW package [19] based on the Cooley-Tukey algorithm [8] to compute the Fourier modes. This implementation also exploits the Hermitian symmetry making this the fastest current implementation known to the authors; this implementation has a well understood complexity of $O(n \log n)$. However, the key constraint relating to this method is its restriction to strictly synchronous data so that the evaluation becomes a simple discrete Fourier transform.

2.2.4. The zero-padded fast Fourier transform

The Zero-padded FFT (ZFFT) implementation extends the FFT implementation by zero padding missing observations allowing the naive computation for asynchrony using a missing data representation (See Algorithm 7). The implementation is achieved by computing the minimum sampling interval Δt and creating a new over-sampled grid with intervals Δt. The observations are then placed at the nearest neighbour on the new over-sampled grid [4]. The FFT algorithm is then applied to the new over-sampled grid. The implementation retains a complexity of $O(n \log n)$ but is slower than the FFT implementation as it does not exploit the Hermitian symmetry and requires the additional step of creating an over-sampled grid. This is our benchmark asynchronous approach to the fast Fourier transform.

Figure 1 demonstrates two points: i.) how the zero-padded implementation works, and ii.) why the implementation does not work for the asynchronous case using an arrival time representation. When asynchrony is induced using a missing data representation, the original grid has equal spacing Δt. Therefore the points on the over-sampled grid will be the same time points as the original grid, and the points will either have zero or the corresponding δ_j - meaning there is no shifting of time points. However, when asynchrony is induced using an arrival time representation, the original grid does not have equal spacing Δt. Therefore the time points from the original grid will be shifted (Seen in the third arrow from the left in Figure 1).

[1] Mancino *et al.* [3] picks N such that MSE is minimised.

[2] The use-case refers to the ability to evaluate synchronous or asynchronous time-series data.

[3] The complexity is given only for the synchronous case.

[4] It is also recommended that this be implement to preserve the filtration structure of time-series events by moving to the nearest right neighbour so that information in the future is shifted, at worst, further into the future, but never into the past relative to a particular time to avoid temporal contamination.

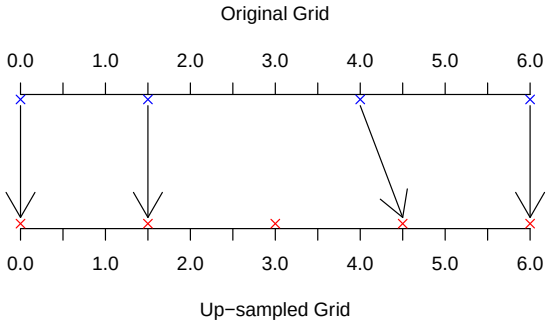

Figure 1: The figure is a toy example to show how the zero-padding works for the zero-padded FFT implementation. The minimum sampling interval Δt = 1.5. A new uniform over-sampled grid is created and observations are placed on the nearest neighbouring point on the over-sampled grid.

2.2.5. *Non-uniform fast Fourier transform*

The Non-Uniform FFT (NUFFT) implementation of the Malliavin-Mancino estimator is the main contribution of this paper. We are interested in the evaluation of:

$$F(k) = \frac{1}{2\pi}\sum_{j=1}^{n-1} f_j e^{-ikx_j}, \tag{4}$$

where $x_j \in [0, 2\pi]$ and $k \in \{-N, ..., N\}$. Therefore, the NUFFT we are interested is known as the 1-dimensional "type 1" NUFFT [10, 11] (also known as the adjoint NUFFT [14]). We use the more popular NUFFT approach - by first convolving the non-uniform source points onto an over-sampled uniform grid, to combine this with a FFT on the uniform grid, and then deconvolving the effects of the convolution on the Fourier space [10, 11, 14].

Here the convolution is achieved with a kernel $\varphi(x)$ [5]. We consider the three most popular kernels: the Gaussian kernel using the fast Gaussian gridding implementation from [10], the Kaiser-Bessel kernel using the implementation approach of [14], and the exponential of semi-circle used by the state-of-the-art FINUFFT package [11].

To set the theoretical scene, let $M = 2N + 1$ be the number of Fourier modes we want returned, σ be the over-sampling ratio (most studies have settled on $\sigma = 2$ [11]), ξ_ℓ be the ℓ^{th} location on the over-sampled grid with $\ell \in \{0, ..., M_r - 1 = \sigma M - 1\}$ and ω is the spreading width with M_{sp} as the spreading in each direction. The Gaussian kernel and its Fourier transform is defined as:

$$\varphi_{G}(x) = e^{-x^2/4\tau}, \tag{5}$$

and

$$\hat{\varphi}_{G}(k) = \sqrt{2\pi}e^{-k^2\tau}, \tag{6}$$

where τ is defined as

$$\tau = \frac{1}{M^2}\frac{\pi}{\sigma(\sigma - 0.5)}M_{sp}. \tag{7}$$

[5] The choice of kernel has a fascinating history and has a significant impact on the speed of NUFFTs. We refer the reader to [11] for further details.

The Kaiser-Bessel pair is defined as:

$$\varphi_{KB}(x) = \frac{1}{\pi}\begin{cases} \dfrac{\sinh\left(b\sqrt{M_{sp}^2 - M_r^2x^2}\right)}{\sqrt{M_{sp}^2 - M_r^2x^2}} & |x| \leq \frac{M_{sp}}{M_r}, \\ \dfrac{\sin\left(b\sqrt{M_r^2x^2 - M_{sp}^2}\right)}{\sqrt{M_r^2x^2 - M_{sp}^2}} & \text{otherwise,} \end{cases} \tag{8}$$

and

$$\hat{\varphi}_{KB}(k) = \frac{1}{M_r}I_0\left(m\sqrt{b^2 - (2\pi k/M_r)^2}\right), \tag{9}$$

where $b = \pi\left(2 - \frac{1}{\sigma}\right)$ and $I_0(\cdot)$ is the modified zero-order Bessel function [14]. Finally, the exponential of semicircle pair is defined as:

$$\phi_{ES}(x) = \begin{cases} e^{\beta\sqrt{1-x^2}-1} & |x| \leq 1, \\ 0 & \text{otherwise,} \end{cases} \tag{10}$$

and

$$\hat{\phi}_{ES}(k) = \int_{-\infty}^{\infty}\phi_{ES}(x)e^{ikx}dx, \tag{11}$$

where $\beta = 2.3\omega$. The kernel is re-scaled to have support between $[-\alpha, \alpha]$ with $\alpha = \pi\omega/M_r$. Thus the re-scaled kernel is:

$$\varphi_{ES}(x) = \phi_{ES}(x/\alpha), \tag{12}$$

and thus

$$\hat{\varphi}_{ES}(k) = \alpha\hat{\phi}_{ES}(\alpha k). \tag{13}$$

The exponential of semicircle kernel has no known analytic Fourier transform, therefore numerical integration is used to obtain $\hat{\varphi}_{ES}(k)$. See [11] for more details on the implementation.

We focus our attention on the implementation for the various Kernels which have different periodicity. The Gaussian and exponential of semi-circle are 2π-periodic with domain on $[0, 2\pi]$ [10, 11], while the Kaiser-Bessel kernel is $1-$periodic with domain on $[0, 1]$. Therefore $\xi_\ell,\ x_j \in [0, 2\pi]$ for the Gaussian and exponential of semi-circle kernel and $\xi_\ell,\ x_j \in [0, 1]$ [6] for the Kaiser-Bessel kernel.

Now let p be the periodicity, then its periodisation is

$$\tilde{\varphi}(x) = \sum_{r=-\infty}^{\infty}\varphi(x - rp). \tag{14}$$

Hence the source strength on the over-sampled grid is given by the periodic discrete convolution

$$f_\varphi(\xi_\ell) = \sum_{j=1}^{n-1} f_j\tilde{\varphi}(\xi_\ell - x), \quad \text{for } \ell = 0, ..., M_r - 1. \tag{15}$$

The full derivation to obtain (15) can be found in either [10, 11, 14]. The second step is to now evaluate the DFT of the over-sampled grid using the standard FFT

[6] [14] have domain on $[-\frac{1}{2}, \frac{1}{2}]$, but we change it to $[0, 1]$ for easier implementation. The actual domain is not important provided the periodicity is correct, this is because all that matters is the distances between x_j and ξ_ℓ.

$$F_\varphi(k) = \sum_{l=0}^{M_r-1} f_\varphi(\xi_\ell) e^{-2\pi i k \ell / M_r}, \quad \text{for } k = -M_r/2, ..., M_r/2 - 1. \tag{16}$$

The final step, as a consequence of the convolution theorem is to correct the effects of the convolution and retain the M central frequencies [11]

$$F(k) = F_\varphi(k)/\hat{\varphi}(k), \quad \text{for } k = -M/2, ..., M/2. \tag{17}$$

At first glance, equation (15) seems a lot more expensive than it actually is. This is based on two observations: firstly, the kernels in equation (5), (8) and (12) are sharply peaked in a manner such that the contribution of f_j to grid points outside the kernel width is zero. Secondly, the evaluation of $\tilde{\varphi}(\cdot)$ is unnecessary, we only need to evaluate $\varphi(\cdot)$ (See Figure 2). This is because the purpose of $\tilde{\varphi}(\cdot)$ is to account for the periodicity when spreading near the end points of the over-sampled grid. Now using these observations, we can efficiently implement equation (15) by looping through the source points, finding the nearest up-sampled grid point ξ_{ℓ^*} that is less than or equal to x_j. Spread to the $s \in \{-M_{sp}, .., M_{sp}\}$ nearest grid points ξ_{ℓ^*-s} with $f_j\varphi(x_j - \xi_{\ell^*} - \frac{sp}{M_r})$, subject to the condition that when $\ell^* - s < 0$, the index becomes $\ell^* - s + M_r$ and when $\ell^* - s \geq M_r$, the index becomes $\ell^* - s - M_r$ to account for the correct indices due to the periodicity.

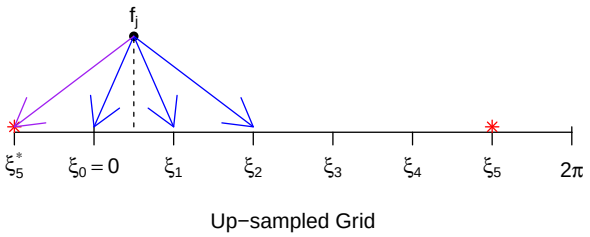

Figure 2: The figure is a toy example to show how the spreading works for a single source point x_j. The up-sampled grid is denoted by ξ_ℓ, and the source strength is spread to the nearest grid points as $f_j\varphi(\xi_\ell - x_j)$. The figure aims to show that we only need to evaluate $\varphi(\xi_\ell - x_j)$ instead of $\tilde{\varphi}(\xi_\ell - x_j)$. The grid points ξ_5^* and ξ_5 (denoted by a red star) is the same point due to the periodicity, but the distance between $\xi_5 - x_j$ is large resulting in $\varphi(\xi_5 - x_j) \approx 0$, but $\tilde{\varphi}(\xi_5 - x_j)$ fixes this by accounting for the periodicity. Therefore, we can reduce the unnecessary computation of Equation (14) by contributing f_j to $f_\varphi(\xi_5)$ with $f_j\varphi(\xi_5^* - x_j)$.

Finally, the amount of spreading in each direction, M_{sp} (in terms of number of grid points) comes from the choice of tolerance, ϵ. We measure tolerance as the relative ℓ^2-norm in the output vector defined as:

$$\epsilon = \frac{\|\boldsymbol{F}_\varphi - \boldsymbol{F}\|_2}{\|\boldsymbol{F}\|_2}. \tag{18}$$

We found that setting $M_{sp} = \lfloor \frac{-\ln(\epsilon)(\sigma - 1/2)}{(\pi(\sigma-1))} + \frac{1}{2} \rfloor$ for the Gaussian kernel, $M_{sp} = \lfloor \frac{1}{2}(\lceil \log_{10}(\frac{1}{\epsilon}) \rceil + 2) \rfloor$ for the Kaiser-Bessel kernel and $M_{sp} = \lfloor \frac{1}{2}(\lceil \log_{10}(\frac{1}{\epsilon}) \rceil + 2) \rfloor + 2$ for the exponential of semi-circle kernel allow us to achieve the desired relative error level [7]. The method can be used for all synchronous and asynchronous cases and has a complexity of $O(M_r \log M_r + n|\log(\epsilon)|)$ [11].

2.3. Insights from NUFFTs

The use of non-uniform FFT methods presents not only a speed advantage, but provides insights in: i.) the averaging scale (N) in the Malliavin-Mancino estimator, and ii.) interpolation of financial data.

Firstly, the Malliavin-Mancino estimator aims to represent the Fourier coefficients of the volatility process as a function of the Fourier coefficients of the price process. Therefore investigation into different time scales of the volatility process is limited to the sampling rate of the price process. The highest sampling rate present in the data is N_0, therefore the Nyquist frequency is $0.5N_0 = N$ - the highest component frequency we can investigate without introducing aliasing. Meaning we are band-limited to frequencies $\leq N$. Hence we can only investigate time scales within the Nyquist frequency *i.e.* the highest component frequency we can investigate is $2\Delta t$, where Δt is the smallest distance between two consecutive prices [3].

To reconstruct the volatility process at the Nyquist frequency, we require at least $2N$ samples. This condition is satisfied by construction of the *Bohr* convolution product, with Fourier modes ranging from $\{-N, ..., N\}$ - resulting in a sampling frequency $M = 2N + 1$ samples.

The relation between the number of Fourier modes and the sampling interval is simply $\frac{T}{M} = \Delta t$. Therefore we can investigate different time scales by investigating different frequency ranges. This is due to a consequence of the sampling theorem - which in essence states that in order to perfectly reconstruct a certain frequency, one needs at least twice the amount of samples. Meaning, by reducing the number of Fourier modes (investigating larger time scales), we are reducing the number of samples, thereby aliasing the larger frequencies. With this in mind, we are able to investigate different time scales by perfectly reconstructing frequencies $< 0.5M$, through the cost of aliasing frequencies $\geq 0.5M$ - a result used by [12, 13].

The insight of NUFFT methods, is that the relation between N and the time scale is demonstrated more intuitively (See Figure 3). For fixed M_{sp}, when N is small, M_r is also small. Therefore the M_r grid points will be more spread out and each grid point will have contributions from multiple source strengths averaged based on the choice of kernel $\varphi(\cdot)$. While for the case when N is large, M_r will also be large. Meaning the M_r grid points are more tightly packed and fewer grid points will have contributions from separate source strengths - in essence there is less averaging.

