MATLAB Software for Recursive Identification of Wiener Systems – Revision 2

Torbjörn Wigren

Systems and Control, Department of Information Technology, Uppsala University, SE-75105 Uppsala, SWEDEN. E-mail: torbjorn.wigren@it.uu.se.

March 2007

Abstract

This reports is intended as a users manual for a package of MATLAB™ scripts and functions, developed for recursive prediction error identification of discrete time nonlinear Wiener systems and nonlinear static systems. Wiener systems consist of linear dynamics in cascade with a static nonlinearity. The core of the package is an implementation of 9 recursive SISO output error identification algorithms. Three main cases are treated. The first set of 5 algorithms identify the IIR linear dynamics in cases where the static nonlinearity is known. It is stressed that the nonlinearity is allowed to be non-invertible. The second set of 2 algorithms simultaneously identifies the linear dynamics and the static non-linearity. The nonlinearity is parameterized as a piecewise linear or a piecewise quadratic nonlinear function. The last set of two algorithms exploits the above parameterization of the static nonlinearity for estimation of static nonlinear systems. Arbitrarily colored additive measurement noise is handled by all algorithms. The software can only be run off-line, i.e. no true real time operation is possible. The algorithms are however implemented so that true on-line operation can be obtained by extraction of the main algorithmic loops. The user must then provide the real time environment. The software package contains scripts and functions that allow the user to either input live measurements or to generate test data by simulation. The functionality for display of results include scripts for plotting of data, parameters and prediction errors. Model validation is supported by several methods apart from the display functionality. First, calculation of the RPEM loss function can be performed, using parameters obtained at the end of an identification run. Pole-zero plots can be used to investigate possible overparameterization in the linear dynamic part of the Wiener model. Finally, the static accuracy as a function of the output signal amplitude can be assessed.

Keywords: Block-orinted model, Identification, MATLAB™, Nonlinear systems, Prediction error method, Recursive algorithm, Recursive identification, Software, Wiener model,

Prerequisites

Recursive identification is today a fairly well established field of technology. The classic text [1] e.g. organizes and analyses recursive identification for linear systems. It is assumed that the reader of this report has a good working knowledge of recursive identification. The scope of this report is not on linear systems, but on nonlinear Wiener systems. Again, the reader is assumed to have a good working knowledge of the field, as defined e.g. by [2] – [4]. Furthermore, the user is assumed to have a working knowledge of MATLAB™. This report hence only describes the parts of [2] - [8] that are required for the description of the software. The main reference behind this report is [2], where all algorithms are described in details. In fact, the simulations of [2] were performed with files that later became revision 1 of this software. The reference [2] is available for download at http://www.it.uu.se/katalog/tw.

Revisions

Revision 1: This revision executes on old versions of MATLAB, like PCMATLAB. It also exploits old toolboxes, in particular the signal processing toolbox and the control systems toolbox. It is available from the author of this report on request.

Revision 2: This revision is ported to current MATLAB. It has been tested on MATLAB 5.3 and MATLAB 7.0. Furthermore, the dependence on toolboxes has been removed – now only MATLAB is required to run the program package. The old short naming (maximum 8 characters, dictated by early MATLAB revisions) of the scripts of revision 1, are retained in the ported revision 2.

Installation

The file SW.zip is copied to the selected directory and unzipped. MATLAB is opened and a path is set to the selected directory using the path browser. The software is then ready for use.

Note: This report is written with respect to the software, as included in the SW.zip file. It may therefore be advantageous to store the originally supplied software for reference purposes.

Error reports

When errors are found, these may be reported in an e-mail to:

torbjorn.wigren@it.uu.se.

1. Introduction

Identification of nonlinear systems is an active field of research today. There are several reasons for this. First, many practical systems show strong nonlinear effects. This is e.g. true for high angle of attack flight dynamics, many chemical reactions and electromechanical machinery of many kinds. Another important reason is perhaps that linear methods for system identification are quite well understood today, hence it is natural to move the focus to more challenging problems.

There are already a number of identification methods available for identification of nonlinear systems. These include grey-box differential equation methods, where numerical integration is combined with optimization in order to optimize the unknown physical parameters that appear in the differential equations. An alternative approach is to start with a discrete time black box model, and to apply existing methodology from the linear field to the solution of the nonlinear identification problem. This is the approach taken in the NARMAX method and its related algorithms. There, a least squares formulation can often be found, a fact that facilitates the solution. Other methods apply neural networks for modeling of nonlinear dynamic systems. There are also models that aims at exploiting the many advantages that result when the dynamics remains linear. These block oriented models typically utilize different cascade structures of linear dynamic and static nonlinear blocks. The Wiener model [2]-[10], where a linear dynamic block is followed by a static nonlinearity is one important case that has received a considerable attention in the literature.

