Application Note

Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

Adaptive RLS Algorithms
Reference Implementations for 32bit Win7

Jiří Kadlec, Raissa Likhonina
kadlec@utia.cas.cz
phone: +420 2 6605 2216
The Czech Academy of Sciences, Institute of Information Theory and Automation

Revision history:

Rev.	Date	Author	Description
1	28.11.2016	J. Kadlec	UTIA ZS DSP demos for Matlab R2016b 64bit
2	08.12.2016	J. Kadlec R. Likhonina	Added identification algorithms: 74f, 75f, 76, 76f, 76l, 77, 77f, 77l Filtration error e(k|k) is used for display and for numeric comparison of algorithms (instead of prediction error e(k|k-1). Tests with selected presentations
3	06.02.2016	R. Likhonina	Design of two applications for Win64 with GUI Based on Matlab 2016.b Compiler: test_algorithms.exe (version 1.6) test_algorithm_comparison.exe (version 1.6)
4	24.10.2017	J. Kadlec	References to related work, licensing conditions. Design of two applications for Win32 with GUI. Apps are generated by commercial Matlab Compiler (2015.b 32b): test_algorithms.exe (version 1.6) test_algorithm_comparison.exe (version 1.6) Apps run on Win7 32b/64b or Win10 32b/64b

Acknowledgements:

This work has been partially supported by the ECSEL JU project SILENSE "(Ultra)Sound Interfaces and Low Energy iNtegrated Sensors" Project No.: ECSEL JU 737487-2 and MSMT 8A17006 (Ministry of Education Youth and Sports of the Czech Republic). See web: http://www.silense.eu/

Table of contents

Table of figures

Table of tables

1. Summary

1.1 Objectives

This Application Note aims to present set of adaptive recursive least squares system identification algorithms based on the Bayesian extensions of real-time adaptive system identification as well as extending the existing recursive least square adaptive algorithms for estimation of time varying parameters in the applications of acoustic signal processing. The included reference adaptive algorithms are implemented in Matlab 2015b (32b). Algorithms serve as "golden" reference models for the embedded implementations on dedicated processors like Arm Cortex A9 and the FPGA programmable logic accelerators in devices like the Xilinx Zynq. Algorithms are numerically robust. Algorithms are implemented in double precision floating point (64b), single precision floating point (32b) and in logarithmic arithmetic with precision 32b and 19bit. All algorithms except for lattice filter are implemented both with exponential forgetting and directional forgetting, which use more complex computation and allows to "forget" previous information only if incoming data are bringing new information with them. In several cases identification process can benefit from this, as it will be shown on examples in the following chapters. Lattice filter, however, is performed with exponential forgetting only.

The Application Note also presents adaptive recursive least squares system identification algorithms taking advantage of dynamic normalization of the core of the algorithm into the guarantied range <-1, 1> for all variables. These algorithmic cores are suitable for the fixed-point implementation (14b). Naturally, the fixed-point implementation with representation of all variables as only 14b fixed-point numbers results in decreasing precision. But it opens possibility of potentially ultralow power implementation of recursive RLS on parallel HW accelerators with custom fixed point arithmetic. This is crucial for implementing in low power embedded systems. As a result then using these algorithms in fixed-point we plan to develop and implement systolic, pipelined SoC IP core in form of HW accelerator in the programmable logic part of the Xilinx Zynq device (28nm) and possibly also in the 16nm Zynq UltraScale+ device.

This application note and the related evaluated package provides reference base for this development. The evaluated package requires Win7 (32b or 64b) PC or Win10 (32b or 64b) PC.

Included scripts and precompiled algorithms can be used with Matlab R2015b (32b).

Alternatively, the package can be also used without Matlab. All scripts are precompiled by the Matlab Compiler R2015.b, 32b as packages supporting installation and standalone execution on Win7 (32b or 64b) PC or Win10 (32b or 64b) PC without Matlab.

1.2 Directory

Download the package and unzip to separate folder. Example

C:\VM_07\R2015b\dsp_1_6\

In this folder, you will find Matlab files executing algorithms and a sub-folder ***"private"*** with Matlab **mexw32** files with precompiled algorithms.

It also comprises sub-folders "**test_algorithms**" and "**test_algorithm_comparison**" for installing and using precompiled scripts.

1.3 Notation

EF	= exponential forgetting
DF	= directional forgetting
DP	= double precision
SP	= single precision
LNS	= Logarithmic numbering system (32b or 19b)
INT	= 14b Integer computation of normalised <-1,1> part of identification algorithm.
NORM	= Normalised <-1,1> part of identification algorithm.
LAT	= Lattice filter.
QR	= Information filter with square root operations.
QRD	= Information filter without square root operations.
QR_rotations	= Information filter with square root operations and orthogonal rotatons in form of (sin, cos).
QRD_rotations	= Information filter without square root operations and orthogonal rotatons in form of (sin, cos).
Inv_QRD	= Inverse update without square root operations.

2. Demos

2.1 Demos with persistent excitation

The present demos describe the performance of FIR filters based on different algorithms in case of persistent excitation. For the purpose of simplicity the third order regression model is used; thus, only three parameters are estimated. The algorithms are implemented in a way that input and output are always in DP, but inside the algorithms calculation can be made in different data formats. The data formats used in the algorithms are double precision (DP) and single precision (SP) arithmetic, logarithmic numbering system (32b or 19b) (LNS) and 14b integer arithmetic (INT). The examples illustrating estimation of two parameters include the following algorithms: lattice filter, inverse information filters without square root operations, information filters with/without square root operations, information filters with/without square root operations and orthogonal rotations in form of (sin, cos), normalized information filters with square root operations. The algorithms are performed using exponential forgetting (EF) or directional forgetting (DF).

The name of algorithms downloaded from the web and presented in this Application Note obtains information about the order of the model. Also it differentiates whether logarithmic numbering system of 32b or 19b is used. The examples of a name structure are

1. **"short_filter_lns19_d1_1_51l"** indicates that the third order model and logarithmic numbering system of 19b is used.
2. **"long_filter_lns32_d1_1_51l"** is for the higher order model (in our case it is the 23rd order model) and logarithmic numbering system of 32b.

If the name does not comprise **"_lns32"** or **"_lns19"**, it means that another data format is used: double precision, single precision or 14b integer computation. For example, Matlab file **short_filter_d1_1_46.m** performs Lattice filter of the third order using double precision. For more information refer to Table 1.

Table 1 lists the algorithms performed in Matlab R2015b. The specification of the model order and logarithmic numbering system is omitted here, as the second part of the name is the same for both filters (long and short one) and for both logarithmic numbering systems. The name of the algorithms in Table 1 begins with **"_"** to show, that the first part is omitted.

Table 1: Demos with persistent excitation

File name	Algorithm	Data format	Forgetting	Normalization	Bits
Demos with Exponential Forgetting					
_d1_1_46.m	LAT	DP	EF		
_d1_1_51.m	Inv_QRD	DP	EF		
_d1_1_51f.m	Inv_QRD	SP	EF		32bit
_d1_1_51l.m	Inv_QRD	LNS	EF		32bit/19bit
_d1_1_70.m	QRD	DP	EF		
_d1_1_70f.m	QRD	SP	EF		32bit
_d1_1_70l.m	QRD	LNS	EF		32bit/19bit
_d1_1_73.m	QR	DP	EF	NORM	
_d1_1_73f.m	QR	SP	EF	NORM	32bit
_d1_1_73l.m	QR	LNS	EF	NORM	32bit/19bit

_d1_1_76.m	QRD_rotations	DP	EF		
_d1_1_76f.m	QRD_rotations	SP	EF		32bit
_d1_1_76l.m	QRD_rotations	LNS	EF		32bit/19bit
_d1_1_77.m	QR_rotations	DP	EF		
_d1_1_77f.m	QR_rotations	SP	EF		32bit
_d1_1_77l.m	QR_rotations	LNS	EF		32bit/19bit
_d1_1_83.m	QR	INT	EF	NORM	14bit
Demos with Directional Forgetting					
_d1_2_51.m	Inv_QRD	DP	DF		
_d1_2_51f.m	Inv_QRD	SP	DF		32bit
_d1_2_51l.m	Inv_QRD	LNS	DF		32bit/19bit
_d1_2_74.m	QR	DP	DF	NORM	
_d1_2_74f.m	QR	SP	DF	NORM	32bit
_d1_2_75.m	QRD	DP	DF		
_d1_2_75f.m	QRD	SP	DF		32bit
_d1_2_84.m	QR	INT	DF	NORM	14bit

The first algorithm in the table is the lattice algorithm. It belongs to the family of RLS algorithms and is cheap as far as computational complexity is concerned. It is based on orthogonal rotations and is suitable for residual extraction problems, when there is no need to compute weights. Thus, the back-substitution step is avoided and computation cost is reduced. This algorithm proves to be sufficiently stable due to the combination of orthogonal transformations and exponential weighting. It is suitable for parallel implementation. Lattice algorithm can be implemented only with exponential weighting. The example of lattice algorithm using exponential forgetting and double precision arithmetic is presented in Matlab file **_d1_1_46.m**.

