SLICOT System Identification Toolbox for Nonlinear Wiener Systems [1]

Rene Schneider[2] Andreas Riedel[2] Vincent Verdult[3] Michel Verhaegen[3]
Vasile Sima[4]

June 2002

[1]This document presents research results of the European Community BRITE-EURAM III Thematic Networks Programme NICONET (contract number BRRT-CT97-5040) and is distributed by the Working Group on Software WGS. *WGS secretariat*: Mrs. Ida Tassens, ESAT - Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001-Leuven-Heverlee, BELGIUM. This report is available by anonymous ftp from `wgs.esat.kuleuven.ac.be/pub/WGS/REPORTS/SLWN2002-6.ps.Z`
[2]Fakultät für Mathematik, TU Chemnitz, D-09107 Chemnitz, Germany
[3]Delft University of Technology, Faculty of Information Technology and Systems, 2600 GA Delft, The Netherlands
[4]National Institute for Research & Development in Informatics, 71316 Bucharest 1, Romania

Contents

1 Introduction

A state space representation of a Wiener system is given as:

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

Here $x(k)$ is the n-dimensional state vector at time k, $u(k)$ is the m-dimensional input (control) vector, $y(k)$ is the ℓ-dimensional output (response) vector, $v(k)$ is a zero-mean stochastic disturbance term, which is statistically independent from $u(t)$ for all k, t pairs. The system matrices A, B, C, D are real matrices of appropriate dimensions and $f(.)$ is a square nonlinear vector function from $\mathbb{R}^\ell$ to $\mathbb{R}^\ell$.

The **Wiener system identification problem** for which this working note presents SLICOT routines is now stated as:

Let the following data sequences of input and output (i/o) data:

$$\begin{bmatrix} u(j) & u(j+1) & \cdots & u(j+N-1) \end{bmatrix}$$

$$\begin{bmatrix} y(j) & y(j+1) & \cdots & y(j+N-1) \end{bmatrix}$$

of the system (1) be given, assume that the input sequence $\{u(k)\}$ is sufficiently persistently exciting as defined in [2] and statistically independent from the perturbation $\{v(k)\}$, the task is to find (a) the order and (b) a statistically consistent estimate of the tuple (A, B, C, D) of the Wiener state space model and the initial conditions (up to a similarity transformation and a scalar input scaling), and (c) an approximation of $f(.)$.

The set-up of this identification problem is represented in the block-scheme of figure 1.

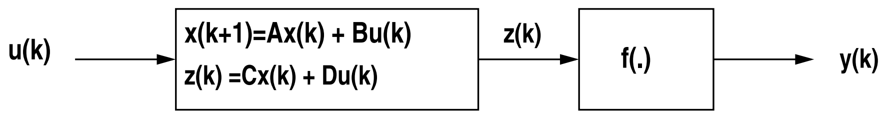

Figure 1: A block-schematic representation of a Wiener system.

A systematic approach to address the above problem is given in [4]. This approach is summarised in Section 2. The structure of the numerical library to identify Wiener systems according to this approach is described in Section 3.1. In this section we summarise the available Fortran 77 routines, developed according to the SLICOT standards. This is done by a brief description of their functionality and an overview of the relevant arguments in the calling list of the essential routines. Section 3.2 provides an interface description enabling the developed Fortran routines to be used in Matlab and Scilab. Finally, Section 4 provides a number of illustrations of the use of the developed software.

2 Outline of the algorithmic steps

In paper [4] the Wiener identification problem is solved in a systematic manner according to the following steps.

2.1 STEP 1: Estimating the linear part of the Wiener system

Given the input and output sequences $\{u(k), z(k)\}$, and assuming the nonlinearity $f(.)$ contains odd terms when evaluating its Taylor series expansion[1], the LTI part of the Wiener system is

[1]If the static nonlinearity $f(.)$ is even, such as for example a quadratic function, a subspace solution has been provided in a similar way as for the mixed odd and even case. For a description of the minor algorithmic modifications we refer to [3].

identified, using the PI or RS [2] subspace identification scheme described in [1] for the identification of MIMO LTI systems. So we can identify the linear dynamics represented by the quadruple (A, B, C, D) as if the nonlinearities were not present and extract information on the structure of the linear dynamics part, such as the order of the state space model and possibly also the dead-time in the input.

2.2 STEP 2: Estimating the parameters of the static nonlinearity $f(.)$

Using estimates of the tuple (A, B, C, D) determined in STEP 1, the sequence $\{\hat{z}(k)\}_{k=1}^{N}$ of this LTI system is computed. With this sequence, finding an initial approximation of the nonlinear part of the Wiener model is just a static nonlinear modelling and estimation problem. This nonlinear part is currently modelled as a single layer neural network, represented as:

$$f_s\left(z(k)\right) = \sum_{i=1}^{\nu}\left(\alpha(s,i)\phi\left(\sum_{j=1}^{\ell}\beta(s,i,j)z_j(k) + b(s,i)\right)\right) + b(s,\nu+1) + \epsilon_s(k)\,. \qquad (2)$$

Here the notation $f_s(.), z_s(k)$ is used to denote the s-th entry of the vector function $f(.)$ and the vector $z(k)$, respectively. Further, 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. The quantity $\epsilon(k)$ is the approximation error. Taking into account the dimensions of the vectors $z(k)$, the integer s runs from 1 to ℓ. The constants $\alpha(s,i), \beta(s,i,j), b(s,i)$ and $b(s,\nu+1)$ in (2) are stacked in the parameter vector $\theta \in \mathbb{R}^{\ell((\ell+2)\nu+1)}$ and are estimated by solving the following nonlinear least squares problem:

$$\min_{\theta} \sum_{k=1}^{N} \left\| \begin{bmatrix} y_1(k) - \sum_{i=1}^{\nu}\left(\alpha(1,i)\phi\left(\sum_{j=1}^{\ell}\beta(1,i,j)z_j(k) + b(1,i)\right)\right) + b(1,\nu+1) \\ \vdots \\ y_\ell(k) - \sum_{i=1}^{\nu}\left(\alpha(\ell,i)\phi\left(\sum_{j=1}^{\ell}\beta(\ell,i,j)z_j(k) + b(\ell,i)\right)\right) + b(\ell,\nu+1) \end{bmatrix} \right\|^2 . \qquad (3)$$

2.3 STEP 3: Estimating the parameters of the full Wiener system

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' parametrised Wiener system with a fixed order of the state vector. In addition to the parameters for the static nonlinearity $f(.)$ defined in STEP 2 in Section 2.2, the full parametrisation of the Wiener system (1) requires a parametrisation of the system matrices $[A, B, C, D]$ and the initial value of the state vector $x(1)$ in (1). Here we consider the pair (A, C) to be parametrised in the so-called output normal form. For MIMO discrete-time state space models the parametrisation was introduced in [5].

[2]The abbreviations PI and RS stand for Past Input and Reconstructed State instrumental variable Subspace Identification methods, respectively, proposed in [1]. The current SLICOT implementations use the PO, i.e., Past input - past Output scheme, which will lead to biased estimates, but it is planned to include the PI scheme in the future.

Definition 1 *The pair (A, C) of linear state space model defined by the quadruple of system matrices $[A, B, C, D]$ is in output normal form if the pair (A, C) satisfies:*

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

where $I_n \in \mathbb{R}^{n \times n}$ denotes the identity matrix of order n.

The definition implicitly constraints the underlying system matrix A to be asymptotically stable.