This report focuses on software that implements the recursive output error identification algorithms presented and analysed in [2]. The details of these methods are not reproduced here, the reader is referred to [2] – [10] for the background. The core of the SW package is a MATLAB implementation of the 9 SISO output error identification algorithms of [2]. Three main cases are treated. The first set of 5 algorithms identify the IIR linear dynamics in cases where the static nonlinearity is known. It is stressed that the nonlinearity is allowed to be non-invertible. The second set of 2 algorithms simultaneously identifies the linear dynamics and the static non-linearity. The nonlinearity is parameterized as a piecewise linear or a piecewise quadratic nonlinear function. The last set of two algorithms exploit the above parameterization of the static nonlinearity for identification of static nonlinear systems. Arbitrarily colored additive measurement noise is handled by all algorithms. Very limited software changes are needed to handle the pure FIR case [9] and to handle e.g. output quantization [10]. It is believed that users with the prerequisites defined above will not encounter any difficulties when such modifications are performed. A software package for

recursive identification of nonlinear systems with more general nonlinear dynamics is described in [11].

The algorithm that are implemented include:

- Algorithm 1: A Gauss-Newton algorithm for the case with a known static nonlinearity. Fix regressor filtering is used and the gradient of the output nonlinearity is approximated by a user chosen constant. The advantages as compared to Algorithm 5 appear in [2] and [6].
- Algorithm 2: A Gauss-Newton algorithm for the case with a known static nonlinearity. Fix regressor filtering and the exact gradient of the static output nonlinearity are used. The advantages as compared to Algorithm 5 appear in [2] and [6].
- Algorithm 3: A stochastic gradient algorithm for the case with a known static nonlinearity. Fix regressor filtering is used and the gradient of the output nonlinearity is approximated by a user chosen constant. The advantages as compared to Algorithm 5 appear in [2] and [6].
- Algorithm 4: A stochastic gradient algorithm for the case with a known static nonlinearity. Fix regressor filtering and the exact gradient of the static output nonlinearity are used. The advantages as compared to Algorithm 5 appear in [2] and [6].
- Algorithm 5: The RPEM for the case with a known static nonlinearity. Adaptive regressor filtering and the exact gradient of the static output nonlinearity are used.
- Algorithm 7: The RPEM for the case with an unknown static nonlinearity. A piecewise linear model of the static nonlinearity is used.
- Algorithm 8: The RPEM for the case with an unknown static nonlinearity. A piecewise quadratic model of the static nonlinearity is used.
- Algorithm 9: The RPEM for the static case without any linear dynamics. A piecewise linear model of the static nonlinearity is used.
- Algroithm 10: The RPEM for the static case without any linear dynamics. A piecewise quadratic model of the static nonlinearity is used.

The present software package is developed and tested using MATLAB 5.3 and MATLAB 7.0. The software package does not rely on any MATLAB toolboxes. It consists of a number of scripts and functions. Briefly, the software package consists of scripts for setup, scripts for generation or measurement of data, scripts for execution of the 9 algorithms, scripts for generation and plotting of results, as well as scripts for validation of identified Wiener models. There is presently no GUI, the scripts must be run from the command window. Furthermore, input parameters need to be configured in one or several of the setup scripts, as well as when running the scripts. The software can only be run off-line, i.e. there is no support for execution in a real time environment. The major parts of the algorithmic loop can however easily be extracted for such purposes.

The report is organized according to the flow of tasks a user encounters when applying the scripts of the package. A detailed description of the software is given for the nonlinear dynamic case, the static case is described more briefly in the end of this report.

2. Software package overview

2.1 Scripts, functions and command flow

Roughly, the scripts and functions can be divided into six groups:

- *Live data*. The script of this group sets up a few variables needed before execution of the algorithms. The script is **InitializeLiveData.m**.
- *Simulated data generation*. The six scripts of this group define example dynamic system, that are then used for generation of simulated data. The scripts are **InitializeDataGeneration.m, pH.m** (pH-control, see [2]), **Piecelin.m** (piecewise linear system, see [2]), **Piecesqu.m** (piecewise quadratic system, see [2]) , **Satur.m** (sensor saturation, see [2]) and **Valve.m** (control valve, see [2]).
- *Recursive identification*. The 14 scripts of this group perform the actual identification tasks. The scripts and functions are **InitializeStaticLinear.m**, **alg1.m** – **alg5.m** (algorithms for known static nonlinearity), **InitializeLinearWiener.m**, **alg7.m**, **InitializeQuadraticWiener.m**, **alg8.m** (algorithms for simultaneous identification of the linear dynamics and the static nonlinearity), and **InitializeStaticPiecewiseLinear.m**, **alg9.m**, **InitializeStaticPiecewiseQuadratic.m**, **alg10.m** (algorithms for identification of static nonlinear systems)
- *Supporting functions - not called by the user*. The function of this group is called by scripts of the previous group. The function is **Eval_fn.m** and it evaluates the known static nonlinearity and the corresponding derivative.
- *Display of results*. There are three scripts in this group. The scripts are **P_data.m**, **P_outerr.m** and **P_param.m**
- *Model validation* There are four scripts in this group. The scripts are **V_crit.m**, **V_outerr.m**, **V_polzer.m** and **V_stacha.m**.

