MATLAB Software for Recursive Identification of Systems With Output Quantization – Revision 1

Torbjörn Wigren

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

April 2007

Abstract

This reports is intended as a users manual for a package of MATLAB™ scripts and functions, developed for recursive identification of discrete time nonlinear Wiener systems, where the static output nonlinearity is a known arbitrary quantization function, not necessarily monotone. Wiener systems consist of linear dynamics in cascade with a static nonlinearity. Hence the systems treated by the software package can also be described as discrete time linear systems, where the output is measured after a known quantization function. The identification algorithms thus identify the linear dynamics of the Wiener system. The core of the package is an implementation of 5 recursive SISO output error identification algorithms. The measurement noise is assumed to affect the system after quantization. The identified linear dynamic part of the system is allowed to be of FIR or IIR type. A key feature of the identification algorithms is the use of a smooth approximation of the quantizer, for derivation of an approximation of the gradient of the algorithm. This is necessary since the derivative of the quantizer consists of a set of pulses, in the quantization steps. Using such an approximation 2 recursive stochastic gradient algorithms and 3 recursive Gauss-Newton algorithms are obtained. The algorithms differ by the choice of gradient approximation. It should be noted that the stochastic gradient algorithms are primarily suited for (high order) FIR systems – they converge very slowly for IIR systems due to the large eigenvalue spread of the Hessian that typically results for IIR 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 with mean residual analysis.

Keywords: Adaptive filtering, Block-oriented model, Identification, MATLAB™, Nonlinear systems, Quantization, 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. Furthermore, the user is assumed to have a working knowledge of MATLAB™. This report hence only describes the parts of [2] - [6] and [9] that are required for the description of the software. The main reference behind this report is [2], where all algorithms are described in detail. The references [3]-[6] cover analysis and various extensions to the case with output quantization. Corresponding techniques for optimal estimation are available in [7] and [8]. The reference [2] is available for download at http://www.it.uu.se/katalog/tw.

Revisions

Revision 1: This revision is obtained by modification of the related software package [10]. The modifications include support for specification of the output quantizer and the corresponding approximation of its derivative. Modifications have also been introduced so that both the FIR and IRR case are handled by all algorithms. The package has been tested on MATLAB 5.3 and MATLAB 7.0. The old short naming (maximum 8 characters, dictated by early MATLAB revisions) of the scripts of [10] has been replaced by more descriptive naming in this software package. Note that a rev. 3.0 of [10] is in preparation and will be available shortly.

Installation

The file QRIS.zip is copied to the selected directory and unzipped. The resulting file SW.zip is then unzipped and all MATLAB scripts appear. 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]-[4], 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 the case where the output measurement of a dynamic system is affected by coarse quantization. Such situations are becoming more important because of the increasing number of dynamic systems that are being controlled or supervised over the internet, where bandwidth and data rate constraints are in effect. These limitations are even more dominant when radio connections are involved, typically exploiting new possibilities offered by cellular telephony. One conclusion of the above discussion is a need to embed the effects of quantization in the model used for controller design and supervision. To reach this goal a suitable model needs to be created, by physical modeling, system identification or by a combination. When system identification is used, the coarse quantization becomes a severe problem due to the discontinuous and highly nonlinear character of the quantization function. This difficulty is particularly troublesome in the output quantization case.

As stated above an approximation of the derivative of the quantizer is needed. This approximation can be obtained in many ways. This document describes the method implemented in the software as default. The idea is to approximate the quantizer with a smooth function and to approximate the derivative of the quantizer with the derivative of the smooth function. It is stressed that the exact quantization function is used in the computation of the model output and the prediction error, in all algorithms. This ensures that the algorithms can converge to the true parameters in ideal situations. The reason is that the approximation of the derivative only affects the adaptation gain – the adaptation error is unaffected by the approximation due to the use of the correct quantization function in the computation of the model output and the prediction error.