A unique parametrisation, which has this property, is defined via a sequence of unitary matrices for $k = 1, 2, \dots, n$:

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

The pair (A, C) defined as

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

satisfies the output normal condition (4) in definition 1. Each matrix T_k is completely defined by a matrix U_k which is parametrised by the entries of the vector $s_k \in \mathbb{R}^\ell$. Since $k = 1, 2, \cdots, n$ the total number of parameters to represent the pair (A, C) equals $n\ell$. From the formulas above, it follows that the parameter vectors s_k are constraint to satisfy $\|s_k\| < 1$. This is not a very nice property, at least if one wants to optimize with respect to s_k. To get rid of this constraint one can use a bijective mapping,

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

Then, it is possible to perform the optimization with respect to $\hat{s}_k := h^{-1}(s_k)$. The vectors $\hat{s}_k$ are not constrained at all. For example, the bijective mapping can be chosen to be

$$h(\hat{s}_k) := \frac{2}{\pi} \cdot \frac{\arctan(\|\hat{s}_k\|)}{\|\hat{s}_k\|} \hat{s}_k \,, \tag{7}$$

where $\|.\|$ is denoting the Euclidean norm (2-norm). Then, the inverse mapping would be

$$h^{-1}(s_k) = \frac{\tan(\|s_k\| \frac{\pi}{2})}{\|s_k\|} s_k \,. \tag{8}$$

This is actually the mapping which is used in the code.

Let these parameter vectors $\hat{s}_k$ together with all the entries of the matrix pair (B, D) (stored column wise) be stacked in the vector θ_{on}, then the parameter estimation problem of a fully parametrised Wiener system can be formulated as:

$$\min_{x(1), \theta_{on}, \theta} \sum_{k=1}^{N} \|y(k) - \hat{y}(k, x(1), \theta_{on}, \theta)\|^2 , \tag{9}$$

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

$$\begin{aligned}
\hat{x}(k+1) &= A(\theta_{on})\hat{x}(k) + B(\theta_{on})u(k), \quad \hat{x}(1) = x(1), \\
\hat{z}(k) &= C(\theta_{on})\hat{x}(k) + D(\theta_{on})u(k), \\
\hat{y}_s(k, x(1), \theta_{on}, \theta) &= \sum_{i=1}^{\nu} \left(\alpha(s,i)\phi \left(\sum_{j=1}^{\ell} \beta(s,i,j)\hat{z}_j(k) + b(s,i) \right) \right) + b(s, \nu+1).
\end{aligned}$$

The initial estimates used in this nonlinear least squares problem are derived by solving STEPS 1 and 2.

Remark 1 *In case the output happens to be a scalar quantity, it would be possible to receive information on the nature of the static nonlinearity just by plotting the pairs* $(y(k), \hat{z}(k))$. *Such a plot may help in the selection of the basis functions in parametrising the static nonlinearity.*

Remark 2 *A more general representation of the Wiener system could be used instead of* (1). *This representation has the following form:*

$$\begin{aligned}
x(k+1) &= Ax(k) + Bu(k) \\
y(k) &= f(x(k)) + v(k)
\end{aligned} \tag{10}$$

The subspace method PI *and* RS *described in [1] is also able to estimate the pair* (A, B) *making use of the input-output sequences* $(u(k), y(k))$. *The nonlinear least squares problem would then require the estimation of the parameters to parametrise this pair in addition to the parameters to parametrise the non-square nonlinear mapping* $f(.)$. *For the parametrisation of the pair* (A, B) *we could use the input normal form, dual to the form defined in Definition 1.*

2.4 The numerical optimization

In case we treat the problems (3) and (9) as unconstraint least squares problems, we initially propose to use the Levenberg-Marquardt algorithm [6]. This algorithm iteratively finds a local minimum of the nonlinear cost function.

These problems have the general structure

$$\min_{\Theta} \|y - \hat{y}(\Theta)\|_2^2 \,, \tag{11}$$

where Θ is the stacked vector of unknown parameters, y is the target vector (e.g., $\text{vec}([y(k)]_{k=1}^N)$ in (9)) and $\hat{y}(\Theta)$ is a given function ($\text{vec}\big([\hat{y}(k, \hat{z}(k), \theta)]_{k=1}^N\big)$ in example 1). For shorter notation: $e(\Theta) = y - \hat{y}(\Theta)$, with $e(.)$ called the *error functions*, or the *residual* vector.

Let the value of this parameter vector in the t-th iteration of the Levenberg-Marquardt algorithm be denoted by $\Theta(t)$, then this iterative algorithm is represented as:

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

with $\Delta\Theta(t)$ the solution of the set of linear equations

$$\left(J^T(t)J(t) + \mu I\right)\Delta\Theta(t) = -J^T(t)e(\Theta(t)), \tag{12}$$

where $J(t)$ is the Jacobian matrix which consists of the following partial derivatives:

$$J_{ij}(t) := \frac{\partial \hat{y}_i(\Theta(t))}{\partial \Theta_j}, \quad i = 1 : N, j = 1 : \text{length}(\Theta).$$

The regularisation 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, which is indicated by increase of the cost function. For big values of μ, the convergence is very slow. So we pursue to keep μ as small as possible: In each step we test if the cost function decreases. If it decreases we accept this step, and check the ratio of the actual decrease compared to the predicted decrease (see [9]). Then μ is decreased if the ratio was acceptable and increased otherwise. If the cost function does not decrease, we reject this step and repeat with an increased μ.

In [9], it was shown that the convergence of the algorithms close to a local minimizer is quadratic for analytically computed Jacobian and linear for numerically computed Jacobian matrices.

Example 1 *Let f_s be defined as in* (2) *with basis functions $\phi(.) = \tanh(.)$, $((z_k, y_k))_{k=1}^N, z_k \in \mathbb{R}^\ell, y_k \in \mathbb{R}^\ell$ a set of N input/output samples. Now, the residual vector e can be denoted as*

$$e(\Theta) = \begin{bmatrix} y_{1,1} - f_1(z_1) \\ \vdots \\ y_{1,N} - f_1(z_N) \\ y_{2,1} - f_2(z_1) \\ \vdots \\ y_{\ell,N} - f_\ell(z_N) \end{bmatrix}.$$

The vector Θ in this case can be partitioned as:

$$\Theta = \left(\begin{array}{c} \beta(1,1,1) \\ \vdots \\ \beta(1,\nu,\ell) \\ \alpha(1,1) \\ \vdots \\ \alpha(1,\nu) \\ b(1,1) \\ \vdots \\ b(1,\nu+1) \\ \hline \beta(2,1,1) \\ \vdots \\ \hline \vdots \end{array}\right) = \left(\begin{array}{c} \theta_1 \\ \hline \theta_2 \\ \hline \vdots \end{array}\right).$$

Now the Jacobian can be computed analytically. Since each of the functions f_s only depends on one part of the parameter vector, J is a block diagonal matrix

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

where the blocks J_s are full matrices (which can be computed analytically). For example, the matrix J_1 has the following form:

$$J_1 = \begin{bmatrix} \frac{\partial f_1(z_1)}{\partial \beta(1,1,1)} & \dots & \frac{\partial f_1(z_1)}{\partial b(1,\nu+1)} \\ \vdots & & \vdots \\ \frac{\partial f_1(z_N)}{\partial \beta(1,1,1)} & \dots & \frac{\partial f_1(z_N)}{\partial b(1,\nu+1)} \end{bmatrix}.$$

An essential step to make the algorithm efficient is to use a good solver for (12), which exploits the sparse structure in the matrix J. Since the regularisation coefficient μ in (12) can always make the system matrix $\left[J^T(t)J(t) + \mu I\right]$ positive definite, we propose the use of the Conjugate Gradients method as described in Section 2.5. It has the advantage that one does not need to compute the system matrix, but only has to provide a multiplication with it. In our case it means that we do not have to do the matrix-matrix multiplication $J^T(t)J(t)$, but only two matrix-vector multiplications $J(t)\Delta\Theta$ and then $J^T(t)(J(t)\Delta\Theta)$ in each CG step. Since we usually need much less CG steps than the size of the parameter vector, this is much cheaper – often we solve the equations system in less time than we would need only to compute the system matrix itself.

