FAST NUMERICAL ALGORITHMS FOR WIENER SYSTEMS IDENTIFICATION *

Vasile Sima
National Institute for Research & Development in Informatics,
Bd. Mareşal Averescu 8–10, 71316 Bucharest 1
Romania
vsima@iciadmin.ici.ro

Abstract A Wiener system consists of a linear dynamic block followed by a static nonlinearity. The identification of a Wiener system means finding a mathematical model using the input and output data. The approach chosen for identification uses a state space representation for the linear part and a single layer neural network to model the static nonlinearity. Fast subspace identification algorithms are used for estimating the linear part, based on the available input-output data. Using the resulted state-space model, an approximate model of the nonlinear part is found by an improved Levenberg-Marquardt (LM) algorithm. Finally, the whole model is refined using a specialized, MINPACK-like, but structure-exploiting LAPACK-based LM algorithm. The output normal form is used to parameterize the linear part. With a suitable ordering of the variables, the Jacobian matrices have a block diagonal form, with an additional block column at the right. This structure is preserved in a QR factorization with column pivoting restricted to each block column. The implementation is memory conserving and about one order of magnitude faster than standard LM algorithms or specialized LM calculations based on conjugate gradients for solving linear systems.

Keywords: Conjugate gradients, least-squares approximation, Levenberg-Marquardt algorithm, optimization, system identification.

1. Introduction

A discrete-time Wiener system has a state space representation

$$x(k+1) \quad = \quad Ax(k) + Bu(k),$$

*Work partially supported by the European Community BRITE-EURAM III *Thematic Networks Programme NICONET* (project BRRT–CT97-5040).

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35690-7_44
V. Barbu et al. (eds.), *Analysis and Optimization of Differential Systems*

$$\begin{aligned} z(k) &= Cx(k) + Du(k), \\ y(k) &= f\left(z(k)\right) + v(k), \end{aligned} \tag{1}$$

where $x(k) \in \mathbb{R}^n$, $u(k) \in \mathbb{R}^m$, and $y(k) \in \mathbb{R}^\ell$ are the state, input, and output vectors at time k, respectively, $v(k)$ is a zero-mean stochastic disturbance term, A, B, C, and D are real matrices of appropriate dimensions, and $f(\cdot)$ is a nonlinear vector function from $\mathbb{R}^\ell$ to $\mathbb{R}^\ell$. Briefly speaking, a Wiener system consists of a linear dynamic block followed by a static nonlinearity.

The *Wiener system identification problem* can be stated as follows: Given the input and output data sequences of the system (1), $\{u(i)\}$, and $\{y(i)\}$, $i = 1, \ldots, N$, where the input sequence $\{u(k)\}$ is assumed sufficiently persistently exciting, and statistically independent from the perturbation $\{v(k)\}$, find: (a) the order n; (b) an estimate of the quadruple (A, B, C, D) of the Wiener state space model and the initial conditions (up to a similarity transformation); (c) an approximation of $f(\cdot)$.

The approach chosen to solve the above identification problem uses a state space representation for the linear part and a single layer neural network to model the static nonlinearity. Fast subspace identification algorithms are used for estimating the linear part, based on the available input-output data. The resulted state-space model is used for finding an approximate model of the nonlinear part by a Levenberg-Marquardt (LM) algorithm. Finally, the whole model is refined using a specialized, MINPACK-like [6], [7], but structure-exploiting LAPACK-based [1] scaling-invariant LM algorithm. The output normal form is used to parameterize the linear part. The parameters corresponding to the nonlinear part come first in the global parameter vector. Using this ordering, the Jacobian matrices in the multi-output case ($\ell > 1$) have a block diagonal form, with an additional block column at the right. This structure is preserved in a QR factorization with column pivoting restricted to each block column, which makes sense in the identification context. The rank deficient case is also covered. Incremental condition estimation is optionally used for finding the ranks of matrices. The approach is implemented in a specialized toolbox of the freely available **S**ubroutine **L**ibrary **I**n **CO**ntrol **T**heory (**SLICOT**). The Jacobian is computed analytically, for the nonlinear part, and numerically, for the linear part. The implementation is memory conserving and significantly faster than standard LM algorithms or specialized LM calculations based on conjugate gradients (without preconditioning) for solving linear systems.

