Superfast Inference for Stationary Gaussian Time Series

Yun Ling and Martin Lysy[*]
Department of Statistics and Actuarial Science
University of Waterloo

March 9, 2020

Abstract

Parametric inference for stationary Gaussian observations is a commonplace task in time series modeling and unsupervised function estimation. Due to the Toeplitz structure of the variance matrix, targetted algorithms for likelihood evaluation offer enormous accelerations over the unstructured setting. However, these "fast" algorithms are $\mathcal{O}(N^2)$ in the number of observations, thus scaling poorly to large datasets. We present here a set of "superfast" methods scaling as $\mathcal{O}(N \log^2 N)$. The starting point is a divide-and-conquer variant of Schur's algorithm for solving Toeplitz systems, which unlike other superfast solvers, has very low overhead and produces superfast log-determinants as well. We generalize the algorithm to arbitrary N, and propose novel superfast algorithms for score and Hessian calculations. This effectively enables superfast inference for stationary Gaussians via a wide array of frequentist and Bayesian methods. We provide an implementation of our methods in the R/C++ package SuperGauss. Numerical results indicate a high degree of speed and accuracy for several popular stationary models.

Keywords: Stationary Gaussian time series; superfast likelihood inference; Generalized Schur algorithm.

[*]E-mail: `mlysy@uwaterloo.ca`. This research was partly supported by Natural Sciences and Engineering Council of Canada (NSERC) grant RGPIN-2014-04255.

1 Introduction

Stationary Gaussian processes are widely used in a variety of statistical applications including time series modeling (Breidt et al. 1998, Harvey 2002, Granger & Joyeux 1980, Hosking 1981), unsupervised function estimation (Smola & Bartlett 2001), differential equation modeling (Archambeau et al. 2007, Calderhead et al. 2009) and signal filtering and smoothing (Särkkä et al. 2014). They have convenient properties for various modeling tasks in machine learning (Williams & Rasmussen 2006), examples range from regression over classification (Neal 1997) to reinforcement learning (Engel et al. 2005).

Many of the applications listed above involve the estimation of the unknown parameters $\boldsymbol{\theta}$ of a stationary Gaussian time series from N consecutive equally spaced observations $\boldsymbol{x} = (x_1, \dots, x_N)$. In the simplest case, we have

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{V_\theta}),$$

where

$$\boldsymbol{V_\theta} = \begin{bmatrix} \gamma_{\boldsymbol{\theta}}(0) & \gamma_{\boldsymbol{\theta}}(1) & \gamma_{\boldsymbol{\theta}}(2) & \dots & \gamma_{\boldsymbol{\theta}}(N-1) \\ \gamma_{\boldsymbol{\theta}}(1) & \gamma_{\boldsymbol{\theta}}(0) & \gamma_{\boldsymbol{\theta}}(1) & \dots & \gamma_{\boldsymbol{\theta}}(N-2) \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \gamma_{\boldsymbol{\theta}}(N-1) & \gamma_{\boldsymbol{\theta}}(N-2) & \gamma_{\boldsymbol{\theta}}(N-3) & \dots & \gamma_{\boldsymbol{\theta}}(0) \end{bmatrix}$$

is a Toeplitz matrix of which the elements $\gamma_{\boldsymbol{\theta}}(h) = \operatorname{cov}(x_n, x_{n+h} \mid \boldsymbol{\theta})$ are parametrized by $\boldsymbol{\theta}$.

The log-likelihood for this problem is

$$\ell(\boldsymbol{\theta} \mid \boldsymbol{x}) = -\frac{1}{2}\left[\boldsymbol{x}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{x} + \log|\boldsymbol{V_\theta}|\right], \tag{1}$$

such that most approaches to parameter inference require repeatedly solving the Toeplitz system $\boldsymbol{V_\theta} \cdot \boldsymbol{z} = \boldsymbol{x}$ and evaluating $\log|\boldsymbol{V_\theta}|$ for different values of $\boldsymbol{\theta}$. Exploiting the Toeplitz structure of the variance matrix, "fast" algorithms for evaluating (1) require only $\mathcal{O}(N^2)$ operations (Levinson 1946, Durbin 1960, Trench 1964, Zohar 1969, Bareiss 1969). This is a massive computational improvement over unstructured variances, for which the corresponding calculations are $\mathcal{O}(N^3)$. However, the quadratic scaling of fast algorithms becomes a serious limitation when N is large.

Beginning with work on displacement ranks of Kailath et al. (1979), it was realized that Toeplitz systems could be solved by "superfast" FFT-based methods scaling as $\mathcal{O}(N\log^2 N)$ (Brent et al. 1980, Bitmead & Anderson 1980, De Hoog 1984, de Hoog 1987, Musicus 1988, Ammar & Gragg 1988, 1987, Chandrasekaran et al. 2007). However, these algorithms have yet to be leveraged for statistical analyses for a several reasons. For one, most of them do not provide direct means of calculating the log-determinant of $\boldsymbol{V_\theta}$ (though superfast methods for this calculation do exist, e.g. Kravanja & Van Barel (2000)). Moreover, many superfast algorithms bury considerable overhead in the big-O notation (Sexton 1982). Third and perhaps most importantly, many superfast algorithms are numerically unstable (Bunch 1985), prompting the developments of stable $\mathcal{O}(N\log^p N)$ solvers with $p > 2$ (e.g., the algorithms of Stewart (2003) and Chen et al. (2006) with $p = 3$ and $p = 5/2$, respectively).

In this paper, we present a set of methods for superfast inference for stationary Gaussian time series. They build upon the superfast Toeplitz solver of Ammar & Gragg (1988), the only algorithm of those mentioned above which provides the log-determinant as well. This algorithm has provably low overhead, crossing over Levinson's fast solver around $N = 260$. As the Ammar-Gragg algorithm is defined only for matrices of size $N = 2^K + 1$, we present a novel extension to arbitrary N with no additional overhead. Moreover, we present new superfast algorithms for the score and Hessian functions, thus providing tools for highly efficient inference for stationary Gaussian process via a broad array of frequentist and Bayesian methods. An implementation of our method is publicly available in the R/C++ library *SuperGauss*.

The remainder of the paper is organized as follows. In Section 2, we provide an overview of the Ammar-Gragg superfast Toeplitz solver and its generalization to arbitrary N. In Section 3 we present superfast gradient and Hessian algorithms for (8) and show how to extend these algorithms to profile likelihoods where the mean of $\boldsymbol{x}$ is given by a regression equation (13). In Section 4, we provide speed and stability comparisons with several fast and superfast algorithms. It is noted that for a variety of commonly-used models for statistical inference, numerical instability does not appear to be a practical issue. In Section 5, we present an application to Gaussian process factor analysis (18). Concluding remarks are offered in Section 6.

2 The Generalized Schur Algorithm for Toeplitz Systems

In this section, we present the generalized Schur Algorithm proposed in (Ammar & Gragg 1988). It is an algorithm that computes the inverse and determinant of size $N = 2^k + 1$ Toeplitz matrix in $\mathcal{O}(N \log^2 N)$ steps. The Ammar-Gragg algorithm, including many of the algorithms mentioned in the introduction (Bareiss 1969, Brent et al. 1980, Bitmead & Anderson 1980, Sexton 1982), are manifestations of Schur algorithm (Ammar 1996). The Ammar-Gragg divide-and-conquer version of Schur algorithm is referred by them as the "generalized" Schur algorithm. For consistency we keep this nomenclature, although "generalized Schur algorithm" is more widely accepted to refer to a fast algorithm for the Cholesky factorization of positive-definite structured matrices (Chandrasekaran & Sayed 1996). In order to give a clearer picture of the algorithm as a whole, technical details are omitted whenever possible. For an in-depth discussion of the mathematics underlying the Ammar-Gragg algorithm, the reader is referred to Ammar & Gragg (1987, 1988).

2.1 Gohberg-Semencul Formula

Let $\boldsymbol{V}$ denote an $N \times N$ symmetric positive definite Toeplitz matrix. From the definition of $\boldsymbol{V}$, it is clear that all entries can be obtained from the first row (or column). A seminal result of Gohberg and Semencul (Gohberg & Semencul 1972) is that the same is true for $\boldsymbol{V}^{-1}$. Namely, let $\boldsymbol{\delta} = [\delta_1, \delta_2, \ldots, \delta_N]$ denote the first row of $\boldsymbol{V}^{-1}$. Then

$$\boldsymbol{V}^{-1} = \frac{1}{\delta_1} \left(\boldsymbol{L}_1 \boldsymbol{L}_1' - \boldsymbol{L}_2 \boldsymbol{L}_2' \right), \tag{2}$$

where

$$\boldsymbol{L}_1 = \begin{bmatrix} \delta_1 & & & \\ \delta_2 & \delta_1 & & \\ \vdots & \ddots & \ddots & \\ \delta_N & \dots & \delta_2 & \delta_1 \end{bmatrix} \quad \text{and} \quad \boldsymbol{L}_2 = \begin{bmatrix} 0 & & & \\ \delta_N & 0 & & \\ \vdots & \ddots & \ddots & \\ \delta_2 & \dots & \delta_N & 0 \end{bmatrix}$$

are lower-triangular Toeplitz matrices (Gohberg & Semencul 1972).

The Gohberg-Semencul formula not only reduces the storage from a whole matrix $\mathcal{O}(N^2)$ to a vector $\mathcal{O}(N)$, but also simplifies the computation that involves solving the Toeplitz system $\boldsymbol{V}\boldsymbol{x} = \boldsymbol{b}$. While matrix-vector multiplication generally takes $\mathcal{O}(N^2)$ steps, the matrix product $\boldsymbol{V}^{-1}\boldsymbol{b}$ can be computed as successive matrix-vector products with triangular Toeplitz matrices. It is well known that each of these multiplications can be obtained in $\mathcal{O}(N \log N)$ steps using the fast Fourier transform (FFT) (Kailath & Sayed 1999). The exact algorithm is provided in Appendix A.

2.2 Generalized Schur Algorithm

For a size $N \times N$ Toeplitz covariance matrix $\boldsymbol{V}$ with first column $\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \dots, \gamma_N]$, consider a rational function

$$\phi_0(x) = \frac{\sum_{j=1}^{N-1} -\gamma_{j+1} x^j}{\sum_{j=1}^{N-1} \gamma_j x^j}.$$

The Schur algorithm is an iterative procedure that can generate a rational function $\phi_n(x)$ from $\phi_0(x)$ using the following linear functional transformation

$$\phi_{i+1}(x) = \frac{1}{x} \cdot \frac{\phi_i(x) - \mu_i}{1 - \mu_i \cdot \phi_i(x)}, \quad \mu_i = \phi_i(0), \quad i = 0, \dots, n-1, \tag{3}$$

where $\{\mu_i\}_{i=1}^n$ are the Schur parameters. For a given n-th order polynomial $\phi(x)$, let $\tilde{\phi}(x) = x^n \phi_n(1/x)$, which is also a polynomial of order at most n. Then the n-th step of Schur's algorithm can be expressed as $\phi_n(x) = T_n^{-1}(\phi_0(x))$, where the ϕ_n are rational functions and T_n has following representation

$$T_n(x) = \frac{\xi_n(x) + \tilde{\eta}_n(x) \cdot x}{\eta_n(x) + \tilde{\xi}_n(x) \cdot x},$$

where ξ_n and η_n are polynomials of degree $< n$ with coefficients depending on those of ϕ_0, such that $\phi_0(x) = T_n(\phi_n(x))$.

It was realized by Ammar & Gragg (1987) that the coefficients of the $N-1$ step of Schur's algorithm $T_{N-1} = \frac{\xi_{N-1}(x) + \tilde{\eta}_{N-1}(x) \cdot x}{\eta_{N-1}(x) + \tilde{\xi}_{N-1}(x) \cdot x}$ produce the first column of $\boldsymbol{V}^{-1}$ via

$$\boldsymbol{\delta} = [\delta_1, \delta_2, \dots, \delta_N] = \frac{1}{\sigma_{N-1}^2}([\eta_{N-1}^{(1)}, \dots, \eta_{N-1}^{(N-1)}, 0] + [0, \xi_{N-1}^{(1)}, \dots, \xi_{N-1}^{(N-1)}]),$$

where $\sigma_{N-1}^2 = \gamma_1 \prod_{j=1}^{N-1}(1 - \mu_j^2)$, $\eta_{N-1}^{(i)}$ and $\xi_{N-1}^{(i)}$ is the i-th coefficient of polynomials $\eta_{N-1}(x)$

and $\xi_{N-1}(x)$ respectively. And the determinant of the Toeplitz matrix is given by

$$|\boldsymbol{V}| = \gamma_1 \prod_{i=1}^{N-1} \sigma_i^2, \quad \sigma_i^2 = \gamma_1 \prod_{j=1}^{i} (1-\mu_j^2).$$

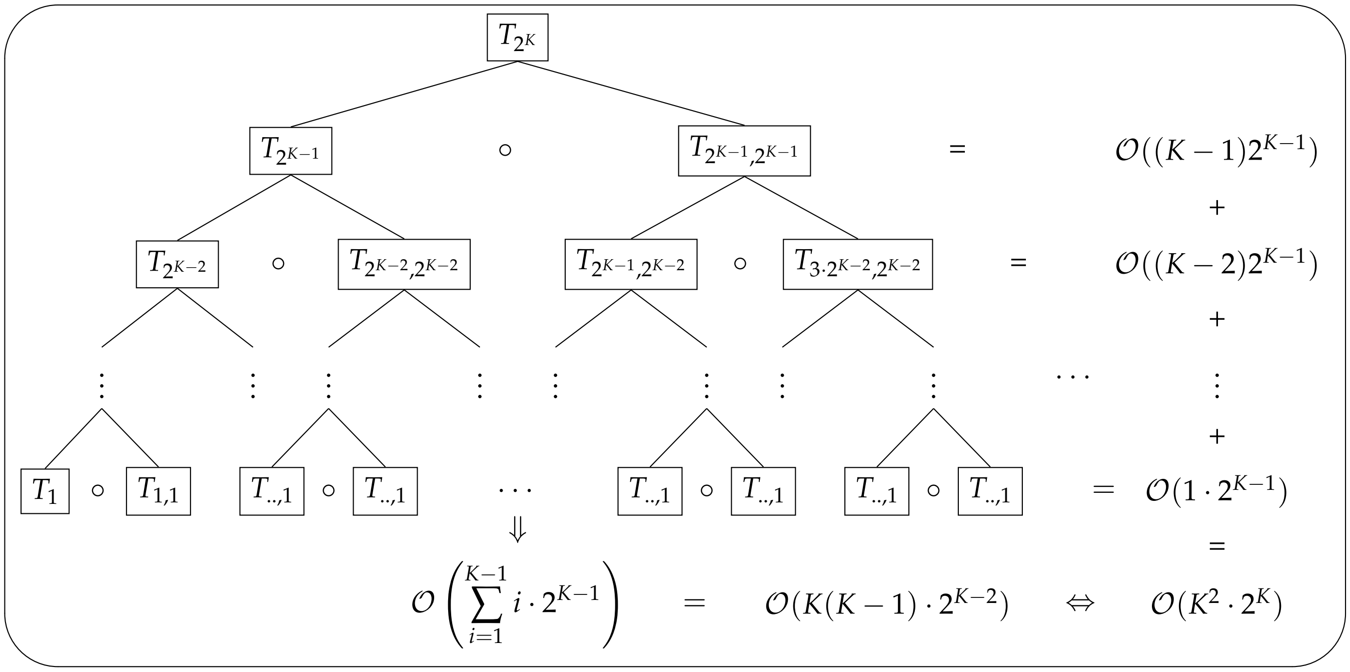
While the sequential calculation of $\{T_1, \ldots, T_{N-1}\}$ requires $O(N^2)$ operations, Ammar & Gragg realized a doubling procedure avoiding most of the intermediary T_n. That is, if for $\phi_0 = T_m(\phi_m)$, define $T_{m,m}(x)$ as the rational function obtained from applying m steps of Schur algorithm to ϕ_m, such that

$$\phi_{2m} = T_{m,m}^{-1}(\phi_m) = T_{m,m}^{-1}(T_m^{-1}(\phi_0)) = T_{2m}^{-1}(\phi_0),$$

i.e. $T_{2m} = T_m \circ T_{m,m}$. The merged transformation $T_{2m} = T_m \circ T_{m,m}$ is generated through polynomial multiplications:

$$\xi_{2m} = \tilde{\eta}_m \xi_{m,m} + \xi_m \eta_{m,m}, \quad \eta_{2m} = \tilde{\xi}_m \xi_{m,m} + \eta_m \eta_{m,m}, \quad \gamma_{2m} = \gamma_m + \lambda^m \gamma_{m,m}. \tag{4}$$

By calculating these multiplications via FFT, the computational cost of calculating T_{2m} is $O(m \log m)$. Thus, for $N = 2^K + 1$, the cost of going through the entire doubling procedure is $\mathcal{O}(K^2 \cdot 2^K) = \mathcal{O}(N \log^2 N)$, which is demonstrated in Figure 1.