Secondly, the interpolation is explicit in NUFFT methods. This is interesting because interpolation of financial data can result in estimates being biased - such as linear interpolation

[7] We tuned the M_{sp} such that it always strictly achieves the desired error level. Therefore we note that our choice of M_{sp} is stricter than that in the literature. Specifically, [11] set $\omega = \lceil \log_{10}(\frac{1}{\epsilon}) \rceil + 1$. NUFFT error is the test script to check that the desired relative ℓ^2 error is strictly achieved and can be found in the GitHub resource [20].

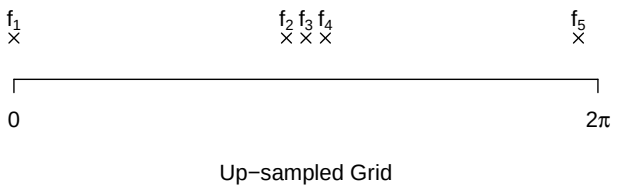

Figure 3: The figure is a toy example to show how the choice of N has an impact on averaging and the time scale. For fixed $M_{sp} = 6$, when N is small (*e.g.* $N = 5$), M_r is also small ($M_r = 22$). Meaning the M_r grid points will be more spaced out and each M_r grid point will have contributions from most of the source strengths f_j - hence investigating smaller time scales by averaging. On the other hand, when N is large (*e.g.* $N = 40$), M_r is also large ($M_r = 162$). Meaning the M_r grid points will be more closely spaced and only some of the M_r grid points will have contributions from separate source points and majority of the M_r grid points will have no contributions. This illustrates the intuitive idea of how changing N allows one to investigate different time scales using the ideas from NUFFTs.

[18] or interpolation based on underlying continuity assumptions such as the Hayashi-Yoshida estimator [4]. We argue these methods are flawed because they do not account for the effect of interpolation - whereas NUFFT methods account for this by deconvolving the interpolation effects in the Fourier space.

3. Algorithm Performance and Benchmarking

The benchmarking is done using Monte Carlo simulations [8]. We compare the relative performance of the algorithms and investigate the various factors influencing speed and accuracy. We use the Geometric Brownian Motion (GBM) which satisfies the following SDEs:

$$\frac{dS_i(t)}{S_i(t)} = \mu_i dt + \sigma_i dW_i(t), \quad i = 1, ..., D, \tag{19}$$

with $\text{Corr}(dW_i, dW_j) = \rho_{ij}$. The GBM is simulated using the first order Euler discretisation (See Algorithm 3) with equal spacing Δt between the observations, which are at the same time across the features - this is known as the synchronous case. Asynchrony is then induced from the synchronous case using two approaches: 1.) the missing data representation, and 2.) the arrival time representation. The missing data representation is achieved by randomly sampling and removing a certain percentage of observations. The arrival time representation is achieved by sampling the synchronous price path using an exponential inter-arrival time with rate λ.

3.1. Benchmark Timing

The common factors affecting the computation time for all the algorithms are: i.) the number of data points, n, ii.) the number of Fourier coefficients, $M = 2N+1$, and iii.) the number of features, D. The parameter specific to the non-uniform FFT method is the tolerance ϵ which determines the spreading width ω.

[8] All the seeds for replication of the work are provided in the respective script files from our GitHub resource [20]

The benchmarking is done using a 2.5GHz Quad-Core Intel i7 with 16GB of RAM on MacOS version 10.15.1 with JuliaPro version 1.2.0. GCC8 is used as a requirement for the Julia interface to FINUFFT provided by [21].

We begin by investigating the common factors which affect computation time for the various algorithms. To this end, we investigate the computation time as a function of the number of data points for a synchronous GBM. The synchronous GBM is used because the Nyquist frequency is $N = \frac{n}{2}$ for the synchronous case. Therefore the number of Fourier coefficients scales linearly with the number of data points.

Figure 4, we compare the following algorithms: the for-loop implementation (MRS 2.2.1 - blue dashes), the vectorised implementation (CFT 2.2.2 - orange dashes), the FFT implementation (2.2.3 green dots), the zero-padded FFT implementation (2.2.4 purple dash-dot) and the fast Gaussian gridding implementation of the NUFFT (FGG 2.2.5 - dark green dash-dot-dot) using the default $\epsilon = 10^{-12}$. The $O(n)$ plots are plotted with compute time [9] on the log scale for better comparison and include the Dirichlet and Fejér representation for $D = 2, 10$ and 100 features respectively [10]. Taking a closer look at Figure 4 we notice several things.

Firstly, the for-loop and vectorised implementation take on the same general shape, but with the vectorised implementation being faster due to the exploitation of the Hermitian symmetry. However, in Figure 4 (a) and (b) we see that because vectorisation requires more memory - this can become a limiting factor. This is because a complex matrix of size $(n \times N)$ has each element requiring 16 bytes for the Complex 64-bit floating point number. For example, with simply 5×10^4 data points, we require 20GB of memory, therefore demanding the use of virtual memory resulting in a deterioration of performance.

Secondly, the difference in speed between the naive methods compared to the fast Fourier transform methods is significant. Looking at Table 1, for 2 features with $n = 10^5$ data points, the for-loop implementation takes 1176 seconds while the fast Gaussian gridding takes 0.119 seconds - 10,000 times faster than the niave for-loop compute time. Thirdly, between the

Method	MRS	KB	ES	FGG	FINUFFT
Time (s)	1176s	2.161s	0.190s	0.119s	0.0331s

Table 1: The table shows the compute time measured in seconds for various algorithms using 2 features with 10^5 data points. The methods considered are: the "for-loop" implementation (MRS), the fast Gaussian gridding (FGG), Kaiser-Bessel (KB), exponential of semi-circle using our naive implementation (ES) and the implementation from FINUFFT (FINUFFT). The NUFFT methods are computed using the default $\epsilon = 10^{-12}$. The times are extracted from Figures 4 and 5.

fast Fourier methods, from fastest to slowest, we have: FFT, zero-padded FFT, and FGG. This is because FFT computes

[9] The compute time is the minimum estimate over 10 replications.

[10] The induced correlation matrix for $D = 10$ and 100 are created using a uniform random matrix and re-scaled appropriately. The function can be found in gencovmatrix provided in our GitHub resources [20]. We note that because a uniform random matrix was used, it only produced positive correlations - which is not an issue for the purposes of timing.

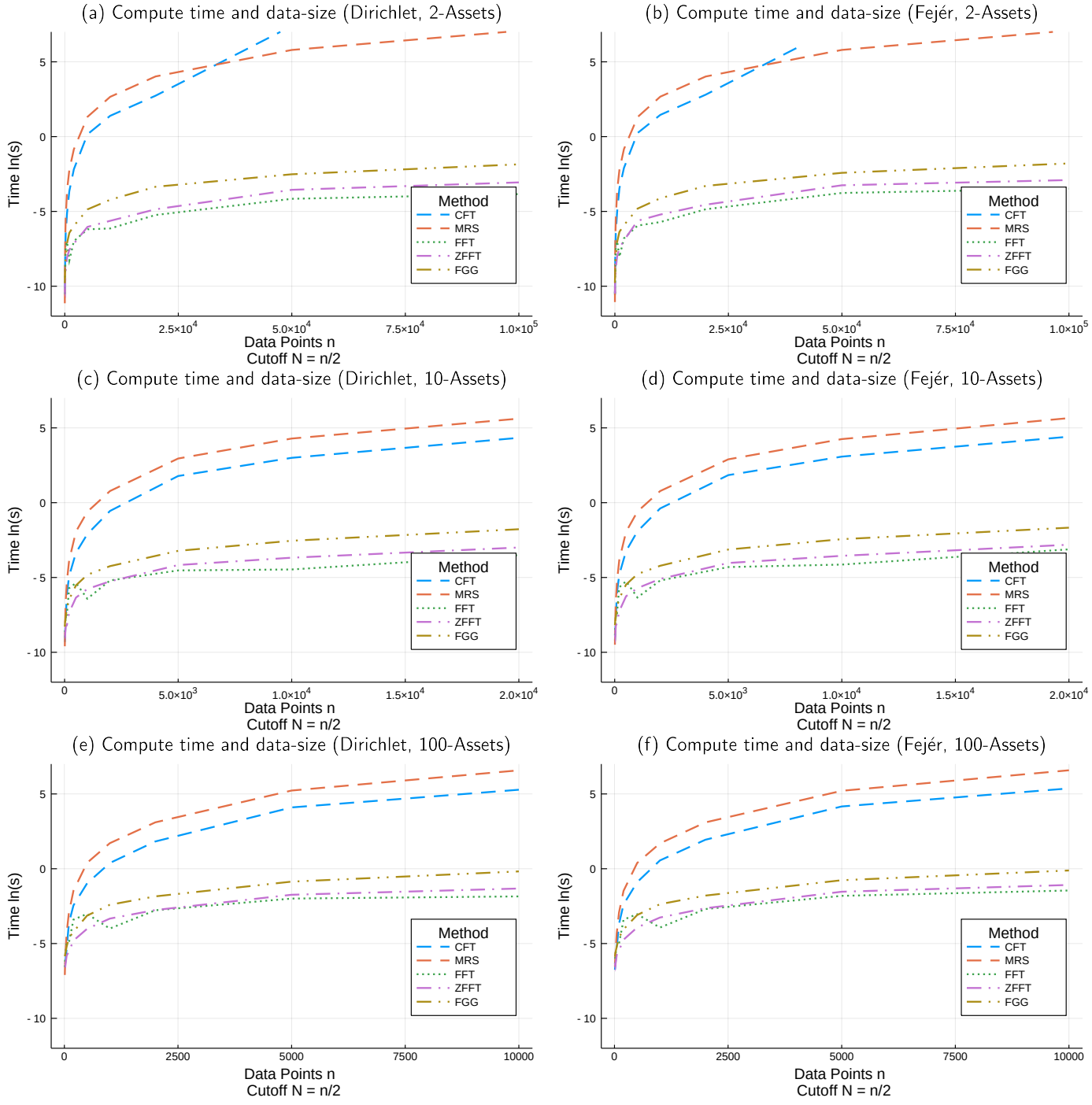

Figure 4: We investigate the algorithm complexity between traditional implementation methods against fast Fourier methods. We plot the logarithm of compute time (measured in seconds) as a function of the number of data points n for various implementation methods. The n is the number of price observations simulated using a Geometric Brownian Motion with Algorithm 3. We investigate the synchronous case since the Nyquist cutoff scales linearly with the number of data points when the data points are uniform in time. Furthermore, we obtain run times for the Dirichlet and Fejér kernel basis for $D = 2, 10$ and 100 features to investigate the impact breadth has on the run time. The traditional methods investigate are the vectorised implementation (CFT - blue dashes) and the "for-loop" implementation (MRS - orange dashes). The fast Fourier methods investigated are the FFT (FFT - green dots), zero-padded FFT (ZFFT - purple dash-dots) and the NUFFT implementation using the fast Gaussian gridding (FGG - dark green dash-dot-dots) with the default $\epsilon = 10^{-12}$. The figures demonstrate the efficacy of fast Fourier methods in reducing compute time for both basis kernels of the Malliavin-Mancino estimator. The figures can be recovered using the Julia script files Dirichlet Timing and Fejer Timing on the GitHub resource [20].

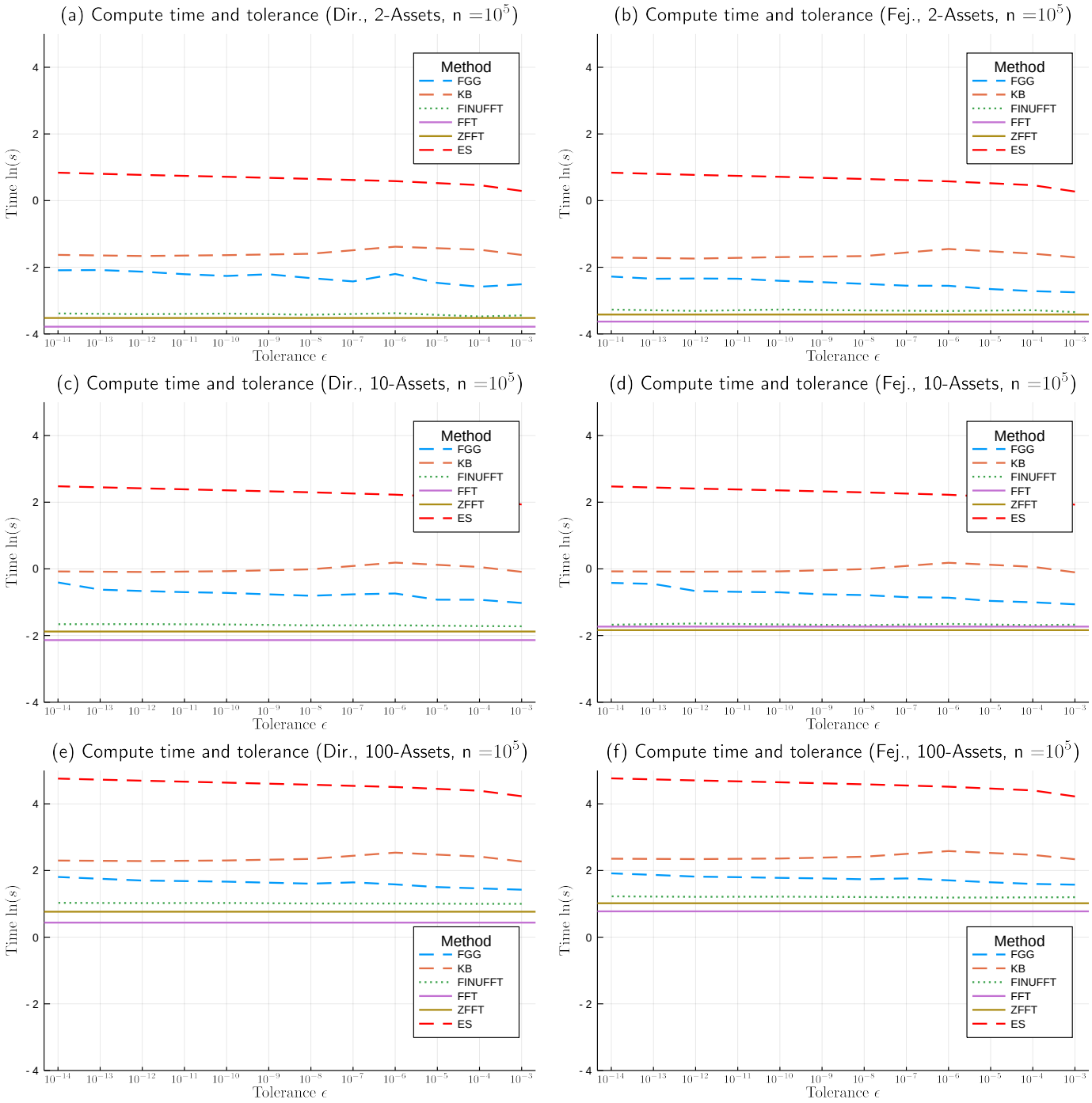

Figure 5: We investigate the algorithm complexity for various non-uniform fast Fourier transform methods by plotting the time (measured in seconds) as a function of the error tolerance ϵ which determines the spreading width ω for the various averaging kernels. The FFT (FFT - purple line) and zero-padded FFT (ZFFT - dark green line) is plotted as a baseline for comparison. We simulate a synchronous Geometric Brownian Motion using Algorithm 3 with $n = 10^5$ data points for $D = 2, 10$ and 100 respectively. The run times are obtained for the two basis kernels: Dirichlet (Dir.) and Fejér (Fej.). The non-uniform FFT methods considered are the Gaussian kernel (FGG - blue dashes), the Kaiser-Bessel kernel (KB - orange dashes) and the exponential of semi-circle kernel with our naive implementation (ES - red dashes) and the implementation by FINUFFT (FINUFFT - green dots). The results are consistent with the results from [11], where the FINUFFT implementation is faster than the fast Gaussian gridding which is faster than the Kaiser-Bessel. The evaluations are all done "on-the-fly" without any pre-computation. The figures can be recovered using the Julia script file Error Timing on the GitHub resource [20].