A systematic approach for solving the Wiener system identification problem is given in [14], and further developed in Section 2. Easy-to-

use software components based on this approach are briefly described in Section 3. Finally, Section 4 presents part of the numerical results obtained using this software.

2. Algorithmic Outline

The Wiener identification problem can be solved in a systematic manner [14], [10]. Algorithmic improvements are summarized below.

Step 1: Estimating the linear part

Given the sequences $\{u(k), y(k)\}$, and assuming that the function $f(\cdot)$ contains odd terms when evaluating its Taylor series expansion, the linear time-invariant (LTI) part of the Wiener system is identified, using one of the subspace identification techniques [13]. If the static nonlinearity $f(\cdot)$ is even, a subspace solution has been provided in a similar way [15]. Therefore, it is possible to identify the linear dynamics as in case the nonlinearities were absent and extract information on the structure of the linear part.

The most time consuming calculation for this step is devoted for finding an upper triangular factor of a QR factorization for a large matrix built from two block-Hankel matrices (with input and output data). Several algorithmic options are available for performing these computations:

- Structure-exploiting correlation calculations and Cholesky factorization [11];
- Fast QR factorization [5];
- Standard QR factorization.

The first two approaches could be one-two orders of magnitude faster than the third approach.

Step 2: Estimating the parameters of $f(\cdot)$

The nonlinear part is modelled as a set of single layer neural networks,

$$f_s\left(z(k)\right) = \hat{f}_s\left(z(k)\right) + \epsilon_s(k)\,, \qquad s = 1, \ldots, \ell\,, \tag{2}$$

$$\hat{f}_s\left(z(k)\right) := \sum_{i=1}^{\nu} \left(\alpha(s,i)\phi \left(\sum_{j=1}^{\ell} \beta(s,i,j) z_j(k) + b(s,i) \right) \right) + b(s,\nu+1)\,.$$

The vector $\epsilon(k)$ is the approximation error. The coefficients $\alpha(s,i)$, $\beta(s,i,j)$, $b(s,i)$ and $b(s,\nu+1)$ are real numbers to be estimated, and

the integer ν represents the number of neurons. These constants are stacked in the parameter vector $\theta \in \mathbb{R}^{\ell((\ell+2)\nu+1)}$,

$$\theta = \left(\theta_1^T \mid \theta_2^T \mid \ldots \mid \theta_\ell^T\right)^T := (\beta(1,1,1),\ldots,\beta(1,\nu,\ell),\ \alpha(1,1),\ldots,\alpha(1,\nu),\ b(1,1),\ldots,b(1,\nu+1) \mid \beta(2,1,1),\ \ldots \mid \ldots \mid \beta(\ell,1,1),\ \ldots)^T ,$$

and are estimated by solving the nonlinear least squares (NLS) problem

$$\min_{\theta} \sum_{k=1}^{N} \left\| \begin{bmatrix} y_1(k) - \hat{f}_1\left(\hat{z}(k)\right) \\ \vdots \\ y_\ell(k) - \hat{f}_\ell\left(\hat{z}(k)\right) \end{bmatrix} \right\|^2 , \tag{3}$$

where $\|.\|$ denotes the Euclidean norm, and the sequence $\{\hat{z}(k)\}_{k=1}^N$ is an estimated output sequence of the linear part, computed using estimates of the quadruple (A, B, C, D) determined in Step 1. Since the parameters appearing in a row of the vector in the square brackets in (3) do not appear in any other row, the above NLS problem can be split into ℓ separate NLS problems, one for each output of the system. Actually, the Jacobian J of the problem (3) is a block diagonal matrix,

$$J = \begin{bmatrix} J_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & J_\ell \end{bmatrix} ,$$

where J_s are full matrices of equal size, which can be computed analytically, if the basis functions were chosen (for instance, $\phi(\cdot) = \tanh(\cdot)$).

This step is referred to as the (nonlinear) initialization step.

Step 3: Estimating all parameters

The estimated system matrices from Step 1 and the estimated parameters in the vector θ from Step 2 are used as initial estimates to compute the parameters in a fully parameterized Wiener system with a fixed order of the state vector. To reduce the number of parameters for the linear part, the so-called *output normal form* parameterization [8], is used; that is, the pair (A, C) is transformed to satisfy

$$A^T A + C^T C = I_n , \tag{4}$$

where $I_n \in \mathbb{R}^{n\times n}$ denotes the identity matrix of order n. The condition (4) requires that the system matrix A be asymptotically stable.