T_{2^K}
$T_{2^{K-1}}$
$\circ$
$T_{2^{K-1},2^{K-1}}$
$=$
$\mathcal{O}((K-1)2^{K-1})$
$+$
$T_{2^{K-2}}$
$T_{2^{K-2},2^{K-2}}$
$T_{2^{K-1},2^{K-2}}$
$T_{3\cdot 2^{K-2},2^{K-2}}$
$\mathcal{O}((K-2)2^{K-1})$
T_1
$T_{1,1}$
$T_{\cdot\cdot,1}$
$\mathcal{O}(1 \cdot 2^{K-1})$
$\Downarrow$
$\mathcal{O}\left(\sum_{i=1}^{K-1} i \cdot 2^{K-1}\right) = \mathcal{O}(K(K-1) \cdot 2^{K-2}) \Leftrightarrow \mathcal{O}(K^2 \cdot 2^K)$

Figure 1: Tree diagram of the generalized Schur algorithm.

The generalized Schur algorithm for size $N = 2^K + 1$ is presented in Algorithm 1. The Ammar-Gragg version contains a number of technical accelerations, for which they prove that an exact operation count crosses over with Levinson's algorithm an $N = 2^8 + 1 = 257$ (Ammar & Gragg 1989). For ease of presentation we do not describe these technical improvements in

Algorithm 1, although we do implement them in the *SuperGauss* library.

Algorithm 1 Generalized Schur Algorithm for $M = 2^k$

```
1: function GSCHUR($\alpha_0^{(M)}, \beta_0^{(M)}$)
                                    ▷ $\alpha_0^{(M)}, \beta_0^{(M)}$: polynomials of degree $M-1$
2:     $\begin{cases} \begin{bmatrix} \xi_{0,1} \\ \eta_{0,1} \end{bmatrix} \leftarrow \begin{bmatrix} \frac{\alpha_0^{(1)}}{\beta_0^{(1)}} \\ 1 \end{bmatrix} \\ \mu_{0,1} \leftarrow \xi_{0,1} \end{cases}$
                                    ▷ $[T_{0,1} \leftarrow \text{GSCHUR}(\alpha_0^{(1)}, \beta_0^{(1)})]$
3:     for $m = 1, 2, 4, \ldots, M/2$ do
4:         $\begin{bmatrix} \alpha_n^{(m)} \\ \beta_n^{(m)} \end{bmatrix} \leftarrow \frac{1}{x^m} \times \begin{bmatrix} \eta_{0,m} & -\xi_{0,m} \\ -\tilde{\xi}_{0,m} & \tilde{\eta}_{0,m} \end{bmatrix} \begin{bmatrix} \alpha_0^{(2m)} \\ \beta_0^{(2m)} \end{bmatrix}$
                                    ▷ Truncate $\alpha_n^{(m)}, \beta_n^{(m)}$ to degree $m-1$
                                    ▷ $[\frac{\alpha_n^{(m)}}{\beta_n^{(m)}} = T_{0,m}^{-1}(\frac{\alpha_0^{(2m)}}{\beta_0^{(2m)}})]$
5:         $\{\xi_{m,m}, \eta_{m,m}, \mu_{m,m}\} \leftarrow \text{GSCHUR}(\alpha_n^{(m)}, \beta_n^{(m)})$    ▷ $[T_{m,m} \leftarrow \text{GSCHUR}(\alpha_n^{(m)}, \beta_n^{(m)})]$
6:         $\begin{cases} \begin{bmatrix} \xi_{0,2m} \\ \eta_{0,2m} \end{bmatrix} \leftarrow \begin{bmatrix} \tilde{\eta}_{0,m} & \xi_{0,m} \\ \tilde{\xi}_{0,m} & \eta_{0,m} \end{bmatrix} \begin{bmatrix} \xi_{m,m} \\ \eta_{m,m} \end{bmatrix} \\ \mu_{0,2m} \leftarrow (\mu_{0,m}, \mu_{m,m}) \end{cases}$
                                    ▷ $[T_{0,2m} = T_{0,m} \circ T_{m,m}]$
7:     end for
8:     return $\{\xi_{0,M}, \eta_{0,M}, \mu_{0,M}\}$
                                    ▷ $\mu_{0,M}$: vector of Schur parameters
9: end function
```

2.3 Extension of Generalized Schur Algorithm

Our extension stems from Ammar & Gragg's observation that for $n \neq m$, by defining $T_{m,n}$ as the n-steps of Schur's algorithm applied to ϕ_m, we have $T_{m+n} = T_m \circ T_{m,n}$, and

$$\xi_{n+m} = \tilde{\eta}_n \xi_{n,m} + \xi_n \eta_{n,m}, \quad \eta_{n+m} = \tilde{\xi}_n \xi_{n,m} + \eta_n \eta_{n,m}, \quad \gamma_{n+m} = \gamma_n + \lambda^n \gamma_{n,m}. \tag{5}$$

Assuming that N is the dimension of the target Toeplitz matrix and $M = N - 1$ is the step of the Schur algorithm, we can first decompose M into the summation of powers of 2:

$$M = \sum_{k=1}^{K} s_k, \quad s_1 < \ldots < s_K,$$

where s_k are powers of 2. For any positive integer M, the corresponding vector $\boldsymbol{s} = [s_1, \ldots, s_K]$ always exists and is unique. With this vector $\boldsymbol{s}$, we can decompose a size M rational function T_M into smaller ones with size being a power of 2

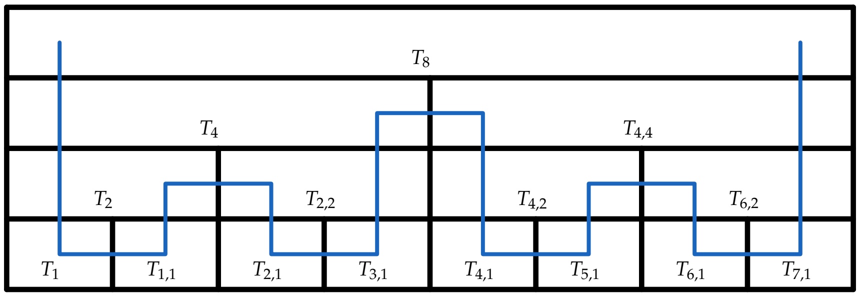
$$T_M = T_{s_1} \circ T_{c_1,s_2} \circ \ldots \circ T_{c_{K-1},s_K}, \quad c_k = \sum_{j=1}^{k} s_j \text{ for } k = 1, \ldots, K-1,$$

where each T_{c_{k-1},s_k} can be directly computed using the original generalized Schur algorithm with its input $\phi_{c_{k-1}}$ provided. We thus propose to calculate the coefficients of T_M by merging pieces of various sizes obtained from the original algorithm for powers of 2. The exact steps are given by Algorithm 2.

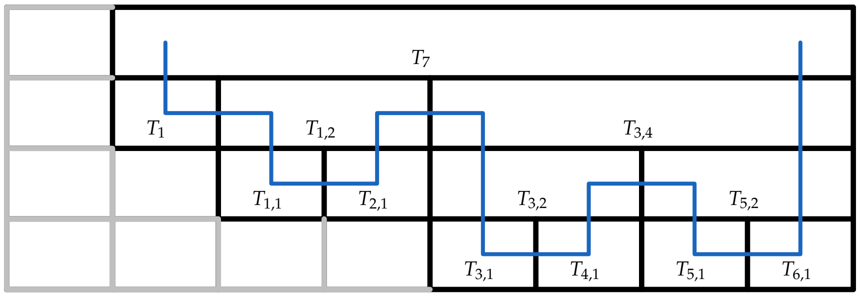
To better explain this procedure, we demonstrate how the extended generalized Schur algorithm solves an $M = 7$ system in Figure 22(b), where

$$T_7 = T_1 \circ T_{1,2} \circ T_{3,4},$$

and compare it with the original generalized Schur algorithm for $M = 8$ in Figure 22(a).



(2(a)) Demonstration of the generalized Schur Algorithm when $M = 8$.



(2(b)) Demonstration of the generalized Schur Algorithm when $M = 7$.

Different rectangles stand for various transformations T, where the width of rectangles is the size of the transformation. The blue line stands for the sequence of mergence steps.

3 Inference for Stationary Gaussian Processes

In this section we present superfast algorithms for the log-likelihood and its derivatives for a general family of Gaussian observations with Toeplitz covariance structure. In addition, we

Algorithm 2 Generalized Schur Algorithm for arbitrary M

```
1:  function GSCHUR2($\alpha_0^{(M)}, \beta_0^{(M)}$)
2:      $s = [s_1, s_2, \dots, s_k]$, where sum$(s) = M$, $s_i$ are power of 2 in ascending order.
3:      $m = s_1$
4:      $\{\xi_{0,m}, \eta_{0,m}, \mu_{0,m}\} \leftarrow$ GSCHUR($\alpha_0^{(m)}, \beta_0^{(m)}$)
                                                    ▷ [$T_{0,m} \leftarrow$ GSCHUR($\alpha_0^{(m)}, \beta_0^{(m)}$)]
5:      (END if $k == 1$)
6:      for $n = 2, \dots, k$ do
7:          $\begin{bmatrix} \alpha_m^{(s_n)} \\ \beta_m^{(s_n)} \end{bmatrix} \leftarrow \frac{1}{x^m} \times \begin{bmatrix} \eta_{0,m} & -\xi_{0,m} \\ -\tilde{\xi}_{0,m} & \tilde{\eta}_{0,m} \end{bmatrix} \begin{bmatrix} \alpha_0^{(m+s_n)} \\ \beta_0^{(m+s_n)} \end{bmatrix}$
                                                    ▷ [$\frac{\alpha_m^{(s_n)}}{\beta_{s_n}^{(m)}} = T_{0,m}^{-1}(\frac{\alpha_0^{(m+s_n)}}{\beta_0^{(m+s_n)}})$]
8:          $\{\xi_{m,s_n}, \eta_{m,s_n}, \mu_{m,s_n}\} \leftarrow$ GSCHUR($\alpha_m^{(s_n)}, \beta_m^{(s_n)}$)
                                                    ▷ [$T_{m,s_n} \leftarrow$ GSCHUR($\alpha_m^{(s_n)}, \beta_m^{(s_n)}$)]
9:          $\begin{cases} \begin{bmatrix} \xi_{0,m+s_n} \\ \eta_{0,m+s_n} \end{bmatrix} \leftarrow \begin{bmatrix} \tilde{\eta}_{0,m} & \xi_{0,m} \\ \tilde{\xi}_{0,m} & \eta_{0,m} \end{bmatrix} \begin{bmatrix} \xi_{m,s_n} \\ \eta_{m,s_n} \end{bmatrix} \\ \mu_{0,m+s_n} \leftarrow (\mu_{0,m}, \mu_{m,s_n}) \end{cases}$
                                                    ▷ [$T_{0,m+s_n} = T_{0,m} \circ T_{m,s_n}$]
10:         $m = m + s_n$
11:     end for
12:     return $\{\xi_{0,N}, \eta_{0,N}, \mu_{0,N}\}$
13: end function
```

show how to extend these algorithms to profile likelihood when the mean process of target time series can be expressed in forms of a regression equation.

Let $\boldsymbol{X}(t) = \boldsymbol{\mu_\theta}(t) + \boldsymbol{Z_\theta}(t)$ be a d-dimensional stochastic process where $\boldsymbol{Z_\theta}(t) = \begin{bmatrix} Z_1(t) & \cdots & Z_d(t) \end{bmatrix}$, $Z_i(t)$ is a mean-zero Gaussian process with separable stationary covariance structure, such that

$$\text{cov}(Z_i(t), Z_j(s)) = \boldsymbol{\Sigma}_{\boldsymbol{\theta}ij} \cdot \gamma_{\boldsymbol{\theta}}(|t-s|),$$

where $\boldsymbol{\Sigma_\theta}$ is a size $d \times d$ covariance matrix.

For equally spaced observations $[\boldsymbol{X}_1, \dots, \boldsymbol{X}_N], \boldsymbol{X}_n = \boldsymbol{X}(n \cdot \Delta t)$, the matrix $\boldsymbol{X}_{N \times d} = \begin{bmatrix} \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_N \end{bmatrix}$ has a matrix-normal distribution

$$\boldsymbol{X} \sim \text{MatNorm}(\boldsymbol{\mu_\theta}, \boldsymbol{V_\theta}, \boldsymbol{\Sigma_\theta}), \tag{6}$$

where $\boldsymbol{V}_{\boldsymbol{\theta}}$ is the among-column covariance matrix with elements $\boldsymbol{V}_{\boldsymbol{\theta}ij} = \gamma_{\boldsymbol{\theta}}(|i-j|\Delta t)$, $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ is the among-row covariance. Its vectorized form $\mathrm{vec}(\boldsymbol{X})$ follows a multivariate normal distribution

$$\mathrm{vec}(\boldsymbol{X}) \sim \mathcal{N}(\mathrm{vec}(\boldsymbol{\mu}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}), \tag{7}$$

where $\mathrm{vec}(\boldsymbol{\mu})$ is the vectorized form of the mean process $\boldsymbol{\mu}_{\boldsymbol{\theta}}$, and $\boldsymbol{\Sigma}_{\boldsymbol{\theta}} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}$ is the Kronecker product between $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ and $\boldsymbol{V}_{\boldsymbol{\theta}}$ and is a matrix of size $Nd \times Nd$.

In either a frequentist or Bayesian inference, the estimation of parameters $\boldsymbol{\theta}$ involves repeated evaluation of the log-likelihood

$$\ell(\boldsymbol{\theta} \mid \boldsymbol{X}) = -\frac{1}{2}\mathrm{tr}\left\{\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X}-\boldsymbol{\mu}_{\boldsymbol{\theta}})'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X}-\boldsymbol{\mu}_{\boldsymbol{\theta}})\right\} - \frac{d}{2}\log|\boldsymbol{V}_{\boldsymbol{\theta}}| - \frac{N}{2}\log|\boldsymbol{\Sigma}_{\boldsymbol{\theta}}|, \tag{8}$$

which requires the inverse and determinant of a size N covariance matrix $\boldsymbol{V}_{\boldsymbol{\theta}}$. With the extended generalized Schur algorithm, both $\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}$ and determinant $|\boldsymbol{V}_{\boldsymbol{\theta}}|$ can be easily computed in superfast steps.