The RLS algorithms based on so-called inverse QRD updating without square root operations are illustrated in demos containing in their name "51". The motivation to use square-root-free Givens rotations is to prevent computational bottleneck caused by square-root computation in hardware implementations. The inverse QRD algorithms prove to be numerically stable. Parallel implementation is relatively complex. The demos presented here include inverse QRD algorithms both with EF and DF. They use different data format inside the computational process: double precision arithmetic, single precision arithmetic and logarithmic numbering system (32b and 19b). For more details, please, see **_d1_1_51.m** in Table 1.

The demos listed in the table also comprise information filters without square root operations, i.e. QRD algorithms. They are based on QRD decomposition of the input matrix to avoid the problem with positive definiteness of the matrix due to rounding errors and, thus, to provide a numerically stable solution. The algorithms presented in demos use exponential forgetting and different data formats: double precision and signal precision arithmetic, and logarithmic numbering system (32b and 19b). See **_d1_1_70.m** in Table 1.

Next three algorithms listed in Table 1 are information filters with square root operations obtaining normalized <-1, 1> identification part (see **_d1_1_73.m**). The normalization aims at solving the problem of overflows and instability of the standard algorithms. Besides, it also reduces the computational complexity and makes the algorithms cheaper for implementation. The examples of normalized QR algorithms also comprise filters both with exponential and directional forgetting and those using different data formats: double precision and single precision arithmetic, logarithmic numbering system (32b and 19b) and 14b integer computation system (see **_d1_1_83.m** in Table 1).

Information filters without square root operations implemented with directional forgetting are also available and presented here in double precision arithmetic and single precision arithmetic (see **_d1_2_74.m** and **_d1_2_75.m** in Table 1).

The algorithms listed as QRD_rotations are information filters implemented without square root operations and orthogonal rotations in form of (sin, cos). Not using square root operations and orthogonal rotations makes the algorithms cheaper in comparison with other algorithms. The present demos show the performance of these algorithms with exponential forgetting and different data format including double and single precision arithmetic and logarithmic numbering system (32b and 19b). See **_d1_1_76.m** in Table 1.

The algorithms named QR_rotations are alternatively information filters with square root operations and orthogonal rotations in form of (sin, cos). They are numerically stable and also computationally cheap. These algorithms are implemented with exponential forgetting and again using different data formats: double precision arithmetic, single precision arithmetic and logarithmic numbering system (32b and 19b). For more details, please, refer to **_d1_1_77.m** in Table 1.

Two last interesting cases are presented by information filters implemented with square root operations, but using 14b integer computation. Though, the accuracy of the algorithms under consideration is not as high as of others previously mentioned; however, they are numerically robust, sufficiently accurate and fast and, what is important, are suitable for implementation in embedded systems. These algorithms use normalization of identification part to fit the interval <-1, 1>, thus, preventing the overflow problems. The algorithms are presented both with exponential forgetting (see **_d1_1_83.m** in Table 1) and directional forgetting (see **_d1_2_84.m** in Table 1).

To discuss the output of the algorithms inverse QRD filter with DP and EF have been used as an example. The filtering results can be viewed in Figure 1.

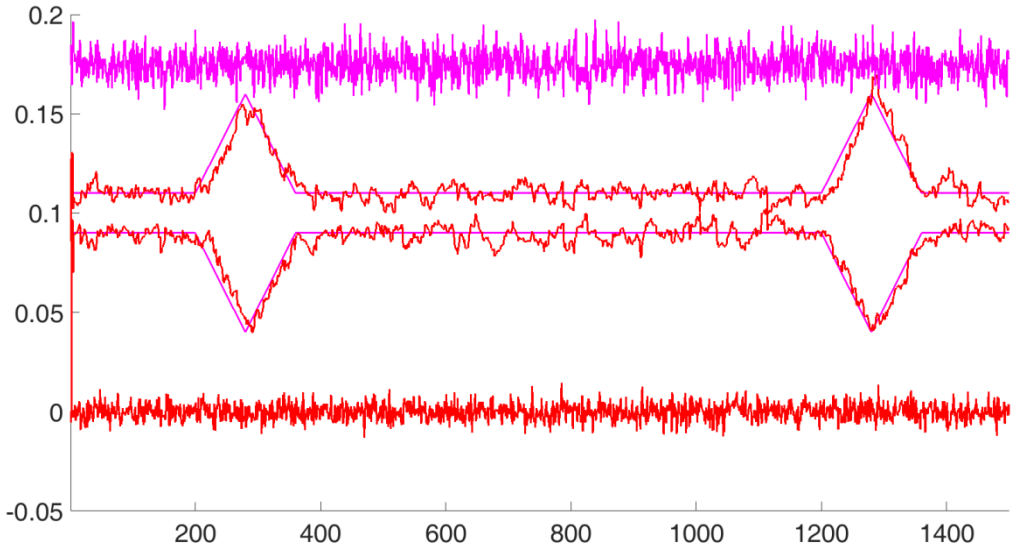


Figure 1: Inverse QRD filter, DP, EF

The upper magenta curve is an input signal causing persistent excitation of the system. The bottom red curve represents filtration error. The interesting is estimation of the parameters, which is shown in the middle part of the graph. Let us remind that for the purpose of simplicity the third order regression model is used to show the performance of the algorithms; thus, there are three parameters estimated. However, on the graph as an example only two parameters are depicted. They are represented by magenta curves in the middle of the graph. Two red curves along the magenta ones show the process of estimation, which is sufficiently accurate and imitates the shape of the magenta lines. The results of the performance of other algorithms in case of persistently excited system are very similar and are omitted here. For more details, please, refer to Matlab files of corresponding algorithm.

2.2 Demos with period of ill-defined excitation

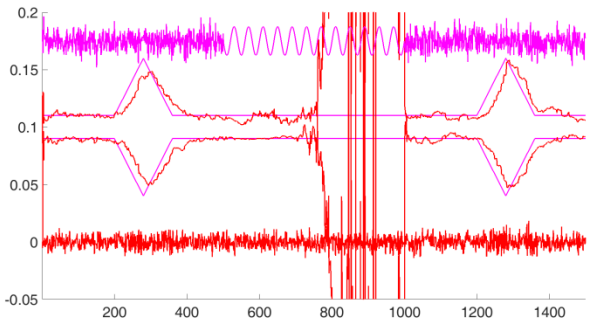
The demos described in this chapter present examples of so-called ill-defined excitation. The first and the last part of the input signal are persistently excited. The middle part of the input is generated in the form of a regular sine, so that the system is ill-excited. It is supposed that in this part the algorithms would have problems with identification. The examples comprise the same algorithms as they have been described in previous chapter, that is: lattice filter, inverse QR filters without square root operations, information filters with square root operations, normalized information filters with square root operations, information filters with/without square root operations, information filters with/without square root operations and orthogonal rotations in form of (sin, cos), normalized information filters with square root operations. The algorithms can be implemented with exponential forgetting (EF) or directional forgetting (DF). Inside the algorithms there can be used different data formats: double precision (DP) and single precision (SP) arithmetic, logarithmic numbering system (32b or 19b) (LNS), 14b integer computation (INT). The end results are in double precision arithmetic. Table 2 lists available algorithms. The structure of algorithm name mentioned in previous chapter is valid for demos with period of ill-defined excitation as well.