Defining

$$\begin{bmatrix} C \\ A \end{bmatrix} = T_1 T_2 \cdots T_n \begin{bmatrix} 0 \\ I_n \end{bmatrix}, \tag{5}$$

where, for $k = 1, 2, \ldots, n$,

$$T_k = \begin{bmatrix} I_{k-1} & & \\ & U_k & \\ & & I_{n-k} \end{bmatrix} \in \mathbb{R}^{(n+\ell)\times(n+\ell)}, \quad U_k = \begin{bmatrix} -s_k & S_k \\ r_k & s_k^T \end{bmatrix},$$

$$t_k = s_k^T s_k, \quad r_k = \sqrt{1 - t_k}, \quad S_k = I_\ell - \frac{1 - r_k}{t_k} s_k s_k^T,$$

then the pair (A, C) satisfies the condition (4). Each unitary matrix T_k is completely defined by a matrix U_k which is parameterized by the entries of the vector $s_k \in \mathbb{R}^\ell$. Hence, the total number of parameters needed for the pair (A, C) equals $n\ell$. From the formulas above, it follows that the parameter vectors s_k must satisfy $\|s_k\| < 1$. To avoid solving a constrained NLS, a bijective mapping,

$$h : \mathbb{R}^\ell \mapsto U_1(0) \subset \mathbb{R}^\ell, \quad s_k = h(\hat{s}_k), \qquad U_1(0) = \{\, s \mid \|s\| < 1 \,\}, \tag{6}$$

is used. Then, it is possible to perform the optimization with respect to the unconstrained vectors $\hat{s}_k := h^{-1}(s_k)$. The mapping used in the codes, and its inverse, are

$$h(\hat{s}_k) := \frac{2}{\pi} \cdot \frac{\arctan(\|\hat{s}_k\|)}{\|\hat{s}_k\|} \hat{s}_k, \qquad h^{-1}(s_k) = \frac{\tan(\|s_k\| \frac{\pi}{2})}{\|s_k\|} s_k. \tag{7}$$

If the vectors $\hat{s}_k$ together with the entries of the matrix pair (B, D), stored column-wise, are stacked in the vector θ_{on}, then the parameter estimation problem of a fully parameterized Wiener system can be stated as

$$\min_{\theta, \theta_{on}, x(1)} \sum_{k=1}^{N} \|y(k) - \hat{y}(k, \theta, \theta_{on}, x(1))\|^2, \tag{8}$$

where $\hat{y}(k, \theta, \theta_{on}, x(1))$ is the output of the Wiener model

$$\begin{aligned} \hat{x}(k+1) &= A(\theta_{on})\hat{x}(k) + B(\theta_{on})u(k), \quad \hat{x}(1) = x(1), \\ \hat{z}(k) &= C(\theta_{on})\hat{x}(k) + D(\theta_{on})u(k), \\ \hat{y}_s(k, \theta, \theta_{on}, x(1)) &= \hat{f}_s(\hat{z}(k)), \qquad s = 1, \ldots, \ell. \end{aligned}$$

The initial estimates used in this NLS problem are obtained by Steps 1 and 2. The Jacobian matrix for (8) has the structure

$$J = \begin{bmatrix} J_1 & 0 & \cdots & 0 & J_1^l \\ 0 & J_2 & \cdots & 0 & J_2^l \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & J_\ell & J_\ell^l \end{bmatrix} \in \mathbb{R}^{\ell N \times (\ell((\ell+2)\nu+1)+n(\ell+m+1)+\ell m)},$$

where the last block column, with full submatrices, corresponds to the linear part. This block column is computed using a forward-difference approximation. The Cholesky factor R of J^TJ, or the R factor of a QR factorization of J have a similar structure. The implementation stores J and R in a compressed form, using only two block columns.

The numerical optimization

The NLS problems (3) and (8) are solved using Levenberg-Marquardt algorithm [6], [7], [9], which iteratively finds a local minimum of the nonlinear cost function. These problems have the general structure

$$\min_{\Theta} \| y - \hat{y}(\Theta) \|^2 , \tag{9}$$

where Θ is the stacked vector of unknown parameters, y is a given vector (e.g., $\mathrm{vec}([y(k)]_{k=1}^N)$) and $\hat{y}(\Theta)$ is a given function. Define $e(\Theta) = y - \hat{y}(\Theta)$, the *residual* vector, whose components are the *error functions*.