3.1 Superfast Computation of the Gradient

To estimate model parameters $\boldsymbol{\theta}$, one popular method is to find the maximum of the likelihood $\ell(\boldsymbol{\theta} \mid \boldsymbol{X})$ (8). Optimization methods that maximize the likelihood typically require first order, even second order derivatives. In this section, we show that the first order derivative of $\ell(\boldsymbol{\theta} \mid \boldsymbol{X})$ with respect to parameter $\theta_i \in \boldsymbol{\theta}$ can also be computed within $\mathcal{O}(N\log^2 N)$ steps.

The first derivative of the log-likelihood with respect to parameter θ_i consists of the five parts

$$\frac{\partial}{\partial\theta_i}\ell(\boldsymbol{\theta} \mid \boldsymbol{X}) = -\frac{1}{2}\mathrm{tr}\{\underbrace{\boldsymbol{\Omega}_{\boldsymbol{\theta}}\boldsymbol{Z}_{\boldsymbol{\theta}}'\boldsymbol{\zeta}_i\boldsymbol{Z}_{\boldsymbol{\theta}}}_{A(\boldsymbol{\theta})} + \underbrace{2\boldsymbol{\Omega}_{\boldsymbol{\theta}}\boldsymbol{Z}_i'\boldsymbol{\zeta}_{\boldsymbol{\theta}}\boldsymbol{Z}_{\boldsymbol{\theta}}}_{B(\boldsymbol{\theta})} + \underbrace{\boldsymbol{\Omega}_i\boldsymbol{Z}_{\boldsymbol{\theta}}'\boldsymbol{\zeta}_{\boldsymbol{\theta}}\boldsymbol{Z}_{\boldsymbol{\theta}}}_{C(\boldsymbol{\theta})}\} - \frac{d}{2}\mathrm{tr}\{\underbrace{\boldsymbol{\zeta}_{\boldsymbol{\theta}}\boldsymbol{V}_i}_{D(\boldsymbol{\theta})}\} - \frac{N}{2}\mathrm{tr}\{\underbrace{\boldsymbol{\Omega}_{\boldsymbol{\theta}}\boldsymbol{\Sigma}_i}_{E(\boldsymbol{\theta})}\},$$

where

$$\boldsymbol{Z}_{\boldsymbol{\theta}} = \boldsymbol{X} - \boldsymbol{\mu}_{\boldsymbol{\theta}}, \quad \boldsymbol{Z}_i = \frac{\partial \boldsymbol{Z}_{\boldsymbol{\theta}}}{\partial\theta_i}, \quad \boldsymbol{\Sigma}_i = \frac{\partial \boldsymbol{\Sigma}_{\boldsymbol{\theta}}}{\partial\theta_i}, \quad \boldsymbol{V}_i = \frac{\partial \boldsymbol{V}_{\boldsymbol{\theta}}}{\partial\theta_i}, \quad \boldsymbol{\Omega}_{\boldsymbol{\theta}} = \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1},$$
$$\boldsymbol{\Omega}_i = \frac{\partial \boldsymbol{\Omega}_{\boldsymbol{\theta}}}{\partial\theta_i} = -\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\boldsymbol{\Sigma}_i\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}, \quad \boldsymbol{\zeta}_{\boldsymbol{\theta}} = \boldsymbol{V}_{\boldsymbol{\theta}}^{-1}, \quad \boldsymbol{\zeta}_i = \frac{\partial \boldsymbol{\zeta}_{\boldsymbol{\theta}}}{\partial\theta_i} = -\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}$$

are the partial derivatives, and $\boldsymbol{V}_{\boldsymbol{\theta}}$'s partial derivative $\boldsymbol{V}_i$ is still a Toeplitz matrix.

With $\boldsymbol{\zeta}_{\boldsymbol{\theta}}$ given in terms of the Gohberg-Semencul formula (2), the computation of part $A(\boldsymbol{\theta})$, $B(\boldsymbol{\theta})$ and $C(\boldsymbol{\theta})$ only involves the multiplication between a Toeplitz matrix or its inverse ($\boldsymbol{V}_{\boldsymbol{\theta}}$, $\boldsymbol{\zeta}_{\boldsymbol{\theta}}$ or $\boldsymbol{V}_i$) and a size $N \times d$ matrix ($\boldsymbol{Z}_{\boldsymbol{\theta}}$ or $\boldsymbol{Z}_i$), which only costs $\mathcal{O}(N\log N)$ steps (since $d \ll N$ in applications, we typically ignore d when examining the complexity). As for the remaining terms, part $E(\boldsymbol{\theta})$ requires the inversion and multiplication of a size $d \times d$ matrix and takes $\mathcal{O}(d^3)$ steps, while the computation of $D(\boldsymbol{\theta})$ is non-trivial. The direct computation of part $\mathrm{tr}\{\boldsymbol{\zeta}_{\boldsymbol{\theta}}\boldsymbol{V}_i\}$ takes $\mathcal{O}(N^2)$ steps, and we here demonstrate how to obtain this term in $\mathcal{O}(N\log N)$ steps.

A Toeplitz covariance matrix $\boldsymbol{V}_i$ with first row $\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \ldots, \gamma_N]$ has displacement rank

2 (Kailath et al. 1979) and can be written as

$$\boldsymbol{V}_i = \frac{1}{\gamma_1}[\boldsymbol{U}_1\boldsymbol{U}_1' - \boldsymbol{U}_2\boldsymbol{U}_2'], \tag{9}$$

where $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ are upper triangular Toeplitz matrices with first row being $[\gamma_1, \dots, \gamma_N]$ and $[0, \gamma_2, \dots, \gamma_N]$ respectively. Combining this with the Gohberg-Semencul representation of $\boldsymbol{\zeta} = \boldsymbol{V}^{-1} = \frac{1}{\delta_1}[\boldsymbol{L}_1\boldsymbol{L}_1' - \boldsymbol{L}_2\boldsymbol{L}_2']$, we have that

$$\begin{aligned}
\mathrm{tr}\{\boldsymbol{\zeta}\boldsymbol{V}_i\} &= \frac{1}{\delta_1\gamma_1}\mathrm{tr}\{[\boldsymbol{L}_1\boldsymbol{L}_1' - \boldsymbol{L}_2\boldsymbol{L}_2'][\boldsymbol{U}_1\boldsymbol{U}_1' - \boldsymbol{U}_2\boldsymbol{U}_2']\} \\
&= \frac{1}{\delta_1\gamma_1}\mathrm{tr}\{\boldsymbol{L}_1\boldsymbol{L}_1'\boldsymbol{U}_1\boldsymbol{U}_1' - \boldsymbol{L}_2\boldsymbol{L}_2'\boldsymbol{U}_1\boldsymbol{U}_1' - \boldsymbol{L}_1\boldsymbol{L}_1'\boldsymbol{U}_2\boldsymbol{U}_2' + \boldsymbol{L}_2\boldsymbol{L}_2'\boldsymbol{U}_2\boldsymbol{U}_2'\} \\
&= \frac{1}{\delta_1\gamma_1}\left(\mathrm{tr}\{\boldsymbol{U}_1'\boldsymbol{L}_1\boldsymbol{L}_1'\boldsymbol{U}_1\} - \mathrm{tr}\{\boldsymbol{U}_1'\boldsymbol{L}_2\boldsymbol{L}_2'\boldsymbol{U}_1\} - \mathrm{tr}\{\boldsymbol{U}_2'\boldsymbol{L}_1\boldsymbol{L}_1'\boldsymbol{U}_2\} + \mathrm{tr}\{\boldsymbol{U}_2'\boldsymbol{L}_2\boldsymbol{L}_2'\boldsymbol{U}_2\}\right).
\end{aligned} \tag{10}$$

Since

$$\boldsymbol{A}_{ij} = \boldsymbol{U}_i'\boldsymbol{L}_j$$

is the product of two lower triangular Toeplitz matrices, we can verify that $\boldsymbol{A}_{ij}$ is also a lower triangular Toeplitz matrix that can be computed in $\mathcal{O}(N\log N)$ steps. The trace of $\boldsymbol{A}_{ij}\boldsymbol{A}_{ij}'$ can be determined in $\mathcal{O}(N)$ steps with following equation

$$\mathrm{tr}\{\boldsymbol{A}_{ij}\boldsymbol{A}_{ij}'\} = \sum_{i=1}^{N}\sum_{j=1}^{i} a_j^2 = \sum_{j=1}^{N}(n-j+1)a_j^2, \tag{11}$$

where $\boldsymbol{a}_{ij} = [a_1, \dots, a_N]$ is the first column of $\boldsymbol{A}_{ij}$. Therefore the calculation of $\mathrm{tr}\{\boldsymbol{A}_{ij}\boldsymbol{A}_{ij}'\}$ is $\mathcal{O}(N\log N)$. All these together leads to the $\mathcal{O}(N\log N)$ complexity of $\mathrm{tr}\{\boldsymbol{\zeta}\boldsymbol{V}_i\}$. In conclusion, the evaluation of the gradient of likelihood $\frac{\partial}{\partial\theta_i}\ell(\boldsymbol{\theta} \mid \boldsymbol{X})$ is superfast.

3.2 Automatic Differentiation

Given the dimension of unknown parameters p, the present algorithm for the gradient $\frac{\partial}{\partial\theta_i}\ell(\boldsymbol{\theta} \mid \boldsymbol{X})$ scales as $\mathcal{O}(p \cdot N\log^2 N)$, which is suitable for $p \ll N$. However, for $p \sim N$ repeated calculation of the trace formula above breaks the superfast scaling. This is an important restriction for applications of automatic differentiation, where derivatives with respect to each element of the autocorrelation function $\boldsymbol{\theta} = \boldsymbol{\gamma} = [\gamma_1, \dots, \gamma_N]$ are desired. A superfast algorithm for this situation is presented here.

Consider the derivative of $\boldsymbol{V}$ with respect to γ_i, a sparse symmetric Toeplitz matrix:

$$\frac{\partial \boldsymbol{V}}{\partial \gamma_i} = \boldsymbol{I}^{(i)} = \begin{bmatrix} \overbrace{0 & \cdots & 0}^{i-1} & 1 & \overbrace{0 & \cdots & 0}^{N-i} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 1 & 0 & \cdots & \cdots & 0 & 1 & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 & 0 \end{bmatrix}.$$

When $i =$, $\boldsymbol{I}^{(1)}$ is an identity matrix. For the derivative of likelihood (8) with respect to vector $\boldsymbol{\gamma}$, we have that $\frac{\partial}{\partial \boldsymbol{\gamma}} \ell(\boldsymbol{\theta} \mid \boldsymbol{X})$ is a vector with elements

$$\frac{\partial}{\partial \gamma_i} \ell(\boldsymbol{\theta} \mid \boldsymbol{X}) = -\frac{1}{2} \text{tr}\{\boldsymbol{\Omega} \boldsymbol{Z}' \boldsymbol{\zeta}_i \boldsymbol{Z}\} - \frac{d}{2} \cdot \text{tr}\{\boldsymbol{\zeta} \boldsymbol{I}^{(i)}\},$$

where $\boldsymbol{\zeta}_i = -\boldsymbol{\zeta} \boldsymbol{I}^{(i)} \boldsymbol{\zeta}$.

Recall that $\boldsymbol{\Omega} = \boldsymbol{\Sigma}^{-1} = \begin{bmatrix} \omega_{11} & \cdots & \omega_{1d} \\ \vdots & \ddots & \vdots \\ \omega_{1d} & \cdots & \omega_{dd} \end{bmatrix}$ is a $d \times d$ symmetric matrix and $\boldsymbol{Z} = \begin{bmatrix} \mathbf{z}_1 & \cdots & \mathbf{z}_d \end{bmatrix}$ is a $N \times d$ matrix, we have

$$\begin{aligned} \text{tr}\{\boldsymbol{\Omega} \boldsymbol{Z}' \boldsymbol{\zeta}_i \boldsymbol{Z}\} =& \text{tr}\{ \begin{bmatrix} \omega_{11} & \cdots & \omega_{1d} \\ \vdots & \ddots & \vdots \\ \omega_{d1} & \cdots & \omega_{dd} \end{bmatrix} \begin{bmatrix} \boldsymbol{Z}_1' \\ \vdots \\ \boldsymbol{Z}_d' \end{bmatrix} \boldsymbol{\zeta}_i \begin{bmatrix} \mathbf{z}_1 & \cdots & \mathbf{z}_d \end{bmatrix} \} \\ &= \sum_{n=1}^{d} \sum_{m=1}^{d} \omega_{nm} \cdot \boldsymbol{Z}_n' \boldsymbol{\zeta}_i \boldsymbol{Z}_m = -\sum_{n=1}^{d} \sum_{m=1}^{d} \omega_{nm} \cdot \boldsymbol{Z}_n' \boldsymbol{\zeta} \boldsymbol{I}^{(i)} \boldsymbol{\zeta} \boldsymbol{Z}_m \end{aligned}$$

For a vector $\boldsymbol{l}^{(n,m)} = [l_1, l_2, \ldots, l_N]$ of following form

$$\boldsymbol{l}^{(n,m)} = \boldsymbol{U}(\boldsymbol{a}_n) \times \boldsymbol{a}_m + \boldsymbol{U}(\boldsymbol{a}_m) \times \boldsymbol{a}_n$$

where $\boldsymbol{a}_n = \boldsymbol{\zeta} \boldsymbol{Z}_n$, $\boldsymbol{a}_m = \boldsymbol{\zeta} \boldsymbol{Z}_m$ and $\boldsymbol{U}(\boldsymbol{a}_n)$ is the upper triangular Toeplitz matrix with first row being $\boldsymbol{a}_n$, we can verify that $\boldsymbol{Z}_n' \boldsymbol{\zeta}_1 \boldsymbol{Z}_m = l_1/2$ and $\boldsymbol{Z}_n' \boldsymbol{\zeta}_i \boldsymbol{Z}_m = l_i$ for $i = 2, \ldots, N$. In other words, by putting $\tilde{\boldsymbol{l}}^{(n,m)} = [l_1/2, l_2, \ldots, l_N]$ we can obtain the vector of $\text{tr}\{\boldsymbol{\Omega} \boldsymbol{Z}' \boldsymbol{\zeta}_i \boldsymbol{Z}\}$ immediately

$$\begin{bmatrix} \text{tr}\{\boldsymbol{\Omega} \boldsymbol{Z}' \boldsymbol{\zeta}_1 \boldsymbol{Z}\} \\ \cdots \\ \text{tr}\{\boldsymbol{\Omega} \boldsymbol{Z}' \boldsymbol{\zeta}_N \boldsymbol{Z}\} \end{bmatrix} = -\sum_{n=1}^{d} \sum_{m=1}^{d} \omega_{nm} \cdot \tilde{\boldsymbol{l}}^{(n,m)}$$

Similarly, considering the Gohberg-Semencul formula (2) that constructs the $\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}$ from its first column $\boldsymbol{\delta} = [\delta_1, \dots, \delta_N]$, we can define a vector $\boldsymbol{v} = [v_1, v_2, \dots, v_N]$ as

$$\boldsymbol{v} = \frac{1}{\delta_1}\{ \begin{bmatrix} \delta_1 & \delta_2 & \delta_3 & \dots & \delta_N \\ 0 & \delta_1 & \delta_2 & \dots & \delta_{N-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \delta_1 \end{bmatrix} \cdot \begin{bmatrix} N \cdot \delta_1 \\ (N-1)\cdot\delta_2 \\ \vdots \\ 1 \cdot \delta_N \end{bmatrix} - \begin{bmatrix} 0 & \delta_N & \delta_{N-1} & \dots & \delta_2 \\ 0 & 0 & \delta_N & \dots & \delta_3 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ (N-1)\cdot\delta_N \\ \vdots \\ 1 \cdot \delta_2 \end{bmatrix} \}.$$

such that $\text{tr}\{\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{I}^{(1)}\} = v_1/2$ and $\text{tr}\{\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{I}^{(i)}\} = v_i$ for $i = 2, \dots, N$. That is to say

$$\begin{bmatrix} \text{tr}\{\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{I}^{(1)}\} \\ \dots \\ \text{tr}\{\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{I}^{(N)}\} \end{bmatrix} = \tilde{\boldsymbol{v}}$$

where $\tilde{\boldsymbol{v}} = [v_1/2, v_2, \dots, v_N]$.