The simplest approximation of the derivative of the quantizer is provided by a constant k_0. The more advanced smooth approximation of the quantizer is given by

$$q(y) \approx \bar{q}(y) = q_1 + \sum_{i=1}^{M-1} (q_{i+1} - q_i) \frac{1}{\pi} \left(\arctan\left(k(y - y_i)\right) + \frac{\pi}{2} \right) . \qquad (1)$$

Here $q(y)$ denotes the exact quantization function, y denotes the output from the linear dynamic part of the system, $\bar{q}(y)$ denotes the approximation of the quantizer, q_i, $i = 1,\ldots, M$ denotes the M levels of the quantizer, y_i, $i = 1,\ldots, M-1$ denotes the $M-1$ switch points of the quantizer and k is a parameter that controls the "sharpness" of the smooth approximation. Note that it is not required that the quantizer is linear or monotone. A differentiation then results in the following approximation of the derivative of the quantizer

$$\frac{dq(y)}{dy} \approx \frac{d\bar{q}(y)}{dy} = \sum_{i=1}^{M-1} (q_{i+1} - q_i) \frac{1}{\pi} \frac{k}{1 + k^2 (y - y_i)^2} . \qquad (2)$$

It is again stressed that equation (1) is never used in the algorithms. The model output and the prediction error is base on the exact quantizer defined by the levels and the switch points.

The software package described in the present report addresses the problem with output quantization, in the cases where the dynamic system is a linear FIR or IIR system and where the quantizer is known. The core of the SW package is a MATLAB implementation of 5 SISO output error identification algorithms, each exploiting a specific approximation of the derivative of the

quantizer. Using these approximation 2 recursive stochastic gradient algorithms and 3 recursive Gauss-Newton algorithms are obtained. It should be noted that the stochastic gradient algorithms are primarily suited for (high order) FIR systems – they converge very slowly for IIR systems due to the large eigenvalue spread of the Hessian that typically results for IIR systems. Arbitrarily colored additive measurement noise is handled by all algorithms.

The algorithms that are implemented include:

- Algorithm 1: A Gauss-Newton algorithm for identification of linear dynamics of a system in the FIR and IIR cases. Fix regressor filtering is used and the gradient of the output quantizer is approximated by a user chosen constant.
- Algorithm 2: A Gauss-Newton algorithm for identification of linear dynamics of a system in the FIR and IIR cases. Fix regressor filtering is used and the gradient of the output quantizer is approximated by the derivative of a smooth approximation of the quantizer, expressed by a sum of scaled arctan-functions. This approximation can be replaced by the user by a straightforward modification of one single MATLAB function.
- Algorithm 3: A stochastic gradient algorithm for identification of linear dynamics of a system in the FIR and IIR cases. The algorithm is only recommended to be used in the FIR case. Fix regressor filtering is used and the gradient of the output quantizer is approximated by a user chosen constant.
- Algorithm 4: A stochastic gradient algorithm for identification of linear dynamics of a system in the FIR and IIR cases. The algorithm is only recommended to be used in the FIR case. Fix regressor filtering is used and the gradient of the output quantizer is approximated by the derivative of a smooth approximation of the quantizer, expressed by a sum of scaled arctan-functions. This approximation can be replaced by the user by a straightforward modification of one single MATLAB function.
- Algorithm 5: A Gauss-Newton algorithm for identification of linear dynamics of a system in the FIR and IIR cases. Adaptive regressor filtering is used and the gradient of the output quantizer is approximated by the derivative of a smooth approximation of the quantizer, expressed by a sum of scaled arctan-functions. This approximation can be replaced by the user by a straightforward modification of one single MATLAB function. This algorithm can be viewed as a close approximation of the RPEM.

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 5 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. 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 three scripts of this group define example dynamic system, that are then used for generation of simulated data. The scripts are **InitializeDataGenerationLinear.m, FIRQuantization.m** (generates inputs and noisy, quantized outputs of a finite impulse response system) and **IIRQuantization.m** (generates inputs and noisy, quantized outputs of an infinite impulse response system).
- *Recursive identification*. The 6 scripts of this group perform the actual identification tasks. The scripts and functions are **InitializeAlgorithmsLinear.m**, **OEAlgorithm1.m – OEAlgorithm4.m** (algorithms with fixed regressor filtering) and **OERPEM.m** (algorithm with adaptive regressor filtering and advanced approximation of the output quantizer).
- *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 quantization function and the corresponding approximation of the derivative. Note that quantization parameters need to be setup in this function.
- *Display of results*. There are three scripts in this group. The scripts are **PlotData.m**, **PlotOutputError.m** and **PlotParameters.m**
- *Model validation* There are four scripts in this group. The scripts are **ValidateCriterion.m**, **ValidateOutputError.m**, **ValidatePoleZero.m** and **ValidateMeanResiduals.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 quantizer and the approximation of 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 six 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 vectors 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 or 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 **InitializeDataGenerationLinear.m**. The parameters that define the data generation are directly written into this script. These parameters define the type of input.
3. Executes **InitializeDataGenerationLinear.m**. This loads the necessary parameters into the MATLAB workspace.
4. Generates data by execution of either of the 2 provided systems, **FIRQuantization.m**. or **IIRQuantization.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