Table 2: Demos with period of ill-defined excitation

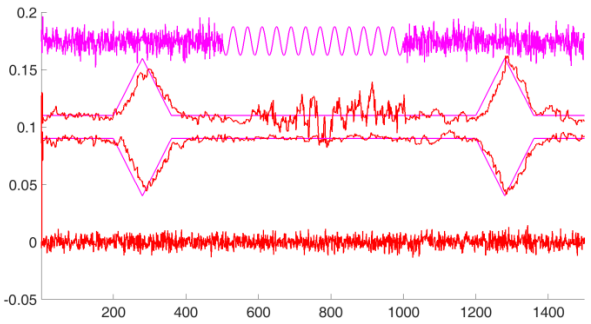
File name	Algorithm	Data format	Forgetting	Normalization	Bits
Demos with Exponential Forgetting					
_d1_3_46.m	LAT	DP	EF		
_d1_3_51.m	Inv_QRD	DP	EF		
_d1_3_51f.m	Inv_QRD	SP	EF		32bit
_d1_3_51l.m	Inv_QRD	LNS	EF		32bit/19bit
_d1_3_70.m	QRD	DP	EF		
_d1_3_70f.m	QRD	SP	EF		32bit
_d1_3_70l.m	QRD	LNS	EF		32bit/19bit
_d1_3_73.m	QR	DP	EF	NORM	
_d1_3_73f.m	QR	SP	EF	NORM	32bit
_d1_3_73l.m	QR	LNS	EF	NORM	32bit/19bit
_d1_3_76.m	QRD_rotations	DP	EF		
_d1_3_76f.m	QRD_rotations	SP	EF		32bit
_d1_3_76l.m	QRD_rotations	LNS	EF		32bit/19bit
_d1_3_77.m	QR_rotations	DP	EF		
_d1_3_77f.m	QR_rotations	SP	EF		32bit
_d1_3_77l.m	QR_rotations	LNS	EF		32bit/19bit
_d1_3_83.m	QR	INT	EF	NORM	14bit
Demos with Directional Forgetting					
_d1_4_51.m	Inv_QRD	DP	DF		
_d1_4_51f.m	Inv_QRD	SP	DF		32bit
_d1_4_51l.m	Inv_QRD	LNS	DF		32bit/19bit
_d1_4_74.m	QR	DP	DF	NORM	
_d1_4_74f.m	QR	SP	DF	NORM	32bit
_d1_4_75.m	QRD	DP	DF		
_d1_4_75f.m	QRD	SP	DF		32bit
_d1_4_84.m	QR	INT	DF	NORM	14bit

The outputs of the algorithms are illustrated by the example of inverse QRD using logarithmic numbering system (both for 32b and 19b) and normalized QR filter using 14b integer arithmetic, both for EF and DF.

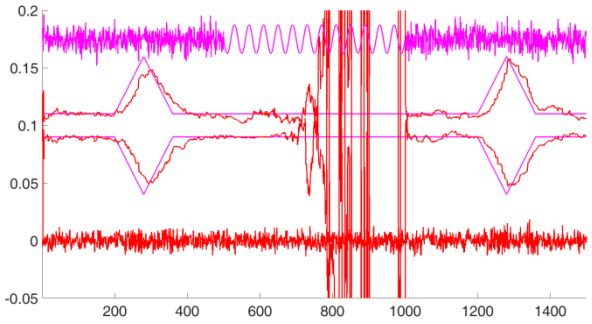
Inverse QRD filter, LNS 32bit, EF



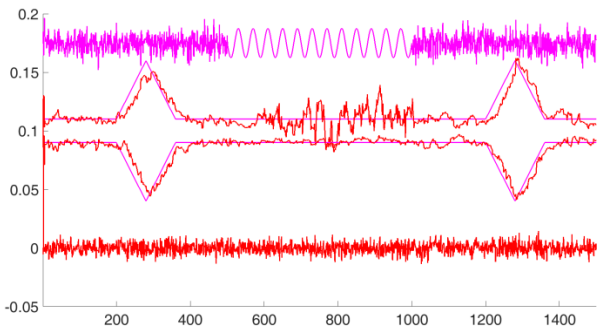
Inverse QRD filter, LNS 32bit, DF



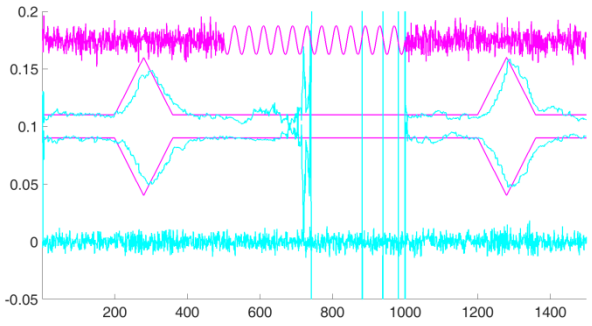
Inverse QRD filter, LNS 19bit, EF



Inverse QRD filter, LNS 19bit, DF



QR filter, INT, EF



QR filter, INT, DF

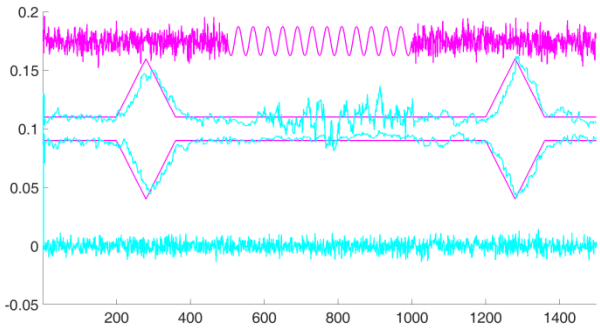


Figure 2: Comparison of outputs from different filters

The upper magenta curve is an input signal, which is created by a regular sine in its middle part. Due to this sinewave the system in this part is ill-excited and the estimation of the parameters is far from accurate. The bottom curve represents filtration errors. Two magenta lines in the middle of the graphs stand for two parameters, which estimation is made

by the corresponding algorithm. Estimation progress itself is shown by red curves for inverse QR filter examples and cyan curves for QR filter examples (two pictures in the bottom).

In Figure 2 the differences between algorithms using EF can be easily viewed (the left part of the figure). In the middle part of the signal the estimation of the parameters is quite a problem for all three algorithms; however, the less accurate is QR filter using 14b integer computation.

The right side of the figure shows the outputs for the same algorithms, but using directional forgetting. It is obvious that in case of ill-defined excitation the algorithms with DF perform much better than those with EF. The reason is that when the input signal contains less information, then in case of using EF all information is gradually wiped out. The algorithms with DF discount old data only when there is actually new information incoming to the model.

The outputs of other filters look more or less similar and omitted here. For seeing them, please, use a corresponding Matlab file.

3. Comparison of results

3.1 Example data

After running any from discussed algorithms the example data including parameter estimations and filtration errors are saved in the form: '*algorithm_name.mat*'.

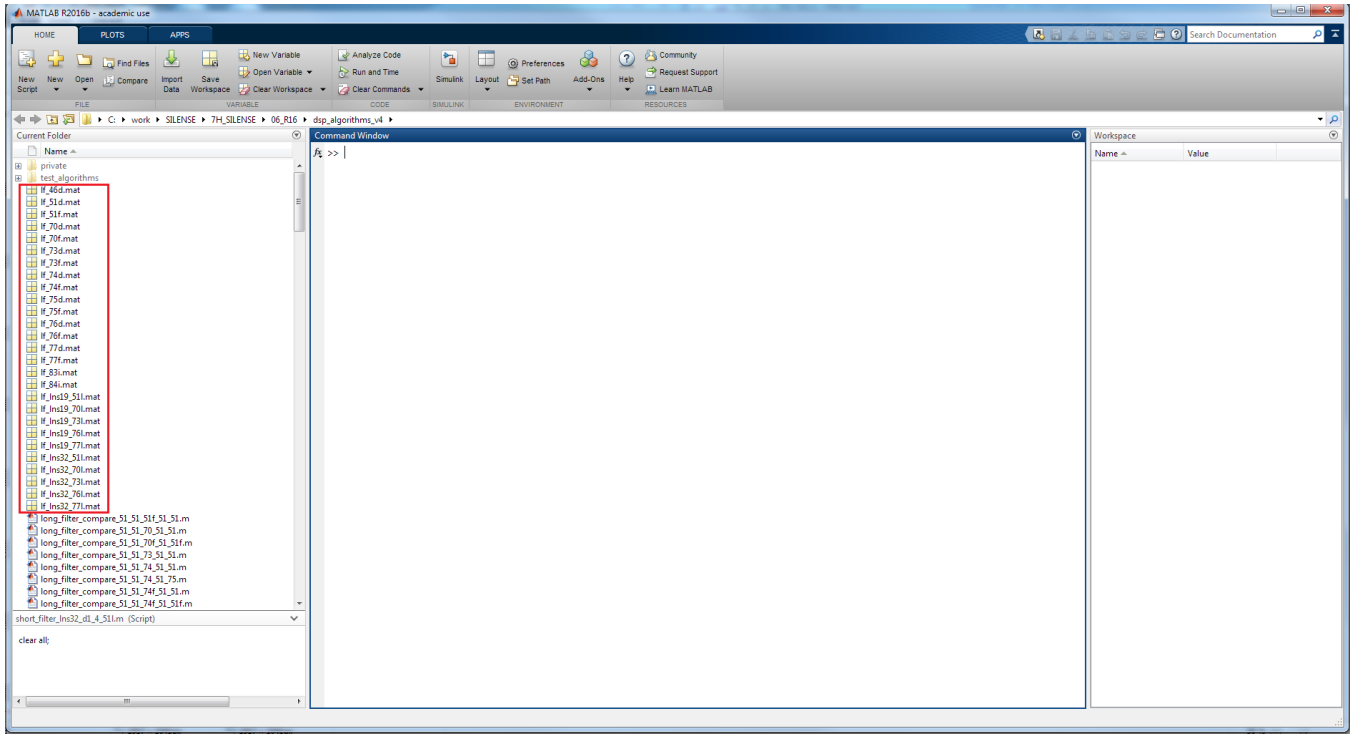


Figure 3: Example data

The examples of saved data are:

