Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Time-series Gaussian Process Regression Based on Toeplitz Computation of $O(N^2)$ Operations and $O(N)$-level Storage

Yunong Zhang, W. E. Leithead, and D. J. Leith

Abstract— **Gaussian process (GP) regression is a Bayesian nonparametric model showing good performance in various applications. However, its hyperparameter-estimating procedure may contain numerous matrix manipulations of $O(N^3)$ arithmetic operations, in addition to the $O(N^2)$-level storage. Motivated by handling the real-world large dataset of 24000 wind-turbine data, we propose in this paper an efficient and economical Toeplitz-computation scheme for time-series Gaussian process regression. The scheme is of $O(N^2)$ operations and $O(N)$-level memory requirement. Numerical experiments substantiate the effectiveness and possibility of using this Toeplitz computation for very large datasets regression (such as, containing 10000$\sim$100000 data points).**

I. INTRODUCTION

Following some initial publications in the late 1990s (e.g., [1]-[3]), interest has grown quickly into the application of Gaussian processes prior models to data analysis; e.g., [4]-[8]. In this paper, these methods are investigated to handle 24000 wind-turbine time-series data; specifically, site measurement of rotor speed for a commercial 1MW machine. The measurement is some unknown combination of the rotor speed and the generator speed (scaled by the gearbox ratio) [8]. Furthermore, the data is corrupted by significant measurement noise. The purpose is to identify both the rotor speed and the rotor acceleration, an initial yet important part of identifying the aerodynamics and drive-train dynamics of variable speed wind turbines [8]. To successfully identify the large-dataset wind-turbine rotor speed and acceleration by using Gaussian process prior models, we have to handle the model-tuning and prediction procedures more efficiently.

The model-tuning procedure of Gaussian process regression could be the optimization procedure based on maximum likelihood estimation (MLE). In standard treatments, it, together with the prediction procedure, usually involves numerous matrix manipulations of $O(N^3)$ operations and $O(N^2)$-level memory storage, where N is the number of data measurements (24000 in this case). Motivated by handling this large dataset of wind-turbine time-series data, efficient computation methods based on Toeplitz matrix structure are presented in the ensuing sections and successfully applied to the wind-turbine data.

This work was supported by Science Foundation Ireland grant, 00/PI.1/C067, and by the EPSRC grant, GR/M76379/01.

The authors are with Hamilton Institute, National University of Ireland, Maynooth, Co. Kildare, Ireland. W. E. Leithead is also with Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, United Kingdom. ynzhang@ieee.org

II. GAUSSIAN PROCESS PRIOR MODELS

Consider a smooth scalar function $f(\cdot)$ dependent on the explanatory variable, $z \in \mathbb{D} \subseteq \mathbb{R}^L$. Suppose N measurements, $\{(z_i, y_i)\}_{i=1}^N$, of the value of the function with additive Gaussian white measurement noise, i.e. $y_i = f(z_i) + n_i$, are available and denote them by $\mathbb{M}$. It is of interest here to use this data $\mathbb{M}$ to learn the function mapping $f(z)$ on the input domain $\mathbb{D}$. Note that this is a regression formulation and it is assumed that the explanatory variable z is noise free. The probabilistic description of the function, $f(z)$, adopted is the stochastic process, f_z, with the mean $E[f_z]$, as z varies, interpreted to be a fit to $f(z)$. By necessity, to define the stochastic process, f_z, the probability distributions of f_z for every choice of value of $z \in \mathbb{D}$ are required together with the joint probability distributions of f_{z_i} for every choice of finite sample, $z_1, \cdots, z_N$, from $\mathbb{D}$. Given the joint probability distribution for f_{z_i}, $i = 1, \cdots, N$, and the joint probability distribution for n_i, $i = 1, \cdots, N$, the joint probability distribution for y_i, $i = 1, \cdots, N$, is readily obtained since the measurement noise, n_i, and the function $f(z_i)$ (and so f_{z_i}) are statistically independent. M is a single event belonging to the joint probability distribution for y_i, $i = 1, \cdots, N$.