$$y(t) = \frac{0.0928q^{-1} + 0.0839q^{-2}}{1 - 1.5641q^{-1} + 0.7408q^{-2}} u(t) \ , \tag{3}$$

$$y_n(t) = q\big(y(t)\big) + e(t) \ , \tag{4}$$

$$q\big(y\big) = \begin{cases} -1.000, & y \le 0.5 \\ 1.000, & y > 0.5 \end{cases} . \tag{5}$$

The procedure is as follows. First select input_type=3 in **InitializeDataGenerationLinear.m**. This input type is a PRBS like signal, but with uniformly distributed amplitudes, cf. [9]. Then execute **InitializeDataGenerationLinear.m** and **IIRQuantization.m** from the MATLAB command line. Note that other input types can be specified, please refer to the source code.

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 **PlotData** generates the following plot.

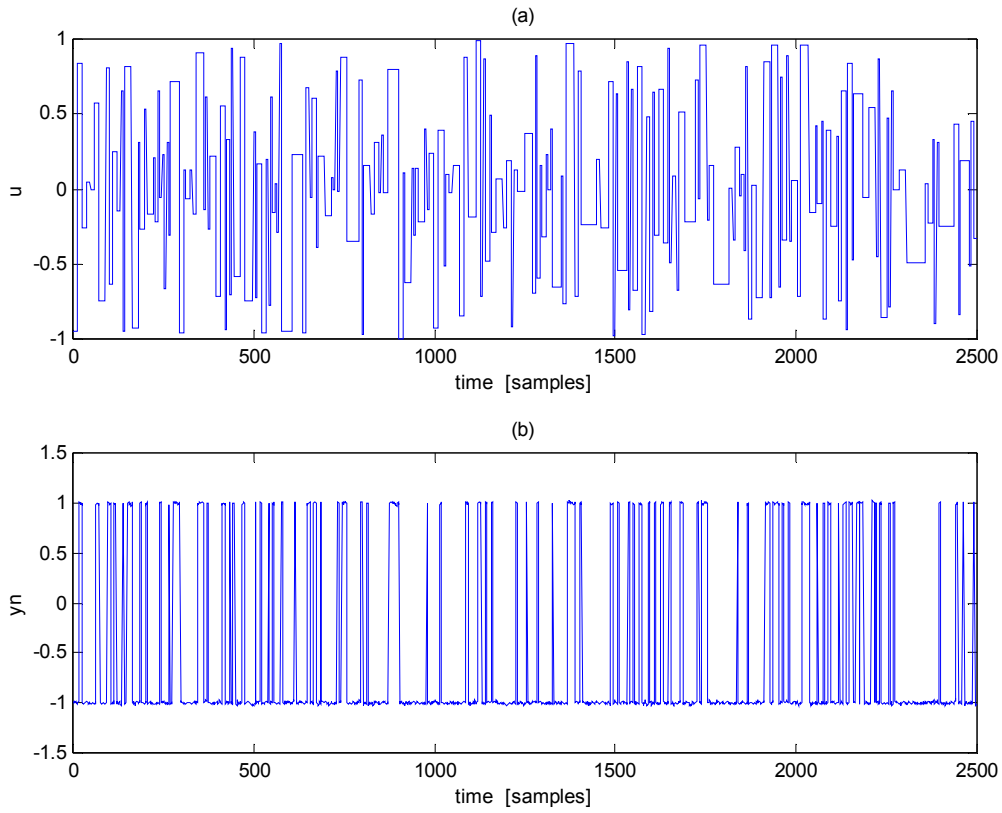

Figure 1: The result of a PlotData command.