2.5 Conjugate gradients method (CG)

Let $A \in \mathbb{R}^{n,n}$ be symmetric positive definite and $b \in \mathbb{R}^n$. To solve the linear equation system $Ax = b$, we use the following Algorithm, called the Conjugate Gradients Method [7]:

Algorithm 1

$$\begin{aligned} k &= 0 \\ q_0 = r_0 &= Ax - b \\ &\mathit{WHILE}\ \langle r, r\rangle > \epsilon^2 \\ \alpha_k &= \tfrac{\langle q_k, r_k\rangle}{\langle q_k, Aq_k\rangle} \\ x_{k+1} &= x_k - \alpha_k q_k \\ r_{k+1} &= r_k - \alpha_k A q_k \\ \beta_k &= \tfrac{\langle r_{k+1}, r_{k+1}\rangle}{\langle r_k, r_k\rangle} \\ q_{k+1} &= r_{k+1} + \beta_k q_k \\ &\mathit{END} \end{aligned}$$

It costs 1 matrix-vector multiplication and 4 vector-vector operations per step. Since the residuals are orthogonal in the scalar product $\langle x, y\rangle_A = y^T Ax$, the algorithm is theoretically finite. But rounding errors cause a loss of orthogonality, so a finite termination cannot be guaranteed. However, one can prove [8] that

$$\|x - x_k\|_A := \sqrt{(x - x_k)^T A (x - x_k)} \leq 2\|x - x_0\|_A \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k .$$

2.6 Other algorithms

Our numerical tests have shown that alternate algorithms, also implemented in the SLICOT Library, could sometimes provide sensibly faster results than the CG method. These algorithms are based on structure-exploiting QR decomposition of the Jacobian matrix $J(t)$ or Cholesky decomposition of the matrix matrix $\left[J^T(t)J(t) + \mu I\right]$, built using the structure.

3 SLICOT Wiener System Identification Toolbox: Software Components

The current version of the SLICOT nonlinear identification toolbox currently consists of over 30 user-callable and computational routines, and six MATLAB interfaces (Mex-files). Included are the auxiliary routines, belonging to the Mathematical Routines, or Transformation Routines chapters of the SLICOT Library, which have been developed for the nonlinear identification toolbox.

3.1 Fortran Routines

The computational and driver Fortran routines are listed in Tables 1 and 2, respectively, together with their function.

Table 1: The computational routines

Routine	Function
`MB02WD`	Solves a system of linear equations with a symmetric positive definite system matrix using the conjugate gradients method summarised in Section 2.5.
`MB02XD`	Solves a set of systems of linear equations, $A^T AX = B$, or, in the implicit form, $f(A)X = B$, with $A^T A$ or $f(A)$ positive definite, using symmetric Gaussian elimination (Cholesky decomposition).
`MB02YD`	Solves a system of linear equations $Ax = b$, $Dx = 0$, in the least squares sense, with D a diagonal matrix, given a QR factorization with column pivoting of A.

Table 1: (continued)

Routine	Function
MB04OW	Performs a special QR factorization $$\left[\begin{array}{c} U \\ x^T \end{array}\right] = Q \left[\begin{array}{c} R \\ 0 \end{array}\right],$$ where $$U = \left[\begin{array}{cc} U_1 & U_2 \\ 0 & T \end{array}\right], \quad R = \left[\begin{array}{cc} R_1 & R_2 \\ 0 & R_3 \end{array}\right],$$ and where U and R are $(m+n) \times (m+n)$ upper triangular matrices, x is an $(m+n)$-vector, U_1 is $m \times m$, T is $n \times n$, stored separately, and Q is an $(m+n+1) \times (m+n+1)$ orthogonal matrix. This routine actually performs an update of the Cholesky decomposition.
MD03AD	Optimises the parameters Θ for a function $f(z, \Theta)$, so that they give the best approximation for $y = f(z, \Theta)$ in a least squares sense. Either a conjugate gradients algorithm or a Cholesky decomposition algorithm is used for solving the linear systems involved.
MD03BA	Interface to SLICOT Library routine MD03BX, for solving standard nonlinear least squares problems using SLICOT routine MD03BD.
MD03BB	Interface to SLICOT Library routine MD03BY, for solving standard nonlinear least squares problems using SLICOT routine MD03BD.
MD03BD	Optimises the parameters Θ for a function $f(z, \Theta)$, so that they give the best approximation for $y = f(z, \Theta)$ in a least squares sense. A MINPACK-like, but structure-exploiting LAPACK-based Levenberg-Marquardt algorithm is used.
MD03BF	Template routine to evaluate the functions and Jacobian matrices for solving a standard nonlinear least squares problem using SLICOT Library routine MD03BD.
MD03BX	Computes the QR factorization with column pivoting of an $m \times n$ matrix J $(m \geq n)$, that is, $JP = QR$, where Q is a matrix with orthogonal columns, P a permutation matrix, and R an upper trapezoidal matrix with diagonal elements of nonincreasing magnitude, and applies the transformation Q^T on the error vector e. The 1-norm of the scaled gradient is also returned.
MD03BY	Finds a value for the parameter λ such that if x solves the system $Ax = b$, $\lambda^{1/2} Dx = 0$, in the least squares sense, where A is an $m \times n$ matrix, D is an $n \times n$ nonsingular diagonal matrix, and b is an m-vector, and if δ is a positive number, then either $\lambda = 0$ and $(\\|Dx\\|_2 - \delta) \leq 0.1\delta$, or $\lambda > 0$ and $\vert\, \\|Dx\\|_2 - \delta \,\vert \leq 0.1\delta$. It is assumed that a QR factorization with column pivoting of A is available, that is, $AP = QR$, where P is a permutation matrix, Q has orthogonal columns, and R is an upper triangular matrix with diagonal elements of nonincreasing magnitude.
NF01AD	Evaluates the output of a Wiener system (with the linear part given by its output normal form parameters) for given input data (see end of Section 2.3).
NF01AY	Evaluates the function $f(y, \Theta)$ for (2) in Section 2.2.

Table 1: (continued)

<table>
<tr><th>Routine</th><th>Function</th></tr>
<tr><td>NF01BA</td><td>Template routine to evaluate the functions and Jacobian matrices for optimizing the parameters of the nonlinear part of a Wiener system (initialization phase), using SLICOT Library routine MD03AD.</td></tr>
<tr><td>NF01BB</td><td>Template routine to evaluate the functions and Jacobian matrices for optimizing all parameters of a Wiener system, using SLICOT Library routine MD03AD.</td></tr>
<tr><td>NF01BD</td><td>A companion of NF01AD, which computes the Jacobian of a Wiener system (this is done analytically using NF01BY, for the parts corresponding to the parameters of the nonlinearity, and numerically by forward difference coefficient method, for the rest).</td></tr>
<tr><td>NF01BE</td><td>Template routine to evaluate the functions and Jacobian matrices for optimizing the parameters of the nonlinear part of a Wiener system (initialization phase), using SLICOT Library routine MD03BD.</td></tr>
<tr><td>NF01BF</td><td>Template routine to evaluate the functions and Jacobian matrices for optimizing all parameters of a Wiener system, using SLICOT Library routine MD03BD.</td></tr>
<tr><td>NF01BP</td><td>Determines a value for the Levenberg-Marquardt parameter λ such that if x solves the system $Jx = b$, $\lambda^{1/2}Dx = 0$, in the least squares sense, where J is an $m \times n$ matrix, D is an $n \times n$ nonsingular diagonal matrix, and b is an m-vector, and if δ is a positive number, then either $\lambda = 0$ and $(\|Dx\|_2 - \delta) \le 0.1\delta$, or $\lambda > 0$ and $|\,\|Dx\|_2 - \delta\,| \le 0.1\delta$. It is assumed that a QR factorization with block column pivoting of J is available, that is, $JP = QR$, where P is a permutation matrix, Q has orthogonal columns, and R is an upper triangular matrix with diagonal elements of nonincreasing magnitude for each block.</td></tr>
<tr><td>NF01BQ</td><td>Determines a vector x which solves the system of linear equations $Jx = b$, $Dx = 0$, in the least squares sense, with D a diagonal matrix, given a QR factorization with block column pivoting of J. The matrix J is the current Jacobian of a nonlinear least squares problem, provided in a compressed form by SLICOT Library routine NF01BD. It is assumed that a block QR factorization, with column pivoting, of J is available.</td></tr>
<tr><td>NF01BR</td><td>Solves one of the systems of linear equations $Rx = b$, or $R^T x = b$, in the least squares sense, where R is an $n \times n$ block upper triangular matrix, with the structure
$$\left[\begin{array}{cccc|c} R_1 & 0 & \cdots & 0 & L_1 \\ 0 & R_2 & \cdots & 0 & L_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & R_\ell & L_\ell \\ 0 & 0 & \cdots & 0 & R_{\ell+1} \end{array}\right],$$
with the upper triangular submatrices R_k, $k = 1 : \ell + 1$, square, and the first ℓ of the same order. The diagonal elements of each block R_k have nonincreasing magnitude. The matrix R is stored in a compressed form.</td></tr>
</table>