These groups of scripts and functions need to be operated in a particular order to make sense. This order will be clear from the examples below. In general, initialization scripts need to be executed before algorithmic scripts of a specific group of scripts. The order between the groups is to start with the *Live Data* or *Simulated Data Generation* groups. Then the *Recursive identification* group is used. Here it is important to remember to include MATLAB code for a known static nonlinearity and the

corresponding derivative in the function **Eval_fn.m**. Following recursive identification the user may wish to proceed with scripts from the groups *Display of results* or *Model validation*.

There are three major ways to exploit the five groups of scripts and functions.

1. In case the user has input and output signals available, the first step is to define and run the script **InitializeLiveData.m.** This sets basic parameters. The user can then proceed directly to use the groups *Recursive Identification* and *Display of results.* The data, which can be simulated or live, should be stored in the (column) matrices u and y_n.
2. In case the user is to perform live measurements, all the steps of the *Live data* group should be executed first. The user can then proceed directly to use the groups *Recursive Identification* and *Display of results.*
3. In case the user intends to use simulated data, this data can be generated by execution of the scripts and functions of the group *Simulated data generation.* The user can then proceed directly to use the groups *Recursive Identification* and *Preparation and display of results.*

2.2 Restrictions

The main restrictions of the software are

- The software is command line driven - no GUI support is implemented.
- The software does not support true real time operation - there is no real time OS support implemented.
- The software has been tested and run using MATLAB 5.3 and MATLAB 7.0.

3. Data input

The generation of data begins the section where the actual software is described. Since the user is assumed to have access to all source files, the descriptions below do not describe code related issues and internal variables. Only the parts that are required for the use of the software package are covered. In case m-files are reproduced, only the relevant parts are included, the reader should be aware that more information can be found in the source code. Note that the setup files are to be treated as templates, the user is hence required to modify right hand sides only - no addition or deletion of code should be used in the normal use of the package.

3.1 Simulated data

The generation of simulated data requires that the user

1. Either uses one of the provided files for data generation, or that the user modifies one of them for the purpose of describing another system.
2. Provides further input data in the script **InitializeDataGeneration.m**. The parameters that define the data generation are directly written into this script. These parameters define the type of input, and if default values provided in each data generation file is to be used.
3. Executes **InitializeDataGeneration.m**. This loads the necessary parameters into the MATLAB workspace.
4. Generates data by execution of either of the 5 provided systems, **pH.m**., **valve.m**, **satur.m**, **piecelin.m**, **piecesqu.m,** or by execution of a script implementing a user defined system. After the execution of this script, variables with sampling instances, input signals and output signals are available in the MATLAB workspace.

Example 1: This and the following examples illustrates the use of the software package for identification of the system. First select input_type=3 and default=0 in **InitializeDataGeneration.m**. This input type is a PRBS like signal, but with uniformly distributed amplitudes, cf. [7]. Then execute **InitializeDataGeneration.m** and **valve.m** from the MATLAB command line.

3.2 Live data

The processing of live data requires that the user

1. Provides basic data on the size (n) of the input and output signal vectors using the script **InitializeLiveData.m**.
2. Executes **InitializeLiveData.m**. This loads the necessary parameters into the MATLAB workspace.
3. Loads the input/output data into the MATLAB workspace, e.g. using a variant of the load command.

3.3. Display of data

After execution of either one of the chain of actions of section 3.1 or section 3.2, data can be plotted. The procedure is as follows.

1. The **PlotData.m** script is executed in the MATLAB command window. This script makes use of the dimensions of the system in order to divide the plot into several sub-windows, and in order to provide the axis text.

   Example 2: The MATLAB command window command **P_data** generates the following plot.