4. Recursive Identification

The preparation for the identification run requires that the user

1. Modifies the **Eval_fn.m** function to describe the quantizer and the corresponding derivative. The user may stick to the approximation principle of (1) and (2). In such a case it is sufficient to select the quantization levels of the model (`quantLevelAlgorithm`), the switch points of the model (`quantGridAlgorithm`) as well as the value of the parameter k (`quantizationSharpness`). The default setting is used below, cf. **Eval_fn.m** and the system description (3) – (5) of Example 1.
2. Provides further input data in the script **InitializeAlgorithmsLinear.m**. 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 value of the constant approximation of the derivative of the quantizer (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. Note that the parameters of the quantizer and the derivative approximation are defined in **Eval_fn.m**.
3. Executes **InitializeAlgorithmsLinear.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. **OERPEM** is typed at the MATLAB command line. In response the parameter vector is identified from the data and typed out. In the IIR case 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. In the FIR case all parameters correspond to the B-polynomial. Note that the order of the F- and B-polynomial should be the same in the IIR case. In the FIR case the user should specify F_0=1 in the script **InitializeAlgorithmsLinear.m**.

The result of the execution of the approximation of the RPEM is

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

```
» OERPEM

theta =

  -1.5493    0.7319    0.0854    0.0989

»
```

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 **PlotParameters.m**. The components of the parameter vector are then plotted as a function of time.

Example 4: The command **PlotParameters** in the MATLAB command window generates the following plot.

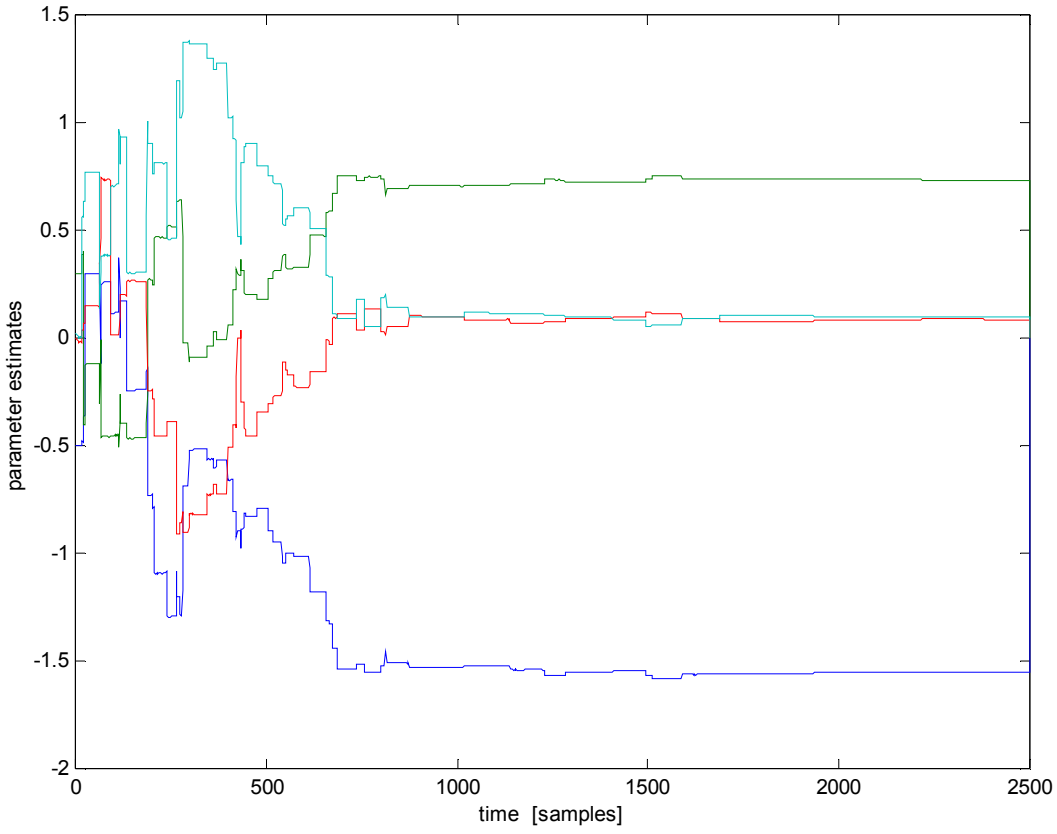

Figure 2: The result of a **PlotParameters** command.