A. Gaussian Process

In the Bayesian probability context, the prior belief is placed directly on the probability distributions describing f_z which are then conditioned on the information, $\mathbb{M}$, to determine the posterior probability distributions. In particular, in the Gaussian process prior model, it is assumed that the prior probability distributions for f_z are all Gaussian with zero mean (in the absence of any evidence, the value of $f(z)$ is as likely to be positive as negative). To complete the statistical description, it requires only a definition of the covariance function $C_f(z_i, z_j) = E[f_{z_i}, f_{z_j}]$, for all z_i and z_j. The resulting posterior probability distributions are also Gaussian. This model is used to carry out inference as follows. Evidently, $\mathbb{P}(f_z|\mathbb{M}) = \mathbb{P}(f_z, \mathbb{M})/\mathbb{P}(\mathbb{M})$ where $\mathbb{P}(\mathbb{M})$ acts as a normalizing constant. Hence, with Gaussian prior assumption,

$$\mathbb{P}(f_z|\mathbb{M}) \propto \exp\left(-\frac{1}{2}\begin{bmatrix} f_z & y^T \end{bmatrix}\begin{bmatrix} \Lambda_{11} & \Lambda_{21}^T \\ \Lambda_{21} & \Lambda_{22} \end{bmatrix}\begin{bmatrix} f_z \\ y \end{bmatrix}\right)$$

where $y := [y_1, \cdots, y_N]^T$ here for notational convenience, Λ_{11} is $E[f_z, f_z]$, the ijth element of matrix Λ_{22} is $E[y_i, y_j]$, and the ith element of vector Λ_{21} is $E[y_i, f_z]$. Both Λ_{11}

0-7803-9568-9/05/$20.00 ©2005 IEEE

TABLE I

MEMORY REQUIREMENT OF A MATRIX IN DOUBLE PRECISION VERSUS ITS DIMENSION N

Dimension N	1000	3000	5000	7000	9000	11000	13000	15000	17000
Storage (MB)	7.7	68.7	190.8	373.9	618.0	923.2	1289.4	1716.6	2204.9

TABLE II

MEMORY REQUIREMENT OF A VECTOR IN DOUBLE PRECISION VERSUS ITS DIMENSION N

Dimension N	1000	7000	15000	20000	30000	50000	100000	500000	1000000
Storage (MB)	.0076	.0534	.1144	.1526	.2289	.3815	.7629	3.8147	7.6294

and Λ_{21} depend on z. It follows from the partitioned matrix-inversion lemma that

$$\mathbb{P}(f_z|\mathbb{M}) \propto \exp\left(-\frac{1}{2}(f_z-\bar{f}_z)\Lambda_z^{-1}(f_z-\bar{f}_z)\right) \quad (1)$$

with $\bar{f}_z := \Lambda_{21}^T\Lambda_{22}^{-1}y$ and $\Lambda_z := \Lambda_{11}-\Lambda_{21}^T\Lambda_{22}^{-1}\Lambda_{21}$ [6][8]. Therefore, the prediction from this model is that the most likely value of $f(z)$ is the mean, $\bar{f}_z$, with variance Λ_z. Note that $\bar{f}_z$ is simply a z-dependent weighted linear combination of the measured data-points, y, using weights $\Lambda_{21}^T\Lambda_{22}^{-1}$. The measurement noise, n_i, $i=1,\cdots,N$, is statistically independent of $f(z_i)$, $i=1,\cdots,N$, and has covariance matrix $[\delta_{ij}]$ with δ_{ij} being Kronecker operator. Hence, the covariances for the measurements, y_i, are simply

$$E[y_i,y_j]=E[f_{z_i},f_{z_j}]+[\delta_{ij}], \quad E[y_i,f_z]=E[f_{z_i},f_z].$$