Denoting $\Theta(l)$ the value of the parameter vector in the l-th iteration of the Levenberg-Marquardt algorithm, then this algorithm computes

$$\Theta(l+1) = \Theta(l) + \Delta\Theta(l),$$

with $\Delta\Theta(l)$ the solution of the positive definite set of linear equations

$$\left(J^T(l)J(l) + \mu I\right) \Delta\Theta(l) = -J^T(l)e(\Theta(l)) . \tag{10}$$

The regularization coefficient $\mu \in (0, \infty)$, called *Levenberg factor*, is essential for the convergence of the algorithm. If μ is too small, the algorithm may diverge. For big values of μ, the convergence is very slow. The algorithm tries to keep μ as small as possible. If the cost function decreases, the current step is accepted, and the ratio of the actual decrease compared to the predicted decrease is checked. Then μ is decreased if the ratio was acceptable and increased otherwise. If the cost function increases, the step is rejected and the calculations are repeated with an increased μ. The algorithm convergence close to a local minimizer is quadratic for analytically computed Jacobian matrices, and linear for numerically computed Jacobian matrices [4].

Two versions of the Levenberg-Marquardt algorithm have been implemented: a MINPACK-like implementation, and a standard implementation. The first scheme uses a structure-exploiting QR decomposition with block-column pivoting of the matrix $J(l)$, while the second scheme uses either a Cholesky decomposition of the matrix $\left[J^T(l)J(l) + \mu I\right]$, built using its structure, or a CG algorithm [3] to solve (10).

3. Software Components

The current version of the SLICOT Wiener system identification toolbox consists of over 30 driver and computational routines, six MATLAB interfaces (MEX-files), and associated M-files. Many details are given in [10].

The MEX-files correspond to the Fortran drivers and other essential routines, and are listed in Table 1. The M-files call the MEX-files and are included for user's convenience. The same interfaces could also be used in a Scilab environment [2].

Table 1. Wiener system identification: MEX-file interfaces to MATLAB/Scilab.

MEX-file	*Function*
`widen t`	Computes a discrete-time model of a Wiener system using a neural network approach and a MINPACK-like Levenberg-Marquardt algorithm.
`widentc`	Computes a discrete-time model of a Wiener system using a neural network approach and a Levenberg-Marquardt algorithm, based on either a Cholesky, or a conjugate gradients algorithm for solving (10).
`Wiener`	Computes the output of a Wiener system.
`ldsim`	Computes the output response of a linear discrete-time system (much faster than the MATLAB function lsim).
`onf2ss`	Transforms a linear discrete-time system given in the output normal form to a state-space representation.
`ss2onf`	Transforms a state-space representation of a linear discrete-time system into the output normal form.

The main MEX-files are `widen t` and `widentc`, but the remaining MEX-files offer additional flexibility. Their calling sequences are

```
[xopt(,perf,nf,rcnd)]=widen t(job,u,y,nn(,s,n,x,iter,nprint,
                              tol,seed,printw,ldwork));
[xopt(,perf,nf,rcnd)]=widentc(job,u,y,nn(,s,n,x,alg,stor,
                              iter,nprint,tol,seed,printw,
                              ldwork))
```

where the parameters put inside the brackets are optional, and have default values. The parameters `u` and `y` denote the input and output trajectories, each row containing all input and output values, respectively, measured at the same time moment. The parameter `job` specifies which part of the Wiener parameterization must be initialized: the linear part only (`job` = 1); the static nonlinearity only (`job` = 2); both linear and nonlinear parts (`job` = 3); nothing, `x` already contains an initial approximation for Θ (`job` = 4). The parameters `nn`, `s`, and `n` denote the number of neurons, the number of block rows in the input and

output block-Hankel matrices processed for estimating the linear part (if `job` is 1 or 3), and the order of the linear part (or an option on how to compute it), respectively. The value of `n` should be given if `job` is 2 or 4. If `job` $\neq 3$, the parameter `x` must contain the part of the vector of initial parameters specified by `job`. The parameters `iter` and `tol` are vectors with two elements, giving the maximal number of iterations and the tolerances for the initialization step, and the whole optimization. The parameter `nprint` specifies the frequency of printing the error norm in the iterative process. If `job` is 2 or 3, the parameter `seed` is a vector of length 4 containing the random number generator seed used to initialize the parameters of the static nonlinearity. Using `seed` enables to obtain reproducible results. The parameter `printw` is a switch for printing the warning messages. The parameter `ldwork` allows the user to specify other sizes than the default ones for working arrays. The parameters `alg` and `stor` specify the algorithm for solving (10) (Cholesky or CG), and how the matrix $J^T(l)J(l)$ is stored for Cholesky algorithm (full or packed), respectively.