1. **lf_46d.mat** – Lattice filter with DP and EF, higher order model
2. **lf_51d.mat** – Inverse QRD filter with DP and EF/DF, higher order model
3. **lf_51f.mat** – Inverse QRD filter with SP and EF/DF, higher order model
4. **lf_lns32_51l.mat** - Inverse QRD filter using LNS 32b and EF/DF, higher order model
5. **lf_lns19_51l.mat** – Inverse QRD filter using LNS 19b and EF/DF, higher order model
6. **lf_70d.mat** – QRD filter with DP and EF, higher order model
7. **lf_70f.mat** – QRD filter with SP and EF, higher order model
8. **lf_lns32_70l.mat** – QRD filter using LNS 32b and EF, higher order model
9. **lf_lns19_70l.mat** – QRD filter using LNS 19b and EF, higher order model
10. **lf_73d.mat** – Normalized QR filter with DP and EF, higher order model
11. **lf_73f.mat** – Normalized QR filter with SP and EF, higher order model
12. **lf_lns32_73l.mat** – Normalized QR filter using LNS 32b and EF, higher order model
13. **lf_lns19_73l.mat** – Normalized QR filter using LNS 19b and EF, higher order model
14. **lf_74d.mat** – QR filter with DP and DF, higher order model
15. **lf_74f.mat** – QR filter with SP and DF, higher order model
16. **lf_75d.mat** – QRD filter with DP and DF, higher order model
17. **lf_75f.mat** – QRD filter with SP and DF, higher order model
18. **lf_76d.mat** – QRD_rotations filter with DP and EF, higher order model
19. **lf_76f.mat** – QRD_rotations filter with SP and EF, higher order model
20. **lf_lns32_76l.mat** – QRD_rotations filter using LNS 32b) and EF, higher order model

21. **lf_lns19_76l.mat** – QRD_rotations filter using LNS 19b and EF, higher order model
22. **lf_77d.mat** – QR_rotations filter with DP and EF, higher order model
23. **lf_77f.mat** – QR_rotations filter with SP and EF, higher order model
24. **lf_lns32_77l.mat** – QR_rotations filter using LNS 32b and EF, higher order model
25. **lf_lns19_77l.mat** – QR_rotations filter using LNS 19b and EF, higher order model
26. **lf_83i.mat** – Normalized QR filter using 14b integer computation and EF, higher order model
27. **lf_84i.mat** – Normalized QR filter using 14b integer computation and DF, higher order model

Similar data are saved for the third order model. The only difference is that the name begins with "sf_" instead of "lf_".

3.2 Comparison scripts

This subchapter aims to present Matlab codes, which compare different algorithms to show the differences between them as far as parameter estimation, filtration errors and accumulated numerical error are concerned. The file name has a structure as in the following example:

- **"short_filter_compare_51_70_73_76_77.m"**.

The first part of the name indicates whether the third order model (***"short_filter_"***) or the 23rd order model (***"long_filter_"***) is used. The first number in the name stands for the reference algorithm; other four numbers after it are the algorithms subject to comparison. Thus, in total four algorithms can be compared with the first reference one. The data used for comparison are those saved after running corresponding algorithm and described in previous subchapters.

If some algorithm uses logarithmic numbering system of 32b or 19b, then it is specified in the name of a file as well:

- **"short_filter_lns32_compare_51_51_51l_51_51f.m"** – for 32b logarithmic numbering system,
- **"short_filter_lns19_ compare_51_51_51l_51_51f.m"** – for 19b logarithmic numbering system.

The examples of several comparison scripts are listed below. However, one can create one's own comparison script on the basis of example scripts and data files to compare any desirable algorithms. The first part of the file name in Table 3 is omitted, because the second part is the same for both the third order and higher model as well as for 32b and 19b logarithmic numbering system. Therefore, it would be pointless to repeat it for each of the variants.

Table 3: Examples of comparison scripts

Comparison script	Algorithms compared	
	Reference algorithm	Other algorithms
_compare_51_51_51f_51_51.m	Inverse QRD filter with DP, EF/DF	Inverse QRD filter with SP, EF/DF
_compare_51_51_51l_51_51.m	Inverse QRD filter with DP, EF/DF	Inverse QRD filter with LNS, EF/DF
_compare_51_51_51l_51_51f.m	Inverse QRD filter with DP, EF/DF	Inverse QRD filter with SP, EF/DF Inverse QRD filter with LNS, EF/DF
_compare_51_51_51l_51_83i.m	Inverse QRD filter with DP, EF/DF	Inverse QRD filter with LNS, EF/DF Normalized QR filter with 14b integer computation, EF
_compare_51_51_70f_51_51f.m	Inverse QRD filter with DP, EF/DF	Inverse QRD filter with SP, EF/DF QRD filter with SP, EF

_compare_51_51_70l_51_51l.m	Inverse QRD filter with DP, EF/DF	Inverse QRD filter with LNS, EF/DF QRD filter with LNS, EF
_compare_51_51_70_51_51.m	Inverse QRD filter with DP, EF/DF	QRD filter with DP, EF
_compare_51_51_73_51_51.m	Inverse QRD filter with DP, EF/DF	Normalized QR filter with DP, EF
_compare_51_51_74f_51_51.m	Inverse QRD filter with DP, EF/DF	Normalized QR filter with SP, DF
_compare_51_51_74f_51_51f.m	Inverse QRD filter with DP, EF/DF	Inverse QRD filter with SP, EF/DF Normalized QR filter with SP, DF
_compare_51_51_74f_75f_51f.m	Inverse QRD filter with DP, EF/DF	Inverse QRD filter with SP, EF/DF Normalized QR filter with SP, DF QRD filter with SP, DF
_compare_51_51_74_51_51.m	Inverse QRD filter with DP, EF/DF	Normalized QR filter with DP, DF
_compare_51_51_74_51_75.m	Inverse QRD filter with DP, EF/DF	Normalized QR filter with DP, DF QRD filter with DP, DF
_compare_51_51_75_51_51.m	Inverse QRD filter with DP, EF/DF	QRD filter with DP, DF
_compare_51_51_84_51_51.m	Inverse QRD filter with DP, EF/DF	Normalized QR filter with 14b integer computation, DF
_compare_51_70f_73f_76f_77f.m	Inverse QRD filter with DP, EF/DF	QRD filter with SP, EF Normalized QR filter with SP, EF QRD_rotations filter with SP, EF QR_rotations filter with SP, EF
_compare_51_70l_73l_76l_77l.m	Inverse QRD filter with DP, EF/DF	QRD filter with LNS, EF Normalized QR filter with LNS, EF QRD_rotations filter with LNS, EF QR_rotations filter with LNS, EF
_compare_51_70_73_76_77.m	Inverse QRD filter with DP, EF/DF	QRD filter with DP, EF Normalized QR filter with DP, EF QRD_rotations filter with DP, EF QR_rotations filter with DP, EF
_compare_51_70_76_70_77.m	Inverse QRD filter with DP, EF/DF	QRD filter with DP, EF QRD_rotations filter with DP, EF QR_rotations filter with DP, EF
_compare_70_70_46_70_70.m	QRD filter with DP, EF	Lattice filter with DP, EF
_compare_70_70_73f_70_70.m	QRD filter with DP, EF	Normalized QR filter with SP, EF
_compare_70_70_73f_70_73l.m	QRD filter with DP, EF	Normalized QR filter with SP, EF Normalized QR filter with LNS, EF
_compare_70_70_73l_70_70.m	QRD filter with DP, EF	Normalized QR filter with LNS, EF

_compare_70_70_73_70_70.m	QRD filter with DP, EF	Normalized QR filter with DP, EF
_compare_70_70_76_70_77.m	QRD filter with DP, EF	QRD_rotations filter with DP, EF QR_rotations filter with DP, EF
_compare_70_73_76_46_77.m	QRD filter with DP, EF	QRD_rotations filter with DP, EF QR_rotations filter with DP, EF Lattice filter
_compare_70_73_76_70_77.m	QRD filter with DP, EF	Normalized QR filter with DP, EF QRD_rotations filter with DP, EF QR_rotations filter with DP, EF

According to performed comparison the minimal differences are between algorithms using the same data format. The variations for parameter estimation in this case ranges up to 10^{-16}, for filtration errors – up to 10^{-17} and for accumulated numerical error – up to 10^{-14}. This proves, that presented algorithms, including inverse QRD filters, QR filters, normalized QR filters, QRD filters, QRD_rotations and QR_rotations filters, when using the same data format, perform almost the same results and are equally accurate. The negligible differences between them can be explained by the way they perform calculations.