In conclusion, we have that

$$\frac{\partial}{\partial\gamma}\ell(\boldsymbol{\theta} \mid \boldsymbol{X}) = \frac{1}{2}\sum_{n=1}^{d}\sum_{m=1}^{d} \omega_{nm} \cdot \tilde{\boldsymbol{l}}^{(n,m)} - \frac{d}{2}\tilde{\boldsymbol{v}}.$$

Since the vectors $\tilde{\boldsymbol{l}}$ and $\tilde{\boldsymbol{v}}$ can be computed in $\mathcal{O}(N \log N)$ steps with $\boldsymbol{\delta}$ provided, we can obtain the gradient vector $\frac{\partial}{\partial\gamma}\ell(\boldsymbol{\theta} \mid \boldsymbol{z})$ in superfast speed.

3.3 Superfast Computation of the Hessian matrix

The second derivative of the log-likelihood with respect to parameter $\theta_i, \theta_j \in \boldsymbol{\theta}$ consists of six terms:

$$\begin{aligned} \frac{\partial^2}{\partial\theta_i\partial\theta_j}\ell(\boldsymbol{\theta} \mid \boldsymbol{X}) = &-\frac{1}{2}\text{tr}\left\{\frac{\partial A(\boldsymbol{\theta})}{\partial\theta_j} + \frac{\partial B(\boldsymbol{\theta})}{\partial\theta_j} + \frac{\partial C(\boldsymbol{\theta})}{\partial\theta_j}\right\} - \frac{N}{2}\text{tr}\left\{\boldsymbol{\Omega}_\theta\boldsymbol{\Sigma}_{ij} - \boldsymbol{\Omega}_\theta\boldsymbol{\Sigma}_j\boldsymbol{\Omega}_\theta\boldsymbol{\Sigma}_i\right\} \\ &- \frac{d}{2}\text{tr}\left\{\boldsymbol{\zeta}_\theta\boldsymbol{V}_{ij}\right\} + \frac{d}{2}\text{tr}\left\{\boldsymbol{\zeta}_\theta\boldsymbol{V}_j\boldsymbol{\zeta}_\theta\boldsymbol{V}_i\right\}, \end{aligned} \tag{12}$$

where

$$\begin{aligned} \frac{\partial A(\boldsymbol{\theta})}{\partial\theta_j} &= \boldsymbol{\Omega}_j\boldsymbol{Z}_\theta'\boldsymbol{\zeta}_i\boldsymbol{Z}_\theta + \boldsymbol{\Omega}_\theta\boldsymbol{Z}_j'\boldsymbol{\zeta}_i\boldsymbol{Z}_\theta + \boldsymbol{\Omega}_\theta\boldsymbol{Z}_\theta'\boldsymbol{\zeta}_{ij}\boldsymbol{Z}_\theta + \boldsymbol{\Omega}_\theta\boldsymbol{Z}_\theta'\boldsymbol{\zeta}_i\boldsymbol{Z}_j \\ \frac{\partial B(\boldsymbol{\theta})}{\partial\theta_j} &= 2\left(\boldsymbol{\Omega}_j\boldsymbol{Z}_i'\boldsymbol{\zeta}_\theta\boldsymbol{Z}_\theta + \boldsymbol{\Omega}_\theta\boldsymbol{Z}_{ij}'\boldsymbol{\zeta}_\theta\boldsymbol{Z}_\theta + \boldsymbol{\Omega}\boldsymbol{Z}_j'\boldsymbol{\zeta}_j\boldsymbol{Z}_\theta + \boldsymbol{\Omega}_\theta\boldsymbol{Z}_i'\boldsymbol{\zeta}_\theta\boldsymbol{Z}_j\right) \\ \frac{\partial C(\boldsymbol{\theta})}{\partial\theta_j} &= \boldsymbol{\Omega}_{ij}\boldsymbol{Z}_\theta'\boldsymbol{\zeta}_\theta\boldsymbol{Z}_\theta + \boldsymbol{\Omega}_i\boldsymbol{Z}_\theta'\boldsymbol{\zeta}_\theta\boldsymbol{Z}_\theta + \boldsymbol{\Omega}_i\boldsymbol{Z}_\theta'\boldsymbol{\zeta}_j\boldsymbol{Z}_\theta + \boldsymbol{\Omega}_i\boldsymbol{Z}_\theta'\boldsymbol{\zeta}_\theta\boldsymbol{Z}_j, \end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{Z}_{ij} &= \frac{\partial^2}{\partial\theta_i\partial\theta_j}\boldsymbol{Z}_{\boldsymbol{\theta}}, \quad \boldsymbol{V}_{ij} = \frac{\partial^2}{\partial\theta_i\partial\theta_j}\boldsymbol{V}_{\boldsymbol{\theta}}, \quad \boldsymbol{\Sigma}_{ij} = \frac{\partial^2}{\partial\theta_i\partial\theta_j}\boldsymbol{\Sigma}_{\boldsymbol{\theta}}, \\
\boldsymbol{\Omega}_{ij} &= \frac{\partial^2}{\partial\theta_i\partial\theta_j}\boldsymbol{\Omega}_{\boldsymbol{\theta}} = \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\boldsymbol{\Sigma}_j\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\boldsymbol{\Sigma}_i\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} + \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\boldsymbol{\Sigma}_i\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\boldsymbol{\Sigma}_j\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} - \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1}\boldsymbol{\Sigma}_{ij}\boldsymbol{\Sigma}_{\boldsymbol{\theta}}^{-1} \\
\boldsymbol{\zeta}_{ij} &= \frac{\partial^2}{\partial\theta_i\partial\theta_j}\boldsymbol{\zeta}_{\boldsymbol{\theta}} = \boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_j\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1} + \boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_j\boldsymbol{V}_{\boldsymbol{\theta}}^{-1} - \boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_{ij}\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}
\end{aligned}$$

are the second order partial derivatives, and $\boldsymbol{V}_{ij}$ is also a Toeplitz matrix.

The first three terms still consist of the multiplication between Toeplitz matrices or inverse ($\boldsymbol{V}_{\boldsymbol{\theta}}$, $\boldsymbol{\zeta}_{\boldsymbol{\theta}}$, $\boldsymbol{V}_i$, $\boldsymbol{V}_j$, $\boldsymbol{V}_{ij}$) and size $N \times d$ matrices ($\boldsymbol{Z}_{\boldsymbol{\theta}}$, $\boldsymbol{Z}_i$, $\boldsymbol{Z}_j$, $\boldsymbol{Z}_{ij}$), which takes $\mathcal{O}(N \log N)$ steps. The fourth term involves matrix computation of several size $d \times d$ matrices and is of complexity $\mathcal{O}(d^3)$. The fifth term can be computed in superfast steps using Equation (10). In the following we present the non-trivial superfast computation for the last part tr $\left\{\boldsymbol{\zeta}_{\boldsymbol{\theta}}\boldsymbol{V}_j\boldsymbol{\zeta}_{\boldsymbol{\theta}}\boldsymbol{V}_i\right\}$.

In addition to the chain rule, the partial derivative of $\boldsymbol{\zeta}_{\boldsymbol{\theta}}$ admits another computation via the Gohberg-Semencul formula

$$\boldsymbol{\zeta} = \boldsymbol{\zeta}(\boldsymbol{\delta}) = \frac{1}{\delta_1}\left[\boldsymbol{L}_1(\boldsymbol{\delta})\boldsymbol{L}_1(\boldsymbol{\delta})' - \boldsymbol{L}_2(\boldsymbol{\delta})\boldsymbol{L}_2(\boldsymbol{\delta})'\right],$$

where $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$ are simple permutations of $\boldsymbol{\delta}$. Therefore we have

$$\begin{aligned}
\boldsymbol{\zeta}_i = &-\frac{\delta_{1,(i)}}{\delta_1}\boldsymbol{\zeta} + \frac{1}{\delta_1}\left[\boldsymbol{L}_1(\boldsymbol{\delta}_{(i)})\boldsymbol{L}_1(\boldsymbol{\delta})' + \boldsymbol{L}_1(\boldsymbol{\delta})\boldsymbol{L}_1(\boldsymbol{\delta}_{(i)})' - \boldsymbol{L}_2(\boldsymbol{\delta}_{(i)})\boldsymbol{L}_2(\boldsymbol{\delta})' - \boldsymbol{L}_2(\boldsymbol{\delta})\boldsymbol{L}_2(\boldsymbol{\delta}_{(i)})'\right] \\
=&h(\boldsymbol{\delta}, \boldsymbol{\delta}_{(i)}),
\end{aligned}$$

where $\boldsymbol{\delta}_{(i)} = \frac{\partial}{\partial\theta_i}\boldsymbol{\delta}$, $\delta_{1,(i)}$ is the first element of $\boldsymbol{\delta}_{(i)}$ and $\boldsymbol{L}_1(\boldsymbol{\delta}_{(i)}), \boldsymbol{L}_2(\boldsymbol{\delta}_{(i)})$ are lower triangular Toeplitz matrices constructed from $\boldsymbol{\delta}_{(i)}$ in the same way as $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$. To obtain this vector $\boldsymbol{\delta}_{(i)}$, recall that $\boldsymbol{\delta}$ is the first column of $\boldsymbol{\zeta}$, such that

$$\boldsymbol{V}\boldsymbol{\delta} = \boldsymbol{e}_1, \qquad \boldsymbol{e}_1 = [1, 0, \ldots, 0]'.$$

Taking derivatives on both sides, we have

$$\boldsymbol{V}_i\boldsymbol{\delta} + \boldsymbol{V}\boldsymbol{\delta}_{(i)} = 0 \Rightarrow \boldsymbol{\delta}_{(i)} = -\boldsymbol{\zeta}\boldsymbol{V}_i\boldsymbol{\delta}.$$

With $\boldsymbol{\delta}$ computed, the matrix-vector product $\boldsymbol{V}_i\boldsymbol{\delta}$ can be obtained in $\mathcal{O}(N \log N)$ steps, after which $\boldsymbol{\delta}_{(i)} = -\boldsymbol{\zeta}\boldsymbol{V}_i\boldsymbol{\delta}$ can also be obtained in $\mathcal{O}(N \log N)$ steps by applying the Gohberg-Semencul decomposition (2).

Thus the computation of $\mathrm{tr}\{\boldsymbol{\zeta} \boldsymbol{V}_i \boldsymbol{\zeta} \boldsymbol{V}_j\} = -\mathrm{tr}\{\boldsymbol{\zeta}_i \boldsymbol{V}_j\}$ can be obtained from

$$\begin{aligned}\mathrm{tr}\{\boldsymbol{\zeta}_i \boldsymbol{V}_j\} =& \frac{1}{\delta_1}\mathrm{tr}\Big\{\Big[\boldsymbol{L}_1(\boldsymbol{\delta}_{(i)})\boldsymbol{L}_1(\boldsymbol{\delta})' + \boldsymbol{L}_1(\boldsymbol{\delta})\boldsymbol{L}_1(\boldsymbol{\delta}_{(i)})' - \boldsymbol{L}_2(\boldsymbol{\delta}_{(i)})\boldsymbol{L}_2(\boldsymbol{\delta})' - \boldsymbol{L}_2(\boldsymbol{\delta})\boldsymbol{L}_2(\boldsymbol{\delta}_{(i)})'\Big] \boldsymbol{V}_j\Big\} \\ & - \frac{\delta_{1,(i)}}{\delta_1}\mathrm{tr}\{\boldsymbol{\zeta} \boldsymbol{V}_j\}.\end{aligned}$$

With the decomposition formula (9) of $\boldsymbol{V}_j$ and the efficient computation of the trace of product between lower and upper Toeplitz matrices (11), we can compute $\mathrm{tr}\{\boldsymbol{\zeta} \boldsymbol{V}_i \boldsymbol{\zeta} \boldsymbol{V}_j\}$ in $\mathcal{O}(N \log N)$ steps. In conclusion, the computation of the Hessian matrix (12) is superfast $\mathcal{O}(N \log^2 N)$.

3.4 Profile Likelihood

In many applications, the mean function $E[\boldsymbol{X}(t)]$ is modeled via a regression equation

$$E[\boldsymbol{X}(t)] = \boldsymbol{\mu}_{\boldsymbol{\theta}}(t) = \sum_{i=1}^{p} \boldsymbol{\beta}_i g_i(t), \tag{13}$$

and a separable covariance structure

$$\mathrm{cov}(X_i(t), X_j(s)) = \boldsymbol{\Sigma}_{ij} \cdot \gamma_{\boldsymbol{\theta}}(|t-s|).$$

In these cases, the distribution of observation matrix $\boldsymbol{X}_{N\times d}$ is given by

$$\boldsymbol{X}_{N\times d} \sim \mathrm{MatNorm}(\boldsymbol{G}\boldsymbol{\beta}, \boldsymbol{\Sigma}, \boldsymbol{V}_{\boldsymbol{\theta}}),$$

where $\boldsymbol{G}_{N\times p} = [\boldsymbol{g}_1, \cdots, \boldsymbol{g}_p]$ is the observation matrix of regression processes, $\boldsymbol{g}_i = [g_i(\Delta t), \ldots, g_i(N \cdot \Delta t)]$, $\boldsymbol{\beta}_{p\times d} = \begin{bmatrix}\boldsymbol{\beta}_1 \\ \vdots \\ \boldsymbol{\beta}_p\end{bmatrix}$ is the vector of coefficients of the regression process, $\boldsymbol{\Sigma}_{d\times d}$ is the covariance matrix, and $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_m\}$ is the parameter set that determines the covariance matrix $\boldsymbol{V}_{\boldsymbol{\theta}}$.

In this case, for fixed $\boldsymbol{\theta}$ the condition maximum likelihood estimates

$$(\hat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}, \hat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}) = \underset{\boldsymbol{\beta},\boldsymbol{\Sigma}}{\arg\max}\left\{-\frac{d}{2}\log|\boldsymbol{V}_{\boldsymbol{\theta}}| - \frac{N}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}\mathrm{tr}[\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}-\boldsymbol{G}\boldsymbol{\beta})'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X}-\boldsymbol{G}\boldsymbol{\beta})]\right\}$$

are given by (Jones et al. 1998, Lysy et al. 2016) as

$$\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}} = (\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G})^{-1}\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X}, \quad \widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}} = \frac{1}{N}(\boldsymbol{X}-\boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X}-\boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}),$$

leading to a profile likelihood function

$$\ell_{\mathrm{prof}}(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{G}) = -\frac{Nd}{2}\log(2\pi) - \frac{d}{2}\log|\boldsymbol{V}_{\boldsymbol{\theta}}| - \frac{N}{2}\log|\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}| - \frac{N}{2}. \tag{14}$$

Optimization over this function greatly reduces the dimensionality. Gradient and Hessian algorithms for the profile likelihood are provided in Appendix B.