Table 1: (continued)

Routine	Function
NF01BS	Computes the QR factorization with block column pivoting of an $m \times n$ matrix J ($m \geq n$), that is, $JP = QR$, where Q is a matrix with orthogonal columns, P a permutation matrix, and R an upper trapezoidal matrix with diagonal elements of nonincreasing magnitude for each block, and applies the transformation Q^T on the error vector e. The 1-norm of the scaled gradient is also returned.
NF01BU	Computes the matrix $J^TJ + cI$, for the Jacobian J given in a compressed form.
NF01BV	Computes the matrix $J^TJ + cI$, for the Jacobian J fully given, for one output variable.
NF01BW	Computes the matrix-vector product $x \leftarrow (J^TJ + cI)x$, where J is given in a compressed form.
NF01BX	Computes $x \leftarrow (A^TA + cI)x$, where A is an $m \times n$ real matrix, and c is a scalar.
NF01BY	Computes analytically the Jacobian for (2); a companion of NF01AY, it is simply the implementation of the calculated Jacobian.
TB01VD	Determines the parameter vector θ of a given quadruple $[A, B, C, D]$ (output normal form parameters).
TB01VY	Determines the system matrices $[A, B, C, D]$ from a parameter vector θ (output normal form parameters).
TF01MX	Computes the output sequence of a linear time-invariant open-loop system given by its discrete-time state-space model with an $(n + p) \times (n + m)$ general system matrix S (the input and output trajectories are stored differently from SLICOT Library routine TF01MD).
TF01MY	Computes the output sequence of a linear time-invariant open-loop system given by its discrete-time state-space model (A, B, C, D), where A is an $n \times n$ general matrix (the input and output trajectories are stored differently from SLICOT Library routine TF01MD).

The design of these subroutines aimed to provide the user the most convenient and flexible way to perform the identification calculations. Since the documentation is quite involved, we will only include here an overview of the main abilities of the driver identification routines, listed in Table 2. Full details are easily available using the on-line html documentation provided at the SLICOT ftp site.

The subroutine statements for the two drivers are as follows:

```
      SUBROUTINE IB03AD( INIT, ALG, STOR, NOBR, M, L, NSMP, N, NN,
     $                   ITMAX1, ITMAX2, NPRINT, U, LDU, Y, LDY, X, LX,
     $                   TOL1, TOL2, IWORK, DWORK, LDWORK, IWARN, INFO )
      SUBROUTINE IB03BD( INIT, NOBR, M, L, NSMP, N, NN, ITMAX1, ITMAX2,
     $                   NPRINT, U, LDU, Y, LDY, X, LX, TOL1, TOL2,
     $                   IWORK, DWORK, LDWORK, IWARN, INFO )
```

and the specification of the arguments is

Table 2: The driver routines

Routine	Function
IB03AD	Driver routine for identification of the parameter vector for a Wiener system in a least-squares sense, using a neural network approach and conjugate gradients or Cholesky algorithms for solving positive definite systems of linear equations.
IB03BD	Driver routine for identification of the parameter vector for a Wiener system in a least-squares sense, using a neural network approach and a MINPACK-like, structure-exploiting Levenberg-Marquardt algorithm.

```
C     .. Scalar Arguments ..
      CHARACTER         ALG, INIT, STOR
      INTEGER           INFO, ITMAX1, ITMAX2, IWARN, L, LDU, LDWORK,
     $                  LDY, LX, M, N, NN, NOBR, NPRINT, NSMP
      DOUBLE PRECISION  TOL1, TOL2
C     .. Array Arguments ..
      DOUBLE PRECISION  DWORK(*), U(LDU, *), X(*), Y(LDY, *)
      INTEGER           IWORK(*)
```

The available options of the driver IB03AD, which can be selected using character strings as actual arguments in the call,[3] are:

INIT Specifies which parts have to be initialized, as follows:
= 'L': initialize the linear part only, X already contains an initial approximation of the nonlinearity;
= 'S': initialize the static nonlinearity only, X already contains an initial approximation of the linear part;
= 'B': initialize both linear and nonlinear parts;
= 'N': do not initialize anything, X already contains an initial approximation.
If INIT = 'S' or 'B', the error functions for the nonlinear part, and their Jacobian matrices, are evaluated by SLICOT Library routine NF01BA (used as a second FCN routine in the MD03AD call for the initialization step).

ALG Specifies the algorithm used for solving the linear systems involving a Jacobian matrix J, as follows:
= 'D': a direct algorithm, which computes the Cholesky factor of the matrix $J^T J + \mu I$ is used, where μ is the Levenberg factor;
= 'I': an iterative Conjugate Gradients algorithm, which only needs the matrix J, is used.
In both cases, matrix J is stored in a compressed form.

STOR If ALG = 'D', specifies the storage scheme for the symmetric matrix $J^T J$, as follows:
= 'F': full storage is used;

[3]One could use, for instance, 'Direct algorithm' for the parameter ALG, to improve the readability of the code.

= 'P': packed storage is used.
The option STOR = 'F' usually ensures a faster execution. This parameter is not relevant if ALG = 'I'.

The input-output parameters for both drivers are described as follows:

NOBR If INIT = 'L' or 'B', NOBR is the number of block rows, s, in the input and output block Hankel matrices to be processed for estimating the linear part. NOBR > 0.
This parameter is ignored if INIT is 'S' or 'N'.

M The number of system inputs. M ≥ 0.

L The number of system outputs. L ≥ 0, and L > 0, if INIT = 'L' or 'B'.

NSMP The number of input and output samples, t.
NSMP ≥ 0, and NSMP $\geq$ 2*(M+L+1)*NOBR - 1, if INIT = 'L' or 'B'.

N The order of the linear part. If INIT = 'L' or 'B', and N < 0 on entry, the order is assumed unknown and it will be found by the routine. Otherwise, the input value will be used. If INIT = 'S' or 'N', N must be non-negative. The values N $\geq$ NOBR, or N $= 0$, are not acceptable if INIT = 'L' or 'B'.

NN The number of neurons which shall be used to approximate the nonlinear part. NN ≥ 0.

ITMAX1 The maximum number of iterations for the initialization of the static nonlinearity.
This parameter is ignored if INIT is 'N' or 'L'. Otherwise, ITMAX1 ≥ 0.

ITMAX2 The maximum number of iterations. ITMAX2 ≥ 0.

NPRINT This parameter enables controlled printing of iterates if it is positive. In this case, FCN is called with IFLAG = 0 at the beginning of the first iteration and every NPRINT iterations thereafter and immediately prior to return, and the current error norm is printed. Other intermediate results could be printed by modifying the corresponding FCN routine. If NPRINT ≤ 0, no special calls of FCN with IFLAG $= 0$ are made.