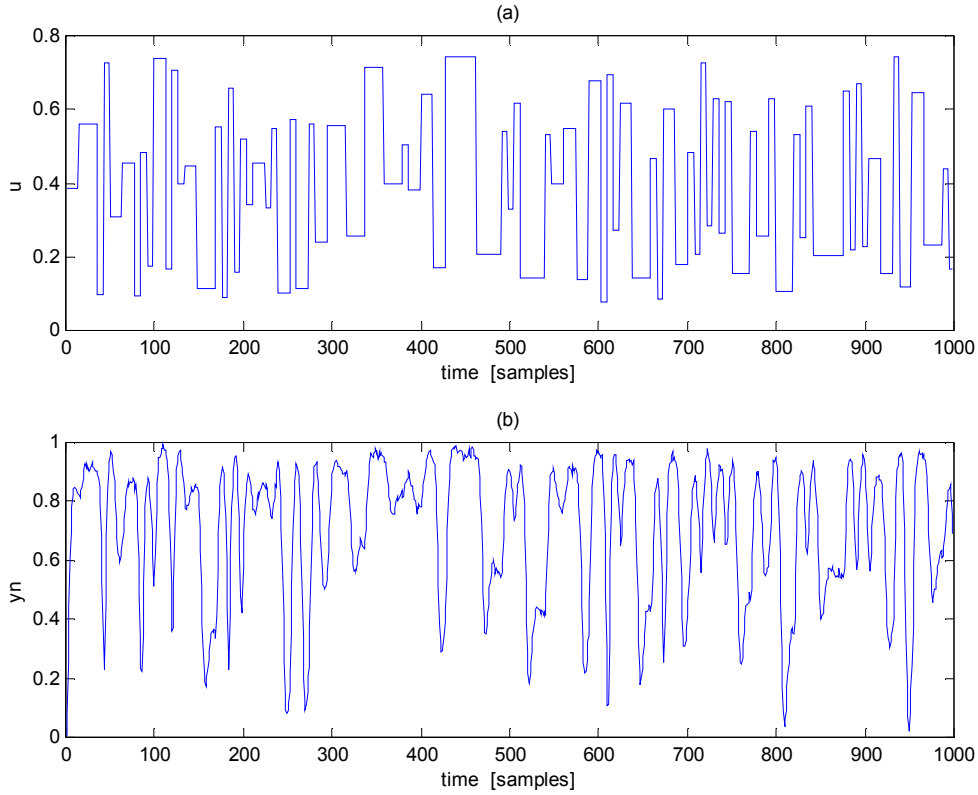

Figure 1: The result of a PlotData command.

4. Recursive Identification

At this point everything is in place for a first identification run.

4.1 Known static nonlinearity

The preparation for the identification run requires that the user

1. Modifies the **Eval_fn.m** function to describe the static nonlinearity and the corresponding derivative. Note that the SW comes with the static nonlinearity for the control valve system implemented.
2. Provides further input data in the script **InitializeStaticLinear.m** (static nonlinear and linear dynamics). The parameters that define the data generation are directly written into this script. These parameters define the initial values for the B- and F-polynomials in the utilized transfer function operator model, the assumed slope of the nonlinearity (k0, not needed for Algorithm 5),

the rate of the gain sequence (lambda0) and the initial value of the gain sequence (lambda(0)), as well as the initial value of the P-matrix (for the Gauss-Newton algorithms) or the scalar p (for the stochastic gradient algorithms). Please refer to the source code of the script for the precise details.

3. Executes **InitializeStaticLinear.m**. This loads the necessary parameters into the MATLAB workspace.

In order to perform an identification run the user is then required to execute the script corresponding to any of the algorithms 1-5. Here it is assumed that the RPEM is run, i.e. **Alg5** is typed at the MATLAB command line. In response the parameter vector is identified from the data and typed out. The parameters corresponding to the F-polynomial normally constitute the first half of the parameter vector theta and the parameters corresponding to the identified B-polynomial constitute the second half of the parameter vector. The result is

Example 3: The command **Alg5** generates the following result in the command window:

```
» Alg5

theta =

  -1.5733    0.6886    0.0640    0.0513
```

.

4.2 Unknown static nonlinearity – piecewise linear case

The preparation for the identification run requires that the user

1. Provides further input data in the script **InitializeLinearWiener.m**. The parameters that define the data generation are written directly into this script. These parameters define the initial values for the B- and F-polynomials in the utilized transfer function operator model. The parameters kmi and kpl, describing the number of left and right grid-points of the nonlinearity, are needed to define the parameterization of the nonlinearity. To initialize the parameter vector for the static nonlinearity, the slope in the mid interval (k0, cf. [2] and [3]), the bias in the mid interval (thetafy0_0), as well as the grid (grid) needs to be specified. The initial value of the parameter vector of the static nonlinearity is then computed as a straight line with slope k0 and bias thetafy0_0 (in the mid-interval, cf. [2] and [3]). Algorithm parameters and initial values are also needed. These include the rate of the gain sequence (lambda0), the initial value of the gain sequence (lambda(0)), as well as the initial value of the P-matrix. Please refer to the source code of the script for the precise details.
2. Executes **InitializeLinearWiener.m**. This loads the necessary parameters into the MATLAB workspace.

In order to perform an identification run the user is then required to execute the script corresponding to algorithm 7, i.e. **Alg7** is typed at the MATLAB command line. In response the parameter vector of the linear dynamic and static nonlinear blocks are identified from the data and typed out. The parameters corresponding to the F-polynomial normally constitute the first half of the parameter vector theta and the parameters corresponding to the identified B-polynomial constitute the second half of the parameter vector of the linear dynamic block. The first parameter of the static nonlinear parameter vector is the bias in the mid-interval. The remaining parameter values are the function values of the user chosen grid points, from left to right. Note that there are no parameters corresponding to the left and right grid point of the mid-interval. .