The following example illustrates comparison results for four algorithms: QRD filter, normalized QR filter, QRD_rotations filter and QR_rotations filter, all using EF and DP. The reference algorithm is inverse QRD filter. Matlab file providing the outputs is **short_filter_compare_51_70_73_76_77.m**.

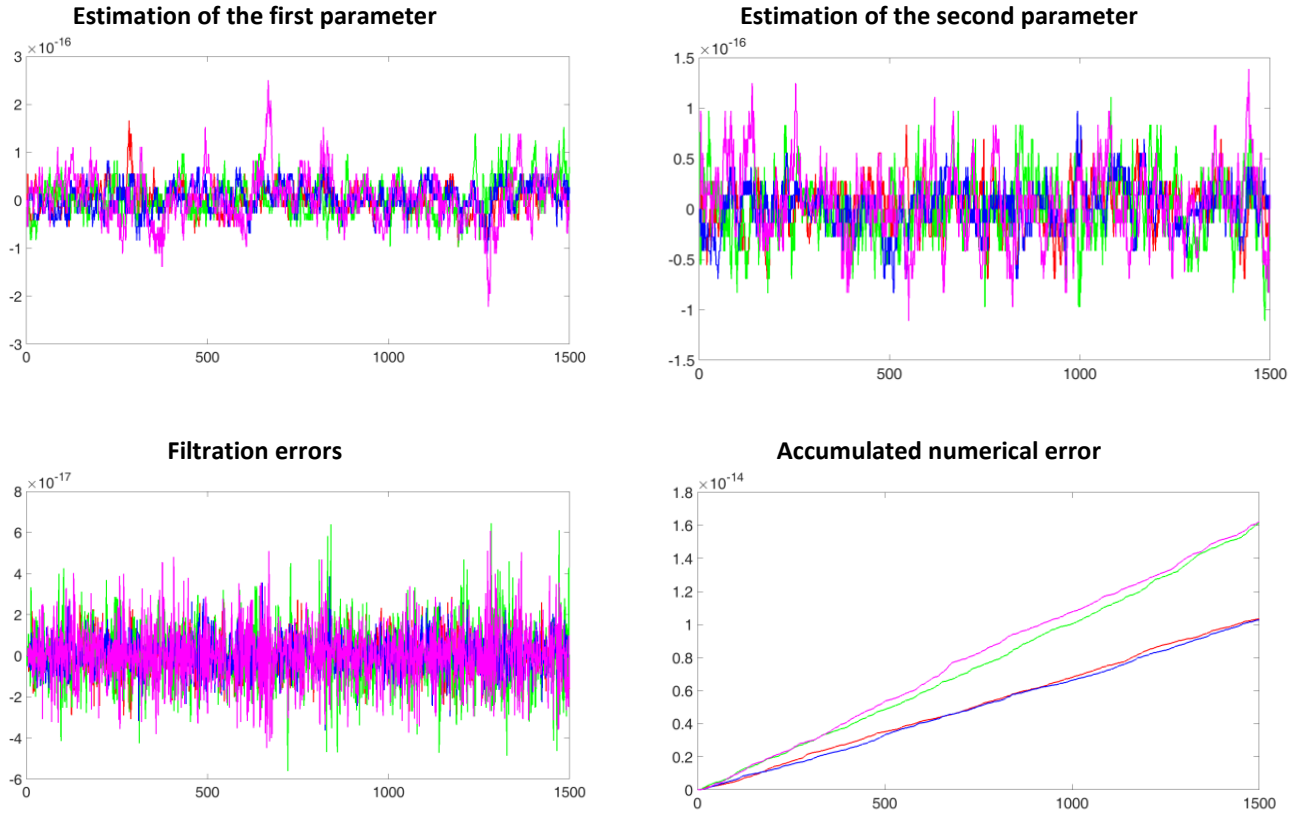


Figure 4: Comparison results for QRD filter, normalized QR filter, QRD_rotations filter and QR_rotations filter (EF, DP)

Red curves on the graphs stand for QRD filter. Green curves present the outputs of normalized QR filter. The results of QRD_rotations filter are colored in blue, while QR_rotations filter is represented by magenta curves.

Two upper graphs show comparison of parameter estimation by different algorithms, while the bottom left picture illustrates filtration errors of the filters compared. The bottom right graph shows how accumulated numerical error is growing in respect to the reference algorithm. It can be seen that the differences are really very small and the algorithms function equally well.

However, the variations in obtained results for the algorithms using data formats and arithmetic with lower precision are already greater and ranges from 10^{-7} to 10^{-8} for parameter estimation, up to 10^{-8} for filtration errors and from 10^{-5} to 10^{-6} for accumulated numerical error.

The example below illustrates comparison results of Matlab file **short_filter_compare_51_51_70f_51_51f.m**, which compares inverse QRD filter with SP and QRD filter with SP with inverse QRD filter with DP as a reference filter.

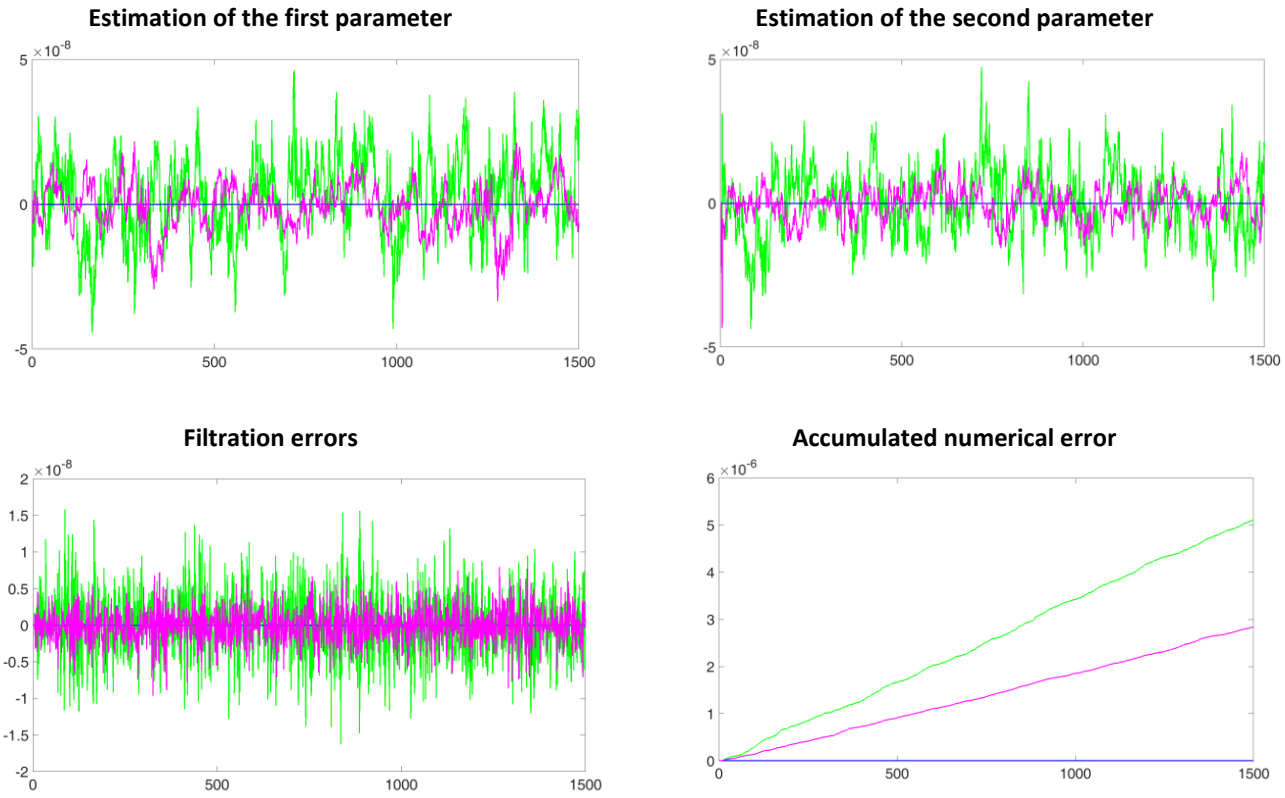


Figure 5: Comparison results for inverse QRD filter with SP and QRD filter with SP

Here the differences in parameter estimation and filtration errors are in the range of 10^{-8}, while accumulated numerical error is in the range of 10^{-6}.

Blue lines on the graph stand for the reference algorithm, i.e. inverse QRD filter with DP. The outputs for QRD filter with SP are colored in green, while the ones for inverse QRD filter with SP are colored in magenta. Though the variations are greater than in the previous examples, they are not so high and the accuracy of the algorithms can be considered to be very similar.

To show the differences in results obtained by the algorithms using single precision arithmetic and logarithmic numbering system Matlab script **short_filter_lns32_compare_51_51_51f_51_51l.m** is created. Here the reference algorithm is inverse QRD filter with DP. The performance of two other algorithms, i.e. inverse QRD with SP and inverse QRD with LNS (32b), is compared with the reference filter. The results are seen in Figure 6.