U The leading NSMP-by-M part of this array must contain the set of input samples,
U = (U(1,1),...,U(1,M); ...; U(NSMP,1),...,U(NSMP,M)).

LDU The leading dimension of array U. LDU $\geq$ MAX(1,NSMP).

Y The leading NSMP-by-L part of this array must contain the set of output samples,
Y = (Y(1,1),...,Y(1,L); ...; Y(NSMP,1),...,Y(NSMP,L)).

LDY The leading dimension of array Y. LDY $\geq$ MAX(1,NSMP).

X On entry, if INIT = 'L', the leading (NN*(L+2) + 1)*L part of this array must contain the initial parameters for the nonlinear part of the system.
On entry, if INIT = 'S', the elements lin1 : lin2 of this array must contain the initial parameters for the linear part of the system, corresponding to the output normal form, computed by SLICOT Library routine TB01VD, where

```
lin1 = (NN*(L+2) + 1)*L + 1;
lin2 = (NN*(L+2) + 1)*L + N*(L+M+1) + L*M.
```

On entry, if `INIT = 'N'`, the elements `1 : lin2` of this array must contain the initial parameters for the nonlinear part followed by the initial parameters for the linear part of the system, as specified above.
This array need not be set on entry if `INIT = 'B'`.
On exit, the elements `1 : lin2` of this array contain the optimal parameters for the nonlinear part followed by the optimal parameters for the linear part of the system, as specified above.

`LX` On entry, this parameter must contain the intended length of `X`. If `N` ≥ 0, then `LX` $\geq$ `NX := lin2` (see parameter `X`). If `N` is unknown (`N` < 0 on entry), a large enough estimate of `N` should be used in the formula of `lin2`.
On exit, if `N` < 0 on entry, but `LX` is not large enough, then this parameter contains the actual length of `X`, corresponding to the computed `N`. Otherwise, its value is unchanged.

The tolerance parameters for the driver `IB03BD` are described as follows:

`TOL1` If `INIT = 'S'` or `'B'` and `TOL1` ≥ 0, `TOL1` is the tolerance which measures the relative error desired in the sum of squares, as well as the relative error desired in the approximate solution, for the initialization step of nonlinear part. Termination occurs when either both the actual and predicted relative reductions in the sum of squares, or the relative error between two consecutive iterates are at most `TOL1`. If the user sets `TOL1 < 0`, then `SQRT(EPS)` is used instead `TOL1`, where `EPS` is the machine precision (see LAPACK Library routine `DLAMCH`). This parameter is ignored if `INIT` is `'N'` or `'L'`.

`TOL2` If `TOL2` ≥ 0, `TOL2` is the tolerance which measures the relative error desired in the sum of squares, as well as the relative error desired in the approximate solution, for the whole optimization process. Termination occurs when either both the actual and predicted relative reductions in the sum of squares, or the relative error between two consecutive iterates are at most `TOL2`. If the user sets `TOL2` < 0, then `SQRT(EPS)` is used instead `TOL2`. This default value could require many iterations, especially if `TOL1` is larger. If `INIT = 'S'` or `'B'`, it is advisable that `TOL2` be larger than `TOL1`, and spend more time with cheaper iterations.

Note: The description of the parameters `TOL1` and `TOL2` for the routine `IB03AD` is slightly different. In particular, these tolerances also refer to the CG algorithm, when `ALG = 'I'`.

The arguments `IWORK` and `DWORK` are working arrays, and `LDWORK` is the length of `DWORK`. Finally, the arguments `IWARN` and `INFO` are the warning and error indicators, respectively.

3.2 Interfaces: Mex-files and m-files

The SLICOT system identification routines can be used from a MATLAB environment using the interfaces described in this section. Both Mex-files and m-files are provided in the toolbox. The Mex-files correspond to the Fortran drivers and other essential routines. Their interface is more complicated than that of the m-files, but they provide more flexibility and generality.

The m-files call the Mex-files and are included for user's convenience. On line information about the Mex-files and m-files can be obtained in the usual manner, e.g., by typing `help wident`, for getting details about the Mex-file `wident`. The same interfaces could also be used in a Scilab environment. The Mex-files are listed in Table 3.

Table 3: SLICOT Wiener system identification: Mex-file interfaces to MATLAB/Scilab.

Mex-file	Function
`wident`	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 conjugate gradients algorithm for solving linear systems of equations.
`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 `wident` and `widentc`, but the remaining Mex-files offer additional working flexibility.

The Mex-files and m-files include several options which could be of interest for the user; all these options have default values, and hence the calling statements could be very simple. One such option is `printw`, a switch for printing the warning messages; it should be set to 1 to print these messages, or to 0, otherwise (the default).

The calling sequences for the Mex-files are shown below.

```
[xopt(,perf,nf,rcnd)] = wident(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))
y                     = Wiener(n,l,nn,theta,u(,ldwork));
[y(,x)]               = ldsim( A,B,C,D,u(,x,ldwork));
[A,B,C,D,x0]          = onf2ss(n,m,l,theta(,apply));
theta                 = ss2onf(A,B,C,D,x0(,apply));
```

where the parameters put inside the brackets are optional, possibly context-dependent. Most of these parameters have clear meaning, or details have been already given in Section 3.1. For instance, the parameters `u` and `y` denote the input and output trajectories; it is assumed that each row of the associated matrices contains all input and output values, respectively, measured at the same time moment. The parameter `job` specifies which part of the Wiener parametrisation must be initialized:

`job` $= 1$: initialize the linear part only;

`job` $= 2$: initialize the static nonlinearity only;

`job` $= 3$: initialize both linear and nonlinear parts;

`job` $= 4$: do not initialize anything, `x` already contains an initial approximation.

The parameters `nn`, `s`, and `n` denote the number of neurons to use for modelling the nonlinear part of the Wiener system, the number of block rows (`NOBR` in Section 3.1) in the input and output block Hankel matrices to be 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. If `job` is 1 or 3, and `n` < 0, the order will be found by the program. Otherwise, `n` should be non-negative (or positive, but less than `s`, if `job` is 1 or 3). 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, corresponding to [`ITMAX1`, `ITMAX2`], and [`TOL1`, `TOL2`], respectively. 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 `ldwork` allows the user to specify other sizes than the default ones for working arrays. Usually, the SLICOT Mex-files are using exactly the sizes needed for the arrays, but providing larger sizes for working arrays could improve the efficiency, e.g., by enabling the calls of the LAPACK block algorithms. The optional input parameter `apply` specifies if the bijective mapping introduced in Section 2.3 should be used or not (default: the mapping is not used). Finally, `theta` denotes the parameters of the linear part, in the output normal form parametrisation.

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 determining the linear part by susbspace techniques, respectively.

The MATLAB m-files which call some of the Mex-files are presented in Table 4. The help m-files are excluded. The *system object* defined in the MATLAB Control Toolbox is used, whenever possible. No m-files corresponding to the Mex-files `wident` and `widentc` are provided, since no simplification of their calling statements is possible.

Table 4: SLICOT Wiener system identification: m-file interfaces.

m-file	Function
`NNout`	Computes the output of a set of neural networks used to model the nonlinear part of a Wiener system.
`dsim`	Computes the output response of a linear discrete-time system (much faster than the Matlab function lsim).
`o2s`	Transforms a linear discrete-time system given in the output normal form to a state-space representation.

Table 4: (continued)

m-file	Function
s2o	Transforms a state-space representation of a linear discrete-time system into the output normal form.