5.2 Output errors

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

1. Execute the script **PlotOutputError.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 **PlotOutputError** in the MATLAB command window command generates the following plot

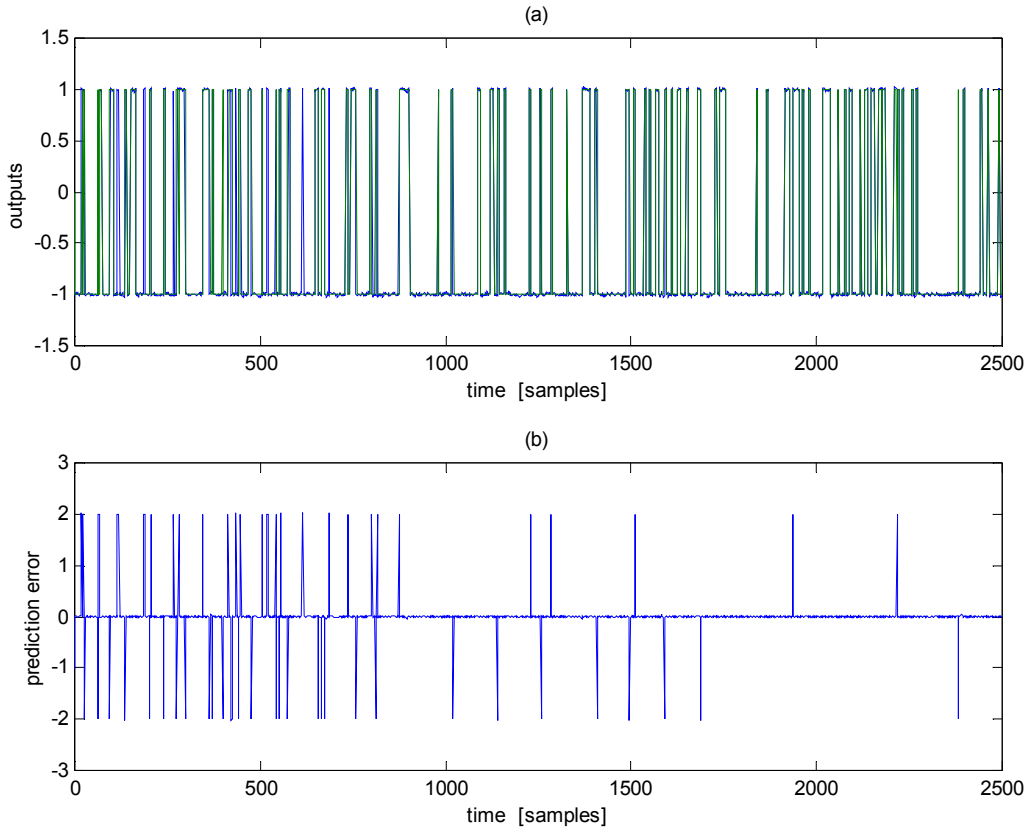

Figure 3: The result of a **PlotOutputError** command.

6. Model validation

Four scripts are provided for model validation purposes. The first method studies the value of the loss function.

6.1 RPEM loss function

The 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 **ValidateCriterion.m**. The loss function is then computed.

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

```
» ValidateCriterion

loss_function =

   0.0129

»                                                                    .
```

6.2 Simulated output errors

The simulated output errors are similar to what is provided by **PlotOutputError**. 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 **ValidateOutputError.m**.