4.3 Unknown static nonlinearity – piecewise quadratic case

The preparation for the identification run requires that the user

1. Provides further input data in the script **InitializeQuadraticWiener.m**. The parameters that define the data generation are written directly into this script. These parameters define the initial values for the B- and F-polynomials in the utilized transfer function operator model. The parameters kmi and kpl, related to the number of left and right grid-points of the nonlinearity, are needed to define the parameterization of the nonlinearity. To initialize the parameter vector for the static nonlinearity, the slope in the mid interval (k0, cf. [2] and [3]), the bias in the mid interval (thetafy0_0), as well as the grid (grid) needs to be specified. The initial value of the parameter vector of the static nonlinearity is then computed as a straight line with slope k0 and bias thetafy0_0 (in the mid-interval, cf. [2] and [3]). It should be noted that the scripts adds intermediate grid points automatically, in between the provided grid points. The grid points half way between the ,main points should hence not be provided by the user, the reader is referred to [2] for details. Algorithm parameters and initial values are also needed. These include the rate of the gain sequence (lambda0), the initial value of the gain sequence (lambda(0)), as well as the initial value of the P-matrix. Please refer to the source code of the script for the precise details.
2. Executes **InitializeQuadraticWiener.m**. This loads the necessary parameters into the MATLAB workspace.

In order to perform an identification run the user is then required to execute the script corresponding to algorithm 8, i.e. **Alg8** is typed at the MATLAB command line. In response the parameter vector of the linear dynamic and static nonlinear blocks are identified from the data and typed out. The parameters corresponding to the F-polynomial normally constitute the first half of the parameter vector theta and the parameters corresponding to the identified B-polynomial constitute the second

half of the parameter vector of the linear dynamic block. The first parameter of the static nonlinear parameter vector is the bias in the mid-interval. The remaining parameter values are the function values of the user chosen grid points, from left to right. Note that there are no parameters corresponding to the left and right grid point of the mid-interval. Note also that there are parameter values corresponding to the automatically added grid points in between the provided ones.

5. Display of results

The display of results is straightforward. By a study of the source code, users should be able to tailor available scripts and also write own ones when needed. The description below assumes that the scripts generating Example 3 has just been run.

5.1 Parameters

In order to plot the parameters the user is required to

1. Execute the script **P_param.m**. The components of the parameter vector are then plotted as a function of time. For results produced by algorithm 7 and algorithm 8, one plot is created for the parameters of the linear dynamic block, and another plot is created for the parameters of the static nonlinear block.

   Example 4: The command **P_param** in the MATLAB command window generated the following plot.

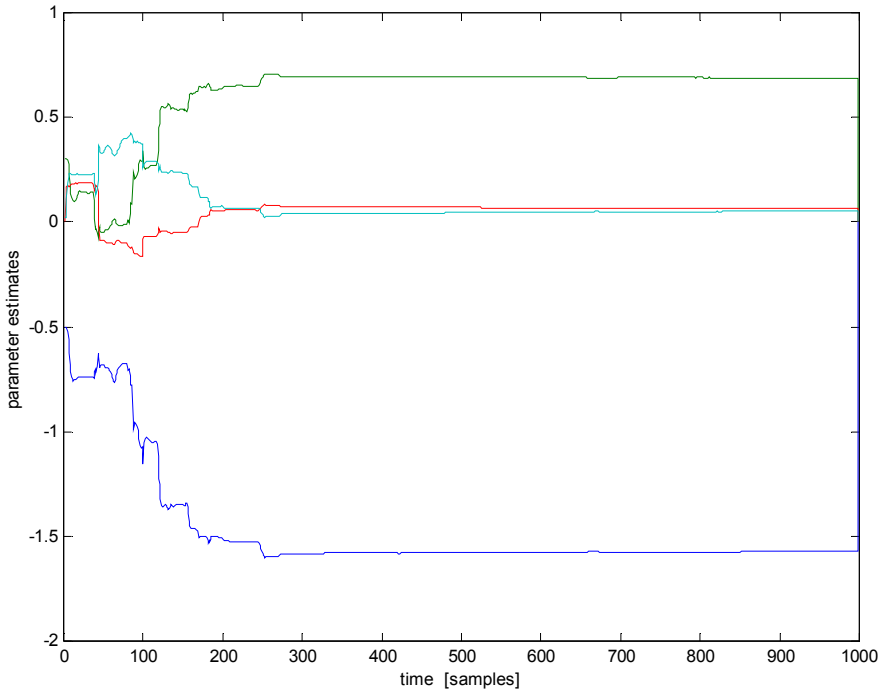

Figure 2: The result of a **P_param** command.

5.2 Output errors

In order to plot the running output errors the user is required to

1. Execute the script **P_outerr.m**. The output signals of the system and the model (using running parameter estimates) are then plotted in the top as a function of time. The bottom plot displays the computed output errors, as a function of time.