The calling sequences for these computational m-files 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));
```

The input argument `wb` contains the weights and biases of the neural network. It is partitioned into `l` vectors of length `nn*(m+2)+1`, `wb = [ wb(1), ..., wb(l) ]`, where `m = size(u,2)`. Each `wb(k)`, `k` $= 1, \ldots,$ `l`, corresponds to one output variable, and is defined as

```
wb(k) = [ w1(1), ..., w1(m), ..., wnn(1), ..., wnn(m),
          ws(1), ..., ws(nn), b(1), ..., b(nn+1) ],
```

where `wi(j)` are the weights of the hidden layer, `ws(i)` are the weights of the linear output layer, and `b(i)` are the biases. The input parameter `x0` is the initial state of the system (default `x0` $=$ 0). The output parameter `x` is the final state of the system. The parameter `sys` is a discrete-time `ss` MATLAB system object, consisting in a state-space realization `sys` $= (A, B, C, D)$. It can be alternately replaced by the matrix tuple. The other arguments have already been defined before.

The toolbox also includes several m-files for testing purposes.

4 Illustration of the use of the Wiener Identification Toolbox

4.1 Solving a linear equations system using MB02WD

With the following calling statement, one can solve a linear equations system $Ax = b$ with the symmetric positive definite system matrix $A \in \mathbb{R}^{n \times n}$ up to a tolerance of $\|Ax - b\|_2 \leq 10^{-6}$,