4 Numerical Experiments

For the purpose of efficient inference of stationary Gaussian processes, we implemented the extended generalized Schur algorithm in **C++** in forms of a head-only library called *SuperGauss*, where the Fast Fourier transformation is implemented using the *fftw* library (Frigo & Johnson 2005). An **R** library, *SuperGauss* (Ling & Lysy 2020) is also provided.

To compare the computation speed of the generalized Schur algorithm, we implemented the Levinson's algorithm in C++ and use the *fftw* library for the FFT computations. The theoretical cross-over point between the generalized Schur algorithm and Levinson algorithm is $N = 257$ (Ammar & Gragg 1989). In our implementation the real cross-over point is around $N = 260$. We also look into the performance of the Fortran implementation of the Hierarchical Structured Solver (HSS) algorithm of Xia et al. (2012), Xi et al. (2014), an $\mathcal{O}(N \log^2 N)$ stable algorithm that works for asymmetric and complex Toeplitz matrices as well. In Figure 3 we present the computation time for the extended Generalized Schur algorithm and Levinson algorithm for matrices of sizes ranging from 100×100 to $10^5 \times 10^5$. To better measure the computation time, we repeat each trial 100 times and record the average value.

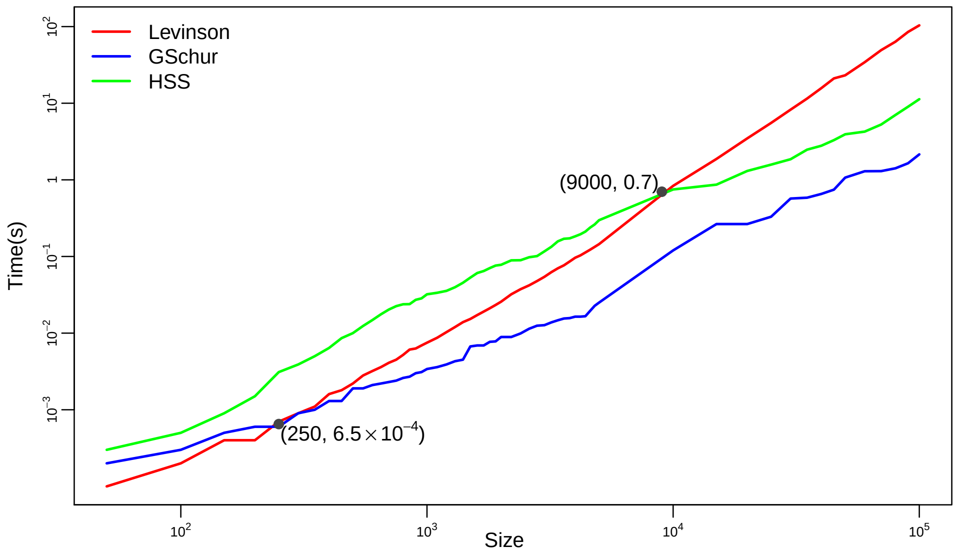


Figure 3: Average time for solving Toeplitz systems $\boldsymbol{V} \cdot \boldsymbol{x} = \boldsymbol{y}$ using different algorithms. A size N Toeplitz system $\boldsymbol{V} \cdot \boldsymbol{x} = \boldsymbol{y}$ comes with an $N \times N$ Toeplitz covariance matrix $\boldsymbol{V}$ and a length-N vector $\boldsymbol{y}$. The first column of $\boldsymbol{V}$ is the ACF of a fractional Gaussian noise process (15) with $\alpha = 0.8$. The vector $\boldsymbol{y}$ has elements $y_i \overset{\text{iid}}{\sim} \mathcal{N}(0, 1)$.

4.1 Numerical Stability Experiments

The main concern about the generalized Schur algorithm is its numerical stability. According to Stewart & Van Dooren (1997), Chandrasekaran & Sayed (1998), the Schur algorithm for

Toeplitz matrix inversion is stable, and extensive numerical experiments on the generalized Schur algorithm display that its growth rates of computation errors are comparable with those of the Szegö recursions, which is equivalent with algorithms like the Levinson algorithm (Ammar & Gragg 1989). However, numerical methods that are based on explicit inversions are usually unstable (Higham 2002), which is exactly the case with the generalized Schur algorithm. In extreme cases where ill-conditioned Toeplitz covariance matrices are generated, the generalized Schur algorithm has worse performance than the Levinson algorithm (Stewart 2003, Chen et al. 2006). In this section, some numerical experiments are conducted to examine the performance of the generalized Schur algorithm with respect to various kinds of stationary Gaussian processes.

For a Toeplitz covariance matrix $\boldsymbol{V}$, its condition number is defined as

$$\kappa(\boldsymbol{V}) = ||\boldsymbol{V}||_p \cdot ||\boldsymbol{V}^{-1}||_p,$$

where matrix norm p can be arbitrary. In this paper, we choose $p = \infty$, whose corresponding matrix norm $||\boldsymbol{V}||_\infty = \max_{1\le i\le N} \sum_{j=1}^{N} |\boldsymbol{V}_{ij}|$ is the maximum absolute row sum of the matrix, and $||\boldsymbol{x}||_\infty = \max_{1\le j\le N} |\boldsymbol{x}_i|$ is the maximum element of the vector. The condition number is the index of the singularity of matrices. A matrix with a high condition number κ is viewed as ill-conditioned and for singular matrices, its $\kappa = \infty$.

Given an $N \times N$ Toeplitz covariance matrix $\boldsymbol{V}$, its measurement error is defined and estimated in the following steps:

1. Simulate a length N vector $\boldsymbol{y} = [y_1, \ldots, y_N]$ whose elements are i.i.d. $y_i \sim \mathcal{N}(0, 1)$.

2. Solve the Toeplitz system $\boldsymbol{V} \cdot \boldsymbol{x} = \boldsymbol{y}$ and obtain the estimation $\hat{\boldsymbol{x}}$

3. Check the measurement error $r(\boldsymbol{V}, \hat{\boldsymbol{x}}, \boldsymbol{y}) = \frac{||\boldsymbol{V}\hat{\boldsymbol{x}}-\boldsymbol{y}||}{||\boldsymbol{V}||\cdot||\hat{\boldsymbol{x}}||+||\boldsymbol{y}||}$.

where the matrix supremum norm is applied here.

4.1.1 Stewart's Example

We can generate an arbitrary $N \times N$ Toeplitz covariance matrix $\boldsymbol{V}$ (with first element $\boldsymbol{V}_{1,1} = 1$) for given Schur parameters $\{\mu_k\}_{k=1}^{N-1}$ using Szegö recurrence (Ammar & Gragg 1987). In a numerical experiment (Stewart 2003), ill-conditioned Toeplitz covariance matrices are generated by manipulating the Schur parameters $\{\mu_k\}_{k=1}^{N-1}$ in the particular way explained in Figure 4.

In Table 1 we demonstrate the estimation errors of 4 different algorithms: the generalized Schur algorithm, Levinson algorithm, Cholesky decomposition, HSS algorithm and an $\mathcal{O}(N \log^{5/2} N)$ preconditioned conjugate gradient algorithm (PCG) developed by Chen et al. (2006) for long-memory processes. In both experimental setups, $\kappa(\boldsymbol{V})$ grows rapidly as matrix size N increases. For the generalized Schur algorithm, its relative error r_{GSchur} grows at a similar rate as $\kappa(\boldsymbol{V})$. On the contrary, the result of LTZ and Cholesky algorithm is accurate and robust against the conditions of $\boldsymbol{V}$. As for the PCG method, its relative error is stable but constantly large.

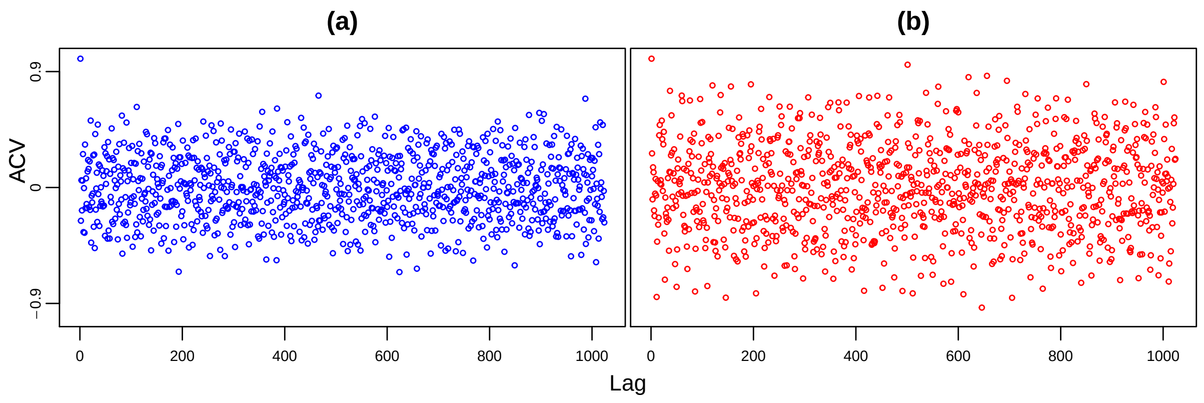


Figure 4: Auto-covariance of Stewart's experiments
(a) Experiment 1, where $\mu_i \sim \text{Unif}(-0.5, 0.5), i = 1, 2, \ldots, N-1$.
(b) Experiment 2, where $\mu_{10} = 1 - 10^{-6}$, $\mu_{15} = -0.99$ and $\mu_i \sim \text{Unif}(-0.3, 0.3)$ for remaining i.

Table 1: Measurement errors of different algorithms in Stewart's examples.
Experiment (a), where $\mu_i \sim \text{Unif}(-0.5, 0.5), i = 1, \cdots, N-1$.
Experiment (b), where $\mu_i \sim \text{Unif}(-0.3, 0.3), i \in \{1, 2, \ldots, N-1\} \setminus \{10, 15\}$, $\mu_{10} = 1 - 10^{-6}$, $\mu_{15} = -0.99$.

	Experiment (a)			Experiment (b)	
	$N = 64$	$N = 128$	$N = 256$	$N = 64$	$N = 128$
$\kappa(\boldsymbol{V})$	2.9×10^4	1.1×10^9	3.5×10^{15}	8.9×10^{13}	1.8×10^{17}
r_{GSchur}	8.1×10^{-13}	6.3×10^{-7}	1.3×10^{-2}	2.7×10^{-4}	2.1×10^{-1}
r_{LTZ}	7.4×10^{-16}	1.7×10^{-14}	6.2×10^{-14}	4.2×10^{-15}	2.8×10^{-14}
r_{Chol}	4.9×10^{-16}	5.6×10^{-16}	8.7×10^{-16}	9.2×10^{-16}	1.0×10^{-15}
r_{PCG}	1.5×10^{-4}	5.7×10^{-5}	4.5×10^{-5}	5.4×10^{-8}	1.2×10^{-8}
r_{HSS}	5.1×10^{-16}	4.5×10^{-10}	1.2×10^{-8}	7.1×10^{-9}	6.3×10^{-9}

Despite the performance of the generalized Schur algorithm in this numerical experiment, from Figure 4 we can see that the auto-covariance generated in such a way hardly exists in real applications. Since our Toeplitz-system solver is developed for statistical applications, we are more interested in the performance of the generalized Schur algorithms under ill-conditioned statistical models. Time series models are roughly categorized into two types for their decay speed: short-memory processes and long-memory process. Scenarios of both types are simulated in the following sections, and the performance of the generalized Schur algorithms is evaluated.

4.1.2 Long-Memory process

Models for long-memory time series are believed to have ill-conditioned covariance matrices because their auto-covariances decline slowly at a power law rate (Chen et al. 2006). Therefore they are inappropriate for the generalized Schur algorithm. In this section, two well-known models for long-memory processes are applied to study their condition numbers $\kappa(\boldsymbol{V})$ and the corresponding impact on the measurement errors of the generalized Schur algorithm.

The autoregressive fractionally integrated moving average (ARFIMA) model (Granger & Joyeux 1980) measures the persistence of shocks by introducing fractional differentiation into autoregressive moving average models. An ARFIMA(p, d, q) model has the following form

$$(1 - \sum_{i=1}^{p} \phi_i B^i)(1 - B)^d X_n = (1 + \sum_{i=1}^{q} \theta_i B^i)\varepsilon_n, \quad \varepsilon_n \overset{\text{iid}}{\sim} N(0, \sigma^2),$$

where B is the lag operator such that $B^k \cdot X_n = X_{n-k}$. The ARFIMA$(0, d, 0)$ model has the following auto-correlation function (ACF)

$$\rho_n = \frac{\Gamma(n+d)\Gamma(1-d)}{\Gamma(n-d+1)\Gamma(d)} \approx n^{2d-1}, \quad n = 1, 2, \ldots .$$

When $d \in (0, 0.5)$, we have that $\sum_{n=1}^{\infty} \rho_n = \infty$, meaning that X_n has long-range persistence.

Another popular stationary model with long-range dependence is the fractional Gaussian noise (fGn), the increment process of fractional Brownian motion $B^{\alpha}(t)$

$$X_n = B_{\alpha}(n+1) - B_{\alpha}(n),$$

whose ACF is

$$\rho_n = \frac{1}{2}\left[(n+1)^{\alpha} + |n-1|^{\alpha} - 2 \cdot n^{\alpha}\right]. \tag{15}$$

For $\alpha \in (1, 2)$, we also have that $\sum_{i=1}^{\infty} \rho_i = \infty$, indicating its long-memory property. In Figure 5 we show the ACF of two long-memory models with different parameters. For the ARFIMA model, the long range dependence is more significant for d closer to 1. For the fGn model, the long-memory property is more obvious for α closer to 2.