Example 5: The command **P_outerr** in the MATLAB command window command generated the following plot

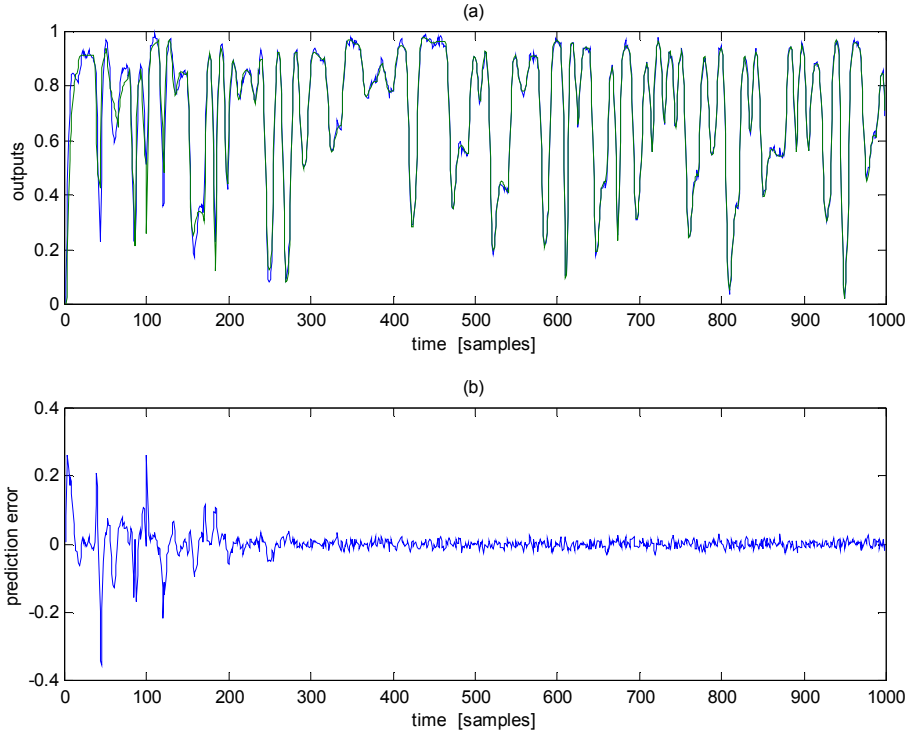

Figure 3: The result of a **P_outerr** command.

6. Model validation

Four scripts are provided for model validation purposes. The first method studies the value of the loss function that is minimized by the RPEM-algorithm.

6.1 RPEM loss function

The RPEM loss function that is computed is given by the average of the squared prediction errors, generated by an identified model using the parameters obtained at the end of the identification run. In order to compute the loss function, the user is required to

1. Execute the script **V_crit.m**. The loss function is then computed.

Example 6: The execution of the command **V_crit** in the MATLAB command window generates:

» V_crit

loss_function =

1.0380e-004 .

6.2 Simulated output errors

The simulated output errors are similar to what is provided by **P_outerr**. The difference is that the simulated output errors are generated with the fix parameter vector, obtained at the end of the identification run. In order to generate the simulated output errors, the user is required to

2. Execute the script **V_outerr.m**.

Example 6: The execution of the command **V_outerr** in the MATLAB command window generates the following plot:

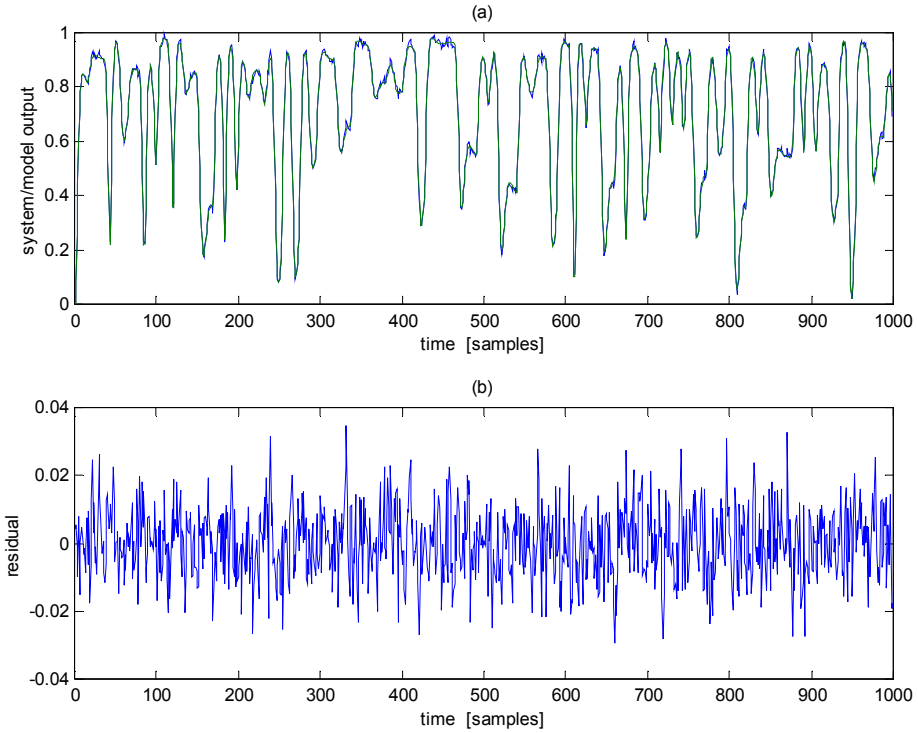

Figure 4: The result of a **V_outerr** command.