```
CALL MB02WD( 'Upper', F, n, ipar, 0, dpar, 0, 100, A, LDA,
             b, 1, x, 1, 1.0D-6, DWORK, LDWORK, IWARN, INFO )
```

where `F` is a void subroutine, declared in an `EXTERNAL` statement. If the tolerance goal was not reached within 100 iterations, the routine will stop with an approximate solution returned in `x`, and `IWARN` set to 1. The routine needs a workspace of size `3*n`. A similar calling statement can be used to solve implicit positive definite systems of linear equations of the form $f(A, x) = b$, if the mapping f is suitably defined in the subroutine `F`.

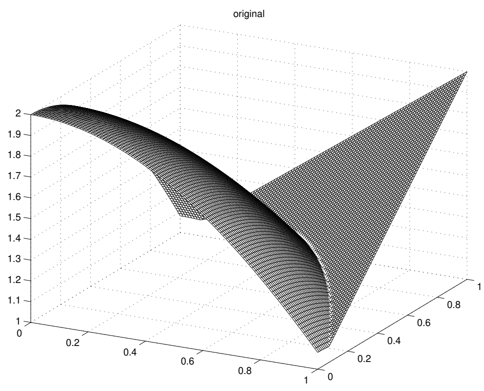

Figure 2: Original function

4.2 Approximation of nonlinear mappings via MD03AD

To provide an example for the capabilities of MD03AD for approximation of nonlinear mappings as described in Section 2.2, we choose a function $f : \mathbb{R}^2 \mapsto \mathbb{R}$, $f(z_1, z_2) := \max(2\cos(\sqrt{z_1^2 + z_2^2}), z_1 + z_2)$, see figure 2. We set ν to 20 which implies $1((2+2)20+1) = 81$ parameters, so $\Theta \in \mathbb{R}^{81}$. For the input we used 10,000 random samples $(z_1, z_2) \in (0,1) \times (0,1)$. Because the estimations for $z(k)$ derived from STEP 1 (see Section 2.1) would be somehow noisy, due to the disturbance term $v(k)$ in (1), we simulated disturbances in (z_1, z_2) and y by

$$y_{a,b} = f(z_1 + a\delta z_1, z_2 + a\delta z_2) + b\delta y\,, \tag{13}$$

where δz_1, δz_2 and δy are equally distributed in $(-1, 1)$. We tested this with three different noise levels (a, b) with the routine MD03AD with tolerance 1.0D-10, and ITMAX $= 55$ Levenberg-Marquardt iterations:

1. Test: $a = b = 0$ (no noise): Results in figure 3;
2. Test: $a = 0.05, b = 0.10$: Results in figure 5;
3. Test: $a = 0.10, b = 0.20$: Results in figure 7.

These plots show the approximated function and the resulting error function $f(z_1, z_2) - \hat{f}(z_1, z_2)$ over $(0,1) \times (0,1)$ for each test, where $\hat{f}(.,.)$ is the right hand side of (2) obtained with the parameters $\left[\hat{\beta}, \hat{\alpha}, \cdots\right]$ calculated in the actual iteration of the Levenberg-Marquardt method.

To give an overview of the results we have also plotted the square of the residual norm $\sum_{k=1}^{10,000} \|y(k) - \hat{f}(\hat{z}_1(k), \hat{z}_2(k))\|_2^2$ over the iterations (figure 9), and listed these results and some additional information in Tables 5, 6 and 7.

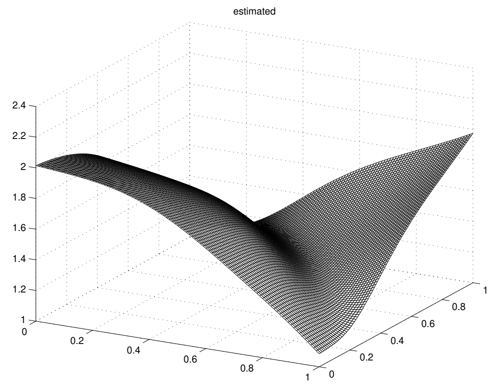

Figure 3: Function estimated without noise ($a = b = 0$) after 55 iterations

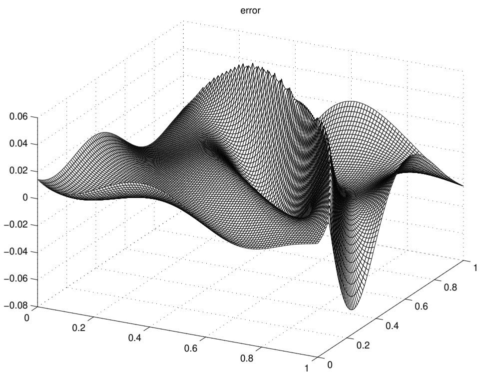

Figure 4: Corresponding error function

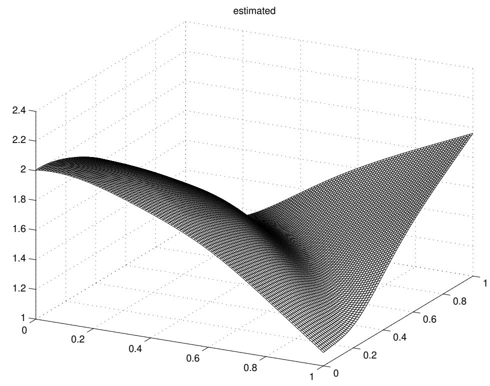

Figure 5: Estimated function with $a = 0.05$ and $b = 0.10$ after 55 iterations

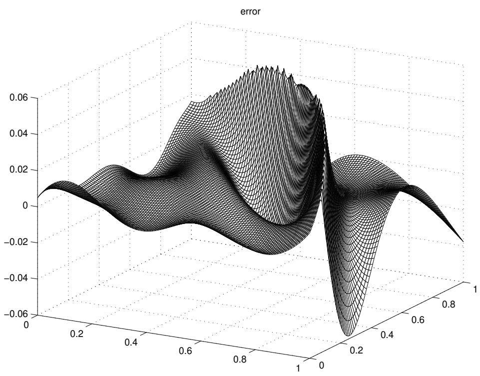

Figure 6: Corresponding error function

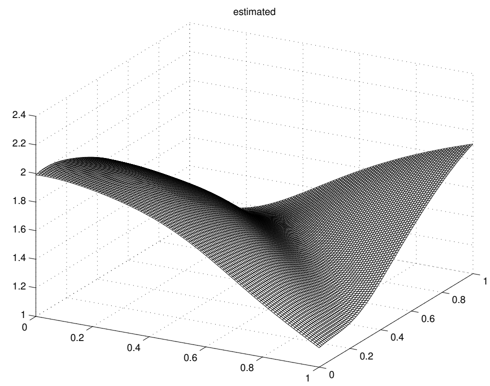

Figure 7: Estimated function with $a = 0.10$ and $b = 0.20$ after 55 iterations

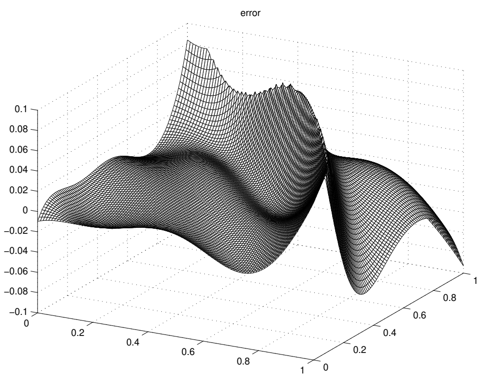

Figure 8: Corresponding error function

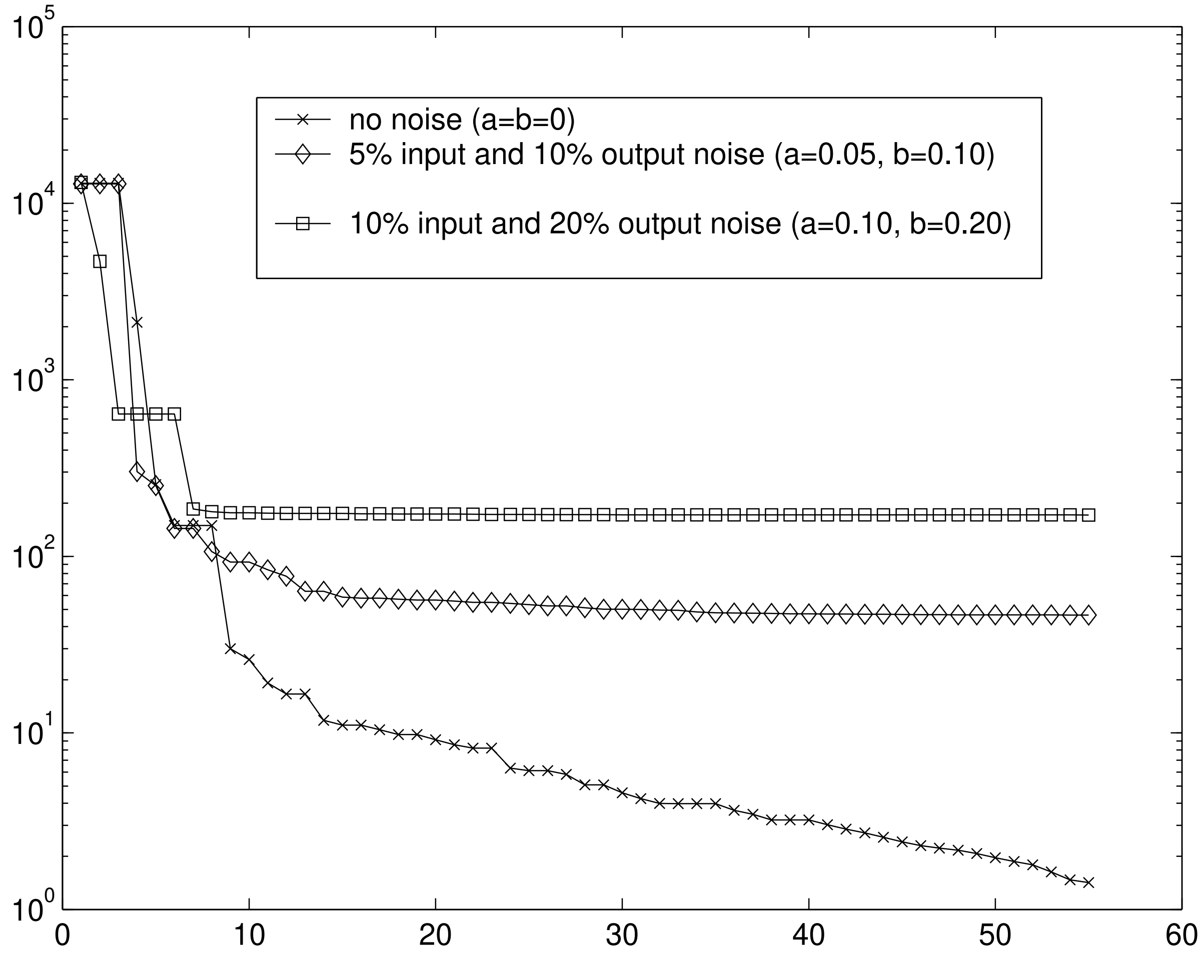

Figure 9: Original function

Table 5: Convergence for $a = 0$, $b = 0$

It.	CG-Steps	Residual	μ
1	7	12965.4160	.10E-02
5	13	255.2610	.10E+01
10	8	26.0323	.10E+03
15	16	11.0807	.11E+01
20	9	9.1472	.81E+01
25	18	6.1170	.49E+00
30	16	4.5851	.12E+01
35	15	3.9776	.77E+00
40	18	3.2200	.11E+01
45	29	2.4134	.53E+00
50	30	1.9647	.21E+00
55	0	1.4201	.86E-01

Table 6: Convergence for $a = 0.05$, $b = 0.10$

It.	CG-Steps	Residual	μ
1	7	12878.2227	.10E-02
5	8	251.5510	.10E+03
10	8	92.7816	.10E+03
15	12	58.8139	.10E+02
20	10	56.5357	.81E+01
25	16	53.3084	.23E+01
30	15	50.1695	.19E+01
35	21	47.9722	.64E+00
40	24	47.2177	.37E+00
45	32	46.8635	.18E+00
50	18	46.6546	.21E+00
55	0	46.5268	.18E+00

Table 7: Convergence for $a = 0.10$, $b = 0.20$

It.	CG-Steps	Residual	μ
1	7	13110.1322	.10E-02
5	17	639.4476	.10E+00
10	14	176.3971	.10E+02
15	6	175.0559	.10E+04
20	22	173.6116	.29E+00
25	21	172.9254	.32E+00
30	37	172.0147	.13E+00
35	32	171.9102	.23E+00
40	18	171.8725	.28E+00
45	22	171.7988	.40E+00
50	23	171.6997	.28E+00
55	0	171.5950	.19E+00

4.3 Identification of a Wiener system using `IB03AD` and `IB03BD`

We conclude this section by summarizing the results obtained using the developed Fortran files and Mex-files for the identification of a Wiener system. The example considered here is also included in the compressed Wiener system identification toolbox file `Wident_mex.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 5000 input and output data samples. Only the first 1000 samples were used to estimate the Wiener system, but all samples serve for validating the identified system. The linear system order was chosen as 4, and the number of neurons in the hidden layer was chosen as 12. The corresponding optimization problem has 1000 error functions and 128 unknown variables. The Mex-file `wident` 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 of the nonlinear part, 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 when using a linear model, but 1.2914 when using a Wiener model. The error norm trajectories after linear and nonlinear identification are plotted in red (upper curve), and green (lower curve), respectively, in Figure 10. The mean values of errors for linear