The parameter `xopt` returns the optimized values of the parameters describing the Wiener system. The optional output parameters `perf`, `nf`, and `rcnd` contain: various performance results, e.g., the maximum residual error norm, the total number of iterations (and CG iterations, if any) performed, the final Levenberg factor; the (total) number of function and Jacobian evaluations; and the reciprocal condition number estimates (if `job` is 1 or 3) for estimating the linear part, respectively.

There are four computational M-files which call some of the MEX-files. Their calling sequences are listed below:

```
y         = NNout(nn,l,wb,u);
[y(,x)]   = dsim(sys,u(,x0));
[sys,x0]  = o2s(n,m,l,theta(,apply));
theta     = s2o(sys,x0(,apply));
```

`NNout` computes the output of a set of neural networks, and the remaining M-files correspond to the last three MEX-files in Table 1. The argument `wb` contains the weights and biases of the neural network. The parameters `x0` and `x` are the initial state (default `x0` $= 0$) and the final state of the system, respectively. The parameter `sys` is a discrete-time `ss` MATLAB system object, consisting in a state-space realization `sys` $= (A, B, C, D)$. It may alternately be replaced by the matrix tuple. The parameter `theta` denotes the parameters of the linear part, in the output normal form parameterization, and `apply` specifies if the bijective mapping (7) should be used or not (default: the mapping is not used). The other arguments have already been defined before.

4. Numerical Results

The first example is also included in the compressed Wiener system identification toolbox file, `Widentmex.zip`, available from the SLICOT ftp site `ftp://wgs.esat.kuleuven.ac.be/`, directory

`pub/WGS/SLICOT/MatlabTools/Windows/SLToolboxes/`

This example has 3 inputs, 2 outputs, and $t = 5000$ input and output data samples. The first 1000 samples were used to estimate the Wiener system, but all samples serve for validating the identified system. The linear system order and the number of neurons in the hidden layer were chosen as $n = 4$ and $\mathtt{nn} = 12$. The corresponding optimization problem has 1000 error functions and 128 unknown variables. The MEX-file `widentc` solved the problem in less than 30 seconds on a 500 MHz PC with 128 Mb memory, for tolerances set to 0.0001. It required 61 iterations for the initialization step, and 12 iterations for the whole optimization. The total number of function and Jacobian evaluations was 431, and 70, respectively. The Euclidean norm of the error was 3.5299 for the linear model, but 1.2914 for the Wiener model. The MEX-file `widentc`, option CG, solved this example in 324 seconds, and the error norm was 1.3066.

Many numerical tests have been performed using the DAISY identification collection (`http://www.esat.kuleuven.ac.be/sista/daisy`). The results enable to compare the implemented algorithms and their options. All algorithms have been initialized with the same seed for the random number generator. Also, the same tolerances (usually, 0.0001) have been used in all compared computations. The most efficient algorithm is problem dependent, but the MINPACK-like approach should be preferred from a numerical point of view. It was almost always the most accurate, and often the most efficient algorithm in our tests.

Some typical results are described below. Table 2 summarizes comparative performance results when using SLICOT MEX-files `widentc` and `widentc` on a set of applications defined in the first three columns of the table. The same numbering scheme for applications as in [12] has been used. The estimation set consisted of the first half of each data set, `nn` was taken as 12, and the data were detrended. The original DAISY Application 16 has 28 outputs, but the first 7 outputs only have been considered, since the calculations are very time consuming. Even with this size reduction, the number of error functions is 4261, and the number of unknown parameters is 977.

As illustrations, the mean values of errors for linear and Wiener identification (computed for a moving window of 40 samples) are plotted in Figure 1 and Figure 2 for Applications 13 and 16, respectively. The improvement when using a nonlinear model is clearly visible.

Table 2. Comparative results for DAISY applications using `widen` and `widentc`.