B. Derivative Gaussian Process

Assume that the related stochastic process, $f_z^{\Delta e_k}$, is well-defined in the limit as $\Delta e_k \to 0$, where $f_z^{\Delta e_k}=(f_{(z+\Delta e_k)}-f_z)/\Delta e_k$ and e_k is a unit basis vector. That is, all the necessary probability distributions for a complete description exist. Denote the derivative stochastic process, i.e. the limiting random process, by $f_z^{e_k}$. The mean $E[f_z^{\Delta e_k}]$ as z varies, is interpreted as a fit to $\partial f(z)/\partial z^k$ when the partial derivative of $f(z)$ in the direction e_k exists. Provided the covariance $E[f_{z_i},f_{z_j}]$ is sufficiently differentiable, it is well known [7]-[9] that $f_z^{e_k}$ is itself Gaussian and that

$$E[f_z^{e_k}]=\partial E[f_z]/\partial z^k \quad (2)$$

where z^k denotes the kth element of z; i.e., the expected value of the derivative stochastic process is just the derivative of the expected value of the stochastic process. Furthermore,

$$E[f_{z_i}^{e_k},f_{z_j}]=\nabla_k^1 E[f_{z_i},f_{z_j}],$$
$$E[f_{z_i}^{e_k},f_{z_j}^{e_l}]=\nabla_k^1\nabla_l^2 E[f_{z_i},f_{z_j}]$$

where $\nabla_k^1 Q(z_i,z_j)$ denotes the partial derivative of $Q(z_i,z_j)$ with respect to the kth element of the first argument, i.e., z_i; and so on.

C. Model Tuning

The prior covariance function is generally dependent on a few hyperparameters, Θ. For example, we could use the following standard covariance function (i.e., the ijth element of covariance matrix Λ_{22}):

$$c_{ij}(\Theta)=a\exp\left(-\frac{1}{2}(z_i-z_j)^T D(z_i-z_j)\right)+w\delta_{ij}$$

where matrix $D=\mathrm{diag}(d_1,d_2,\cdots,d_L)$ characterizes a set of length scales d_l corresponding to each input, and the set of hyperparameters $\Theta := [a,d_1,d_2,\cdots,d_L,\ w] \in \mathbb{R}^{L+2}$ with each hyperparameter constrained to be nonnegative. Before applying the GP model for prediction purposes such as (1) and (2), as a common practice, the hyperparameters Θ could be adapted by maximizing the likelihood, $\mathbb{P}(\mathbb{M}|\Theta)$, or equivalently minimizing the following negative log-likelihood, $\mathcal{L}(\Theta)$ [1]-[3]:

$$\mathcal{L}(\Theta)=\frac{1}{2}\log\left(\det C(\Theta)\right)+\frac{1}{2}y^T C^{-1}(\Theta)y. \quad (3)$$

with $C := \Lambda_{22}$, and $\log\det C(\Theta)$ denotes the logarithm of the determinant of matrix $C(\Theta)$. As the likelihood is in general nonlinear and multimodal, efficient optimization routines usually entail the gradient information:

$$\frac{\partial\mathcal{L}}{\partial\Theta_i}=\frac{1}{2}\mathrm{tr}\left(C^{-1}\frac{\partial C}{\partial\Theta_i}\right)-\frac{1}{2}y^T C^{-1}\frac{\partial C}{\partial\Theta_i}C^{-1}y \quad (4)$$

where $\mathrm{tr}(\cdot)$ denotes the trace operator of a matrix.

III. TOEPLITZ-BASED COMPUTATION

Looking at the optimization and prediction procedures such as (1)-(4), we can see that there are three basic types of matrix manipulations involved as follows:

$$\log\det C,\ C^{-1}y,\ \text{and}\ \mathrm{tr}(C^{-1}P) \quad (5)$$

with P denoting any $\partial C/\partial\Theta_i$ hereafter for notional convenience. In standard treatments, the number of operations in solving for C^{-1} and $\log\det$ is of $O(N^3)$ [10][11], whilst the memory space to store C, C^{-1} and P is $O(N^2)$ (see Table I for specific values). For large data sets such as of 24000 data-points, fast algorithms, that require less memory allocation, are required for the basic matrix manipulations as in (5), especially when tuning the hyperparameters of Gaussian process prior models.