N Fourier modes, the zero-padded FFT computes M Fourier modes and the FGG computes M_r Fourier modes. On top of that, the FFT requires no steps before performing the FFT, whereas the zero-padded FFT needs to zero-pad missing data and the FGG requires the convolution and deconvolution step.

Finally, D, the breadth of features can impact computation time depending on the choice of N. The case when N is the same across all features is simple. We then only need to compute the M Fourier coefficients for D features - this is presented in Figure 4. When N is the same across all features, we are presented with two advantages: i.) the time scale investigated will be the same for all the features, and ii.) if one uses the Fejér basis kernel, we can guarantee positive semi-definiteness in the covariance matrix [3, 22].

The case when N changes for the different features becomes more nuanced. For example, in the arrival time representation for our simulated event data different features have different Nyquist frequencies. Here we need to compute $\frac{D(D-1)}{2}$ pairwise estimates for each $\hat{\Sigma}^{ij}_{n,N}$ entry. A potential problem arises when N is independently obtained to investigate the co-movement between events for each feature pair [4], or when the Dirichlet basis kernel is used, because we are not guaranteed a positive semi-definite matrix. This can present challenges, for example, when an invertible covariance matrix is a necessary requirement; as in the case of portfolio optimisation. This can be ameliorated by transforming the non-positive semi-definite covariance matrix estimate to the closest positive semi-definite matrix, under some appropriate norm [23] or using extensions of these type of transformations [24]. However, doing so comes with an additional computational cost.

We then investigate the impact of the tolerance, ϵ; and how this affects the computation time for the various NUFFT methods. This is explored by plotting the computation time as a function of ϵ. We use the synchronous GBM with $n = 10^5$ data points.

Figure 5, we compare the following algorithms: the FFT implementation (2.2.3 purple line) and the zero-padded FFT implementation (2.2.4 dark green line) as a baseline for comparison. The NUFFT methods include the fast Gaussian gridding with the Gaussian kernel ([10] blue dashes), the Kaiser-Bessel kernel ([14] orange dashes) and the exponential of semi-circle using our naive implementation and the FINUFFT package ([11] red dashes and green dots respectively). The $O(n)$ plots are plotted on the log scale as the minimum compute time estimate over 10 replications. The figures include the Dirichlet and Fejér representation for $D = 2, 10$ and 100 features respectively with $n = 10^5$.

Taking a closer look at the inner workings for each algorithm, we see that the zero-padded FFT implementation needs to assign the n data points to the over-sampled grid and the NUFFT methods need to assign ωn points to the over-sampled grid. Furthermore, the zero-padded FFT requires no evaluations whereas the NUFFT methods require ωn evaluations of $f_j\varphi(\cdot)$.

The key differences between the NUFFT algorithms is in how they reduce the number of evaluations required. The technique used in the fast Gaussian gridding is to reduce the number of exponential evaluations for $\varphi_G(\cdot)$. This is achieved by separating the exponential into three components:

$$e^{-(x_j-2\pi m/M_r)^2/4\tau} = \left[e^{-x_j^2/4\tau}\right]\left[e^{x_j\pi/M_r\tau}\right]^m\left[e^{-(\pi m/M_r)^2/\tau}\right]. \quad (20)$$

By spitting the exponential this way, instead of ω exponential evaluations for each source point, we only need two exponential evaluations per source point. Reducing the number of exponential evaluations from ωn to $\omega + 2n$.

The advantage with using the Kaiser-Bessel kernel is that it is both smooth and has narrow support [11]. This can be exploited to cut the number of kernel evaluations by reducing ω to achieve a comparable level of accuracy, *e.g.* for roughly 12 digit accuracy - the Gaussian kernel requires $\omega = 24$, while the Kaiser-Bessel kernel requires $\omega = 13$ for the same tolerance level [11].

Finally, the exponential of semi-circle has narrow support similar to that of the Kaiser-Bessel kernel, but is simpler and faster to evaluate. The downfall is that there is no known analytic Fourier transform, thus incurring additional cost of numerical integration to evaluate (13).

Our implementation of the exponential of semi-circle is naive compared to [11]. We do not exploit the piecewise polynomial kernel approximation to accelerate the evaluation of (12). Furthermore, we use naive numerical integration to compute (13) using the adaptive Gauss-Kronrod quadrature QuadGK rather than the Gauss-Legendre quadrature with "phase winding" [11].

The naive implementation of the exponential of semi-circle serves two purposes: i.) allowing the like-for-like comparison between the various kernels and their algorithms based on our implementation, and, ii.) illustrating the importance of the implementation techniques used by [11]. This is seen more clearly in Figure 5, without the implementation techniques, the exponential of semi-circle is significantly slower than the Gaussian or Kaiser-Bessel kernel. This is due to the numerical integration required for (13). However, with their implementation techniques in place, [11] are able to reduce the compute time of the exponential of semi-circle to a similar time as our zero-padded FFT.

The results in Figure 5 are consistent with that of [11]. Between the non-uniform FFT methods considered: the FINUFFT implementation of the exponential of semi-circle is the fastest, followed by the fast Gaussian gridding, the Kaiser-Bessel kernel evaluated "on-the-fly" [11], and lastly the exponential of semi-circle using the naive implementation.

3.2. Benchmark Accuracy

We have demonstrated the merit of fast Fourier techniques in terms of speed. We now turn towards accuracy to see the use case for the various fast Fourier methods and the conditions where they fail. This is done by testing the fast Fourier methods on the synchronous case, the missing data representation and the arrival time representation. Furthermore, we look

[11] Without any pre-computations, preventing large RAM overhead.

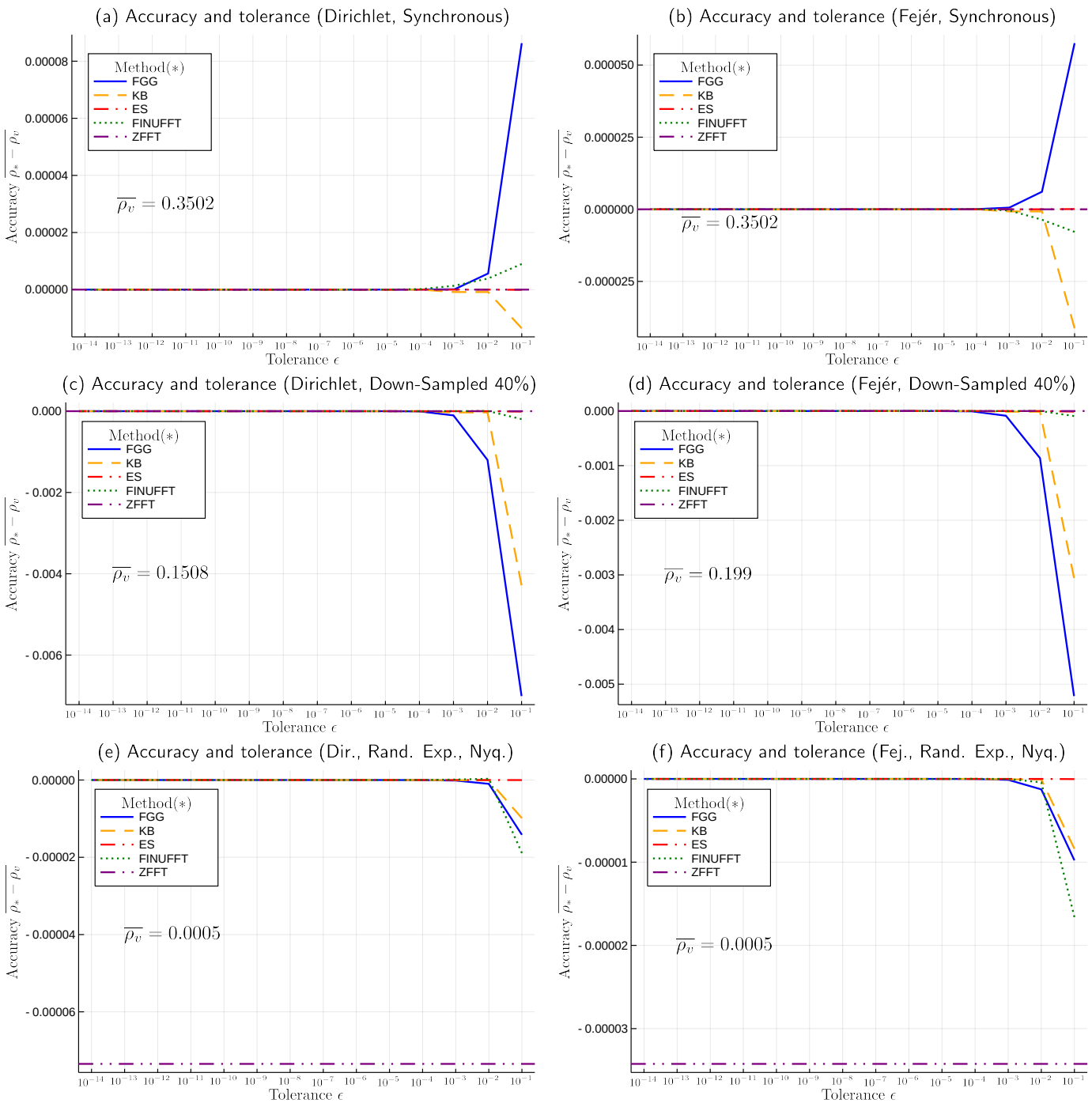

Figure 6: We investigate the accuracy of various fast Fourier methods as a function of the error tolerance ϵ for various simulation settings. Accuracy is measured as the difference between the estimates of the various fast Fourier methods ρ_* and the estimate using the vectorised implementation (CFT) ρ_v averaged over the 100 replications. The average correlation estimate from the vectorised implementation is provided as an inset in each figure. The base-line price process is the synchronous Geometric Brownian Motion with 10^4 data points simulated using Algorithm 3. The three simulation cases are: the synchronous case (a) and (b), the missing data representation (c) and (d), and the arrival time representation (e) and (f). The missing data representation is down-sampled by 40% while the arrival time representation is sampled with an exponential inter-arrival time (Rand. Exp.) with mean 30 and 45 for the first and second price path respectively. The fast Fourier methods investigated are: the fast Gaussian gridding (FGG - blue line), the Kaiser-Bessel kernel (KB - orange dashes), the exponential of semi-circle with our naive implementation (ES - red dash-dots) and the FINUFFT implementation (FINUFFT - green dots) and finally, the zero-padded FFT (ZFFT - purple dash-dot-dot). The accuracy is tested for the two basis kernels: Dirichlet (Dir.) and Fejér (Fej.). Furthermore, the Nyquist frequency (Nyq.) is used for all the correlation estimate, therefore it must be noted that for the arrival time representation, N changes for each replication. We see firstly, provided the tolerance $\epsilon < 10^{-4}$, the NUFFT methods can accurately recover the estimates, secondly, the divergence from the CFT implementation when $\epsilon \geq 10^{-4}$ may simply be an artefact of random errors from the lack of precision requested, and finally, the zero-padded FFT fails for arrival time representation. The figures can be recovered using the Julia script file AccSynDS and AccRE on the GitHub resource [20].