6.3 Pole – zero plots

It is well known that overparameterization of a linear model can manifest itself in pole zero cancellations. This is also the fact for the linear dynamic block of the Wiener model. In order to generate a pole-zero plot, the user is required to

3. Execute the script **V_polzer.m**.

Example 7: The execution of the command **V_polzer** in the MATLAB command window generates the following plots:

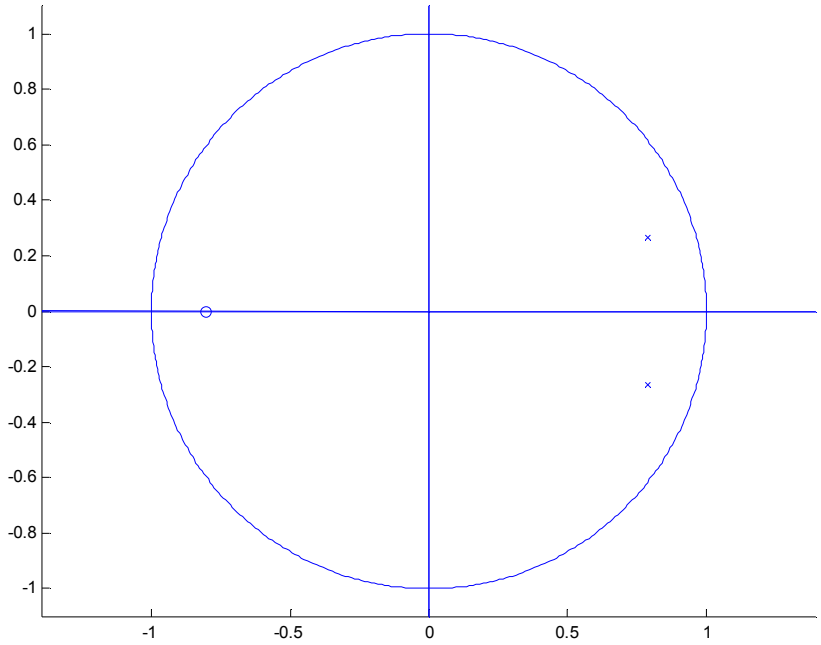

Figure 5: The first result of a **V_polzer** command. – Overview.

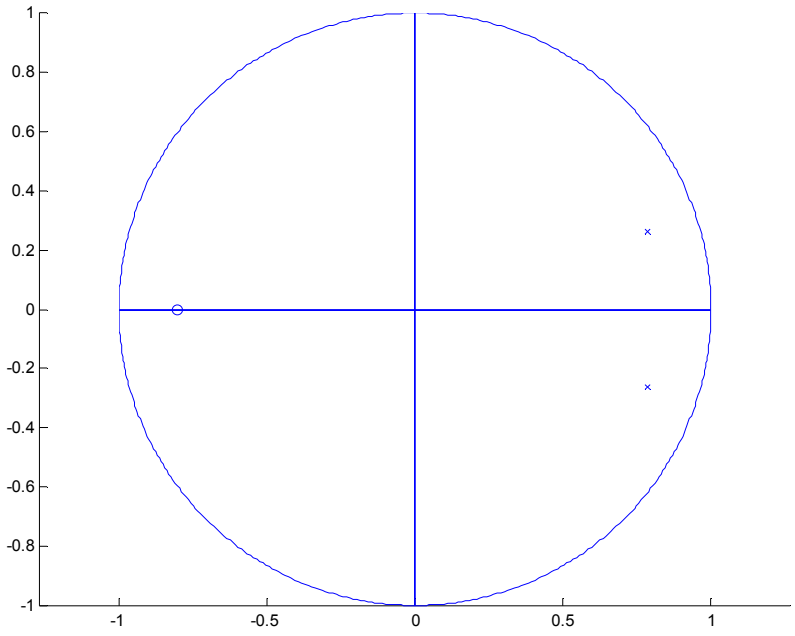

Figure 6: The second result of the **V_polzer** command. Detailed view. For the model at hand, there is little difference between Figure 5 and the present plot. Other scenarios generate larger differences.

6.4 Mean residual analysis

Mean residual analysis is a method that evaluates the obtained static characteristics of an identified model of any kind. It operates by sorting residual errors into bins, the bin being decided by the value of the measured output signal with the same time index as the residual error. The mean of the residuals are then computed, in each bin, and plotted against the range of the output signal. The number of samples of each bin is also plotted. The user is referred to [7] for further details. In order to perform mean residual analysis, the user is required to

1. Provide the intervals used to divide the output signal range, by typing them at the command line in the row vector variable I_l.
2. Execute the script **V_stacha.m**.