In order to verify the degree of ill-conditioning for long-memory processes, we generate the covariance matrices for ARFIMA$(0, 0.49, 0)$ and fGn with $\alpha = 1.9$ of different sizes, ranging

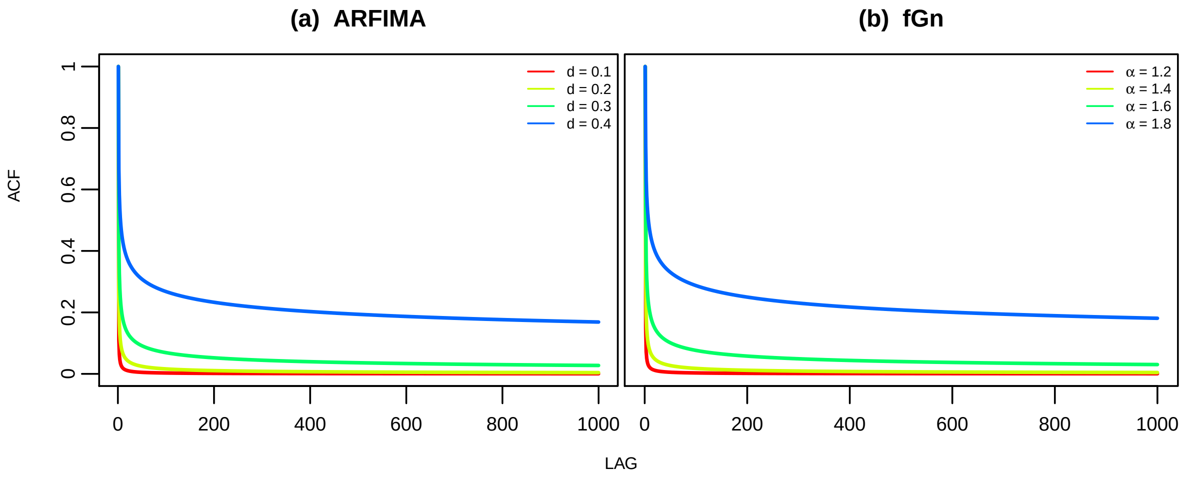


Figure 5: Auto-correlation function of long-memory models.
(a) ARFIMA$(0, d, 0)$ where $d = 0.1, 0.2, 0.3, 0.4$.
(b) fGn model where $\alpha = 1.2, 1.4, 1.6, 1.8$.

from 2000 to 10^5. By repeating the procedures of the previous section, we compute the condition number $\kappa(\boldsymbol{V})$ and measure the relative errors of the generalized Schur algorithm, LTZ and PCG algorithm (Cholesky decomposition is a $\mathcal{O}(N^3)$ algorithm, we are not going to apply it for time series longer than 1000). In Table 2 we show the relative errors for different long-memory

Model	Size N	$\kappa(\boldsymbol{V})$	r_{GSchur}	r_{LTZ}	r_{PCG}	r_{HSS}
ARFIMA(0,d,0) d = 0.49	2000	8.2×10^3	9.9×10^{-13}	6.6×10^{-15}	5.6×10^{-16}	1.3×10^{-15}
	5000	1.9×10^4	1.9×10^{-12}	9.3×10^{-15}	3.1×10^{-15}	1.3×10^{-14}
	10^4	3.5×10^4	1.2×10^{-12}	1.5×10^{-14}	6.7×10^{-16}	1.1×10^{-15}
	10^5	4.1×10^5	2.1×10^{-11}	3.6×10^{-13}	1.3×10^{-15}	1.6×10^{-14}
fBM α = 1.9	2000	2.9×10^3	4.1×10^{-14}	8.0×10^{-15}	4.6×10^{-16}	4.9×10^{-16}
	5000	5.9×10^3	2.6×10^{-13}	1.1×10^{-14}	7.6×10^{-16}	8.9×10^{-15}
	10^4	1.0×10^4	2.0×10^{-13}	9.2×10^{-15}	7.3×10^{-16}	6.7×10^{-15}
	10^5	2.1×10^5	3.6×10^{-12}	1.2×10^{-14}	8.1×10^{-16}	3.1×10^{-14}

Table 2: Measurement errors of different algorithms, long-memory models.

processes. Judging from the condition number $\kappa(\boldsymbol{V})$, we discover that the covariance matrices of long-memory processes are ill-conditioned, but not to an extreme degree like Stewart's examples. The relative errors of the generalized Schur algorithm are systematically larger than the result of the LTZ algorithm but still within a tolerable range.

4.1.3 Short Memory Process

Time series with exponential decay ACF, as an important model for short-memory processes, is established to be well-conditioned. These time series are also stationary Gaussian processes and are widely applied in applications including social communication (Karagiannis et al. 2010) and neuronal performance (Stein 1965, Byron et al. 2009). In this section, the performance of the generalized Schur algorithm is evaluated when analyzing processes with exponential decay ACF of the following form

$$\gamma_n = \exp\{-\lambda \cdot n^d\}, \quad n = 1, 2, \ldots, N, \tag{16}$$

where parameters λ and d determine the decreasing speed of γ_n. For larger λ and d, γ_n drops more rapidly. In Table 3 we show the measurement error of three different Toeplitz system solvers, where all algorithms have similarly good performance.

d	$\kappa(\boldsymbol{V})$	r_{GSchur}	r_{LTZ}	r_{PCG}	r_{HSS}
1	6.3	8.4×10^{-16}	3.2×10^{-16}	4.0×10^{-16}	5.3×10^{-16}
2	2.3	7.0×10^{-16}	4.9×10^{-16}	4.2×10^{-16}	6.2×10^{-16}
3	2.0	9.9×10^{-16}	5.4×10^{-16}	4.0×10^{-16}	4.9×10^{-16}
4	2.0	8.8×10^{-16}	3.6×10^{-16}	5.4×10^{-16}	5.6×10^{-16}

Table 3: Measurement errors of different algorithms, exponential decay models.
Data size $N = 10^5$, $\lambda = 1$.

It is worthwhile mentioning that PCG algorithm solves the Toeplitz system by recursively updating its output, where the number of iterations is related to the condition number of Toeplitz matrix $\kappa(\boldsymbol{V})$. During our experiments with the PCG algorithm, we discovered that this algorithm solves a long-memory Toeplitz system much faster than a short-memory system. More specifically, let $\boldsymbol{V}_1$ be the covariance matrix of a long-memory process and $\boldsymbol{V}_2$ be the covariance matrix of an exponential decay process. If $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ are equivalently ill-conditioned (we can achieve this by having a very small λ in (16)), PCG algorithm will take many more iterations to invert matrix $\boldsymbol{V}_2$ than $\boldsymbol{V}_1$.

4.2 Parameter Estimation for Long-Memory Models

One main purpose of the evaluation of the log-likelihood for stationary Gaussian processes is to estimate the parameters. In the previous section we investigate the measurement error of the generalized Schur algorithm with respect to various time processes but are still in lack of a straightforward impression of the impact of these errors. In order to reveal a potential bias when applying the generalized Schur algorithm, we design a numerical experiment to measure the accuracy of estimation results using the generalized Schur algorithm. Among all common statistical models, long-memory series are established to be ill-conditioned and turn out to be most unsuitable for the generalized Schur algorithm. In order to see the limitation of the generalized Schur algorithm, M long-dependency time series $\boldsymbol{X} = \left[\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(M)}\right]$ are generated, where $\boldsymbol{X}^{(m)} = [X_1^{(m)}, \ldots, X_N^{(m)}]$ i.i.d. follows ARFIMA$(0, d, 0)$ model for $m = 1, \ldots, M$, i.e.

$$(1-B)^d X_n^{(m)} = \varepsilon_n, \quad \varepsilon \overset{\text{iid}}{\sim} N(0, \sigma^2), \quad n = 1, 2, \ldots, N.$$

In the simulation, we generate $M = 500$ time series with true parameters $d = 0.45, \sigma = 1$ and length of data $N = 10^4$. This is a long-memory time series model with only two unknown parameters $\boldsymbol{\theta} = \{d, \sigma\}$, which can be estimated by maximizing the following likelihood

$$\ell(\boldsymbol{\theta} \mid \boldsymbol{X}^{(m)}) = -\frac{1}{2}\left[\boldsymbol{X}^{(m)\prime}\boldsymbol{V}_{\boldsymbol{\theta}}\boldsymbol{X}^{(m)} + \log|\boldsymbol{V}_{\boldsymbol{\theta}}| + N\log(2\pi)\right],$$

where $\boldsymbol{V}_{\boldsymbol{\theta}}$ is the Toeplitz covariance matrix whose first column is the auto-covariance of ARFIMA$(0, d, 0)$ multiplying σ^2.

In addition to the MLE estimates

$$\hat{\boldsymbol{\theta}}_m = \arg\max_{\boldsymbol{\theta}}\{\ell(\boldsymbol{\theta} \mid \boldsymbol{X}^{(m)})\},$$

we can also compute their covariance matrix using the observed Fisher information

$$\operatorname{cov}(\hat{\boldsymbol{\theta}}_m) = -\left[\frac{\partial^2}{\partial\boldsymbol{\theta}^2}\ell(\boldsymbol{\theta} \mid \boldsymbol{X}_i)\bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_m}\right]^{-1}.$$

Based on $\hat{\boldsymbol{\theta}}_m$ and $\operatorname{cov}(\hat{\boldsymbol{\theta}}_m)$, we evaluate the quality of estimation using three statistics of the estimator, including the bias, the MSE and the true coverage rate:

- Bias, $\frac{1}{M}\sum_{m=1}^{M}\hat{\boldsymbol{\theta}}_m - \boldsymbol{\theta}_0$ where $\boldsymbol{\theta}_0 = \{0.45, 1\}$ is the true parameter value.
- MSE, $\frac{1}{M}\sum_{m=1}^{M}(\hat{\boldsymbol{\theta}}_m - \boldsymbol{\theta}_0)^2$ measures the average of squares of errors.
- True coverage $\mathrm{P}_\alpha(\hat{\boldsymbol{\theta}})$, for different confidence intervals for each parametric estimator it is calculated as

  $$\mathrm{P}_\alpha(\hat{\boldsymbol{\theta}}) = \frac{1}{M}\sum_{m=1}^{M}\mathbb{1}\{\boldsymbol{\theta}_0 \in \hat{\boldsymbol{\theta}}_m \pm q_\alpha \cdot \operatorname{se}(\hat{\boldsymbol{\theta}}_m)\}, \tag{17}$$

  where q_α is the normal quantile for significance level α, $\operatorname{se}(\hat{\boldsymbol{\theta}}_m)$ is the square root of the diagonal elements of the covariance matrix $\operatorname{cov}(\hat{\boldsymbol{\theta}}_m)$. When correct models are applied, their corresponding coverage rate should be close to the theoretical coverage rate, which is the significance level α.

In Table 4 we show the estimation results using the generalized Schur algorithm, Levinson algorithm and PCG. For all methods the estimation errors in both parameters $\{\sigma, d\}$ are negligible, and the coverage rate r_α suggests that the estimations of confidence intervals for various significant levels $\alpha = 90\%, 95\%, 99\%$ are also accurate.

To conclude the experiments on numerical stability, we first propose an empirical relation between the measurement errors of the generalized Schur algorithm and the condition number of Toeplitz matrices

$$r_{\text{GSchur}} \quad \propto \quad \kappa(\boldsymbol{V}),$$

which explains the concerns about the numerical stability of the generalized Schur algorithm. However, with further investigation of the covariance matrices involved in statistical applications, including long-memory processes whose covariance matrix is most ill-conditioned, their

	algorithm	bias	MSE	$P_{90\%}$	$P_{95\%}$	$P_{99\%}$
	GSchur	-8.4×10^{-5}	9.9×10^{-5}	90	95	99
σ	Levinson	1.5×10^{-5}	1.3×10^{-4}	91	95	99
	PCG	6.9×10^{-6}	2.6×10^{-5}	92	97	100
	GSchur	-1.2×10^{-3}	1.0×10^{-4}	93	97	98
d	Levinson	-7.4×10^{-4}	8.9×10^{-5}	93	96	99
	PCG	1.0×10^{-3}	7.4×10^{-5}	91	94	98

Table 4: Estimation results of different algorithms, ARFIMA$(0, d, 0)$ model.
The true parameterare $d = 0.45$, $\sigma = 1$.

condition number is still within a tolerable degree such that the relative error of the superfast Toeplitz-system solver is acceptable. The result of parametric estimation experiment also supports this point of view. In general, the generalized Schur algorithm is applicable in the majority of statistical applications, and its disadvantage in numerical stability will hardly hinder the correct inference of models.

5 Application: Gaussian Process Factor Analysis

Summarizing a high dimensional data set with a low dimensional embedding is a standard approach for exploring the data structure. Typical techniques which can be used for dimensionality reduction includes linear discriminant analysis, principal component analysis (PCA) and factor analysis. The Gaussian process factor analysis (GPFA), whose motivation can be traced back to the use of PCA for extracting informative low dimensional views of high-dimensional neural data (Byron et al. 2009), actually accomplishes the dimensionality reduction and smoothing operations in a common probabilistic framework. In this section we describe the GPFA model and later propose a superfast Gibbs sampling for the inference.

Let $\boldsymbol{y}(t) = \left[\, y_1(t) \ \cdots \ y_D(t) \,\right] \in R^{1\times D}$ be the vector of the high-dimensional processes recorded at time t, where D is the number of processes recorded. In the framework of GPFA, we try to extract a corresponding low-dimensional signal $\boldsymbol{x}(t) = \left[\, x_1(t) \ \cdots \ x_K(t) \,\right] \in R^{1\times K}$ at time t, where K is the number of factors used to explain $\boldsymbol{y}$. Each factor $x_k(t)$ has mean 0 and a stationary covariance function

$$\operatorname{cov}\left(x_k(t), x_k(s)\right) = f_k(|t-s|, \boldsymbol{\theta}_k).$$

For the discrete observation $\boldsymbol{y} = [\boldsymbol{y}_1, ..., \boldsymbol{y}_N] \in R^{N\times D}, \boldsymbol{y}_n = \boldsymbol{y}(n\Delta t)$ at evenly distributed time series $\boldsymbol{t} = [\Delta t, \ldots, N\Delta t]$, we define a conditional Gaussian distribution of $\boldsymbol{y}_n$ given $\boldsymbol{x}(t)$:

$$\boldsymbol{y}_n \mid \boldsymbol{x}(t) \sim \mathcal{N}\left(\boldsymbol{x}(n\Delta t)\boldsymbol{\beta}, \boldsymbol{\Sigma}\right),$$

where $\boldsymbol{\beta}_{K\times D}$ is the coefficient of factors and $\boldsymbol{\Sigma}_{D\times D} = \begin{bmatrix} \sigma_1^2 & & \\ & \ddots & \\ & & \sigma_D^2 \end{bmatrix}$ is the diagonal covariance matrix.

The matrix form of the GPFA model is

$$\boldsymbol{y} = \boldsymbol{x}\boldsymbol{\beta} + \boldsymbol{\varepsilon}\boldsymbol{\Sigma}^{1/2}, \tag{18}$$

where the signal matrix $\boldsymbol{x} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N] \in R^{N\times K}$, $\boldsymbol{x}_n = \left[x_1(n\Delta t) \cdots x_K(n\Delta t) \right]$, $\boldsymbol{\varepsilon}_{N\times D}$ is the matrix of white noises $\boldsymbol{\varepsilon}_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$ and the factor observations $\boldsymbol{x}_k$ are assumed to follow independent multivariate normal distributions

$$\boldsymbol{x}_k \overset{\text{ind}}{\sim} \mathcal{N}(\boldsymbol{0}, \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}), \tag{19}$$

where $\boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}$ is the covariance matrix with elements $\boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}(n,m) = f_k(|n-m|\Delta t, \boldsymbol{\theta}_k)$.

Normally the parameters of the GPFA model are learnt in a straightforward way using the expectation-maximization (EM) algorithm, where the conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{y})$ is required in the E-step:

$$\boldsymbol{x}_k \mid \boldsymbol{y} \sim \mathcal{N}\left((\boldsymbol{\beta}_{k:} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)})\boldsymbol{\Omega}^{-1}\text{VEC}(\boldsymbol{y}), {\boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}}^{-1} - (\boldsymbol{\beta}_{k:} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)})\boldsymbol{\Omega}^{-1}(\boldsymbol{\beta}_{k:}^T \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}) \right) \tag{20}$$

where $\boldsymbol{\beta}_{k:}$ is the k-th row of $\boldsymbol{\beta}$, $\boldsymbol{\Omega} = \sum_{k=1}^{K} \boldsymbol{\beta}_{k:}^T\boldsymbol{\beta}_{k:} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)} + \boldsymbol{\Sigma} \otimes \text{I}_N$ is a $Nd \times Nd$ matrix. The evaluation of $E_{\boldsymbol{x}|\boldsymbol{y}}\left[\ell(\boldsymbol{\beta}, \boldsymbol{\Sigma}, \boldsymbol{\theta} \mid \boldsymbol{x}, \boldsymbol{y})\right]$, where $\ell(\boldsymbol{\beta}, \boldsymbol{\Sigma}, \boldsymbol{\theta} \mid \boldsymbol{x}, \boldsymbol{y})$ is the log-likelihood of (18), requires the inversion of $\boldsymbol{\Omega}$, which can only be achieved with the Cholesky decomposition in $\mathcal{O}(d^3N^3)$ steps. The overall computation cost of the EM algorithm is too expensive.