Now consider a time series with fixed interval between the measurements. The explanatory variable, z, in the Gaussian process prior model is thus a scalar, the time t. When the

Algorithm 1.1 Modified Trench's Algorithm with Any c_0 and Returning $\log\det C$

INPUT: vector $c=[c_0,c_1,c_2,\cdots,c_{N-1}]^T$
OUTPUT: matrix $\bar{C}\in R^{N\times N}$ as C^{-1} and scalar ℓ as $\log\det C$
Call Algorithm 1.2 with input c to solve for v and ℓ
$\bar{C}(1,1:N)=v(N:1)$, $\bar{C}(1:N,1)=v(N:1)$
$\bar{C}(N,1:N)=v(1:N)$, $\bar{C}(1:N,N)=v(1:N)$
for $i=2:\text{floor}((N-1)/2)+1$,
  for $j=i:N-i+1$,
    $\bar{C}(i,j)=\bar{C}(i-1,j-1)+\frac{v(N+1-j)v(N+1-i)-v(i-1)v(j-1)}{v(N)}$
    $\bar{C}(j,i)=\bar{C}(i,j)$
    $\bar{C}(N-i+1,N-j+1)=\bar{C}(i,j)$
    $\bar{C}(N-j+1,N-i+1)=\bar{C}(i,j)$
  end
end

Algorithm 1.2 Vector-Inverse Algorithm with Any c_0 and Returning $\log\det C$

INPUT: vector $c=[c_0,c_1,c_2,\cdots,c_{N-1}]^T$
OUTPUT: vector $v\in R^N$ and scalar ℓ
Calculate vector $\varsigma=c(2:N)/c_0$
call Algorithm 1.3 with inputs $N-1$ and ς to solve for vector z and ℓ
$\ell=\ell+N\log c_0$
$v(N)=1/((1+\varsigma^T z)c_0)$
$v(1:N-1)=v(N)z(N-1:1)$

covariance function depends on the difference in the explanatory variable, as is almost always the case, the covariance matrix $C(\Theta)$ and its derivative matrices P are Toeplitz; i.e.,

$$C=\begin{pmatrix} c_0 & c_1 & c_2 & \cdots & c_{N-1}\\ c_1 & c_0 & c_1 & \cdots & c_{N-2}\\ c_2 & c_1 & c_0 & \ddots & \vdots\\ \vdots & \vdots & \ddots & \ddots & c_1\\ c_{N-1} & c_{N-2} & \cdots & c_1 & c_0\end{pmatrix}=Toeplitz(c)$$

with $c:=[c_0,c_1,c_2,\cdots,c_{N-1}]^T$ represents the first column vector of C. Furthermore, $C(\Theta)$ is positive definite.

The Toeplitz nature of $C(\Theta)$ is exploited to derive fast algorithms that require less memory allocation. It is well-known that positive-definite Toeplitz matrices can be elegantly manipulated in $O(N^2)$ operations. For example, Trench's algorithm inverts C with $13N^2/4$ operations, and Levinson's algorithm solves for $C^{-1}y$ with $4N^2$ operations [12]. However, direct application of these algorithms to Gaussian process regression may fail even with medium-scale datasets due to lack of memory space (see Table I). For example, on a Pentium-IV 3GHz 512MB-RAM PC, a MATLAB-JAVA algorithm usually fails around $N=5000\sim 7000$ because storing C^{-1} almost uses up the available system memory. The solution is to adapt the fast algorithms so as to use only vector-level storage. It follows from Table II that a vector-storage approach is more theoretically able to handle very large datasets, such as 1-million data points, in terms of memory requirements.

Three versions of fast Toeplitz-computation algorithms have been developed and coded. Due to space limitation, only two of them are detailed in the ensuing two sections; namely, a full-matrix version as the preliminaries and a vector-storage version as the final algorithms.

IV. FULL-MATRIX TOEPLITZ ALGORITHMS

The simplest way of introducing Toeplitz computation to Gaussian process regression is to compute C^{-1} directly as the basis of the other types of matrix computation. Specifically, Trench's algorithm of $O(N^2)$ operations can be readily modified to solve C^{-1} (denoted by $\bar{C}$ hereafter) whilst simultaneously determining $\log\det C$ as the logarithm sum of reflection coefficients. Then, given $\bar{C}$, the computation $\text{tr}(C^{-1}P)=\sum_i^N\sum_j^N \bar{c}_{ij}p_{ji}$ is easily performed in $O(N^2)$ operations, where $\bar{c}_{ij}$ and p_{ij} are the ijth elements of $\bar{C}$ and P, respectively.