Example 8: The intervals are first provided by typing them at the MATLAB command line. The command **V_stacha** then generates the plot of figure 7.

```
I_l =

  Columns 1 through 7

   -0.2000   -0.1000         0    0.1000    0.2000    0.3000    0.4000

  Columns 8 through 14

    0.5000    0.6000    0.7000    0.8000    0.9000    1.0000    1.1000

  Column 15

    1.2000

» V_stacha.
```

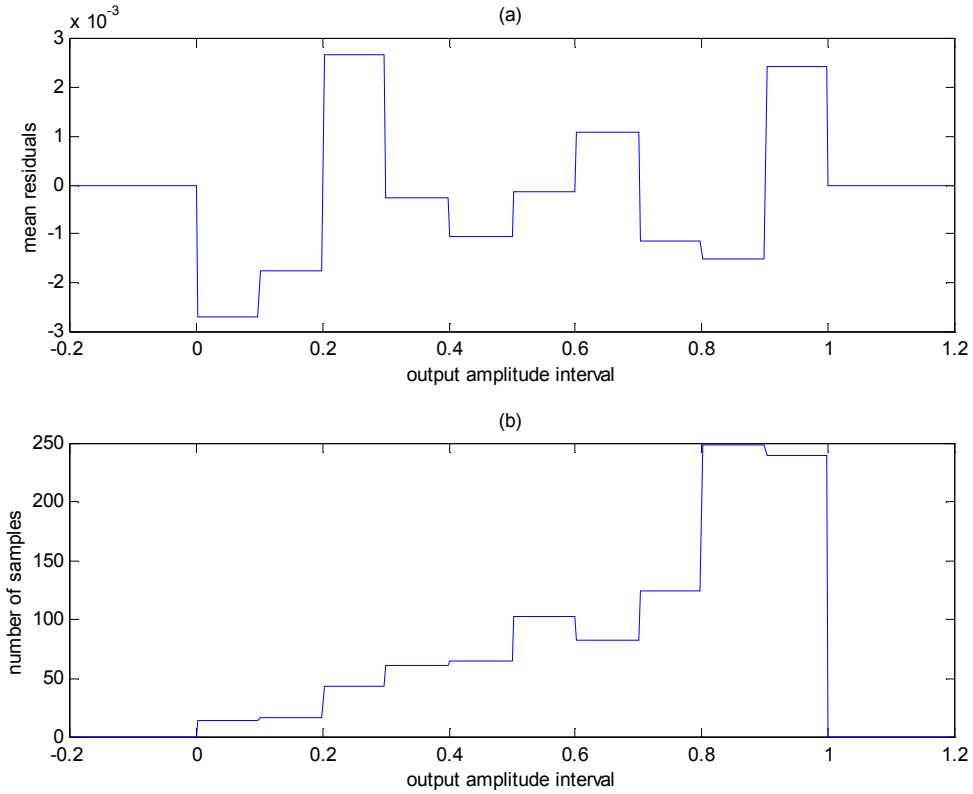

Figure 7: The effect of a **V_stacha** command

7. Summary

This report describes a software package for identification of nonlinear Wiener systems. Future work in this field, that result in useful MATLAB routines, will be integrated with the presently available functionality. Updated versions of this report will then be made available.

11. References

[1] L. Ljung and T. Söderström. *Theory and Practice of Recursive Identification.* Cambridge, Ma: MIT Press, 1983.

[2] T. Wigren, Recursive identification based on the nonlinear Wiener model, Ph.D. thesis, Acta Universitatis Upsaliensis, Uppsala Dissertations from the Faculty of Science 31, Uppsala University, Uppsala, Sweden, December, 1990.

[3] T. Wigren, Recursive prediction error identification using the nonlinear Wiener model, Automatica, vol. 29, no. 4, pp. 1011-1025, 1993.

[4] T. Wigren, Convergence analysis of recursive identification algorithms based on the nonlinear Wiener model, IEEE Trans. Automat. Contr., vol. AC-39, no. 11, pp. 2191-2206, 1994.

[5] T. Wigren, Approximate gradients, convergence and positive realness in recursive identification of a class of nonlinear systems, Int. J. Adaptive Contr. Signal Processing, vol. 9, no. 4, pp. 325-354, 1995.

[6] T. Wigren, Circle criteria in recursive identification, IEEE Trans. Automat. Contr. , vol. 42, no. 7, pp. 975-979, 1997.

[7] T. Wigren, "User choices and model validation in system identification using nonlinear Wiener models", Prep. 13:th IFAC Symposium on System Identification , Rotterdam, The Netherlands, pp. 863-868, August 27-29, 2003. Invited session paper.

[8] T. Wigren, Reduction of amplitude dependent gain variations in control of non-linear Wiener type systems, submitted to 7th IFAC Symposium on Nonlinear Control Systems, NOLCOS 2007 , Pretoria, South Africa, August 22-24, 2007.

[9] T. Wigren, Output error convergence of adaptive filters with compensation for output nonlinearities, IEEE Trans. Automat. Contr., vol. 43, no. 7, pp. 975-978, 1998.

[10] T. Wigren, Adaptive filtering using quantized output measurements, IEEE Trans. Signal Processing, vol. 46, no. 12, pp. 3423-3426, 1998.

[11] T. Wigren, "MATLAB software for recursive identification and scaling using a structured nonlinear black—box model – revision 2*", Technical Reports from the department of Information Technology 2005-022,* Uppsala University, Uppsala, Sweden, August, 2005.