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

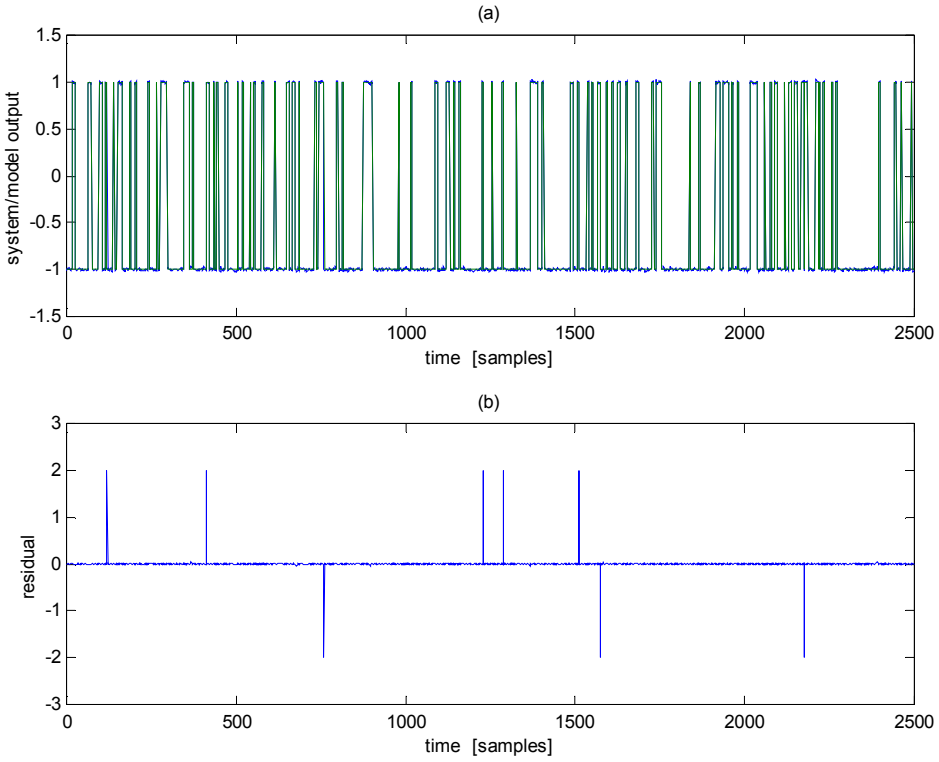

Figure 4: The result of a **ValidateOutputError** command.

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 **ValidatePoleZero.m**.

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

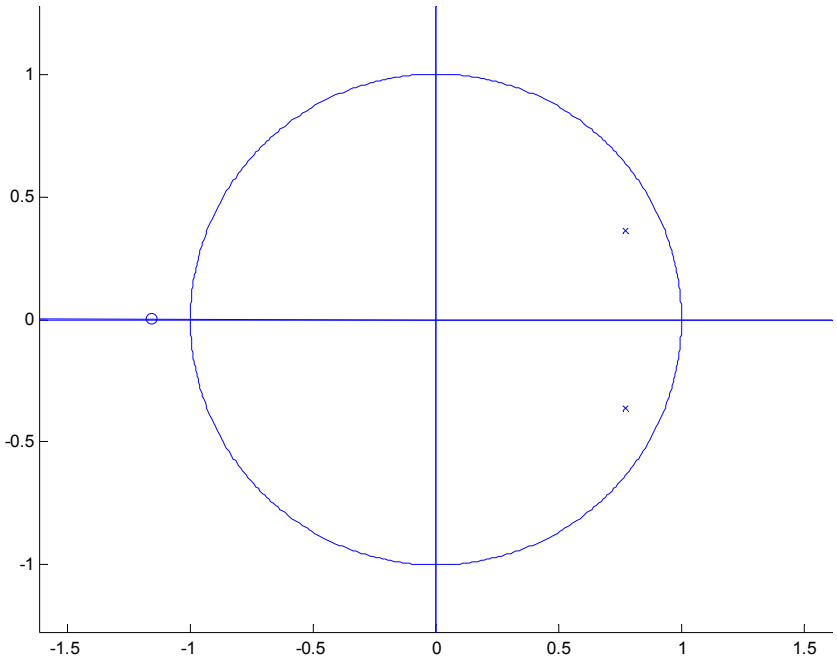

Figure 5: The first result of a **ValidatePoleZero** command. – Overview.