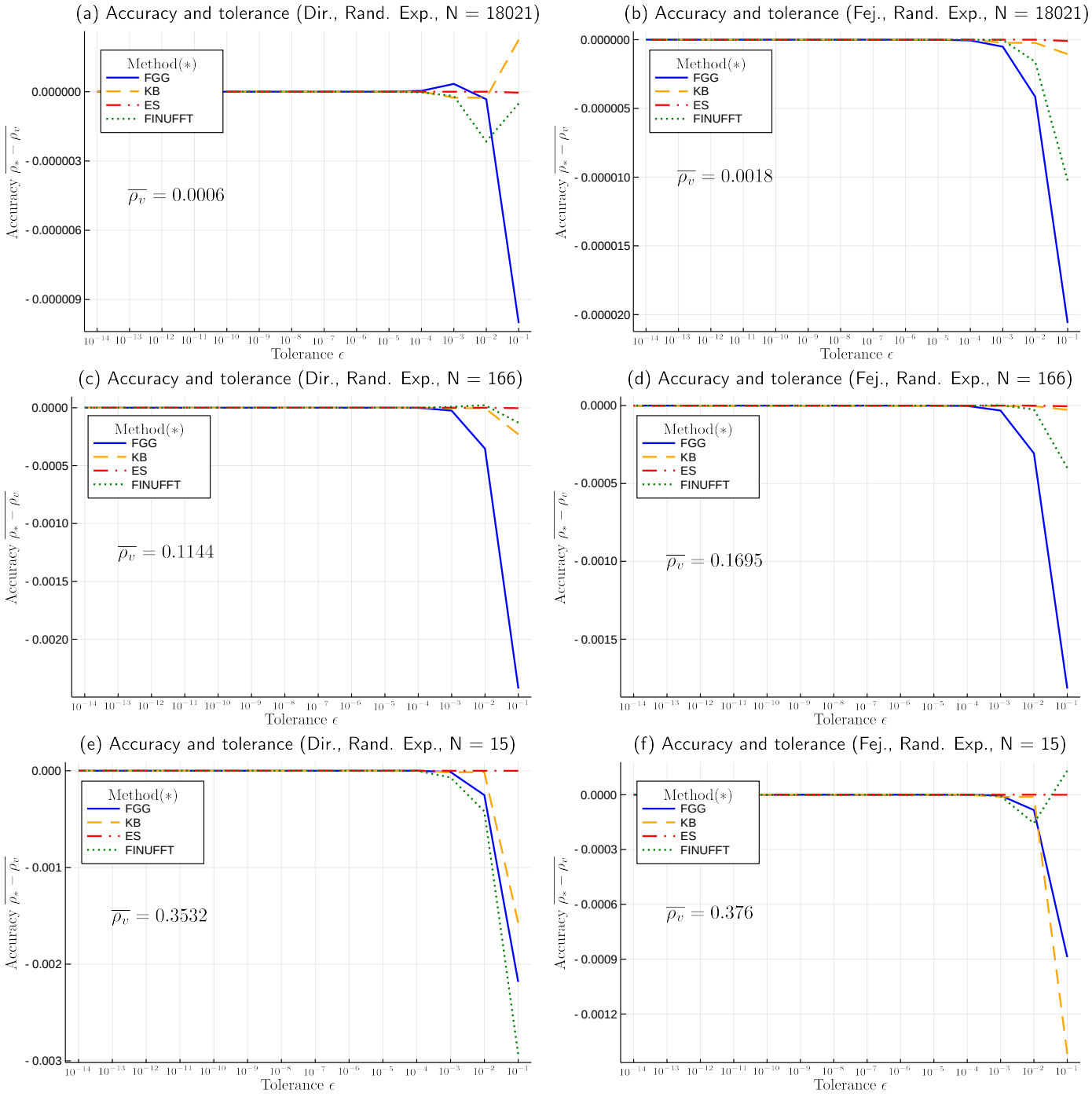

Figure 7: We investigate the inter-play between kernel averaging and time-scale averaging. We plot the accuracy of various fast Fourier methods as a function of the error tolerance ϵ for various choices of N. Accuracy is measured as the difference between the estimates of the various fast Fourier methods ρ_* and the estimate using the vectorised implementation (CFT) ρ_v averaged over the 100 replications. The average correlation estimate from the vectorised implementation is provided as an inset in each figure. The base-line price process is the synchronous Geometric Brownian Motion with 10^4 data points simulated using Algorithm 3. The synchronous price paths are then sampled with an exponential inter-arrival time (Rand. Exp.) with mean 30 and 45 for the first and second price path respectively to create the arrival time representation of asynchrony. The fast Fourier methods investigated are: the fast Gaussian gridding (FGG - blue line), the Kaiser-Bessel kernel (KB - orange dashes), the exponential of semi-circle with our naive implementation (ES - red dash-dots) and the FINUFFT implementation (FINUFFT - green dots). The accuracy is tested for the two basis kernels: Dirichlet (Dir.) and Fejér (Fej.). We see that there is no clear relation between the two types of averaging, rather the divergence for higher tolerance levels seems be an artefact of errors arising from the lack of precision requested. The figures can be recovered using the Julia script file AccRE on the GitHub resource [20].

at the inter-relation between two types of averaging: i.) kernel averaging - the convolution step in the NUFFT algorithms, and ii.) time-scale averaging - the choice of N in the Malliavin-Mancino estimator and the relation it has to the sampling interval investigated. This is tested by investigating various kernel widths ω for various choices of N to see if spurious relations might emerge between the two types of averaging.

The setting for the following two experiments are as follows: a bivariate Geometric Brownian Motion with $n = 10^4$ is simulated using Algorithm 3. The daily parameters for the GBM are: $\mu_1 = 0.01, \mu_2 = 0.01, \sigma_1^2 = 0.1, \sigma_2^2 = 0.2, \rho_{12} = 0.35$. We set $\Delta t = \frac{1}{86400}$, therefore each unit interval can be thought of as a second in Calendar time. From the synchronous case we create the missing data representation by randomly removing 40% of data points from each path; and the arrival time representation is achieved by sampling each price path with an exponential inter-arrival time with mean 30 and 45 for the first and second price paths respectively.

Accuracy here is measured as the difference between the estimates from the various fast Fourier methods and the estimates from the vectorised implementation averaged over 100 replications. We do not measure accuracy in terms of Mean Square Error (MSE), because of the various experiment settings and our choice of the Nyquist frequency. From our previous work [4], we know that for the down-sampled and arrival time representation, there is an increase in MSE due to the downward bias arising from the Nyquist choice [22] - which we attribute towards the Epps effect [25] [12]. This is clearly seen from the average correlation estimate from the vectorised implementation (provided as insets in Figure 6) - as the level of asynchrony increases, the correlation decays [4].

Beginning with Figure 6, we investigate the accuracy for the various fast Fourier methods: the zero-padded FFT (ZFFT - purple dash-dot-dots) and for the non-uniform fast Fourier methods: the Gaussian kernel (FGG - blue line), the Kaiser-Bessel kernel (KB - yellow dashes), the exponential of semi-circle kernel using our naive implementation (ES - red dash-dots) and the FINUFFT implementation (FINUFFT - green dots). This is done for three scenarios - the synchronous case, the missing data representation and the arrival time representation.

Figure 6 demonstrates several things. Firstly, provided the tolerance $\epsilon < 10^{-4}$, all the NUFFT methods can accurately recover the estimates. Furthermore, we see that the ES kernel (red dashes) recover the correct estimates for $\epsilon = 10^{-1}$. This is not a property of the exponential of semi-circle kernel, as the FINUFFT implementation diverges from $\epsilon = 10^{-4}$ (due to their more lenient choice of ω), rather, this is a result of our choice of M_{sp} for the ES implementation - to ensure the requested tolerance is always strictly met. Concretely, this means each source point must be spread in each direction for a minimum of $M_{sp} = 4$ grid points for the Gaussian kernel, $M_{sp} = 3$ grid points for the Kaiser-Bessel kernel and $M_{sp} = 3$ grid points for the exponential of semi-circle to perfectly recover the vectorised estimate [13]. Secondly, when tolerance $\epsilon \geq 10^{-4}$, the non-uniform FFT methods diverge away from the vectorised implementation. There is no clear pattern in the divergence for the various kernels, therefore it seems the errors are a simple artefact arising from the lack of precision requested in $\boldsymbol{F}_\varphi$. Finally, the zero-padded FFT recovers the correct estimate for the synchronous case and missing data representation, but more importantly, it fails for the arrival time representation. Figure 1 demonstrates why the zero-padding fails for the arrival time representation. For the arrival time representation, the source points do not have equal spacing Δt, therefore the events need to be shifted to the nearest synchronous grid point, which leads to the smearing of time points, resulting in the incorrect estimate.

The failure of the zero-padded FFT for the truly asynchronous case is the main motivation as to why non-uniform FFT methods are required. Non-uniform FFT methods overcome this through a convolution and deconvolution step to correct the effects of shifting the points to a uniform grid. However, what is currently unclear is the relationship, if any, between the kernel averaging and the time-scale averaging. This relationship is investigated using the arrival time representation (with the same parameters as before) for various choices of N.

Previously in Figure 6, the arrival time representation had N changing for each replication. In Figure 7, we fix three cases of N for the various replications. The first N is computed as the minimum Nyquist frequency across the 100 replications resulting in $N = 18,021$. The second N is computed based on the smallest *average* sampling interval resulting in $N = \lfloor \frac{10,000}{30\times 2} \rfloor = 166$. Finally, the last N is chosen to be arbitrarily small subject to the condition that the corresponding M_r is larger than ω for $\epsilon = 10^{-14}$ [14]. We pick $N = 15$ for the final case. The zero-padded FFT is excluded because the implementation only computes the case when N is the Nyquist frequency.

Looking at Figure 7, there is no clear relation between the two types of averaging. For any choice of N, we can recover the vectorised estimate provided the tolerance $\epsilon < 10^{-4}$. There is no clear pattern in the divergence for the various kernels, the inaccuracy in the estimates are due to the lack of precision requested in $\boldsymbol{F}_\varphi$ - caused by the lack of spreading to nearby grid points for each source point.

Finally, the average correlation estimate from the vectorised implementation provided as insets in Figure 7 is consistent with the results from [4, 12]. As the sampling frequency increases, the sampling interval decreases - resulting in the Epps effect [12].

To briefly summarise the various benchmarking. We have shown the efficacy of NUFFT methods over the "for-loop" and vectorised implementation in terms of speed while accurately recovering the correct estimates - provided the tolerance requested $\epsilon < 10^{-4}$. Lastly, from a pure algorithmic perspective,

[12] The Epps effect is the decay in correlation arising from smaller sampling intervals.

[13] The M_{sp} requirement is calculated based on $\epsilon = 10^{-4}$ for the Gaussian and Kaiser-Bessel kernel and $\epsilon = 10^{-1}$ for the ES kernel.

[14] This is to ensure the up-sampled grid is larger than the total spreading width.

the fast Gaussian gridding is the fastest algorithm without sophisticated implementation techniques.

4. Correlations and time-scale averaging

4.1. Simulated data

We take a deeper look into the relationship between the time-scale and the Epps effect [25]. Concretely, different time scales can be investigated with the Malliavin-Mancino estimator through the choice of N - specifically $\Delta t = \frac{T}{2N+1}$ (See Section 2.3). This is how Renò [12] and Precup and Iori [13] investigate the Epps effect. Precup and Iori were able to demonstrate the higher the level of asynchrony, the larger the drop in correlation for the Epps effect. This is demonstrated in Figure 6 with the average correlation provided as insets for varying level of asynchrony. Additionally, Renò was able to demonstrate the Epps effect as a function of sampling frequency under the arrival time representation of asynchrony.

Following the work of [12, 13], Tóth and Kertész [26] and Mastromatteo *et al.* [27] were able to analytically derive the Epps effect arising from the arrival time representation as:

$$\tilde{\rho}^{ij}_{\Delta t} = c\left(1 + \frac{1}{\Delta t}\left(e^{-\lambda \Delta t} - 1\right)\right). \tag{21}$$

Here c is the induced correlation and the sampling intensity is λ; the same for the price paths [27].

We compare the relationship between the time-scale averaging in the Malliavin-Mancino estimator used by [12, 13] to the analytic formula characterising the Epps effect arising from Poissonian sampling in Equation (21). This is done by simulating T data points from a bivariate Geometric Brownian Motion with the same parameters as Section 3.2. We consider one hour, one trading day and one trading weeks' worth of simulated data, with a price realisation sampled each second. Thus, assuming that each trading day is 8 hours in Calendar time, we have $T = 3600, 28800$ and 144000 synchronous data points for the various cases. The synchronous price paths are then sampled using an exponential inter-arrival time with the same rate λ to create the arrival time representation of the asynchronous price paths.

Figure 8 we plot Equation (21) as a function of Δt (see the green dashes labelled "MMZ" in Figure 8) ranging from 1 to 100 seconds and contrast this with the estimated correlations. The corresponding N [15] for the Malliavin-Mancino estimator is given by:

$$N = \left\lfloor \frac{1}{2}\left(\frac{T}{\Delta t} - 1\right)\right\rfloor. \tag{22}$$

The Malliavin-Mancino correlation estimates are computed using the fast Gaussian gridding implementation of the non-uniform fast Fourier transform with $\epsilon = 10^{-12}$ for the various choices of Δt. This is done for the Dirichlet basis kernel (see the blue line with label "MM Dirichlet" in Figure 8) and the Fejér basis kernel (see the red line with label "MM Fejér" in Figure 8). Furthermore, this process is repeated 100 times so that the variability between the measured estimates can be investigated for various n and N. Here, $n \approx T/\lambda$ on average, based on the Poissonian sampling.

Figure 8, we plot the averaged correlation estimates over the various replications, with the error bars (obtained using the sample standard deviation and a t-distribution with 99 degrees of freedom) representing 68% of the variability between the estimated paths. We see that firstly, the precision of the estimates are more precise as n and N increase. The exact contributions of n and N leading to the increased precision for larger T is unclear, as larger T implies larger n and N. However, for a fixed T, we see the effect larger N has on the precision (ignoring the variability from n changing based on the replications).

Secondly, we see that the Dirichlet kernel plausibly recovers the theoretical Epps curve (we found several realisations from the 100 replications which correctly recovers the curve), while the Fejér is biased upwards with respect to the Dirichlet basis and the theoretical Epps effect. This is because the Fejér kernel places more weight on the lower frequencies and less weight on the higher frequencies - making it more stable under microstructure noise [22]. Furthermore, due to the weighting of frequencies in the Fejér kernel - we get smoother estimates compared to the Dirichlet kernel (See Figure B.10 in Appendix B for indication of individual realisations).

Now the question as to which kernel basis is more effective is tricky. The answer to this depends on how one views the Epps effect: as a bias that requires correction [22, 28–30] or if it is a fundamental property of market microstructure [4, 26]. Although the Epps effect is well known and the factors causing it are well investigated [12, 13, 26, 27, 29, 31–33]. What is unclear is whether some of the explanations are the result of normative models being forced onto the data, or are in-fact reflecting fundamental properties of the underlying stochastic processes in the presence of order-flow dynamics and market structure [4]. Therefore it is unclear whether the Epps effect is indeed a bias or a fundamental property of market microstructure.

Thus the answer as to which kernel basis is more effective is two fold. If one views the Epps effect as a fundamental property of market microstructure then the Dirichlet kernel is the correct representation because it (plausibly) recovers the theoretical Epps effect. However, if one views the Epps effect as a bias which requires correction, then the Fejér kernel is more effective as more weight is placed on lower order frequencies to avoid market microstructure noise. Furthermore, the Fejér kernel can be coupled with the method in [22] to pick N such that MSE is minimised.

The implication of picking N to minimise MSE is that control over the time-scale investigated is relinquished. From Figure 8, it is clear that in order for MSE to be minimised, N must be small, meaning that larger time intervals are investigated. This is precarious because decision making in finance takes place on a particular time-scale. Therefore, without control over N, decisions in finance which are time-scale dependent such as state detection may be adversely affected.