As Trench's algorithm uses Durbin's algorithm to solve Yule-Walker equations, we present the modified Trench's and Durbin's algorithms respectively as in Algorithm 1.1 and Algorithm 1.3. We use the same parentheses and colon notation as in [12]. As shown in the following proposition (with proof omitted due to space limitation), one of the contributions of our full-matrix algorithms is the simultaneous computation of $\log\det C$.

Proposition 1. $\log\det C$ can be obtained by Algorithms 1.2 and 1.3 during solving for C^{-1}. □

Remark 1. Note that the modified Trench's algorithm is separated as two parts: one is matrix-free Algorithm 1.2, and another is the remaining part of Algorithm 1.1 that generates $\bar{C}$. Algorithms 1.2 and 1.3 do not contain any matrix or matrix-related computation/storage, thus able to perform very high-dimension Toeplitz computation. In view of this possibility, Algorithms 1.2 and 1.3 can be reemployed in the ensuing section as a part of the vector-storage version of Toeplitz-computation.

Algorithm 1.3 Modified Durbin's Algorithm with $\log\det C$ Returned

INPUT: integer m and vector $\varsigma \in R^m$

OUTPUT: vector $z \in R^m$ and scalar ℓ

$z(1) = -\varsigma(1)$, $\beta = 1$, $\alpha = -\varsigma(1)$, $\ell = 0$

for $i = 1 : m-1$,

$\beta = (1-\alpha^2)\beta$

$\ell = \ell + \log\beta$

$\alpha = -\frac{\varsigma(i+1)+\varsigma(i:1)^T z(1:i)}{\beta}$

$z(1:i) = z(1:i) + \alpha z(i:1)$

$z(i+1) = \alpha$

end

$\beta = (1-\alpha^2)\beta$

$\ell = \ell + \log\beta$

where the local variables α, β and z may be reused for notational convenience.

TABLE III

ACCURACY AND SPEEDUP OF FULL-MATRIX TOEPLITZ COMPUTATION

	$N = 1000$	$N = 2000$	$N = 3000$
Accuracy on			
$\log\det C$:	4.1×10^{-15} (2.2×10^{-14})	3.4×10^{-15} (4.2×10^{-14})	3.5×10^{-15} (7.3×10^{-14})
$C^{-1}y$:	1.3×10^{-12} (6.5×10^{-12})	7.0×10^{-13} (1.8×10^{-12})	2.2×10^{-12} (1.3×10^{-11})
$\mathrm{tr}(C^{-1}P)$:	8.1×10^{-14} (8.0×10^{-13})	1.7×10^{-13} (1.1×10^{-12})	5.5×10^{-14} (7.7×10^{-13})
Speedup:	70.45 (16.38)	91.64 (14.19)	90.84 (14.90)

TABLE IV

ACCURACY AND EXECUTION TIME OF WEDGE-FORM ALGORITHMS

	accuracy	time (in seconds)
$N = 4000$	6.0×10^{-13} (1.4×10^{-12})	2.1 (0.44)
$N = 5000$	5.1×10^{-13} (1.9×10^{-12})	4.9 (2.39)
$N = 6000$	4.6×10^{-13} (1.2×10^{-12})	12.1 (7.50)
$N = 7000$	5.2×10^{-13} (1.8×10^{-12})	28.0 (14.16)
$N = 8000$	8.8×10^{-13} (2.9×10^{-12})	40.9 (24.19)

Remark 2. A large number of numerical experiments are performed to verify the correctness and numerical stability of the modified algorithms and their implementation. The covariance matrix is randomly generated. The results are compared to those of the standard MATLAB INV routine [13]. See Table III where every test of different dimension is based on 100 random covariance matrices, and the mean and standard deviation of relative errors and speedup ratios are shown (with standard deviation in brackets). Numerical tests show that the Trench's algorithm is stable enough in our Gaussian process context in the sense that it can work well for $w/a \geqslant 10^{-9}$, though it is a little weaker than the MATLAB INV (the latter can work well for $w/a \geqslant 10^{-13}$).