	Problem size		*Execution time (sec.)*			*Sum of squares*		
#	t	n,m,ℓ	`widen`	*Chol.*	*CG*	`widen`	*Chol.*	*CG*
2	1247	5,3,6	479.72	1158.98	10505.32	1.50e+1	8.39e+0	8.42e+0
5	2001	6,2,1	25.26	32.79	44.27	1.30e+1	1.30e+1	1.31e+1
6	867	10,3,3	360.36	40.43	108.86	5.46e+0	8.59e+0	8.81e+0
7	4000	6,1,1	8.73	17.25	46.57	1.30e+1	1.28e+1	1.28e+1
9	7500	5,1,2	1338.92	211.58	662.45	6.95e+0	1.34e+1	1.30e+1
10	9600	9,4,4	1471.29	588.85	3606.03	6.57e+2	2.10e+3	1.91e+3
11	1000	1,1,1	4.18	0.77	0.83	8.21e-1	9.53e-1	9.53e-1
12	1000	4,1,1	4.28	0.72	0.77	1.75e+0	1.79e+0	1.79e+0
14	1024	6,1,1	4.89	4.23	7.30	3.47e+0	3.39e+0	3.38e+0
15	1024	4,1,1	2.58	9.23	11.54	1.78e-1	1.71e-1	1.71e-1
16	8523	20,2,7	7956.51	3481.84	98595.72	1.55e+2	1.79e+2	1.55e+2
19	1680	3,2,1	6.04	3.63	6.76	3.87e+0	3.82e+0	3.82e+0
20	801	7,1,1	4.89	36.20	86.67	2.06e+1	1.04e+2	6.20e+1
21	99999	2,0,1	25.43	18.29	14.28	75532.0	75533.0	75533.0

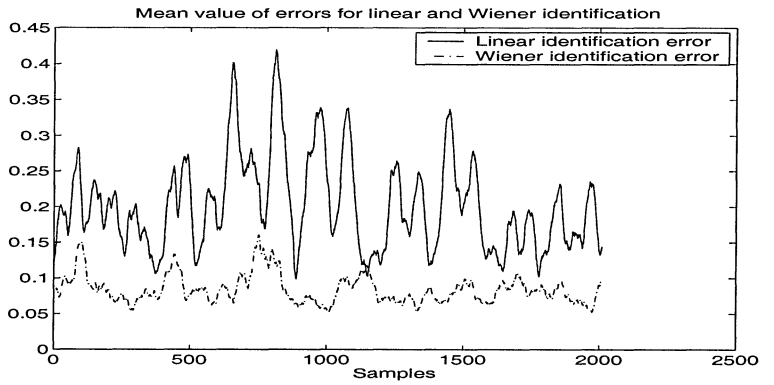

Figure 1. The mean values of errors for linear and Wiener identification, Application 13; all data samples used for estimation.

Finally, Table 3 summarizes comparative performance results when using SLICOT MEX-files `widen` and `widentc` for Application 16 for the first 7 outputs only. The numbers appearing in parantheses are the performance results corresponding to the initialization step.

5. Summary

Algorithmic, implementation and numerical details concerning nonlinear, multivariable Wiener systems identification have been investigated.

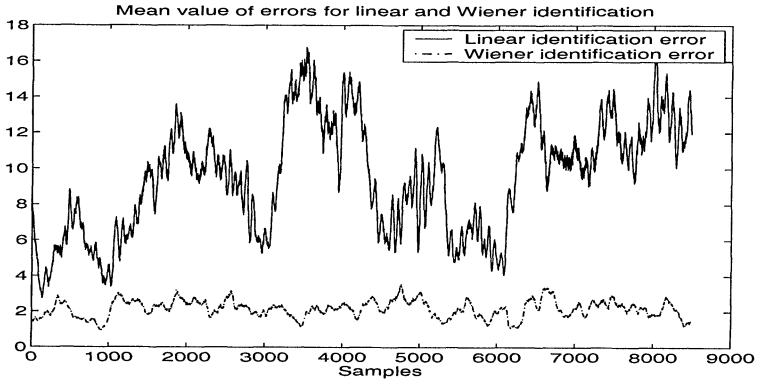

Figure 2. The mean values of errors for linear and Wiener identification, Application 16 (the first 7 outputs only); the first half of the data set used for estimation.

Table 3. Comparative results for Application 16 using `widen`t and `widentc`.