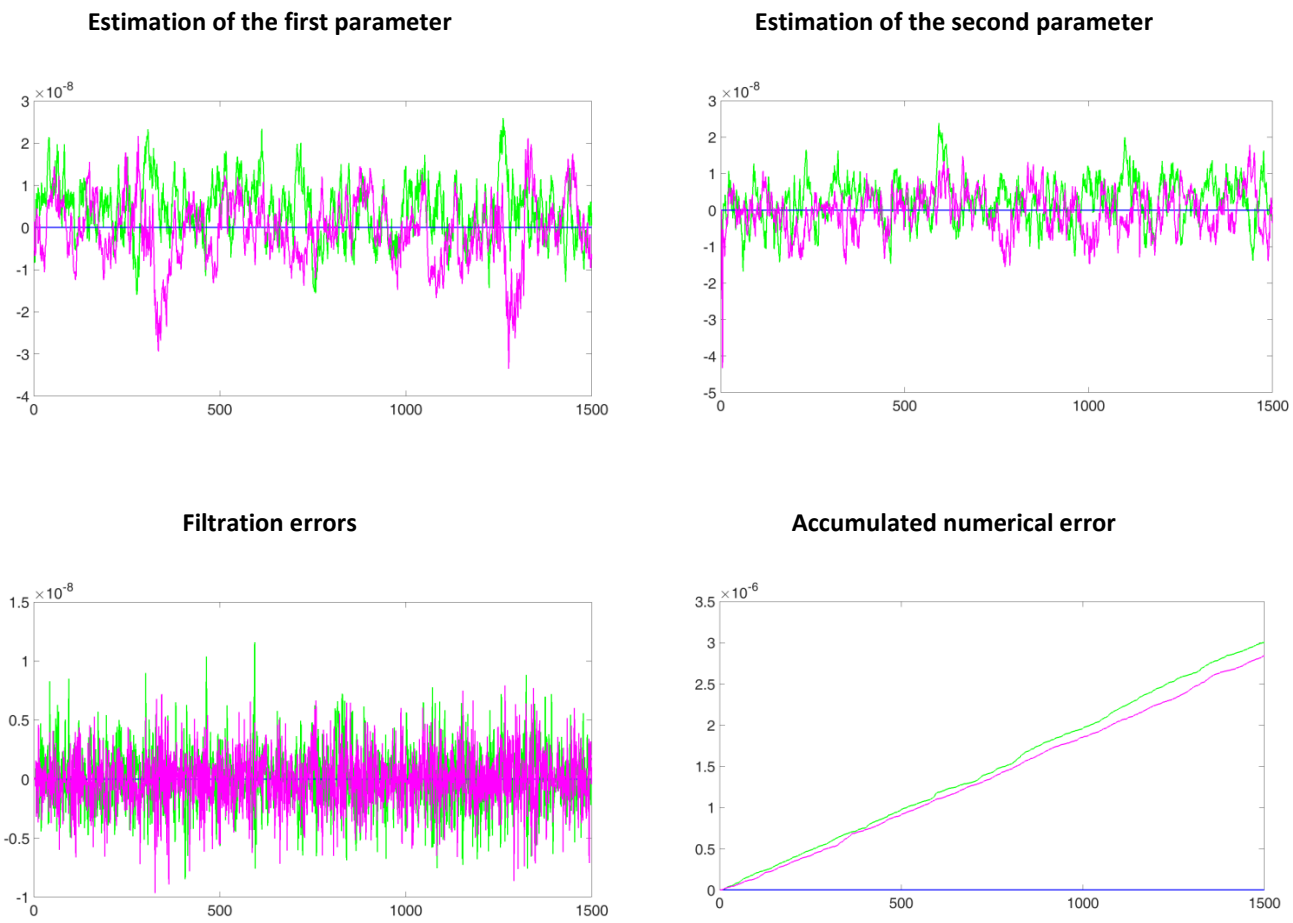


Figure 6: Comparison results for inverse QRD with SP and inverse QRD with LNS

The reference algorithm is presented with blue lines on the graphs. Inverse QRD with SP gives the outputs colored in magenta, while the results of inverse QRD with LNS are shown in green. The differences in parameter estimation and filtration errors from the reference algorithm are in the range of 10^{-8}. The variations in accumulated numerical error are also small and are not greater than 10^{-6}. These differences are explained by the method of computation the algorithm use to obtain the results. Moreover, it can be seen, that the accumulated numerical error for inverse QRD with LNS (32b) is a little bit greater than for inverse QRD with SP. But the difference is really not so high.

The next graph shows the differences in estimation of the first parameter in the range of 10^{-7}, the differences in estimation of the second parameter and in filtration errors in the range of 10^{-8} and the differences in accumulated numerical error in the range of 10^{-6}. The algorithms compared are inverse QRD filter with DP, QRD filter with SP, normalized QR filter with SP, QRD_rotations filter with SP and QR_rotations filter with SP. Matlab code used for these purposes is **short_filter_compare_51_70f_73f_76f_77f.m.**

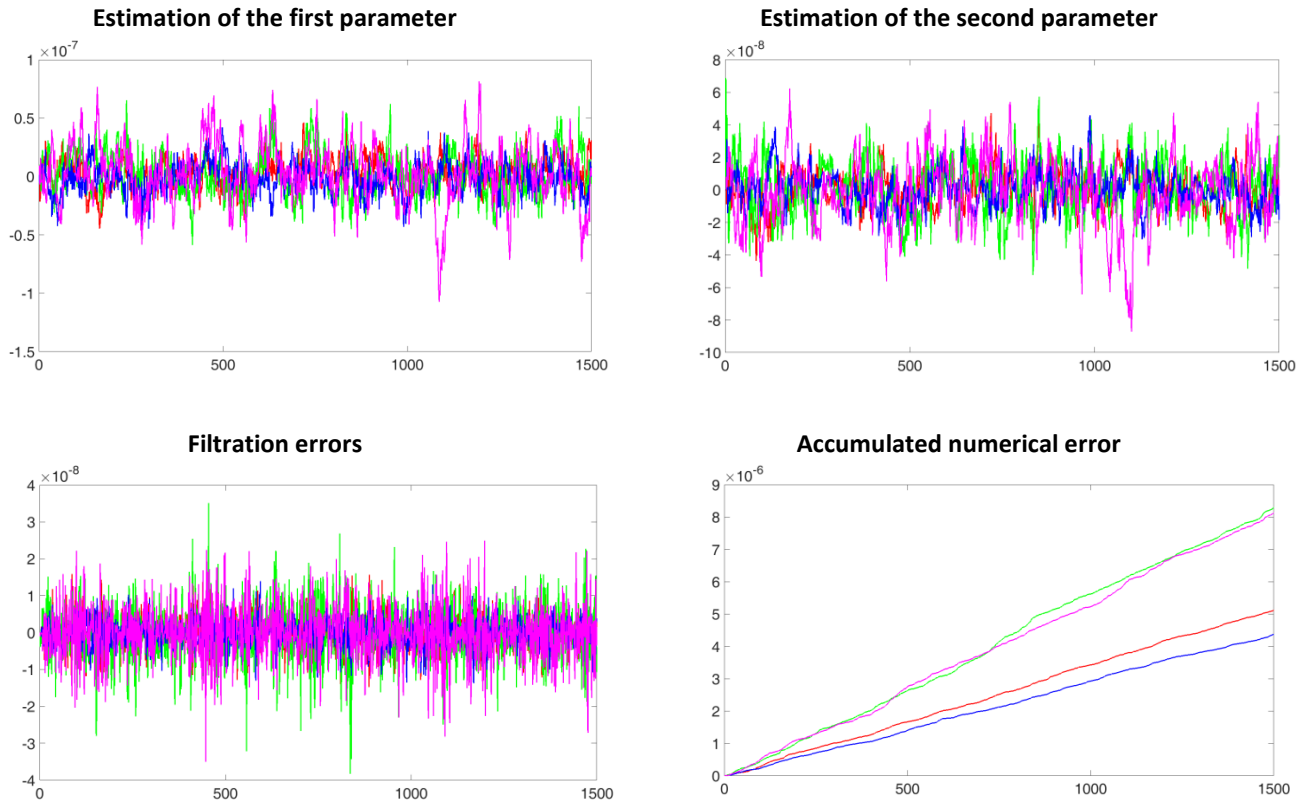


Figure 7: Comparison results for QRD filter with SP, normalized QR filter with SP, QRD_rotations filter with SP and QR_rotations filter with SP

The outputs of QRD filter with SP are in red, while the results of normalized QR filter with SP are in green. Blue curves present QRD_rotations filter with SP and magenta curves stand for QR_rotations filter with SP.