Remark 3. A natural extension of full-matrix algorithms is wedge-form algorithms. Because of the symmetric and persymmetric properties of $\bar{C}$ [12], we could only store the upper wedge of $\bar{C}$, roughly using one-fourth memory-space as in full-matrix algorithms. Numerical performance of our wedge-form Toeplitz computation is shown in Table IV.

V. VECTOR-STORAGE TOEPLITZ ALGORITHMS

As discussed above, the full-matrix and wedge-form Toeplitz algorithms work well for medium-scale regression tasks with a speed-up around 100. However, the matrix-level memory allocation is still an issue for large datasets with N greater than 10000 such as the 24000 wind-turbine data. It follows from Table II that if possible, a specialized vector-level storage version of the algorithms is attractive for specific computation task, such Gaussian regression for time series. The modified matrix-free Durbin's algorithm is used to compute $\log\det C$ and Levinson's algorithm to compute $C^{-1}y$. The remaining manipulation, i.e., $\mathrm{tr}(C^{-1}P)$, is obtained with the aid of the following proposition with proof omitted due to space limitation.

Proposition 2. As implemented in Algorithm 3.2, $\mathrm{tr}(C^{-1}P)$ can be computed as

$$\psi_1 p_1 + 2\sum_{i=2}^{N} \psi_i p_i \tag{6}$$

where $p = [p_1, p_2, \cdots, p_N]$ is the representative vector of P, and ψ_i denotes the elements summation of the ith diagonal of C^{-1} which is implemented as in Algorithm 3.1. □

Remark 4. Before applying them to Gaussian process regression, a large number of numerical experiments are also performed for the efficient and economic vector-storage version of Toeplitz algorithms. Table V shows the numerical accuracy and execution time of the algorithms, where the involved notation is the same as previous sections'. They substantiate the efficacy of the vector-storage Toeplitz computation on large datasets.

Algorithm 3.1 Vector-storage Diagonal-sum Algorithm

INPUT: vector $c = [c_0, c_1, c_2, \cdots, c_{N-1}]^T$

OUTPUT: vector φ containing diagonal sums and ℓ as $\log\det C$

Call Algorithm 1.2 with input c to solve for v and ℓ

for $i = 1 : N$

  $\varphi(1) = v(N)v(N-i+1)$, $\varphi(N-i+1) = \varphi(1)$

  for $j = 2 :$ floor$((N-i)/2) + 1$

    $\varphi(j) = \varphi(j-1) + v(N-j-i+2)v(N-j+1) - v(j-1)v(j+i-2)$

    $\varphi(N-i-j+2) = \varphi(j)$

  end

  $\varphi(N-i+1) = \sum_{j=1}^{N-i+1} \varphi(j)$

end

Algorithm 3.2 Vector-storage Algorithm of Solving tr$(C^{-1}P)$

INPUT: vectors φ and p

OUTPUT: scalar α as tr$(C^{-1}P)$

$\alpha = \varphi(N)p_1$

for $i = 2 : N$

  $\alpha = \alpha + 2p_i\varphi(N-i+1)$

end

$\alpha = \alpha / v(N)$

TABLE V

ACCURACY AND EXECUTION TIME OF VECTOR-STORAGE ALGORITHMS

	accuracy	time (in seconds)
$N = 10000$	1.4×10^{-13} (2.0×10^{-13})	29.1 (22.5)
$N = 20000$	2.0×10^{-13} (3.7×10^{-13})	264.7 (154.2)
$N = 30000$	1.4×10^{-13} (2.1×10^{-13})	730.8 (393.3)
$N = 40000$	2.5×10^{-13} (5.6×10^{-13})	1555.3 (784.8)
$N = 50000$	2.8×10^{-13} (1.4×10^{-12})	2497.1 (1339.3)
$N = 60000$	1.4×10^{-13} (2.2×10^{-13})	3641.7 (1957.0)

VI. WIND-TURBINE ROTOR-SPEED DATA

The measurement data for the wind-turbine rotor speed consist of a run of 600 second sampled at 40Hz. The data has a long length-scale component due to variations in the aerodynamic torque, caused by changes in the wind speed and the pitch angle of the rotor blades, and a short length-scale component due to the structural and electro-mechanical dynamics of the machine.