[15] We note that (22) may not always be a perfect conversion due to the range of Fourier modes in (1).

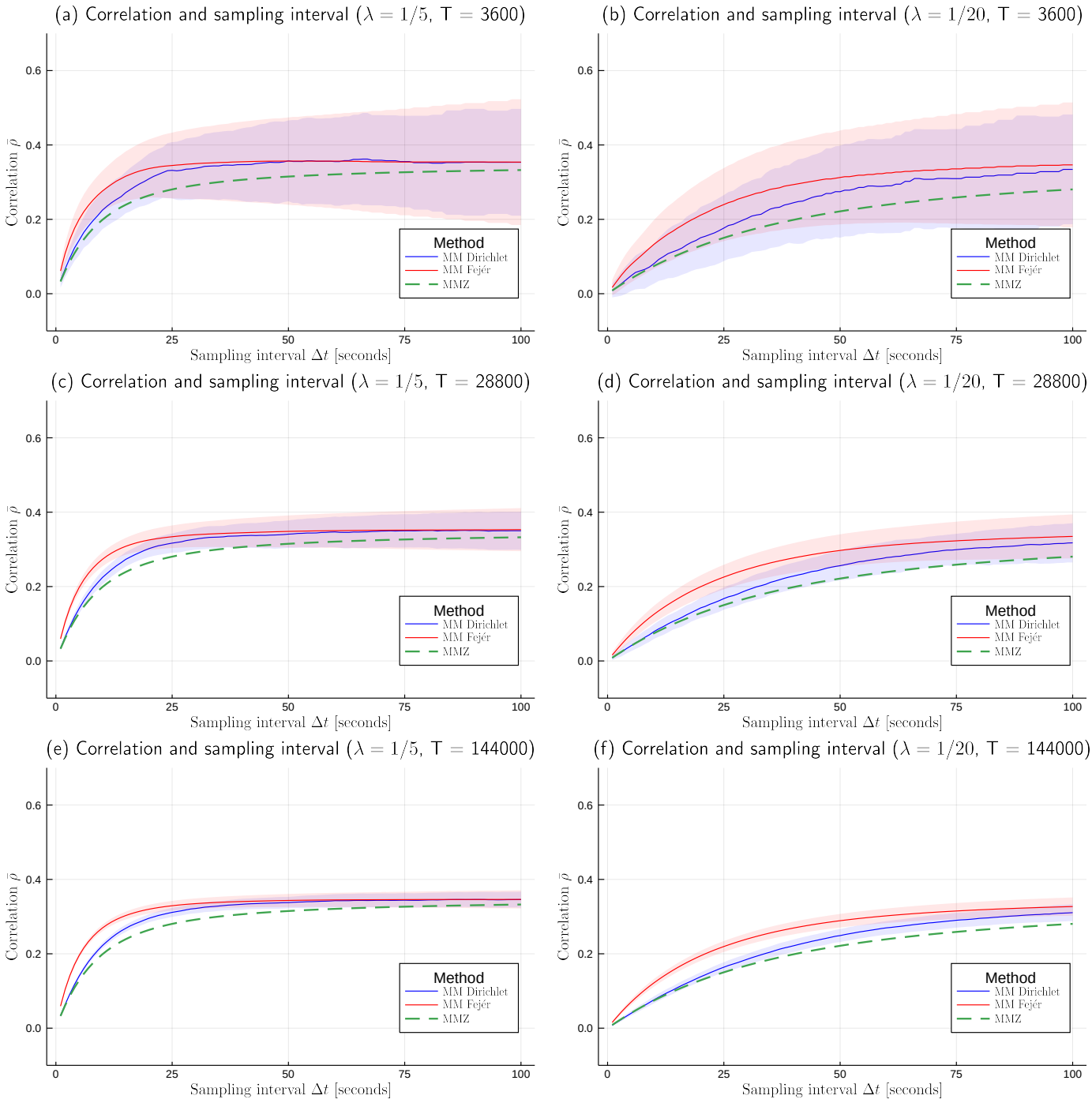

Figure 8: We investigate the link between the time-scale averaging in the Malliavin-Mancino estimator determined by the number of Fourier coefficients and the analytic formula characterising the Epps effect arising from Poissonian sampling using Equation (21). The conversion from Δt to N in the estimators is given by Equation (22). The arrival time representation samples T synchronous data points (each points representing a second in a day) with an inter-arrival time with rates $\lambda = \frac{1}{5}$ and $\lambda = \frac{1}{20}$ for (a), (c), (e) and (b), (d), (f) respectively. The induced correlation across the replications is 0.35. Here the green dashes ("MMZ") is the plot of equation (21). The Malliavin-Mancino estimates for each Δt are obtained using the Dirichlet and Fejér basis kernels; these are respectively the blue line ("MM Dirichlet") and red line ("MM Fejér"). Here we repeat this process 100 times to obtain 100 arrival time representation paths, based on 100 different GBM paths to obtain 100 estimates for each Δt. The average correlation estimate at each Δt is then plotted with error bars (computed using a t-distribution with 99 degrees of freedom and the sample standard deviation) representing 68% of the variability between the estimated paths. The Dirichlet basis kernel (plausibly) recovers the Epps effect given by equation (21), while the Fejér basis kernel is biased upwards with respect to the Dirichlet basis kernel (and the theoretical Epps effect) because there is induced averaging. For the same reason the estimate curves using the Dirichlet basis kernel are more volatile than those estimated using Fejér kernel basis (See Figure B.10). The figures can be recovered using the Julia script file MMZandMM on the GitHub resource [20].

4.2. Real-world data

The correlations at various high-frequency time scales for 10 equity assets listed on the Johannesburg Stock Exchange (JSE) are given as a real-world example. To estimate correlation we used Trade and Quote (TAQ) data for the 10 equities extracted from Bloomberg Pro and processed to remove repeated time stamps by aggregating trades with the same time stamp using a volume weighted average. The processed TAQ data can be found in [34]. The 10 equities considered are: FirstRand Limited (FSR), Shoprite Holdings Ltd (SHP), Absa Group Ltd (ABG), Nedbank Group Ltd (NED), Standard Bank Group Ltd (SBK), Sasol Ltd (SOL), Mondi Plc (MNP), Anglo American Plc (AGL), Naspers Ltd (NPN) and British American Tobacco Plc (BTI). The period considered is the week from 24/06/2019 to 28/06/2019. Given that the data is for a 5 day period, with equities trading 8 hours a day. This yields $T = 5 \times 28,800 = 144,000$ seconds in the period of consideration.

Tickers	Vol. Traded	Unique Trades	$1/\hat{\lambda}$ [sec]
BTI	3143263	7893	17.83 ± 0.64
NPN	2791054	12378	11.38 ± 0.32
AGL	5751811	9091	15.49 ± 0.50
MNP	1701907	6562	21.43 ± 0.93
SOL	6048773	10343	13.62 ± 0.43
SBK	9427755	7441	18.93 ± 0.65
NED	4518354	7090	19.85 ± 0.69
ABG	6607644	6572	21.36 ± 0.78
SHP	3758655	5549	25.35 ± 1.01
FSR	38493240	10412	13.53 ± 0.39

Table 2: Table 2 provides a summary of the 10 equities considered for the week from 24/06/2019 to 28/06/2019. The table indicates the volume traded, the number of unique trades and the mean inter-arrival time between trades measured in seconds with the 95% confidence interval provided.

Table 2, provides the volume traded, the number of unique trades and the mean inter-arrival time between the trades (measured in seconds with a 95% confidence interval provided computed using a t-distribution and the standard errors) for the 10 equities used in the analysis. It is important to notice is that the measured intensities $\hat{\lambda}$'s [16] are not the same across the assets. Therefore, in order to use (21), we make the simplifying assumption that the $\hat{\lambda}$'s are approximately the same and take on the larger intensity of the two, *i.e.* $\hat{\lambda} = \max(\hat{\lambda}_i, \hat{\lambda}_j)$. But, we highlight that an extension of (21) to model different intensities and lead-lags is provided by Mastromatteo *et al.* [27]. They considered multiple intensities $\lambda_i \neq \lambda_j$ in order to decouple effects from asynchronous sampling and the effects from lead-lag. We are interested in the general concave shape of the theoretical model and do not use the extended model for the theoretical Epps effect (See [26, 27] and the plots therein) but point out that under estimation not all of the measured Epps curves can conform to the theoretical models.

Before comparing the theoretical Epps effect against the measured Epps effect, we perform some Exploratory Data Analysis to identify the interesting correlation pairs out of the 45 available pairs. We first plotted the 45 correlation pairs as a function of the sampling interval Δt for the Dirichlet and Fejér basis kernel in Figure B.10. The conversion for Δt to N is given by (22), assuming $T = 144,000$. The correlation estimates are estimated using the fast Gaussian gridding implementation of the non-uniform fast Fourier transform with $\epsilon = 10^{-12}$. We obtain correlation matrices for 100 Δt's ranging from 1 to 100. The compute time for 100 different N's took a total of 7.28 seconds using the Dirichlet basis and 9.10 seconds using the Fejér basis - demonstrating the efficacy of our fast Fourier method. Two initial observations are made from Figure B.10: firstly, the Fejér kernel produces smoother estimates compared to the Dirichlet kernel. Secondly, nearly all the correlation pairs exhibit the Epps effect, where the correlation rises as Δt increases, and as such generally conform to the theoretical models in the literature [26, 27, 33]. However, there are exceptions where correlation pairs do not exhibit the behaviours easily accounted for by the prevailing models. Rather, it seems that the correlation drops as Δt increases, to the point where the sign of the correlation switches.

To get a better understanding of the correlation structure in Figure B.10, snapshots of the correlation structures are plotted as heat-maps for $\Delta t = 1, 30, 60$ and 100 seconds for the Dirichlet and Fejér basis [17] in Figure B.11. The heat-maps provide clearer insight as to which asset pairs exhibit a strong decay in the correlation - seen in the top right quadrant of the heat-maps - which are the equities from the banking sector. More importantly, the correlation pair FSR/AGL is interesting. When $\Delta t = 1$, the pair is positively correlated, but when $\Delta t = 100$, the pair becomes negatively correlated - a result which does not fit in with the theory or phenomenology of the Epps effect.

Figure 9 investigates this in more detail by plotting the correlation as a function of Δt for two asset pairs. Furthermore, indicative error bars are obtained through block bootstrap. This is achieved by splitting the data into 100 Calendar time blocks and estimating the Δt's with 1 block removed each time. Here we remove the block by assuming it is missing data, therefore T remains the same across the various replications. The errors bars are 95% of the variability between the estimates for each Δt - obtained from the block bootstrap and computed using a t-distribution with 99 degrees of freedom and the sample standard deviation. These errors are then overlaid on the mean estimates from the block bootstrap. Here the line with errors bars are from the block bootstrap, while the dashes are the measured correlation from the full path. The first pair FSR/SBK (blue line/dashes) is a clear demonstration of the Epps effect, while the second pair FSR/AGL (red line/dashes) does not behave in accordance to the Epps effect.

The most notable models that decompose the Epps effect in the current literature are from Tóth and Kertész [26, 33], where they decompose the correlation at longer time scales Δt as a function of auto- and cross- correlations on shorter time scale Δt_0; and Mastromatteo *et al.* [27], where they decouple the

[16] The $\hat{\lambda}$'s are indicative, estimated from the TAQ data.

[17] Both the Dirichlet and Fejér kernel produced positive semi-definite covariance matrices.

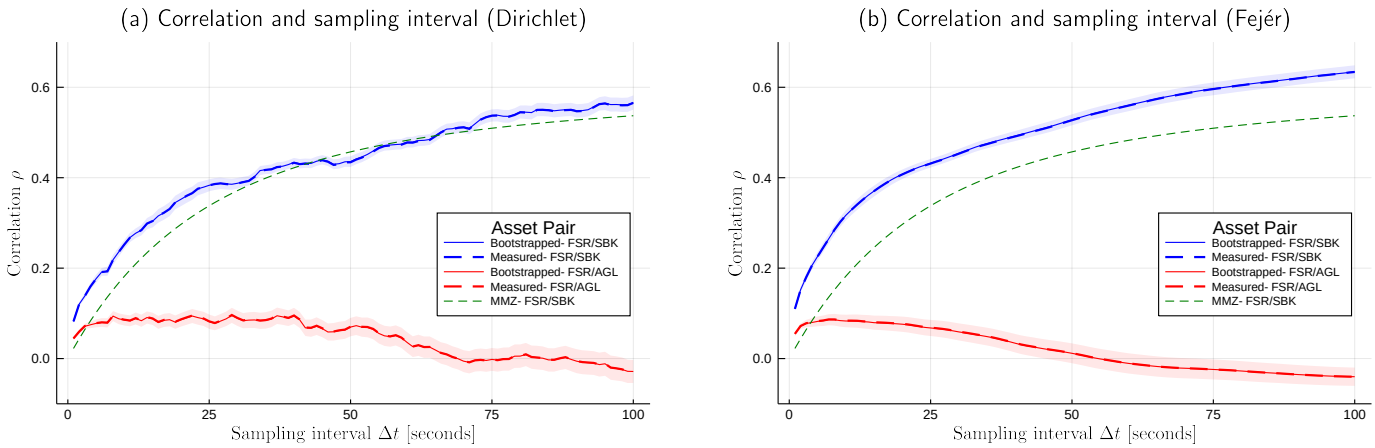

Figure 9: We investigate the two most interesting correlation pairs from the 10 available equity data by plotting the correlation as a function of the sampling interval Δt measured in seconds (the complete set is in Figures B.10 (a) and (b) of Appendix B). The conversion between Δt and N is given by (22) assuming $T = 144,000$. The correlation pairs considered are FSR/SBK (blue) and FSR/AGL (red). Here the lines with error bars are the mean estimates and 95% variability between the paths obtained from block bootstrap, while the dashes are the measured estimates from the complete time series. Furthermore, we plot a simple theoretical Epps effect arising from asynchrony for the FSR/SBK (green dashes) pair (See Equation (21)). This is done by assuming the inter-arrival time of trades follow an exponential distribution with the same rate $\hat{\lambda} = 1/13.5$. Furthermore, we found that $c = 0.621$ provided a relatively good fit. The theoretical Epps is not plotted for the FSR/AGL pair because the correlation dynamics do not exhibit the Epps effect. The empirical reality of curves, such as the upper blue curve for the FSR/SBK pair, suggest that current theoretical Epps effect models can plausibly model the correlation dynamics for some asset pairs; while curves such as the lower red curve for the FSR/AGL pair, suggest that the current theoretical explanations for the Epps effect are possibly insufficient. The figures can be recovered using the Julia script file Empirical on the GitHub resource [20].

effects from lead-lag and Poissonian sampling. However, these models of the Epps effect only account for a drop in magnitude due to the concave nature, not a change in sign.