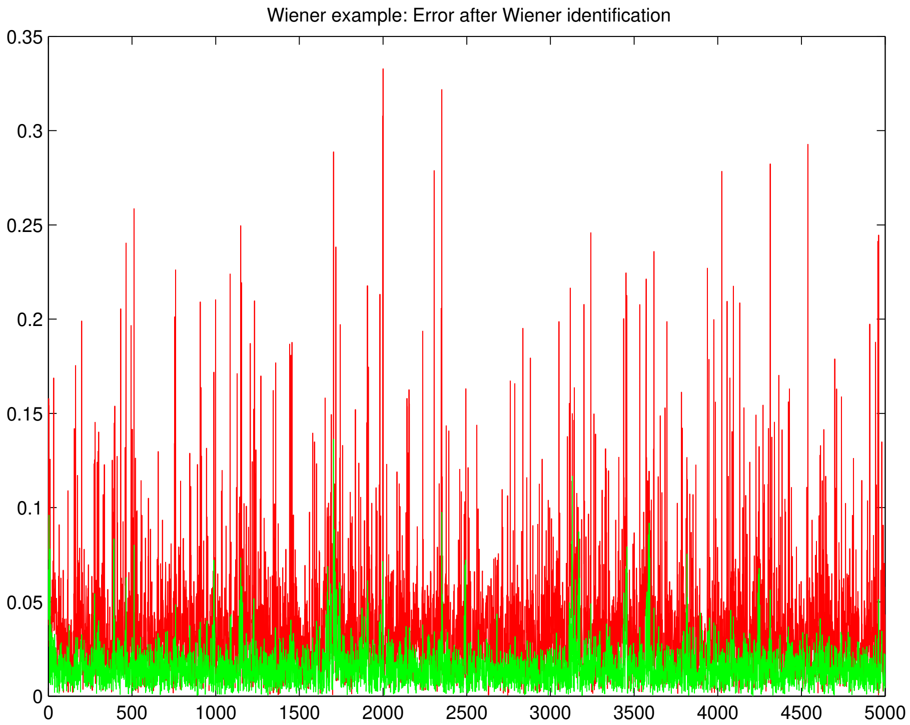

Figure 10: Error norm trajectories after linear (red, upper curve) and nonlinear (green, lower curve) identification.

and Wiener identification (computed for a moving window of 40 samples) are plotted similarly in Figure 11. The improvement when using a nonlinear model is clearly visible.

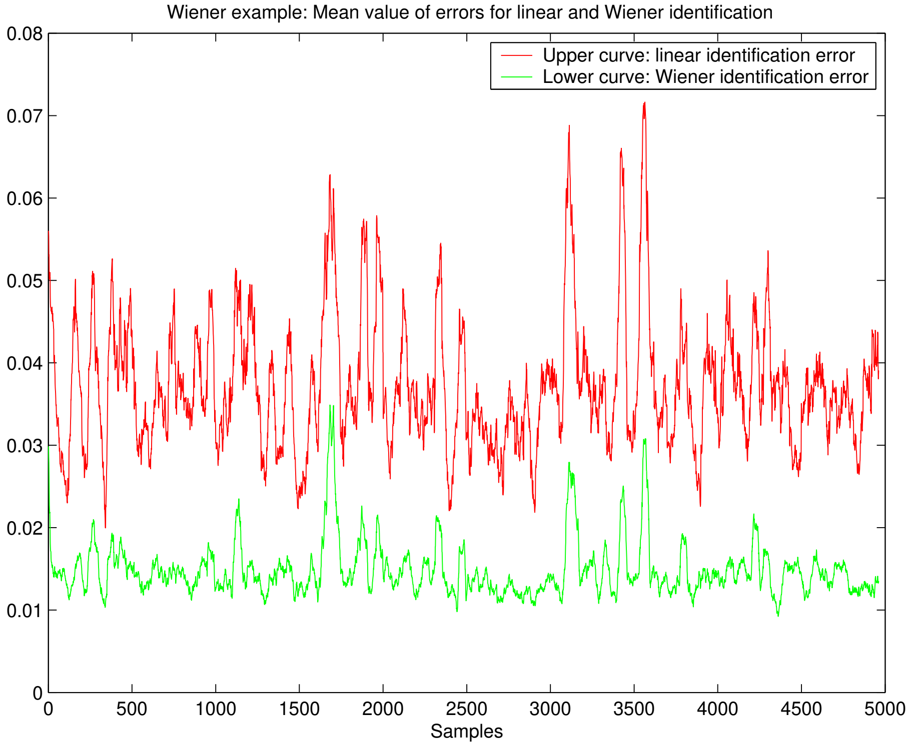

Figure 11: Mean values of errors for linear (red, upper curve) and Wiener (green, lower curve) identification.

Table 8 summarizes comparative performance results when using SLICOT routines `IB03AD` and `IB03BD`, called via the Mex-files `widentc` and `wident`, respectively, for the example discussed above. The option `STOR = 'Full'` has been used for the Cholesky algorithm. The numbers appearing in parantheses are the performance results corresponding to the initialization step of the nonlinear part.

All algorithms have been initialized with the same seed for the random number generator. Also, the same tolerances (`TOL1` = `TOL2` = 0.0001) have been used in all computations. Clearly, the MINPACK-like, but structure-exploiting LAPACK-based Levenberg-Marquardt algorithm, implemented in SLICOT routine `IB03BD` is the most efficient one (concerning all efficiency-related performance parameters), closely followed by the Cholesky-based algorithm. Incidentally, the Cholesky algorithm returned the smallest Euclidean norm of the error (over the whole set of 5000 data samples) using a Wiener model. But `IB03BD` routine produced a smaller value for the sum of squares, which is the criterion actually minimized by the algorithm (over the first 1000 data samples).

5 Inventory of future work

Besides the work described above, there is naturally some other work to do, e.g.,

Table 8: Comparative performance results using SLICOT Mex-files `wident` and `widentc`.

Performance parameter	`wident`	`widentc`, Cholesky	`widentc`, CG
Execution time (sec.)	29.49	33.45	324.78
Sum of squares	2.5381 (1.8255)	2.5525 (1.8406)	2.536 (1.8668)
Number of iterations	12 (61)	16 (103)	70 (56)
Number of CG iterations	0 (0)	0 (0)	12974 (605)
Final Levenberg factor	0.0019 (0.00053)	0.0060 (0.0050)	0.0084 (0.079)
Total number of function evaluations	431	695	2332
Total number of Jacobian evaluations	70	119	126
Euclidean norm of the error using a linear model	3.5299	3.5299	3.5299
Euclidean norm of the error using a Wiener model	1.2914	1.1024	1.3066

- Adapt the SLIDENT toolbox so that it can also handle the PI-MOESP scheme of [1].
- Implementation of other possibilities to aproximate nonlinear functions (neural networks with other activation functions, polynomial approximation,...).
- Benchmark testing.

6 Conclusions

Algorithmic, implementation and numerical details concerning nonlinear, multivariable Wiener systems identification have been investigated. A systematic procedure for solving this problems has been used. This procedure involves three steps: (1) estimating the linear part of the Wiener system, using efficient subspace-based techniques; (2) estimating the nonlinear part of the system, modelled by a neural network with one hidden layer, using a Levenberg-Marquardt algorithm; (3) optimising the whole system (with linear part parameterised by the output normal form) using again a Levenberg-Marquardt algorithm. 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 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] M. Verhaegen, "Subspace Model Identification. Part III: Analysis of the ordinary Output-Error State Space Model Identification algorithm," Int. J. Control, Vol. 58, No. 3, pp. 555–586, 1993.

[2] M. Verhaegen, "Identification of the deterministic part of MIMO state space models given in innovation form from input-output data," Automatica, Special Issue on Statistical Signal Processing and Control, Vol. 30, No. 1, pp. 61–74, 1994.

[3] D. Westwick, M. Verhaegen, "Identifying MIMO Wiener systems using subspace model identification methods", Signal Processing, Vol. 52, pp. 235–258, 1996.

[4] M. Verhaegen, "Identification of the temperature-product quality relationship in a Multi-component distillation column," Invited Paper, Chemical Engineering Communications, Vol. 163, pp. 111–132, 1998.

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

[6] W. H. Press, B. P. Flannery, S. A. Teukolski and W. T. Vetterling, "Numerical Recipes", pp. 523–526, Cambridge, 1986.

[7] G. H. Golub, C. F. van Loan, "Matrix Computations", pp. 520–528, Baltimore, 1996.

[8] G. Luenberger, "Introduction to Linear and Nonlinear Programming", p. 187, New York, 1973.

[9] C. T. Kelley, "Iterative methods for optimization", Philadelphia, Pa., 1999.