To compare the same algorithm as in the previous example, but using LNS (32b) Matlab file **short_filter_lns32_compare_51_70l_73l_76l_77l.m** is created. Here red curves stand for the outputs of QRD filter with LNS (32b), while green ones are applied for normalized QR filter with LNS (32b). The results of QRD_rotations filter with LNS (32b) are colored in blue and those of QR_rotaions filter are presented in magenta.

The ranges of the differences from the reference algorithms in parameter estimation, filtration errors and accumulated numerical error are almost the same as in previous example and as follows:

- up to 10^{-8} for parameter estimation and filtration errors,
- up to 10^{-5} for accumulated numerical error.

It should be noted also, that the red curves of QRD filter are hidden under the blue curves of QRD_rotations filter, because the results of these two algorithms in this case are very similar.

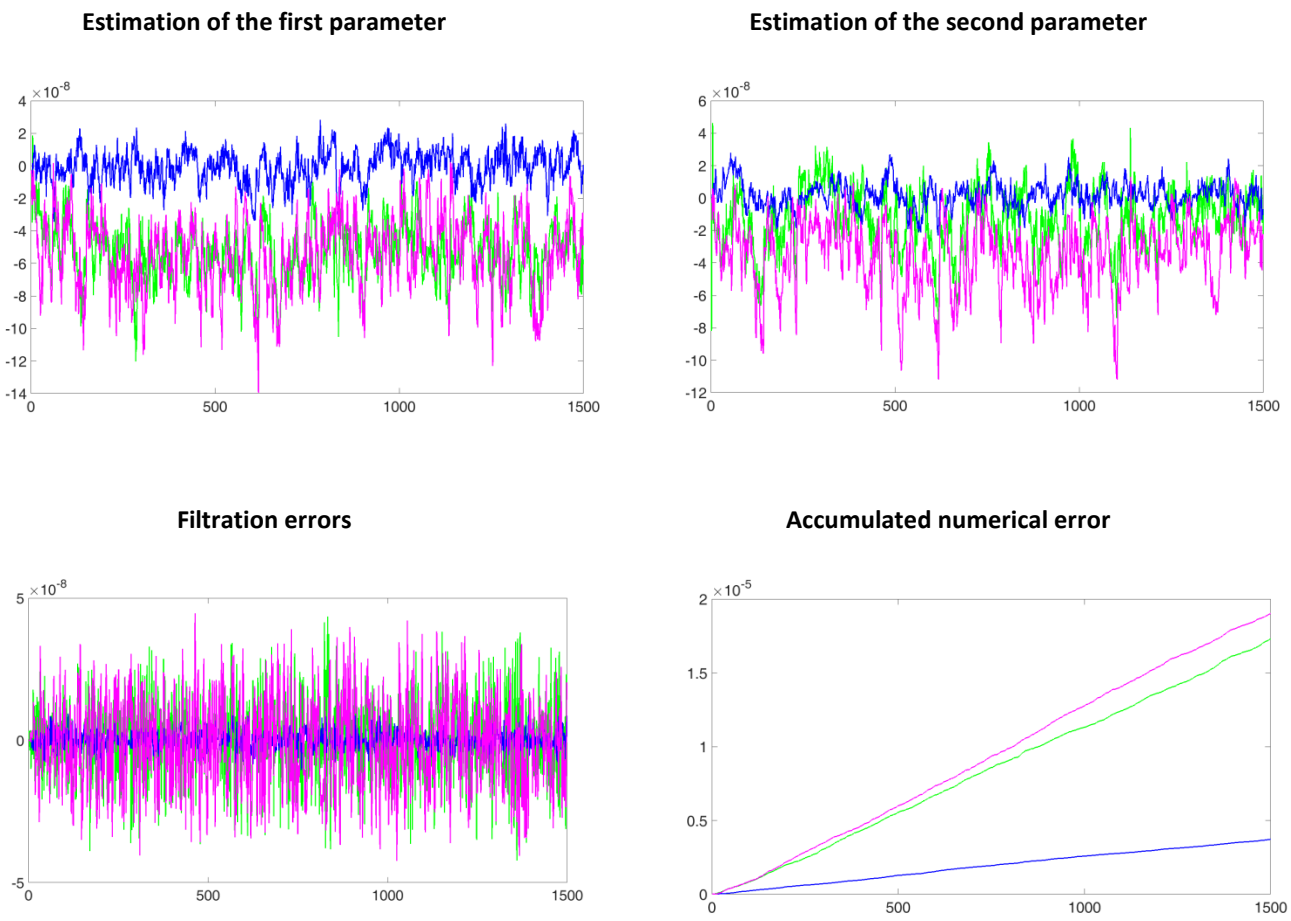


Figure 8: Comparison results for QRD filter, normalized QR filter, QRD_rotations filter and QR_rotations filter, all with LNS (32b)

However, if 19b logarithmic numerical system is used, the results are much less precise, as it can be seen in the example of **short_filter_lns19_compare_51_70l_73l_76l_77l.m**.

The algorithms compared with the reference algorithm (inverse QRD with DP) are the following:

QRD filter with LNS (19b), colored in red,
normalized QR filter with LNS (19b), colored in green,
QRD_rotations filter with LNS (19b), colored in blue,
QR_rotations filter with LNS (19b), colored in magenta.

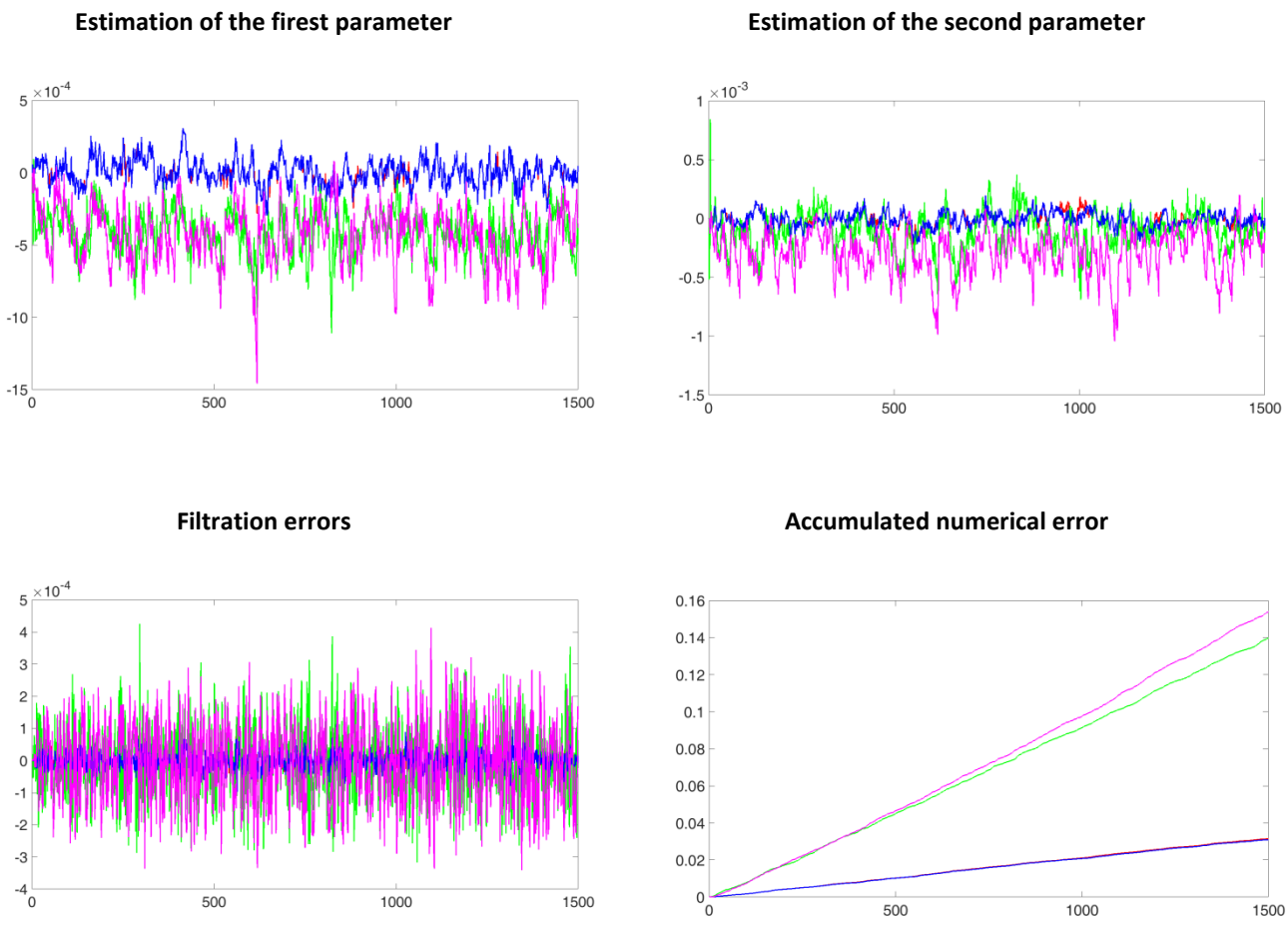


Figure 9: Comparison results for QRD filter, normalized QR filter, QRD_rotations filter and QR_rotations filter, all with LNS (19b)

The differences of above mentioned algorithms with the reference one are:

up to 10^{-4} for estimation of the first parameter and filtration errors,
up to 10^{-3} for estimation of the second parameter,
up to 10^{-1} for accumulated numerical error.