Remark 5. Applying the vector-storage Toeplitz algorithms to the Gaussian process regression of rotor-speed data, we have Fig. 1 where solid lines correspond to the extracted long-scale component of the rotor-speed data $\{y_j\}$. Applying formula (2) further yields the derivative Gaussian process regression, i.e., the extracted rotor acceleration signal from $\{y_j\}$, as shown in Fig. 2.

Remark 6. The extraction is performed very well. Twenty numerical tests, starting from different random initial conditions, all converge to this long length-scale regression result successfully. The average execution time for training and handling the 24000 wind-turbine data is 2.615 hours by using the 'preprocessing' technique of simply subtracting the mean from signal $\{y_j\}$. The average time for computing $\log\det C$, $C^{-1}y$ and tr$(C^{-1}P)$ once in this rotor-speed data regression is 29.35 seconds. Table VI shows that the execution time even could be as small as only 40 minutes (i.e., 0.65 hours) by starting from good random initial conditions. In Table VI, ITE denotes the number of MLE optimization iterations, and FUN denotes the number of involved function/gradient evaluations.

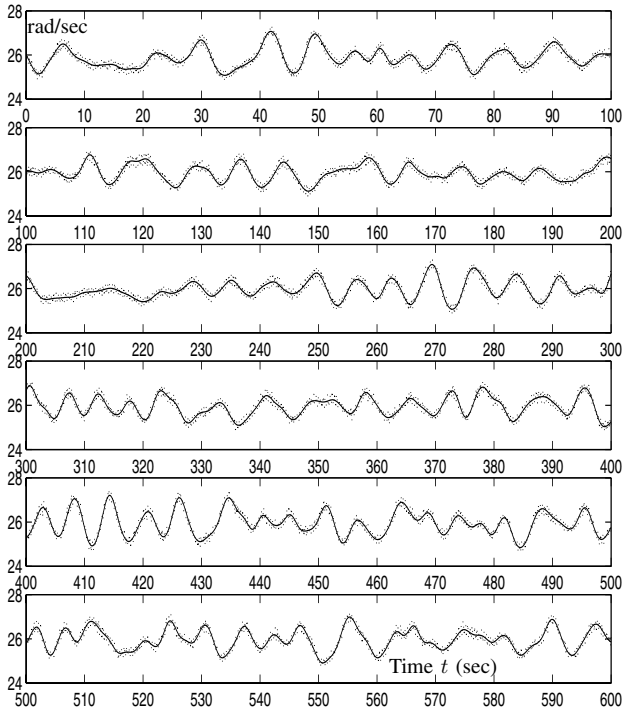

Fig. 1. Extracted long length-scale component from rotor-speed signal $\{y_j\}$, with dotted lines corresponding to $\{y_j\}$.

TABLE VI

NUMERICAL ASPECTS OF THE 24000 ROTOR-SPEED DATA REGRESSION

	Without preprocessing			With preprocessing			
Test #	ITE	FUN	Time in hours	Test #	ITE	FUN	Time in hours
1	170	850	6.35	1	33	165	1.24
2	242	1210	9.03	2	11	55	0.68
3	232	1160	8.22	3	89	445	3.5
4	96	480	3.65	4	16	80	0.95
5	119	595	4.90	5	10	50	0.65
6	60	300	2.50	6	34	170	1.66
7	212	1060	8.63	7	12	60	0.70
8	181	905	7.27	8	90	450	4.49
9	228	1140	8.30	9	17	85	0.99
10	68	340	2.63	10	230	1150	11.29
Average	160.8	804.0	6.148	Average	54.2	271.0	2.615