Performance parameter	`widen`t	`widentc`, *Cholesky*	`widentc`, *CG*
Execution time (sec.)	7956.51	3481.84	98595.72
Sum of squares	154.7	179.34	155.33
Number of iterations	31 (157)	19 (178)	64 (214)
Number of CG iterations	0 (0)	0 (0)	96206 (19769)
Total number of function evaluations	6658	4392	14155
Total number of Jacobian evaluations	184	197	278
Euclidean norm of the error for a Wiener model	225.08	249.03	225.7

A systematic three-step procedure for solving this problem has been used. Either a conjugate gradients algorithm or a direct, Cholesky-based algorithm is used for solving the linear systems of equations appearing in the computational process. Alternately, a MINPACK-like, specialized LAPACK-based Levenberg-Marquardt algorithm can be used, which proved to be very accurate and fast. The techniques are implemented in the new nonlinear system identification toolbox for the SLICOT Library. This toolbox includes interfaces (MEX-files and M-files) to the MATLAB and Scilab environments, which improve the user-friendliness of the collection. The results obtained show that the algorithms included in the toolbox are operational, and very fast.

References

[1] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK Users' Guide: Third Edition.* Software · Environments · Tools. SIAM, Philadelphia.

[2] Delebecque, F. and Steer, S. (1997). *Integrated Scientific Computing with Scilab.* Birkhäuser, Boston.

[3] Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations.* M. D. Johns Hopkins University Press, Baltimore, Maryland, third edition.

[4] Kelley, C. T. (1999). *Iterative Methods for Optimization.* SIAM, Philadelphia, PA.

[5] Mastronardi, N., Kressner, D., Sima, V., Van Dooren, P., and Van Huffel, S. (2001). A fast algorithm for subspace state-space system identification via exploitation of the displacement structure. *J. Comput. Appl. Math.*, 132(1):71–81.

[6] Moré, J. J. (1978). The Levenberg-Marquardt algorithm: Implementation and theory. In Watson, G. A., editor, *Numerical Analysis*, volume 630 of *Lecture Notes in Mathematics*, pages 105–116. Springer-Verlag, Berlin, Heidelberg and New York.

[7] Moré, J. J., Garbow, B. S., and Hillstrom, K. E. (1980). User's guide for MINPACK-1. Report ANL-80-74, Applied Math. Division, Argonne National Laboratory, Argonne, Illinois.

[8] Peeters, R., Hanzon, B., and Olivi, M. (1999). Balanced realizations of discrete-time stable all-pass systems and the tangential Schur algorithm. In *Proceedings of the European Control Conference 31 August–3 September 1999, Karlsruhe, Germany.* Session CP-6, Discrete-time Systems.

[9] Press, W. H., Teukolsky, S. A., Wetterling, W. T., and Flannery, B. P. (1992). *Numerical Recipes. The Art of Scientific Computing.* Cambridge University Press, New York, second edition.

[10] Schneider, R., Riedel, A., Verdult, V., Verhaegen, M., and Sima, V. (2002). SLICOT system identification toolbox for nonlinear Wiener systems. SLICOT Working Note 2002-6, Katholieke Universiteit Leuven (ESAT/SISTA), Leuven, Belgium. Available from `ftp://wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/SLWN2002-6.ps.Z`.

[11] Sima, V. (1999). Cholesky or QR factorization for data compression in subspace-based identification ? In *Proceedings of the Second NICONET Workshop on "Numerical Control Software: SLICOT, a Useful Tool in Industry", December 3, 1999, INRIA Rocquencourt, France*, pages 75–80.

[12] Sima, V. and Van Huffel, S. (2001). Performance Investigation of SLICOT System Identification Toolbox, In *Proceedings of the European Control Conference, ECC 2001, 4–7 September, 2001, Seminário de Vilar, Porto, Portugal*, pages 3586-3591.

[13] Verhaegen, M. (1993). Subspace model identification. Part 3: Analysis of the ordinary output-error state-space model identification algorithm. *Int. J. Control*, 58(3):555–586.

[14] Verhaegen, M. (1998). Identification of the temperature-product quality relationship in a multi-component distillation column. *Chemical Engineering Communications*, 163:111–132.

[15] Westwick, D. and Verhaegen, M. (1996). Identifying MIMO Wiener systems using subspace model identification methods. *Signal Processing*, 52:235–258.