In order to reduce the computational cost, we propose the following superfast Gibbs sampling for parameter estimation. Each step of sampling can be efficiently done in $\mathcal{O}(N \log^2 N)$ steps. For a prior

$$\sigma_d^2 \sim \text{Inv-Gamma}(\alpha_d, \beta_d), \quad \boldsymbol{\beta}_{:d} \sim \mathcal{N}(\boldsymbol{\Psi}_d, \boldsymbol{S}_d), \quad d = 1, 2, \ldots, D, \tag{21}$$

the Gibbs sampling updates its various components using the analytical distributions

$$\begin{aligned} \boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{\beta}, \boldsymbol{\Sigma}, \boldsymbol{\theta} &\sim p(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{\beta}, \boldsymbol{\Sigma}, \boldsymbol{\theta}) \\ \sigma_d^2 \mid \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\beta} &\sim \text{Inv-Gamma}\left(\alpha_d^\star, \beta_d^\star\right) \\ \boldsymbol{\beta}_{:d} \mid \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\Sigma} &\sim \mathcal{N}\left(\boldsymbol{\Psi}_d^\star, \boldsymbol{S}_d^\star\right) \\ \boldsymbol{\theta} \mid \boldsymbol{x} &\sim q(\boldsymbol{\theta} \mid \boldsymbol{x}) \end{aligned} \tag{22}$$

where $\alpha_d^\star = \alpha_d + \dfrac{N}{2}$, $\beta_d^\star = \beta_d + \dfrac{(\boldsymbol{y}_d - \boldsymbol{x}\boldsymbol{\beta}_{:d})^T(\boldsymbol{y}_d - \boldsymbol{x}\boldsymbol{\beta}_{:d})}{2}$, $\boldsymbol{\Psi}_d^\star = \left[\boldsymbol{S}_d^{-1} + \dfrac{\boldsymbol{x}'\boldsymbol{x}}{\sigma_d^2}\right]^{-1}\left[\boldsymbol{S}_d^{-1}\boldsymbol{\Psi}_d + \dfrac{\boldsymbol{x}'\boldsymbol{y}_d}{\sigma_d^2}\right]$ and $\boldsymbol{S}_d^\star = \left[\boldsymbol{S}_d^{-1} + \dfrac{\boldsymbol{x}'\boldsymbol{x}}{\sigma_d^2}\right]^{-1}$.

The conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{\beta}, \boldsymbol{\Sigma}, \boldsymbol{\theta})$ is not trivial. Consider

$$\boldsymbol{y}_k^\star = \boldsymbol{y} - \sum_{i \neq k} \boldsymbol{x}_i\boldsymbol{\beta}_{i:} = \boldsymbol{x}_k\boldsymbol{\beta}_{k:} + \boldsymbol{\varepsilon}\boldsymbol{\Sigma}^{1/2}.$$

We find that $\boldsymbol{x}_k \mid \boldsymbol{y}_k^\star$ follows a multivariate normal distribution

$$\boldsymbol{x}_k \mid \boldsymbol{y}_k^\star \sim \mathcal{N}\left((\boldsymbol{\beta}_{k:} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)})\boldsymbol{\Omega}_k^{-1}\mathrm{vec}(\boldsymbol{y}_k^\star), {\boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}}^{-1} - (\boldsymbol{\beta}_{k:} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)})\boldsymbol{\Omega}_k^{-1}(\boldsymbol{\beta}_{k:}^T \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)})\right) \tag{23}$$

where $\boldsymbol{\Omega}_k = \boldsymbol{\beta}_{k:}^T\boldsymbol{\beta}_{k:} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)} + \boldsymbol{\Sigma} \otimes \mathrm{I}_N$ is also an $Nd \times Nd$ matrix. Unlike the computation that requires $p(\boldsymbol{x}_k \mid \boldsymbol{y})$ (20), the calculation involving $\boldsymbol{\Omega}_k$ can be greatly simplified with the Woodbury matrix identity (Higham 2002)

$$\boldsymbol{\Omega}_k^{-1} = \boldsymbol{\Sigma}^{-1} \otimes \mathrm{I}_N - (\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta}_{k:}^T\boldsymbol{\beta}_{k:}\boldsymbol{\Sigma}^{-1}) \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}\boldsymbol{Q}^{-1}$$

where $c = \boldsymbol{\beta}_{k:}\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta}_{k:}^T$ is a scale and $\boldsymbol{Q} = c \cdot \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)} + \mathrm{I}_N$ is an $N \times N$ Toeplitz matrix. We find that

$$(\boldsymbol{\beta}_{k:} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)})\boldsymbol{\Omega}_k^{-1}\mathrm{vec}(\boldsymbol{y}_k^\star) = \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}\boldsymbol{Q}^{-1}\boldsymbol{y}_k^\star\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta}_{k:}^T$$

and

$$\boldsymbol{V}_{\boldsymbol{\theta}}^{(k)} - (\boldsymbol{\beta}_{k:} \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)})\boldsymbol{\Omega}_k^{-1}(\boldsymbol{\beta}_{k:}^T \otimes \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}) = \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}\boldsymbol{Q}^{-1},$$

that is to say

$$\boldsymbol{x}_k \mid \boldsymbol{y}_k^\star \sim \mathcal{N}\left(\boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}\boldsymbol{Q}^{-1}\boldsymbol{y}_k^\star\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta}_{k:}^T, \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}\boldsymbol{Q}^{-1}\right),$$

whose simulation can be achieved efficiently by generating

$$\boldsymbol{x}_k = \boldsymbol{Q}^{-1}\boldsymbol{\varepsilon}_1 + \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}\boldsymbol{Q}^{-1}\boldsymbol{\varepsilon}_2,$$

where

$$\boldsymbol{\varepsilon}_1 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}), \qquad \boldsymbol{\varepsilon}_2 \sim \mathcal{N}(\boldsymbol{y}_k^\star\boldsymbol{\Sigma}^{-1}\boldsymbol{\beta}_{k:}^T, c \cdot \mathrm{I}_N).$$

Since both $\boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}$ and $\boldsymbol{Q}$ are size $N \times N$ Toeplitz matrices, sampling from $p(\boldsymbol{x} \mid \boldsymbol{y}, \boldsymbol{\beta}, \boldsymbol{\Sigma}, \boldsymbol{\theta})$ is superfast.

As for the posterior $q(\boldsymbol{\theta} \mid \boldsymbol{x})$, we have that

$$q(\boldsymbol{\theta} \mid \boldsymbol{x}) = \prod_{k=1}^{K} q(\boldsymbol{\theta}_k \mid \boldsymbol{x}_k), \qquad q(\boldsymbol{\theta}_k \mid \boldsymbol{x}_k) \quad \propto \quad L(\boldsymbol{\theta}_k \mid \boldsymbol{x}_k)$$

where $L(\boldsymbol{\theta}_k \mid \boldsymbol{x}_k)$ is the likelihood of (19)

$$L(\boldsymbol{\theta}_k \mid \boldsymbol{x}_k) = \frac{\exp\left(-\frac{1}{2}\boldsymbol{x}_k{\boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}}^{-1}\boldsymbol{x}_k\right)}{\sqrt{|2\pi\boldsymbol{V}_{\boldsymbol{\theta}}^{(k)}|}},$$

To verify the quality of the proposed Gibbs sampler, we simulated a length $N = 2000$, $d = 10$ dimensional data $\boldsymbol{y}$ containing two factors $\boldsymbol{x}_1, \boldsymbol{x}_2$, where $\boldsymbol{x}_1$ is a short-memory process with exponential decay ACF and $\boldsymbol{x}_2$ is a long-memory fGn process

$$\begin{aligned}\mathrm{ACF}_{x_1}(n) &= \exp\{-\lambda \cdot n^2\} \\ \mathrm{ACF}_{x_2}(n) &= \frac{1}{2}\left[(n+1)^\alpha + |n-1|^\alpha - 2\cdot n^\alpha\right].\end{aligned} \tag{24}$$

For $\boldsymbol{\beta}$ and $\boldsymbol{\Sigma}$, their elements are drawn from uniform distributions

$$\beta_{ij} \overset{\text{iid}}{\sim} \mathrm{Unif}(-10,10), \quad \sigma_j \overset{\text{iid}}{\sim} \mathrm{Unif}(0,3), \quad 1 \le i \le 2, 1 \le j \le 10.$$

In Figure 6 we show the posterior distribution of α and λ, and in Table 4 we demonstrate the point estimation of $\boldsymbol{\beta}$ and $\boldsymbol{\Sigma}$ with the standard deviation. The estimated coefficients $\{\hat{\alpha}, \hat{\lambda}, \hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\Sigma}}\}$ are very close to their true value, indicating that the result of the proposed Gibbs sampling procedure for GPFA model (18) is consistent and asymptotically unbiased.

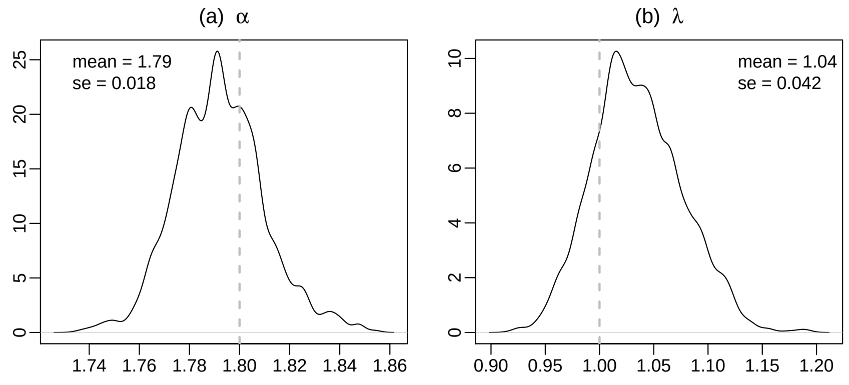


Figure 6: Posterior distribution of the factor parameters $\{\alpha, \lambda\}$ using the proposed Gibbs sampling procedure.
(a) The posterior density of estimated α.
(b) The posterior density of estimated λ.

6 Discussion

In this paper, we have provided computationally efficient algorithms for the inference of stationary Gaussian processes. Realizing that the covariance matrix for stationary Gaussian process is Toeplitz, we implemented the generalized Schur algorithm that solves the Toeplitz systems in superfast steps ($\mathcal{O}(N \log^2 N)$) and extend the range of this algorithm from $N = 2^K + 1$ to an arbitrary N. With a superfast solution to the Toeplitz system, the evaluation of the log-likelihood (8) and its derivatives for a general family of Gaussian observations with Toeplitz covariance structure can also be accomplished in superfast speed, which greatly reduce the time cost for parameter estimation, in either frequentist or Bayesian approaches. Profile likelihood for a special condition (13) is also provided for dimensionality reduction.

Based on our **R**/**C++** implementation of the extended generalized Schur algorithm, exten-

	$\beta_{1,1}$	$\beta_{1,2}$	$\beta_{1,3}$	$\beta_{1,4}$	$\beta_{1,5}$	$\beta_{1,6}$	$\beta_{1,7}$	$\beta_{1,8}$	$\beta_{1,9}$	$\beta_{1,10}$
True	9.2	6.4	1.2	-3.9	4.0	-2.2	-1.1	5.2	4.1	-5.2
Mean	9.2	6.2	0.98	-4.0	4.1	-2.4	-1.2	5.0	4.3	-5.2
SD	0.16	0.093	0.064	0.11	0.10	0.11	0.082	0.10	0.13	0.11
	$\beta_{2,1}$	$\beta_{2,2}$	$\beta_{2,3}$	$\beta_{2,4}$	$\beta_{2,5}$	$\beta_{2,6}$	$\beta_{2,7}$	$\beta_{2,8}$	$\beta_{2,9}$	$\beta_{2,10}$
True	-7.9	3.4	4.2	7.0	-6.8	7.3	5.5	5.5	-8.5	6.2
Mean	-7.6	3.5	4.2	6.9	-6.7	7.2	5.4	5.5	-8.4	6.1
SD	0.35	0.19	0.15	0.26	0.25	0.26	0.20	0.23	0.32	0.25
	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8	σ_9	σ_{10}
True	2.0	2.7	0.71	1.8	0.81	2.9	0.55	2.6	1.7	2.3
Mean	2.0	2.6	0.78	1.9	0.96	2.9	0.61	2.4	1.5	2.3
SD	0.077	0.091	0.035	0.037	0.036	0.038	0.031	0.091	0.044	0.039

Table 5: Estimated $\boldsymbol{\beta}$ and $\boldsymbol{\Sigma}$ for the simulated GPFA data.

sive numerical experiments are conducted to compare the superfast method and other distinguished Toeplitz-system solves in aspects of overall computation speed and numerical stability. Despite that the generalized Schur algorithm can be unstable in very extreme cases, we show that for statistical applications its measurement error remains within a tolerable extent and will return unbiased estimates. Finally, we introduce a GPFA model for smoothing and dimensionality reduction and propose a superfast Gibbs sampling procedure that returns consistent and asymptotically unbiased estimates.

A Multiplication of Toeplitz Matrix and Vector

Assuming that $\boldsymbol{V}$ is a Toeplitz matrix of the following form

$$\boldsymbol{V} = \begin{bmatrix} \gamma_1 & \gamma_2 & \gamma_3 & \cdots & \gamma_N \\ \gamma_{-2} & \gamma_1 & \gamma_2 & \cdots & \gamma_{N-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \gamma_{-N} & \gamma_{1-N} & \gamma_{2-N} & \cdots & \gamma_1 \end{bmatrix},$$

and $\boldsymbol{x} = [x_1, x_2, \cdots, x_N]$ is a length-N vector. The normal computation of $\boldsymbol{V} \times \boldsymbol{x}$ requires $\mathcal{O}(N^2)$ steps, but with the Fast Fourier Transformation (FFT) (Kailath & Sayed 1999) we can achieve this in $\mathcal{O}(N \log N)$ steps.