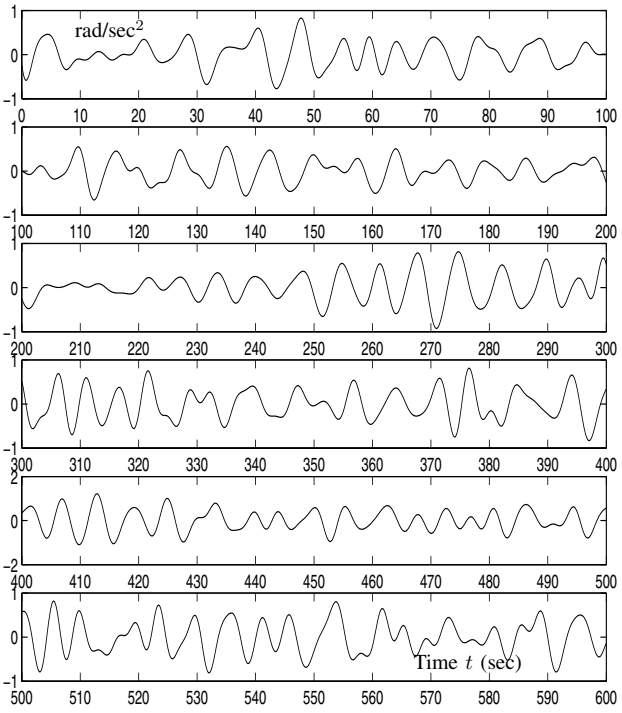

Fig. 2. Rotor acceleration generated from rotor-speed measurements $\{y_j\}$.

VII. CONCLUSIONS

From poor-quality wind-turbine rotor speed measurements, the rotor speed and acceleration have been estimated by using time-series Gaussian process regression. In so doing, efficient algorithms for the manipulation of large matrices $(24,000 \times 24,000)$ have been developed within the Gaussian process regression context. The vector-storage Toeplitz algorithms are of $O(N^2)$ computation complexity and $O(N)$ memory requirement. Numerical experiments, including the wind-turbine example, have substantiated the efficacy and possibility of the Toeplitz-based Gaussian process regression on handling (very) large datasets.

REFERENCES

[1] C.E. Rasmussen, Evaluation of Gaussian processes and other methods for non-linear regression, *Ph.D. thesis*, University of Toronto, Canada, 1996.

[2] D.J.C. MacKay, Introduction to Gaussian processes, *Neural Networks and Machine Learning, F: Computer and Systems Sciences* (C.M. Bishop, Ed.), *168*: 133-165, Springer: Berlin, Heidelberg, 1998.

[3] C.K.I. Williams, Prediction with Gaussian processes: from linear regression to linear prediction and beyond, *Learning in Graphical Models* (Jordan, M. I., Ed.), 599-621, 1999.

[4] S. Sambu, M. Wallat, T. Graepel, & K. Obermayer, "Gaussian process regression: active data selection and test point rejection", *in the IEEE International Joint Conference on Neural Networks*, *3*: 241-246, 2000.

[5] T. Yoshioka, & S. Ishii, "Fast Gaussian process regression using representative data", *in the IEEE International Joint Conference on Neural Networks*, *1*: 132-137, 2001.

[6] D.J. Leith, W.E. Leithead, E. Solak, & R. Murray-Smith, "Divide and conquer identifi cation using Gaussian process priors", *in the 41st IEEE Conference on Decision and Control, 1*: 624-629, 2002.

[7] E. Solak, R. Murray-Smith, W.E. Leithead, D.J. Leith, & C.E. Rasmussen, "Derivative observations in Gaussian process models of dynamic systems", *in Advances in Neural Information Processing Systems* (Becker, S., Thrun, S., & Obermayer, K., Eds.), *15*: 1033-1040, MIT Press, 2003.

[8] W.E. Leithead, F. Hardan, & D.J. Leith, "Identifi cation of aerodynamics and drive-train dynamics for a variable speed wind turbine", *in European Wind Energy Conference,* Madrid, 2003.

[9] A. O'Hagan, On curve fi tting and optimal design for regression, *Journal of Royal Statistical Society, B*, vol. 40, pp. 1-42, 1978.

[10] W. E. Leithead, Y. Zhang, & D. J. Leith, "Effi cient Gaussian process based on BFGS updating and logdet approximation," *in the 16th IFAC world congress,* Prague, July 2005.

[11] Y. Zhang, "Revisit the analog computer and gradient-based neural system for matrix inversion", *in the IEEE International Symposium on Intelligent Control,* pp. 1411-1416, 2005.

[12] G.H. Golub & C.F. Van Loan, *Matrix Computations*. Baltimore: Johns Hopkins University Press, 1996.

[13] The MathWorks, Inc. *MATLAB Optimization Toolbox*, ver. 2.3, 2003.