A simple theoretical Epps effect arising from Poissonian sampling given by (21) is plotted for the correlation pair FSR/SBK (green dashes) in Figure 9. This is done by assuming the inter-arrival time of trades follow an exponential distribution with larger of the two rates: $\max(\hat{\lambda}_{FSR}, \hat{\lambda}_{SBK}) = \hat{\lambda} \approx 1/13.5$. Furthermore, we found that $c = 0.621$ produced a relatively good fit for the measured correlations using the Dirichlet basis. No theoretical Epps curve is plotted for the correlation pair FSR/AGL because the correlation dynamics do not exhibit the Epps effect.

Figure 9 is important because it illustrates a case where the Epps effect can be modelled, and a case where it cannot be modelled. Although an over-simplistic theoretical Epps effect model was used, what is demonstrated is that one can find a more complicated theoretical Epps effect model which can explain the correlation between FSR/SBK. This means that majority of the correlation pairs in Figure B.10 can be modelled using some combination of lead-lag and asynchrony. Mastromatteo *et al.* do caution that a significant portion of the Epps effect cannot be explained by current models of the Epps effect, meaning there are other factors which can affect the dynamics of the observed correlation [27]. This is best shown in the FSR/AGL pair from Figure 9. The correlation dynamics do not even exhibit an Epps-like effect. Suggesting either: i.) current theoretical explanations for the Epps effect are possibly insufficient, or, ii.) the correlation dynamics under market-microstructure cannot be explained with only the Epps effect.

Now to check that the observed dynamics between FSR/AGL is not a result of estimation uncertainty, but rather truly reflecting the underlying correlation dynamics; we investigate cases where pathological correlation choices were made. Figure C.12, we perform the same experiment as Figure 8, but for three features with induced correlation choices $\rho_{12} = -0.5$, $\rho_{13} = 0.7$ and $\rho_{23} = 0.01$ for (a) and $\rho_{12} = 0.5$, $\rho_{13} = 0.7$ and $\rho_{23} = 0.01$ for (b). We see that for $\rho \approx 0$, when n and N is not large enough, estimation uncertainty arises. Thus the correlation estimates can switch signs; however, this does not account for the drop in correlation magnitude as Δt increases, as is the case for FSR/AGL. Meaning there are empirical correlation dynamics which cannot be explained by simulation or theoretical models.

5. Conclusions

We provide a fast novel implementation of the Malliavin-Mancino Fourier estimators using non-uniform fast Fourier transforms and promote the use of fast Gaussian gridding with the Fejér basis function as our preferred implementation.

Here we first compared three averaging kernels: the Gaussian, Kaiser-Bessel and exponential of semi-circle kernel. Based on the like-for-like algorithmic comparison, the fast Gaussian gridding is the fastest out of the three non-uniform fast Fourier methods. However, with appropriate implementation techniques the exponential of semi-circle kernel can be made to be faster than the fast Gaussian gridding [11]. All three non-uniform fast Fourier method significantly outperform naive implementations of the Malliavin-Mancino estimators.

Second, we demonstrate the requirement for using the non-uniform fast Fourier methods as motivated by the failure of the zero-padded fast Fourier transform on the arrival time representation of asynchrony.

Third, we demonstrate that there is no adverse interplay between kernel averaging and time-scale averaging (arising from the choice of N) - provided there is sufficient spreading to enough nearby grid points. Concretely, when using our choice of M_{sp}, provided $\epsilon < 10^{-4}$, the non-uniform fast Fourier methods can accurately recover the same estimates as the naive implementation, and therefore feasible for all cases of asynchrony.

Fourth, we provide the link between the work done by Renò [12] and Precup and Iori [13] with the work from Tóth and Kertész [26] and Mastromatteo, Marsili and Zoi [27]. It is here where we see clear differences between the basis kernels used in the Malliavin-Mancino estimator and their use cases. The Dirichlet kernel plausibly recovers the theoretical Epps effect while the Fejér kernel produces smoothed estimates that have an upward bias relative to the Dirichlet kernel. However, the Fejér kernel guarantees positive semi-definiteness while the Dirichlet does not for the case when N is the same across all features.

Finally, we demonstrate the efficacy of our non-uniform fast Fourier methods with one week of Trade and Quote data from the JSE. We argue that the current theoretical explanations for the Epps effect are possibly insufficient in explaining the entirety of the empirical correlation dynamics under specific market microstructures.

Future work aims to expand our empirical understanding of correlation dynamics on various streaming event-data sources using this convenient estimation tool as highlighted in Appendix B and to incorporate the estimators NUFFT implementation with an extension that makes the estimate robust to samples that explicitly incorporate jumps [35].

6. Acknowledgements

The authors would like to thank Melusi Mavuso and Roger Bukuru. The data was sourced from Bloomberg Professional via the University of Cape Town Library service.

Appendix A. Algorithms

Algorithm 1 Malliavin-Mancino Estimators

Require:

1. **P**: (n x D) matrix of sampled prices. Non-trade times are represented using *NaNs* or *NAs*.
2. **T**: (n x D) matrix of sampled times. Non-trade times are represented using *NaNs* or *NAs*.
3. N (Optional): cutoff frequency (Integer) used in the convolution. Default is set to be the Nyquist cutoff.
4. tol (Optional): error tolerance for NUFFTs. Determines the number of grid points to spread. Default is set to 10^{-12}.

Step I. Initialisation.

I.1. Re-scale the sampled times (T) (See Algorithm 2).

I.2. Compute the Nyquist cutoff (N) - unless specified otherwise through input parameter (See Algorithm 4).

Step F: Compute the Fourier coefficients, $k \in \{-N, ..., N\}$.
for i = 1 to D **do**

F.1. Extract the re-scaled sampled times for the i^{th} object: $\tilde{\mathbf{t}} = \mathbf{T}(i)$, excluding any *NaNs* or *NAs*.

F.2. Extract and compute the logarithm of the sampled prices for the i^{th} object: $\tilde{\mathbf{p}} = \ln(\mathbf{p}(\tilde{t}))$, excluding any *NaNs* or *NAs*.

F.3. Compute the returns: $\delta_j = \tilde{p}(\tilde{t}_{j+1}) - \tilde{p}(\tilde{t}_j)$

F.4. Compute the Fourier coefficients:

$$c_k^+(i) = \sum_{j=1}^{n-1} e^{ik\tilde{t}_j}\delta_j; \quad c_k^-(i) = \sum_{j=1}^{n-1} e^{-ik\tilde{t}_j}\delta_j$$

end for
Step C: Convolution.

C.1. The Dirichlet implementation:

$$\hat{\Sigma}_{ij} = \frac{1}{2N+1}\sum_{k=-N}^{N}[c_k^+(i)c_k^-(j)]$$

C.2. The Fejér implementation:

$$\hat{\Sigma}_{ij} = \frac{1}{N+1}\sum_{k=-N}^{N}\left(1 - \frac{|k|}{N}\right)[c_k^+(i)c_k^-(j)]$$

Correlation: $R_{ij} = \frac{\Sigma_{ij}}{\sqrt{\Sigma_{ii}}\sqrt{\Sigma_{jj}}}$

return (**Σ**, **R**)

Table A.3: The Malliavin-Mancino estimators (See Algorithm 1) computes the Dirichlet or Fejér implementation of the Malliavin-Mancino estimator [1, 2] using a complex exponential formulation of the Fourier transform. The algorithm is a mere sketch provided by [17] and is based on their MATLAB implementation [16]. The various implementations differ only at steps F.4 and C when compared to the standard vanilla implementation given by [3].

Algorithm 2 Time-rescaling Algorithm

Require:

1. **T**: (n x D) matrix of sampled times. Non-trade times are represented using *NaNs* or *NAs*.

Set: $t_{\min}$ = minimum value of **T**
Set: $t_{\max}$ = maximum value of **T**
for j = 1 to D **do**
  for i = 1 to n **do**

$$\tilde{t}_{ij} = \frac{2\pi(t_j - t_{min})}{t_{\max} - t_{\min}}$$

  end for
end for
return ($\tilde{\mathbf{t}}$)

Table A.4: The Time-rescaling Algorithm (See Algorithm 2) rescales the trading times from $[0, T]$ to $[0, 2\pi]$. The Julia implementation can be found in any of the Dirichlet or Fejér implementations in the GitHub resource [20] and is an auxiliary function based on the MATLAB implementation from [16, 17].

Algorithm 3 GBM Algorithm

Require:

1. n: number of price points to simulate.
2. $\boldsymbol{\mu}$: (D x 1) vector of drift parameters.
3. $\boldsymbol{\Sigma}$: (D x D) covariance matrix.
4. start price: (D x 1) vector of $S(0)$.

Procedure for the i^{th} feature:

1. Generate: $\mathbf{Z} \sim \mathcal{N}_D(\mathbf{0}, \mathbf{I}_{DxD})$.
2. Set: $S_i(t_{k+1}) = S_i(t_k) \exp\left[(\mu_i - \frac{1}{2}\sigma_i^2)(t_{k+1} - t_k) + \sqrt{t_{k+1} - t_k} \sum_{k=1}^{d} A_{ik} Z_k\right]$.

return (**S**)

Table A.5: The Geometric Brownian Motion (GBM) Algorithm (See Algorithm 3) simulates a correlated multivariate GBM using the first order Euler discretisation. It is subject to the initial condition $S(0)$ = start price. **A** is the Cholesky decomposition of Σ. The Julia implementation can be found at GBM in the GitHub resource [20] and was provided by [36].

Algorithm 4 Nyquist Frequency

Require:

1. $\tilde{\mathbf{t}}$: (n x D) of re-scaled sampled times. Non-trade times are represented using *NaNs* or *NAs*.

Set: Δt = minimum distance between sampled times.
Set: $N_0 = \frac{2\pi}{\Delta t}$.
Set: N = $\lfloor \frac{N_0}{2} \rfloor$, where $\lfloor x \rfloor$ denotes the floor of x.
return (N)

Table A.6: The Nyquist frequency Algorithm (See Algorithm 4) computes the Nyquist cutoff used in our data informed approach [4]. The Julia implementation can be found in any of the Dirichlet or Fejér implementations in the GitHub resource [20] and is an auxiliary function based on the MATLAB implementation from [16, 17].

Algorithm 5 "for-loop" implementation

Require:

1. $\mathbf{f} = (f_j)_{j=1}^{n-1}$: vector of source strengths (corresponding to δ_j).
2. $\mathbf{x} = (x_j)_{j=1}^{n-1}$: vector of sample times (corresponding to the source points $\tilde{t_j}$).
3. N: the cutoff frequency.

for s =1 to 2N + 1 **do**
  $k = s - N - 1$

$$c_s^+ = \sum_{j=1}^{n-1} e^{ikx_j} f_j$$

$$c_s^- = \sum_{j=1}^{n-1} e^{-ik\ x_j} f_j$$

end for
return (c_k^+, c_k^-)

Table A.7: The for-loop implementation [3] (See Algorithm 5) computes the Fourier coefficients using for-loops. The Julia implementation can be found in MScorrDK or MScorrFK on the GitHub resource [20] and correspond to the Dirichlet and Fejér representation respectively. The implementation is based on the MATLAB implementation from [3].

Algorithm 6 Vectorised code implementation

Require:

1. $\mathbf{f} = (f_j)_{j=1}^{n-1}$: column vector of source strengths (corresponding to δ_j).
2. $\mathbf{x} = (x_j)_{j=1}^{n-1}$: column vector of sample times (corresponding to source points $\tilde{t}_j$).
3. N: the cutoff frequency.

Set: $\mathbf{k} = (1, 2, \ldots, N)^T$ a column vector 1 to N.
Compute: $\mathbf{c}_{1:N} = \mathbf{f}^\intercal \exp(-i\,\mathbf{x}\,\mathbf{k}^\intercal)$
Compute: $c_0 = \sum_{j-1}^{n-1} f_j$
Piece $\mathbf{c}_{1:N}$, $\overline{\mathbf{c}_{1:N}}$ and c_0 together to obtain c_k^+ and c_k^-
return (c_k^+, c_k^-)

Table A.8: The vectorised code implementation [16, 17] (See Algorithm 6) replaces for-loops to vectorise the computation of the Fourier coefficients and exploits the Hermitian symmetry of the real source strengths. The Julia implementation can be found in CFTcorrDK or CFTcorrFK on the GitHub resource [20] and correspond to the Dirichlet and Fejér representation respectively.

Algorithm 7 Zero-padded FFT implementation

Require:

1. f_j: column vector of source strengths (corresponding to δ_j).
2. $x_j \in [0, 2\pi]$: column vector of sample times (corresponding to the source points $\tilde{t}_j$).
3. $N^* = \lfloor \frac{2\pi}{\Delta t} \rceil$, where $\lfloor x \rceil$ denotes rounding x to the nearest Integer and Δt is the minimum distance between sampled times from Algorithm 4.

Set: n_j = length of x_j.
Initialise: $(\tilde{f}_\ell)_{\ell=1}^{N^*} = \mathbf{0}$, a zero vector of length N^*.
for j = 1 to n_j **do**
  $\ell = \lfloor \frac{N^* x_j}{2\pi} \rceil + 1$
  $\tilde{f}_\ell = f_j$
end for
return ($\tilde{f}_\ell$ for FFT computation)

Table A.9: The zero-padded FFT implementation (See Algorithm 7) creates a uniform grid and shifts the non-uniform source points to the nearest grid point on an up-sampled uniform grid. The Julia implementation can be found in FFTZPcorrDK or FFTZPcorrFK on the GitHub resource [20] and correspond to the Dirichlet and Fejér representation respectively. Note that the index ℓ is set for languages with array indices starting from 1.

Algorithm 8 Fast Gaussian Gridding NUFFT implementation

Require:

1. f_j: vector of source strengths (corresponding to δ_j).
2. $x_j \in [0, 2\pi]$: vector of sample times (corresponding to the source points $\tilde{t}_j$).
3. $M = 2N + 1$: the number of Fourier modes computed.
4. ϵ: error tolerance.