We first generate a circulant embedding form of the Toeplitz matrix $\boldsymbol{V}$

$$\boldsymbol{V}_0 = \left[\begin{array}{ccccc|ccccc} \gamma_1 & \gamma_2 & \gamma_3 & \cdots & \gamma_N & 0 & \gamma_{-N} & \gamma_{1-N} & \cdots & \gamma_{-2} \\ \gamma_{-2} & \gamma_1 & \gamma_2 & \cdots & \gamma_{N-1} & \gamma_N & 0 & \gamma_{-N} & \cdots & \gamma_{-3} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots & \vdots & \cdots & \ddots \\ \gamma_{-N} & \gamma_{1-N} & \gamma_{2-N} & \cdots & \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 & \cdots & 0 \\ \hline 0 & \gamma_{-N} & \gamma_{1-N} & \cdots & \gamma_{-2} & \gamma_1 & \gamma_2 & \gamma_3 & \cdots & \gamma_N \\ \gamma_N & 0 & \gamma_{-N} & \cdots & \gamma_{-3} & \gamma_{-2} & \gamma_1 & \gamma_2 & \cdots & \gamma_{N-1} \\ \vdots & \vdots & \vdots & \cdots & \ddots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \gamma_2 & \gamma_3 & \gamma_4 & \cdots & 0 & \gamma_{-N} & \gamma_{1-N} & \gamma_{2-N} & \cdots & \gamma_1 \end{array}\right] = \left[\begin{array}{c|c} \boldsymbol{V} & \boldsymbol{M} \\ \hline \boldsymbol{M} & \boldsymbol{V} \end{array}\right],$$

and set $\boldsymbol{x}_0 = [\boldsymbol{x}, \boldsymbol{0}]'$ such that $\boldsymbol{x}_0$ is a length $2N$ vector. Then we have that

$$\boldsymbol{V}_0 \times \boldsymbol{x}_0 = [\boldsymbol{V} \times \boldsymbol{x}, \boldsymbol{M} \times \boldsymbol{x}]',$$

where the first half is exactly what we want. Since circulant matrix can be diagonalized by the Fourier matrix

$$\boldsymbol{V}_0 = \boldsymbol{F}_N^* \times \operatorname{diag}(\boldsymbol{F}_N \times \boldsymbol{\gamma}) \times \boldsymbol{F}_N,$$

where $\boldsymbol{F}_N$ is a size N Fourier matrix and $\boldsymbol{F}_N^*$ is its conmugate transpose, $\boldsymbol{\gamma}$ is the first column of matrix $\boldsymbol{V}_0$ and the result of $\boldsymbol{F}_N \times \boldsymbol{\gamma}$ equals $\boldsymbol{\mathcal{F}}\boldsymbol{\gamma}$, the fast Fourier transformation on $\boldsymbol{\gamma}$. Here we present a $\mathcal{O}(N \log N)$ complexity procedure in computing $\boldsymbol{y}_0 = \boldsymbol{V}_0 \times \boldsymbol{x}_0$

- compute $\boldsymbol{f} = \boldsymbol{\mathcal{F}}\boldsymbol{\gamma}$,
- compute $\boldsymbol{g} = \boldsymbol{\mathcal{F}}\boldsymbol{x}_0$,
- compute $\boldsymbol{h} = \boldsymbol{f} \cdot \boldsymbol{g}$ (element-wise product for vectors),
- compute $\boldsymbol{y}_0 = \boldsymbol{\mathcal{F}}^{-1}\boldsymbol{h}$.

By extracting the first N elements of $\boldsymbol{y}_0$ we can obtain the result of $\boldsymbol{V} \times \boldsymbol{x}$.

B Superfast Computation of the Gradient and Hessian matrix of Profile Likelihoood

For the profile likelihood

$$\ell_{\text{prof}}(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{G}) = -\frac{Nd}{2}\log(2\pi) - \frac{d}{2}\log|\boldsymbol{V}_{\boldsymbol{\theta}}| - \frac{N}{2}\log|\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}| - \frac{N}{2},$$

where

$$\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}} = (\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G})^{-1}\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X}, \quad \widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}} = \frac{1}{N}(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}),$$

its first derivative with respect to parameter $\theta_i \in \boldsymbol{\theta}$ is

$$\frac{\partial}{\partial \theta_i} \ell_{\text{prof}}(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{G}) = -\frac{d}{2}\text{tr}\{\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\} - \frac{N}{2}\text{tr}\{\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}\widehat{\boldsymbol{\Sigma}}_i\}$$

where

$$\begin{aligned}\widehat{\boldsymbol{\Sigma}}_i = \frac{\partial \widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}}{\partial \theta_i} = &- \widehat{\boldsymbol{\beta}}_i' \boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}) - (\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})' \boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\widehat{\boldsymbol{\beta}}_i - \\ &(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})' \boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}),\end{aligned}$$

and

$$\begin{aligned}\widehat{\boldsymbol{\beta}}_i = \frac{\partial \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}}{\partial \theta_i} = &\left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} - \\ &\left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X}.\end{aligned}$$

Since the computation of $\widehat{\boldsymbol{\Sigma}}_i$ and $\widehat{\boldsymbol{\beta}}_i$ is $\mathcal{O}(N \log N)$, and $\widehat{\boldsymbol{\Sigma}}_i$ is a size $d \times d$ matrix whose computation of $\text{tr}\{\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}\widehat{\boldsymbol{\Sigma}}_i\}$ is $\mathcal{O}(d^3)$, the total computation of $\frac{\partial}{\partial \theta_i}\ell_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{G})$ is superfast.

For the second derivative of the profile likelihood (14) with respect to $\theta_i, \theta_m \in \boldsymbol{\theta}$, we have

$$\frac{\partial^2}{\partial \theta_i \partial \theta_m}\ell_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{G}) = -\frac{d}{2}\text{tr}\{\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_{im} - \boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\} - \frac{N}{2}\text{tr}\{\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}\widehat{\boldsymbol{\Sigma}}_{im} - \widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}\widehat{\boldsymbol{\Sigma}}_m\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}\widehat{\boldsymbol{\Sigma}}_i\}, \tag{25}$$

where

$$\begin{aligned}\widehat{\boldsymbol{\Sigma}}_{im} = \frac{\partial^2 \widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}}{\partial \theta_i \partial \theta_m} = &- \widehat{\boldsymbol{\beta}}_{im}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}) + 2 \cdot \widehat{\boldsymbol{\beta}}_i'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}^{-1}(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}) + \widehat{\boldsymbol{\beta}}_i'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\widehat{\boldsymbol{\beta}}_m + \\ &\widehat{\boldsymbol{\beta}}_m'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\widehat{\boldsymbol{\beta}}_i + 2 \cdot (\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\widehat{\boldsymbol{\beta}}_i - (\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\widehat{\boldsymbol{\beta}}_{im} + \\ &(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}) - \\ &(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_{im}\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}) + \\ &(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}})'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}(\boldsymbol{X} - \boldsymbol{G}\widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}),\end{aligned}$$

and

$$
\begin{aligned}
\widehat{\boldsymbol{\beta}}_{im} = \frac{\partial^2 \widehat{\boldsymbol{\beta}}_{\boldsymbol{\theta}}}{\partial\theta_i \partial\theta_m} = & \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} - \\
& \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} + \\
& \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_{im}\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} - \\
& \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} + \\
& \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} - \\
& \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} + \\
& \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G} \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} - \\
& \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} + \\
& \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_{im}\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X} + \\
& \left(\boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{G}\right)^{-1} \boldsymbol{G}'\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{X}.
\end{aligned}
$$

Despite the complex expression of $\widehat{\boldsymbol{\beta}}_{im}$ and $\widehat{\boldsymbol{\Sigma}}_{im}$, their computation is still $\mathcal{O}(N \log N)$. Since part $\operatorname{tr}\{\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_{im} - \boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_m\boldsymbol{V}_{\boldsymbol{\theta}}^{-1}\boldsymbol{V}_i\}$ is proved to be superfast and $\operatorname{tr}\{\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}\widehat{\boldsymbol{\Sigma}}_{im} - \widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}\widehat{\boldsymbol{\Sigma}}_m\widehat{\boldsymbol{\Sigma}}_{\boldsymbol{\theta}}\widehat{\boldsymbol{\Sigma}}_i\}$ only involves some $d \times d$ matrices, the total computation of the Hessian term $\frac{\partial^2}{\partial\theta_i\partial\theta_m}\ell_{\text{prof}}(\boldsymbol{\theta}|\boldsymbol{X}, \boldsymbol{G})$ is also superfast.

References

Ammar, G. S. (1996), 'Classical foundations of algorithms for solving positive definite toeplitz equations', *Calcolo* **33**(1-2), 99–113.

Ammar, G. S. & Gragg, W. B. (1987), The generalized schur algorithm for the superfast solution of toeplitz systems, *in* 'Rational approximation and its applications in mathematics and physics', Springer, pp. 315–330.

Ammar, G. S. & Gragg, W. B. (1988), 'Superfast solution of real positive definite toeplitz systems', *SIAM Journal on Matrix Analysis and Applications* **9**(1), 61–76.

Ammar, G. S. & Gragg, W. B. (1989), 'Numerical experience with a superfast real toeplitz solver', *Linear Algebra and its Applications* **121**, 185–206.

Archambeau, C., Cornford, D., Opper, M. & Shawe-Taylor, J. (2007), 'Gaussian process approximations of stochastic differential equations', *Journal of machine learning research* **1**, 1–16.

Bareiss, E. H. (1969), 'Numerical solution of linear equations with toeplitz and vector toeplitz matrices', *Numerische Mathematik* **13**(5), 404–424.

Bitmead, R. R. & Anderson, B. D. (1980), 'Asymptotically fast solution of toeplitz and related systems of linear equations', *Linear Algebra and its Applications* **34**, 103–116.

Breidt, F. J., Crato, N. & De Lima, P. (1998), 'The detection and estimation of long memory in stochastic volatility', *Journal of econometrics* **83**(1), 325–348.

Brent, R. P., Gustavson, F. G. & Yun, D. Y. (1980), 'Fast solution of toeplitz systems of equations and computation of padé approximants', *Journal of Algorithms* **1**(3), 259–295.

Bunch, J. R. (1985), 'Stability of methods for solving toeplitz systems of equations', *SIAM Journal on Scientific and Statistical Computing* **6**(2), 349–364.

Byron, M. Y., Cunningham, J. P., Santhanam, G., Ryu, S. I., Shenoy, K. V. & Sahani, M. (2009), Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity, *in* 'Advances in neural information processing systems', pp. 1881–1888.

Calderhead, B., Girolami, M. & Lawrence, N. D. (2009), Accelerating bayesian inference over nonlinear differential equations with gaussian processes, *in* 'Advances in neural information processing systems', pp. 217–224.

Chandrasekaran, S., Gu, M., Sun, X., Xia, J. & Zhu, J. (2007), 'A superfast algorithm for toeplitz systems of linear equations', *SIAM Journal on Matrix Analysis and Applications* **29**(4), 1247–1266.

Chandrasekaran, S. & Sayed, A. H. (1996), 'Stabilizing the generalized schur algorithm', *SIAM Journal on Matrix Analysis and Applications* **17**(4), 950–983.

Chandrasekaran, S. & Sayed, A. H. (1998), 'A fast stable solver for nonsymmetric toeplitz and quasi-toeplitz systems of linear equations', *SIAM Journal on Matrix Analysis and Applications* **19**(1), 107–139.

Chen, W. W., Hurvich, C. M. & Lu, Y. (2006), 'On the correlation matrix of the discrete fourier transform and the fast solution of large toeplitz systems for long-memory time series', *Journal of the American Statistical Association* **101**(474), 812–822.

de Hoog, F. (1987), 'A new algorithm for solving toeplitz systems of equations', *Linear Algebra and its Applications* **88**, 123–138.

De Hoog, F. R. (1984), *On the solution of Toeplitz systems of equations*, Australian National University, Centre for Mathematical Analysis.

Durbin, J. (1960), 'The fitting of time-series models', *Revue de l'Institut International de Statistique* pp. 233–244.

Engel, Y., Mannor, S. & Meir, R. (2005), Reinforcement learning with gaussian processes, *in* 'Proceedings of the 22nd international conference on Machine learning', ACM, pp. 201–208.

Frigo, M. & Johnson, S. G. (2005), 'The design and implementation of fftw3', *Proceedings of the IEEE* **93**(2), 216–231.

Gohberg, I. & Semencul, A. (1972), 'On the inversion of finite toeplitz matrices and their continuous analogs', *Mat. issled* **2**, 201–233.

Granger, C. W. & Joyeux, R. (1980), 'An introduction to long-memory time series models and fractional differencing', *Journal of time series analysis* **1**(1), 15–29.

Harvey, A. C. (2002), Long memory in stochastic volatility, *in* 'Forecasting volatility in the financial markets', Elsevier, pp. 307–320.

Higham, N. J. (2002), *Accuracy and stability of numerical algorithms*, Siam.

Hosking, J. R. (1981), 'Fractional differencing', *Biometrika* **68**(1), 165–176.

Jones, D. R., Schonlau, M. & Welch, W. J. (1998), 'Efficient global optimization of expensive black-box functions', *Journal of Global optimization* **13**(4), 455–492.

Kailath, T., Kung, S.-Y. & Morf, M. (1979), 'Displacement ranks of a matrix', *Bulletin of the American Mathematical Society* **1**(5), 769–773.

Kailath, T. & Sayed, A. H. (1999), *Fast reliable algorithms for matrices with structure*, SIAM.

Karagiannis, T., Le Boudec, J.-Y. & Vojnovic, M. (2010), 'Power law and exponential decay of intercontact times between mobile devices', *IEEE Transactions on Mobile Computing* **9**(10), 1377–1390.

Kravanja, P. & Van Barel, M. (2000), 'Coupled vandermonde matrices and the superfast computation of toeplitz determinants', *Numerical Algorithms* **24**(1-2), 99–116.

Levinson, N. (1946), 'The wiener (root mean square) error criterion in filter design and prediction', *Journal of Mathematics and Physics* **25**(1), 261–278.

Ling, Y. & Lysy, M. (2020), *SuperGauss: Superfast Likelihood Inference for Stationary Gaussian Time Series*. R package version 1.0.2.

Lysy, M., Pillai, N. S., Hill, D. B., Forest, M. G., Mellnik, J. W., Vasquez, P. A. & McKinley, S. A. (2016), 'Model comparison and assessment for single particle tracking in biological fluids', *Journal of the American Statistical Association* **111**(516), 1413–1426.

Musicus, B. R. (1988), *Levinson and fast Choleski algorithms for Toeplitz and almost Toeplitz matrices*, Citeseer.

Neal, R. M. (1997), 'Monte carlo implementation of gaussian process models for bayesian regression and classification', *arXiv preprint physics/9701026* .

Särkkä, S., Hartikainen, J., Svensson, L. & Sandblom, F. (2014), Gaussian process quadratures in nonlinear sigma-point filtering and smoothing, *in* '17th International Conference on Information Fusion (FUSION)', IEEE, pp. 1–8.

Sexton, H. (1982), Analysis of an algorithm of bitmead and anderson for the inversion of toeplitz systems., Technical report, DTIC Document.

Smola, A. J. & Bartlett, P. L. (2001), Sparse greedy gaussian process regression, *in* 'Advances in neural information processing systems', pp. 619–625.

Stein, R. B. (1965), 'A theoretical analysis of neuronal variability', *Biophysical Journal* **5**(2), 173–194.

Stewart, M. (2003), 'A superfast toeplitz solver with improved numerical stability', *SIAM journal on matrix analysis and applications* **25**(3), 669–693.

Stewart, M. & Van Dooren, P. (1997), 'Stability issues in the factorization of structured matrices', *SIAM Journal on Matrix Analysis and Applications* **18**(1), 104–118.

Trench, W. F. (1964), 'An algorithm for the inversion of finite toeplitz matrices', *Journal of the Society for Industrial and Applied Mathematics* **12**(3), 515–522.

Williams, C. K. & Rasmussen, C. E. (2006), *Gaussian processes for machine learning*, Vol. 2, MIT press Cambridge, MA.

Xi, Y., Xia, J., Cauley, S. & Balakrishnan, V. (2014), 'Superfast and stable structured solvers for toeplitz least squares via randomized sampling', *SIAM Journal on Matrix Analysis and Applications* **35**(1), 44–72.

Xia, J., Xi, Y. & Gu, M. (2012), 'A superfast structured solver for toeplitz linear systems via randomized sampling', *SIAM Journal on Matrix Analysis and Applications* **33**(3), 837–858.

Zohar, S. (1969), 'Toeplitz matrix inversion: The algorithm of wf trench', *Journal of the ACM (JACM)* **16**(4), 592–601.