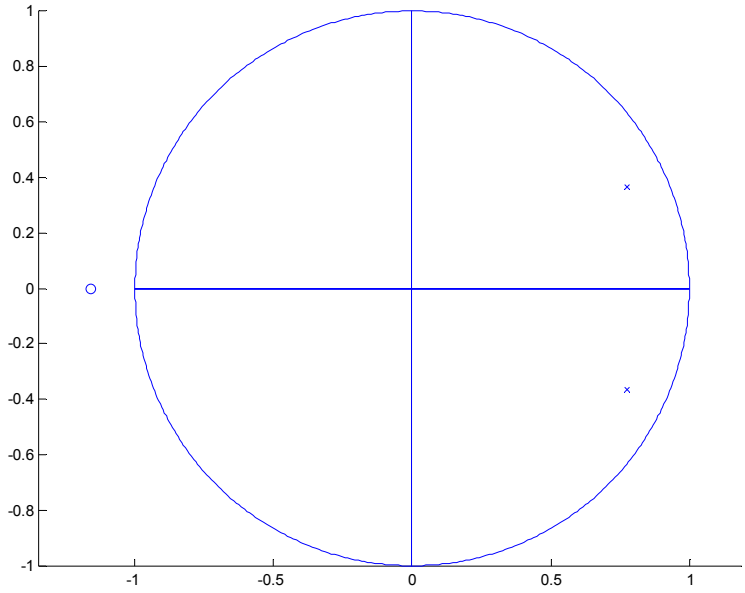

Figure 6: The second result of the **ValidatePoleZero** command. Detailed view. For the model at hand, there is little difference between the figures. Other scenarios generate larger differences.

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 [9] 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 **ValidateMeanResiduals.m**.

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

```
» I_l=(-1.1:0.2:1.1)

I_l =

  Columns 1 through 7

   -1.1000   -0.9000   -0.7000   -0.5000   -0.3000   -0.1000    0.1000

  Columns 8 through 12

    0.3000    0.5000    0.7000    0.9000    1.1000

» ValidateMeanResiduals.
```

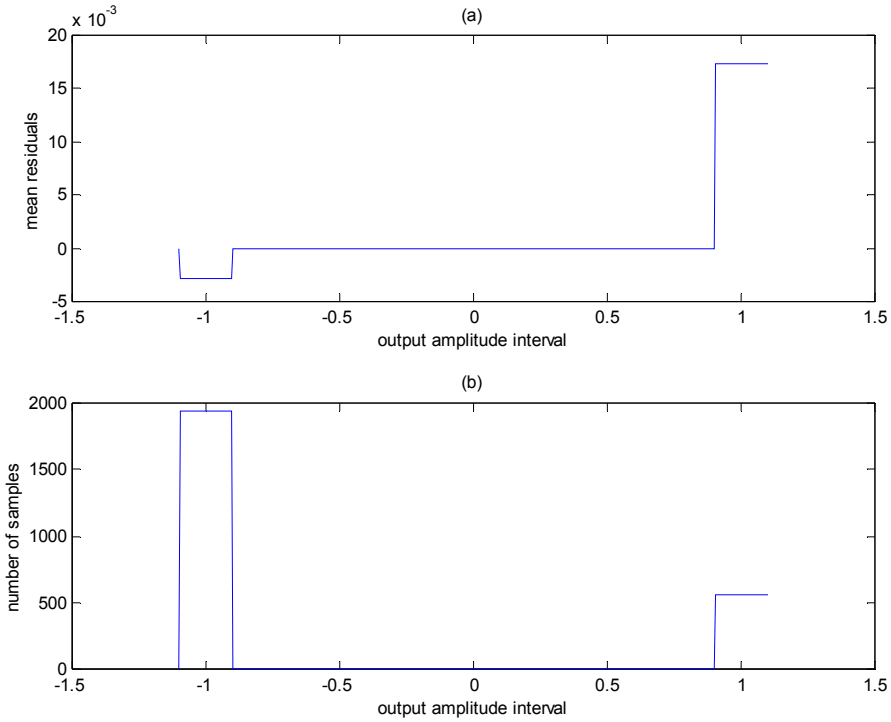

Figure 7: The effect of a **ValidateMeanResiduals** command

7. Summary

This report describes a software package for identification of linear systems with output quantization.. 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.

8. 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, 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.

[4] 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.

[5] T. Wigren and B. Carlsson, "On the use of quantized lambda-sensor measurements in recursive identification of air-fuel mixing dynamics", Preprints of the IFAC Workshop on Intelligent Components for Vehicles , Seville, Spain, pp. 77-82, March 23-24, 1998.

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

[7] E. Sviestins and T. Wigren, "Optimal recursive state estimation with quantized measurements", IEEE Trans. Automat. Contr. , vol. 45, no. 4, pp. 762-767, 2000.

[8] E. Sviestins and T. Wigren, "Nonlinear techniques for Mode C climb/descent rate estimation in ATC systems", IEEE Trans. Contr. Sys. Tech. , vol. 9, pp. 163-174, 2001.

[9] 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.

[10] T. Wigren, "MATLAB software for Recursive Identification of Wiener Systems - Revision 2", available at http://www.it.uu.se/research/publications/reports/2007-010/WRIS.zip , Uppsala University, Uppsala, Sweden, March, 2007.