Step I. Initialisation:

I.1. Set: n_j = length of x_j; $\sigma = 2$.

I.2. Set: $M_r = \sigma M$; $M_{sp} = \lfloor \frac{-\ln(\epsilon)(\sigma - 1/2)}{(\pi(\sigma-1))} + \frac{1}{2} \rfloor$.

I.3. Set: $\lambda = \frac{\sigma^2 M_{sp}}{\sigma(\sigma - 0.5)}$; $h_x = \frac{2\pi}{M_r}$; $t_1 = \frac{\pi}{\lambda}$.

I.4. Set: $\tau = \frac{\pi\lambda}{M_r^2}$.

I.5. Initialise: $(\tilde{f}_\ell)_{\ell=1}^{M_r} = \mathbf{0}$, a zero vector of length M_r.

for k = 1 to M_{sp} **do**
  $E_{3,k} = \exp(-t_1 k^2)$
end for
Step C: Convolution (See 15).
for j = 1 to n_j **do**
  $b_0 = \lfloor \frac{x_j}{h_x} \rfloor$, index of nearest up-sampled grid $\xi_{b_0} \leq x_j$.
  d = $\frac{x_j}{h_x} - b_0$.
  $E_0 = \mathbf{0}$, a zero vector of length $2M_{sp}$.
  $E_1 = e^{-t_1 \mathrm{d}^2}$; $E_{0,M_{sp}} = E_1$; $E_2 = e^{2t_1 \mathrm{d}}$.
  for k = 1 to M_{sp} **do**
    $E_{0,M_{sp}+k} = E_{3,k} E_1 E_2^k$.
  end for
  for k = 1 to $M_{sp} - 1$ **do**
    $E_{0,M_{sp}-k} = E_{3,k} E_1 E_2^{-k}$.
  end for
  $b_d = \min(M_{sp} - 1, b_0)$; $b_u = \min(M_{sp}, M_r - b_0 - 1)$.
  for k = $-M_{sp} + 1$ to $-b_d - 1$ **do**
    $\tilde{f}_{b_0+k+M_r+1} = \tilde{f}_{b_0+k+M_r+1} + f_j E_{0,M_{sp}+k}$.
  end for
  for k = $-b_d$ to b_u **do**
    $\tilde{f}_{b_0+k+1} = \tilde{f}_{b_0+k+1} + f_j E_{0,M_{sp}+k}$.
  end for
  for k = $b_u + 1$ to M_{sp} **do**
    $\tilde{f}_{b_0+k-M_r+1} = \tilde{f}_{b_0+k-M_r+1} + f_j E_{0,M_{sp}+k}$.
  end for
end for
Step F: Compute FFT on over-sampled grid (See (16)).

F.1. Find Fourier coefficients $F_G(k)$ via FFT on the grid $\tilde{f}_\ell$.

Step D: Deconvolution (See (17)).

D.1. Compute: $F(k) = \sqrt{\frac{\pi}{\tau}} e^{k^2\tau} F_G(k) \frac{1}{M_r}$.

return $(F(k), k \in \{-M, ..., M\})$

Table A.10: The non-uniform FFT implementation (See Algorithm 8) creates an up-sampled uniform grid and convolves the non-uniform source points onto the uniform grid, applies the FFT on the up-sampled grid and deconvolves the convolution effects in the Fourier space. Algorithm 8 is specific for the Gaussian kernel as it uses the fast Gaussian gridding implementation by [10] to reduce the number of exponential evaluations. The Julia implementation can be found in NUFFT-FGG on the GitHub resource [20]. The Algorithm is a replication of the FORTRAN source code from [10], but adjusted for languages with array indices starting from 1.

Algorithm 9 Kaiser-Bessel NUFFT implementation

Require:

1. f_j: vector of source strengths (corresponding to δ_j).
2. $x_j \in [0,1]$: vector of sample times (corresponding to the source points $\tilde{t}_j$ re-scaled as $\frac{\tilde{t}_j}{2\pi}$).
3. $M = 2N + 1$: the number of Fourier modes computed.
4. ϵ: error tolerance.

Step I. Initialisation:

I.1. Set: n_j = length of x_j; $\sigma = 2$.

I.2. Set: $M_r = \sigma M$; $M_{sp} = \lfloor \frac{1}{2}(\lceil \log_{10}(\frac{1}{\epsilon}) \rceil + 2) \rfloor$.

I.3. Initialise: $(\tilde{f}_\ell)_{\ell=1}^{M_r} = \mathbf{0}$, a zero vector of length M_r.

Step C: Convolution (See 15).

for j = 1 to n_j **do**

$b_0 = \lfloor x_j M_r \rfloor$, index of nearest up-sampled grid

$\xi_{b_0} \leq x_j$.

d = $x_j - \frac{b_0}{M_r}$.

$b_d = \min(M_{sp} - 1, b_0)$; $b_u = \min(M_{sp}, M_r - b_0 - 1)$.

for k = $-M_{sp}$ to $-b_d - 1$ **do**

$\tilde{f}_{b_0+k+M_r+1} = \tilde{f}_{b_0+k+M_r+1} + f_j \varphi_{KB}(d - \frac{k}{M_r})$.

end for

for k = $-b_d$ to b_u **do**

$\tilde{f}_{b_0+k+1} = \tilde{f}_{b_0+k+1} + f_j \varphi_{KB}(d - \frac{k}{M_r})$.

end for

for k = $b_u + 1$ to M_{sp} **do**

$\tilde{f}_{b_0+k-M_r+1} = \tilde{f}_{b_0+k-M_r+1} + f_j \varphi_{KB}(d - \frac{k}{M_r})$.

end for

end for

Step F: Compute FFT on over-sampled grid (See (16)).

F.1. Find Fourier coefficients $F_{KB}(k)$ via FFT on the grid $\tilde{f}_\ell$.

Step D: Deconvolution (See (17)).

D.1. Compute: $F(k) = \frac{1}{M_r} F_{KB}(k) / \hat{\varphi}_{KB}(k)$.

return $(F(k), k \in \{-M, ..., M\})$

Table A.11: The non-uniform FFT implementation (See Algorithm 9) creates an up-sampled uniform grid and convolves the non-uniform source points onto the uniform grid, applies the FFT on the up-sampled grid and deconvolves the convolution effects in the Fourier space. Algorithm 9 uses the Kaiser-Bessel kernel here, but the algorithm can be applied to any kernel that is 1-periodic. The algorithm improves upon Algorithm 5.2 in [14] by removing the unnecessary computation of $\tilde{\varphi}(\cdot)$ using techniques from Algorithm 8 as seen in Figure 2. However the algorithm is structured without the precomputation step used in [14] and evaluates the algorithm "on-the-fly". The Julia implementation can be found in NUFFT-KB on the GitHub resource [20]. The Algorithm is set for languages with array indicies starting from 1.

Algorithm 10 Exponential of semi-circle NUFFT implementation

Require:

1. f_j: vector of source strengths (corresponding to δ_j).
2. $x_j \in [0, 2\pi]$: vector of sample times (corresponding to the source points $\tilde{t}_j$).
3. $M = 2N + 1$: the number of Fourier modes computed.
4. ϵ: error tolerance.

Step I. Initialisation:

I.1. Set: n_j = length of x_j; $\sigma = 2$.

I.2. Set: $M_r = \sigma M$; $M_{sp} = \lfloor \frac{1}{2}(\lceil \log_{10}(\frac{1}{\epsilon}) \rceil + 2) \rfloor + 2$.

I.3. Initialise: $(\tilde{f}_\ell)_{\ell=1}^{M_r} = \mathbf{0}$, a zero vector of length M_r.

Step C: Convolution (See 15).

for j = 1 to n_j **do**

$b_0 = \lfloor x_j M_r \rfloor$, index of nearest up-sampled grid

$\xi_{b_0} \leq x_j$.

d = $x_j - \frac{b_0}{M_r}$.

$b_d = \min(M_{sp} - 1, b_0)$; $b_u = \min(M_{sp}, M_r - b_0 - 1)$.

for k = $-M_{sp}$ to $-b_d - 1$ **do**

$\tilde{f}_{b_0+k+M_r+1} = \tilde{f}_{b_0+k+M_r+1} + f_j \varphi_{ES}(d - \frac{2\pi k}{M_r})$.

end for

for k = $-b_d$ to b_u **do**

$\tilde{f}_{b_0+k+1} = \tilde{f}_{b_0+k+1} + f_j \varphi_{ES}(d - \frac{2\pi k}{M_r})$.

end for

for k = $b_u + 1$ to M_{sp} **do**

$\tilde{f}_{b_0+k-M_r+1} = \tilde{f}_{b_0+k-M_r+1} + f_j \varphi_{ES}(d - \frac{2\pi k}{M_r})$.

end for

end for

Step F: Compute FFT on over-sampled grid (See (16)).

F.1. Find Fourier coefficients $F_{ES}(k)$ via FFT on the grid $\tilde{f}_\ell$.

Step D: Deconvolution (See (17)).

D.1. Compute: $\hat{\varphi}_{ES}(k) = \alpha \int_{-\infty}^{\infty} \phi_{ES}(x) e^{i\alpha k x} dx$ using numerical integration.

D.2. Compute: $F(k) = \frac{2\pi}{M_r} F_{ES}(k) / \hat{\varphi}_{ES}(k)$.

return $(F(k), k \in \{-M, ..., M\})$

Table A.12: The non-uniform FFT implementation (See Algorithm 10) creates an up-sampled uniform grid and convolves the non-uniform source points onto the uniform grid, applies the FFT on the up-sampled grid and deconvolves the convolution effects in the Fourier space. Algorithm 10 uses the exponential of semi-circle kernel. The algorithm is a naive implementation based on the steps provided in [11] and does not exploit the piecewise polynomial kernel approximation nor the Gauss-Legendre quadrature for implementation acceleration used in FINUFFT [11]. The implementation relies on the QuadGK package to perform the numerical integration using adaptive Gauss-Kronrod quadrature. The Julia implementation can be found in NUFFT-ES on the GitHub resource [20]. The Algorithm is set for languages with array indices starting from 1.

Appendix B. Epps effect EDA for 10 JSE stocks

Here we consider the 10 stocks as described in Table 2 and provide the correlation heat-maps in Figure B.11, and Epps effect plots in Figure B.10.

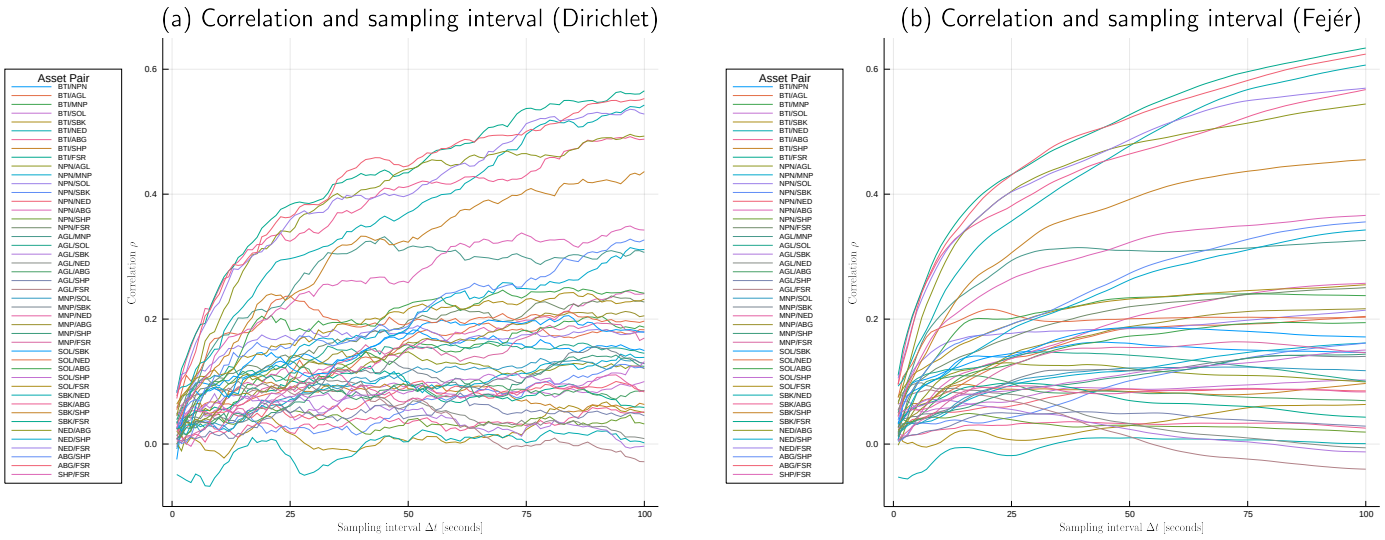

Figure B.10: Figure B.10 investigates the Epps effect on the JSE by plotting the correlation estimate from the Malliavin-Mancino estimator as a function of the sampling interval Δt. The conversion between Δt and N is given by (22), assuming $T = 144,000$ seconds in the 5 day period. The correlation pairs are plotted for all 10 equities. Sub-figure (a) is the estimates using the Dirichlet basis kernel and (b) the Fejér basis kernel. It is clear that the Fejér kernel produces smoother estimates compared to the Dirichlet kernel due to the induced averaging. Finally, we see that most of the equity pairs produce the Epps effect demonstrated in Figure 8, but more interesting is that there is an equity pair where the correlation switches signs for different Δt. The figures can be recovered using the Julia script file Empirical on the GitHub resource [20].

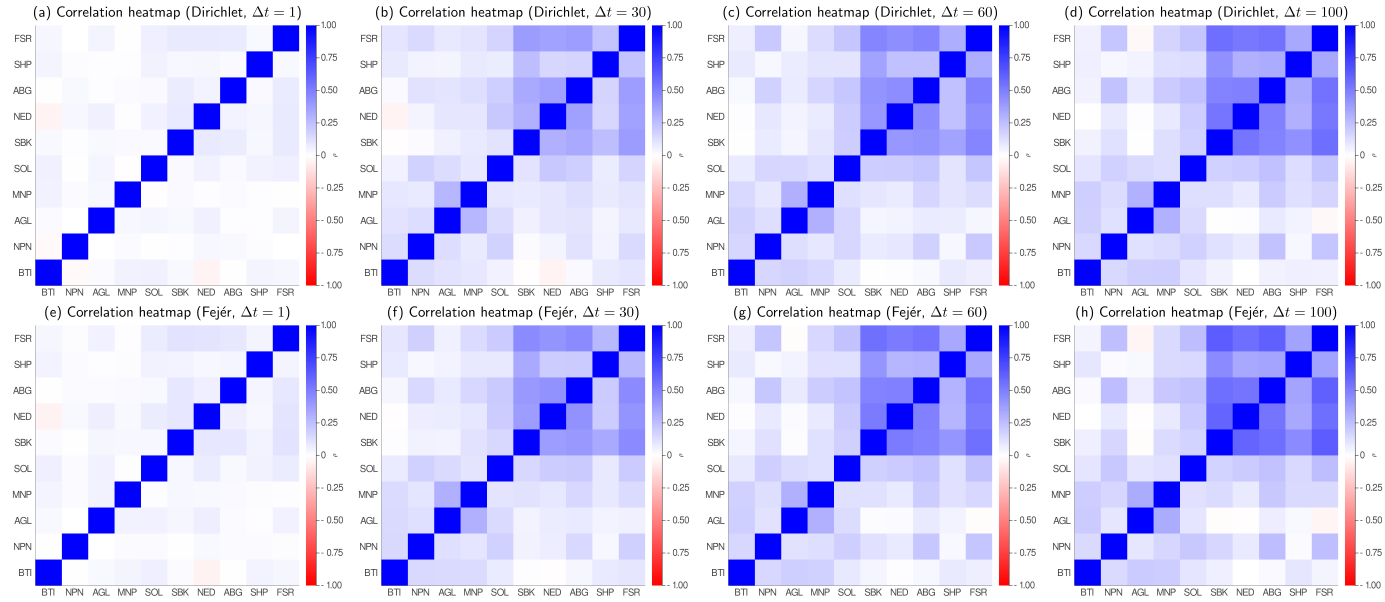

Figure B.11: Figure B.11 investigates the Epps effect on the JSE by plotting correlation structure as heat-maps for snapshots from Figure B.10. The snapshots are taken for $\Delta t = 1, 30, 60$ and 100 seconds for (a) to (d) and (e) to (h) respectively. (a) to (d) plots the correlation structure using the Dirichlet basis and (e) to (h) the Fejér basis. We see that the correlations at short time scales are generally positively correlated - with the top right quadrant being the most positively correlated as they are from the banking sector. More interestingly, we see that the correlation pair FSR/AGL goes from positively correlated to negatively correlated as Δt increases. The figures can be recovered using the Julia script file Empirical on the GitHub resource [20].

Appendix C. Simulated 3-asset case

Here we consider 3 simulated correlated stocks to demonstrate the interplay of the combination of negative and positive correlations in Figure C.12. The change of scale can lead to spurious negative and positive correlations when there is insufficient data.

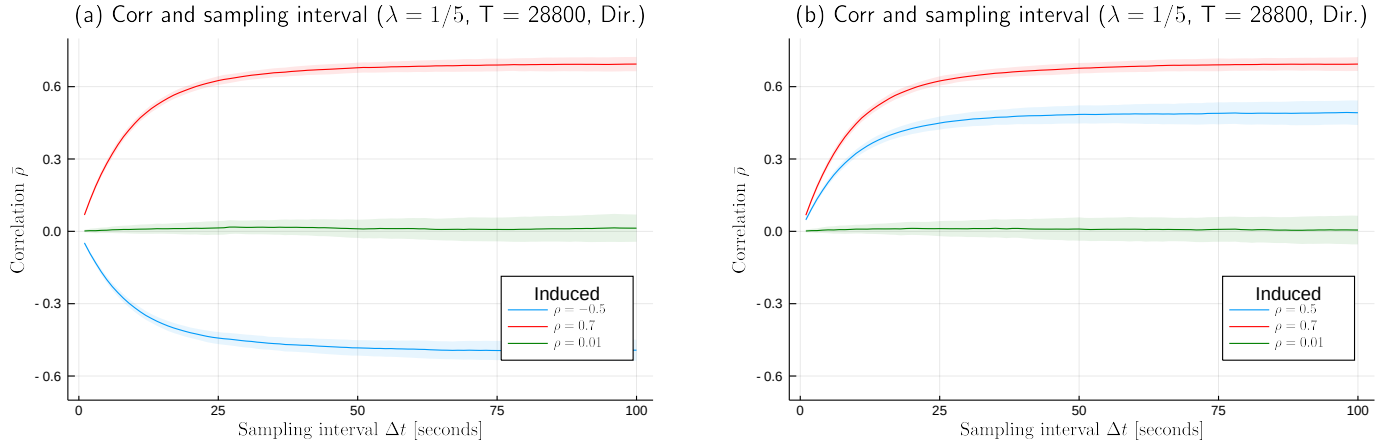

Figure C.12: Figure C.12 investigates whether the interplay from correlation combinations or estimation uncertainty can explain the correlation dynamics found in FSR/AGL from Figure 9. We simulate $T = 28800$ data points for a three feature Geometric Brownian Motion with induced correlation choices $\rho_{12} = -0.5$ (blue line), $\rho_{13} = 0.7$ (red line) and $\rho_{23} = 0.01$ (green line) for (a) and $\rho_{12} = 0.5$ (blue line), $\rho_{13} = 0.7$ (red line) and $\rho_{23} = 0.01$ (green line) for (b). The synchronous case is then sampled with an exponential inter-arrival process with rate $\lambda = 1/5$. The Dirichlet estimates are obtained for Δt ranging from 1 to 100 with the conversion to N given in Equation (22). This process is repeated 100 times and the average correlation estimate at each Δt is then plotted with error bars (computed using a t-distribution with 99 degrees of freedom and the sample standard deviation) representing 68% of the variability between the estimation paths. We see that for $\rho \approx 0$, when n and N is not large enough, estimation uncertainty arises, explaining the switching of signs; but it does not account for the correlation magnitude dropping as Δt increases. The figures can be recovered using the Julia script file 3Asset on the GitHub resource [20].

References

[1] P. Malliavin, M. E. Mancino, A fourier transform method for nonparametric estimation of multivariate volatility, Ann. Statist. 37 (4) (2009) 1983–2010. doi:10.1214/08-AOS633.
URL https://doi.org/10.1214/08-AOS633

[2] P. Malliavin, M. E. Mancino, Fourier series method for measurement of multivariate volatilities, Finance and Stochastics 6 (1) (2002) 49–61. doi:10.1007/s780-002-8400-6.
URL https://doi.org/10.1007/s780-002-8400-6

[3] M. Mancino, M. Recchioni, S. Sanfelici, Fourier-Malliavin Volatility Estimation Theory and Practice, Springer International Publishing, 2017. doi:10.1007/978-3-319-50969-3.

[4] P. Chang, R. Bukuru, T. Gebbie, Revisting the epps effect using volume time averaging: An exercise in r (2019).
URL https://arxiv.org/abs/1912.02416

[5] D. Hendricks, T. Gebbie, D. Wilcox, Detecting intraday financial market states using temporal clustering, Quantitative Finance 16 (11) (2016) 1657–1678.

[6] D. Hendricks, Using real-time cluster configurations of streaming asynchronous features as online state descriptors in financial markets, Pattern Recognition Letters 97 (2017) 21 – 28.

[7] D. Hendricks, D. Wilcox, A reinforcement learning extension to the almgren-chriss framework for optimal trade execution, 2014 IEEE Conference on Computational Intelligence for Financial Engineering & Economics (CIFEr) (2014) 457–464.

[8] J. Cooley, J. Tukey, An algorithm for the machine calculation of complex fourier series, Mathematics of Computation 19 (90) (1965) 297–301. doi:10.1090/S0025-5718-1965-0178586-1.

[9] A. Dutt, V. Rokhlin, Fast fourier transforms for nonequispaced data, SIAM Journal on Scientific Computing 14 (6) (1993) 1368–1393. arXiv:https://doi.org/10.1137/0914081, doi:10.1137/0914081.
URL https://doi.org/10.1137/0914081

[10] L. Greengard, J.-Y. Lee, Accelerating the nonuniform fast fourier transform, SIAM Review 46 (3) (2004) 443–454. arXiv:https://doi.org/10.1137/S003614450343200X, doi:10.1137/S003614450343200X.
URL https://doi.org/10.1137/S003614450343200X

[11] A. H. Barnett, J. F. Magland, L. af Klinteberg, A parallel non-uniform fast fourier transform library based on an "exponential of semicircle" kernel, SIAM J. Scientific Computing 41 (2018) C479–C504. doi:10.1137/18m120885x.

[12] R. Renò, A closer look at the epps effect, International Journal of Theoretical and Applied Finance 06 (11 2001). doi:10.2139/ssrn.314723.

[13] O. V. Precup, G. Iori, Cross-correlation measures in the high-frequency domain, The European Journal of Finance 13 (4) (2007) 319–331. arXiv:https://doi.org/10.1080/13518470600813565, doi:10.1080/13518470600813565.
URL https://doi.org/10.1080/13518470600813565

[14] D. Potts, G. Steidl, Fast summation at nonequispaced knots by nfft, SIAM Journal on Scientific Computing 24 (6) (2003) 2013–2037. arXiv:https://doi.org/10.1137/S1064827502400984, doi:10.1137/S1064827502400984.
URL https://doi.org/10.1137/S1064827502400984

[15] C. Malherbe, Fourier method for the measurement of univariate and multivariate volatility in the presence of high frequency data, Msc. dissertation, University of Cape Town (2007).

[16] C. Malherbe, D. Hendricks, T. Gebbie, D. Wilcox, Matlab functions `ftcorrgpu.m` and `fftcorrgpu.m` (2005).

[17] D. Hendricks, T. Gebbie, D. Wilcox, High-speed fourier method estimation of covariances from asynchronous data, working paper (2017).

[18] E. Barucci, R. Renò, On measuring volatility and the garch forecasting performance, Journal of International Financial Markets, Institutions and Money 12 (2002) 183–200. doi:10.1016/S1042-4431(02)00002-1.

[19] M. Frigo, S. G. Johnson, The design and implementation of FFTW3, Proceedings of the IEEE 93 (2) (2005) 216–231, special issue on "Program Generation, Optimization, and Platform Adaptation". doi:10.1109/JPROC.2004.840301.

[20] P. Chang, E. Pienaar, T. Gebbie, Julia code: Malliavin-mancino estimators implemented with the non-uniform fast fourier transform (2020). doi:10.25375/uct.11903418.
URL https://github.com/CHNPAT005/PCEPTG-MM-NUFFT

[21] L. af Klinteberg, Julia interface to finufft (2018).
URL https://github.com/ludvigak/FINUFFT.jl

[22] M. E. Mancino, S. Sanfelici, Estimating Covariance via Fourier Method in the Presence of Asynchronous Trading and Microstructure Noise, Journal of Financial Econometrics 9 (2) (2011) 367–408. arXiv:https://academic.oup.com/jfec/article-pdf/9/2/367/2437637/nbq031.pdf, doi:10.1093/jjfinec/nbq031.
URL https://doi.org/10.1093/jjfinec/nbq031

[23] F. Lindskog, Linear correlation estimation (2001).

[24] L. Matoti, Building a statistical linear factor model and a global minimum variance portfolio using estimated covariance matrices, Msc. dissertation, University of Cape Town (2009).

[25] T. W. Epps, Comovements in stock prices in the very short run, Journal of the American Statistical Association 74 (366) (1979) 291–298.
URL http://www.jstor.org/stable/2286325

[26] B. Tóth, J. Kertész, Modeling the Epps effect of cross correlations in asset prices, in: J. Kertész, S. Bornholdt, R. N. Mantegna (Eds.), Noise and Stochastics in Complex Systems and Finance, Vol. 6601, International Society for Optics and Photonics, SPIE, 2007, pp. 89 – 97. doi:10.1117/12.727127.
URL https://doi.org/10.1117/12.727127

[27] I. Mastromatteo, M. Marsili, P. Zoi, Financial correlations at ultra-high frequency: theoretical models and empirical estimation, The European Physical Journal B 80 (2) (2011) 243–253. doi:10.1140/epjb/e2011-10865-y.
URL https://doi.org/10.1140/epjb/e2011-10865-y

[28] T. Hayashi, N. Yoshida, On covariance estimation of non-synchronously observed diffusion processes, Bernoulli 11 (2) (2005) 359–379. doi:10.3150/bj/1116340299.
URL https://doi.org/10.3150/bj/1116340299

[29] M. C. Münnix, R. Schäfer, T. Guhr, Statistical causes for the epps effect in microstructure noise, International Journal of Theoretical and Applied Finance 14 (08) (2011) 1231–1246. arXiv:https://doi.org/10.1142/S0219024911006838, doi:10.1142/S0219024911006838.
URL https://doi.org/10.1142/S0219024911006838

[30] L. Zhang, Estimating covariation: Epps effect, microstructure noise, Journal of Econometrics 160 (2010) 33–47. doi:10.1016/j.jeconom.2010.03.012.

[31] M. C. Münnix, R. Schäfer, T. Guhr, Impact of the tick-size on financial returns and correlations, Physica A: Statistical Mechanics and its Applications 389 (21) (2010) 4828 – 4843. doi:https://doi.org/10.1016/j.physa.2010.06.037.
URL http://www.sciencedirect.com/science/article/pii/S0378437110005662

[32] A. Saichev, D. Sornette, A simple microstructure return model explaining microstructure noise and epps effects, International Journal of Modern Physics C 25 (06) (2014) 1450012. arXiv:https://doi.org/10.1142/S0129183114500120, doi:10.1142/S0129183114500120.
URL https://doi.org/10.1142/S0129183114500120

[33] B. Tóth, J. Kertész, The epps effect revisited, Quantitative Finance 9 (7) (2009) 793–802. arXiv:https://doi.org/10.1080/14697680802595668, doi:10.1080/14697680802595668.
URL https://doi.org/10.1080/14697680802595668

[34] P. Chang, E. Pienaar, T. Gebbie, Johannesburg stock exchange trade and quote data (2020). doi:10.25375/uct.11903442.

[35] C. Cuchiero, J. Teichmann, Fourier transform methods for pathwise covariance estimation in the presence of jumps, Stochastic Processes and their Applications 125 (1) (2015) 116 – 160. doi:https://doi.org/10.1016/j.spa.2014.07.023.
URL http://www.sciencedirect.com/science/article/pii/S0304414914001823

[36] P. Glasserman, Monte Carlo methods in financial engineering, Springer, New York, 2004.