Nevertheless, the worst results are obtained when comparing normalized QR algorithm with 14b integer computation with the reference filter using DP. The differences in parameter estimation are in the range of 10^{-4}, the differences in filtration errors are up to 10^{-5} and the differences in accumulated numerical error are not greater than 10^{-2}. Though the variations are higher than in other cases, however, the accuracy is sufficient. Besides, the algorithms using 14b integer computation are cheaper and result in lower power consumption. Matlab script **short_filter_lns32_compare_51_51_51l_51_83i.m** provides comparison of inverse QRD filter with LNS and normalized QR filter with 14b integer computation with the reference algorithm, i.e. inverse QRD filter with DP.

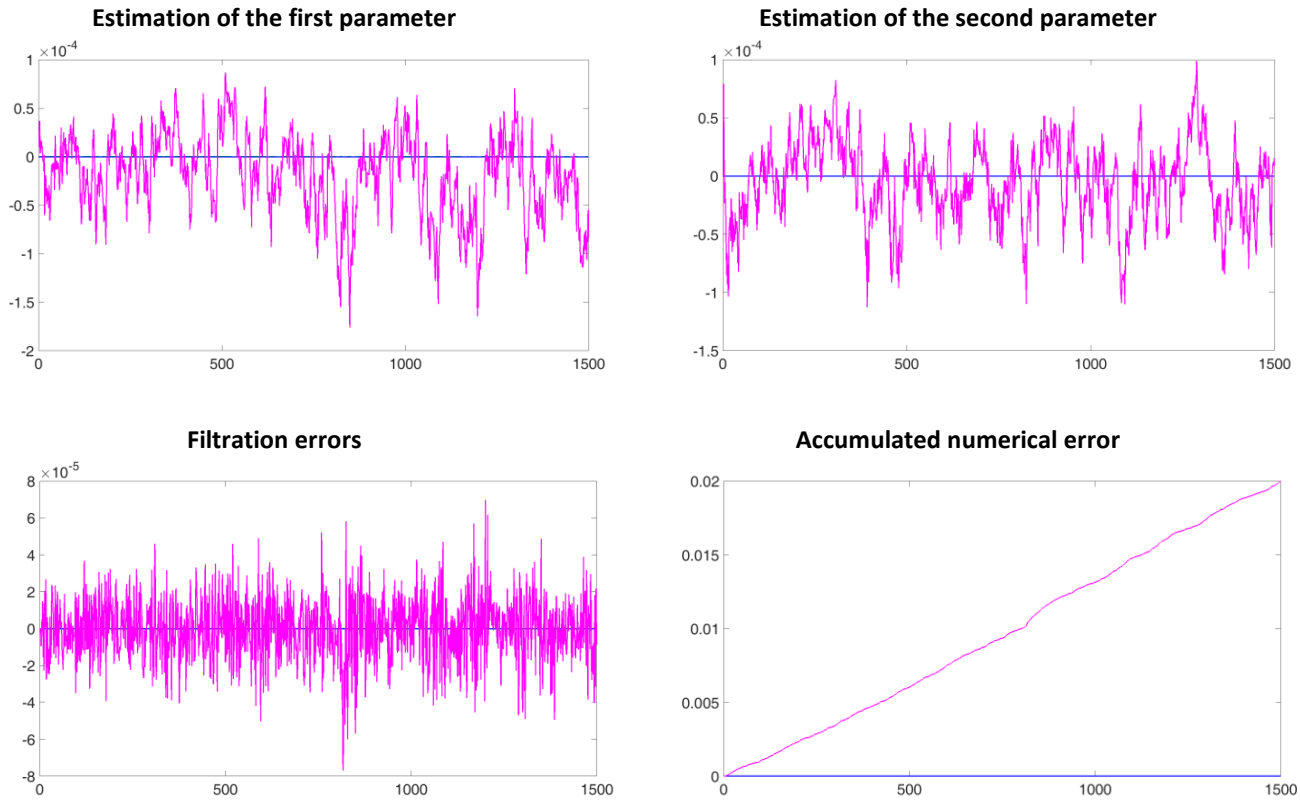


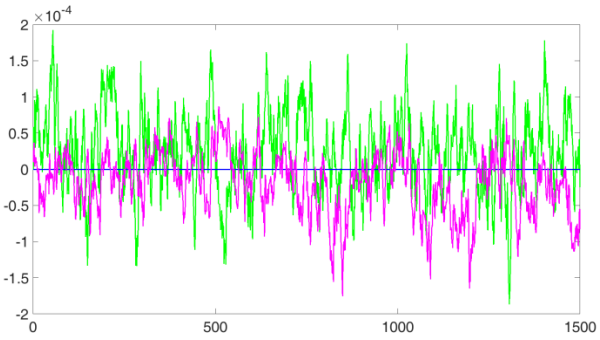
Figure 10: Comparison results for inverse QRD filter with LNS (32b) and normalized QR filter with 14b integer computation

The reference algorithm is presented by blue lines on the graphs. Magenta curves illustrate the outputs of normalized QR filter using 14b integer computation. Green curves should provide the outputs for inverse QRD filter with LNS; however, they are not seen on the graphs, because the difference between this algorithm and the reference one is much smaller, than the difference between normalized QR filter with 14b integer computation and the reference filter.

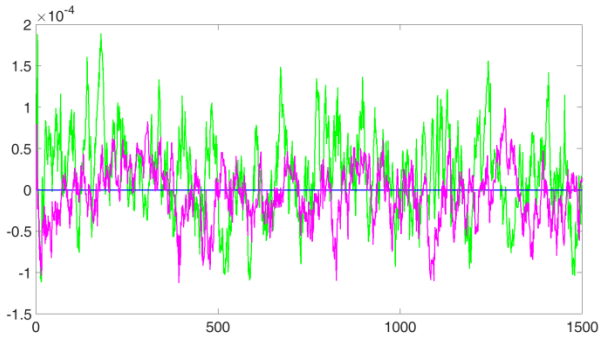
The same example, but with 19b logarithmic numbering system, is presented by Matlab file **short_filter_lns19_compare_51_51_51l_51_83i.m.** Here again two algorithms are compared with the reference inverse QRD filter with DP. These two algorithms are:

- inverse QRD filter with LNS (19b), in green,
- QR filter using 14b integer computation, in magenta.

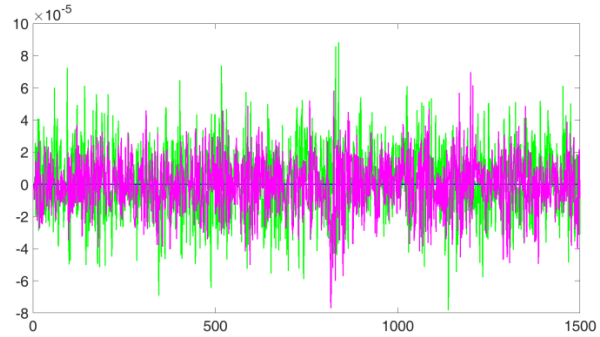
Estimation of the first parameter



Estimation of the second parameter



Filtration errors



Accumulated numerical error

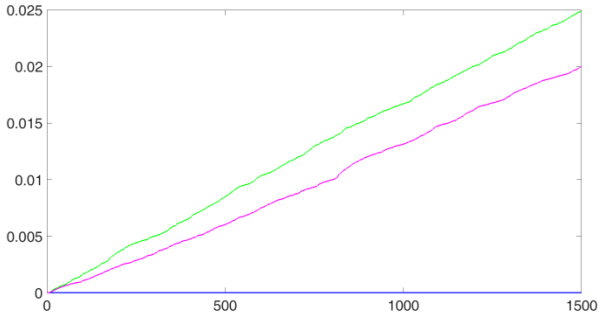


Figure 11: Comparison results for inverse QRD filter with LNS (19b) and normalized QR filter with 14b integer computation

The differences from the reference algorithm in the obtained results are the following:

- up to 10^{-4} for parameter estimation and filtration errors,
- up to 10^{-2} for accumulated numerical error.

Because precision of 19b logarithmic numbering system is smaller than in the previous example and more or less similar with the one for QR filter with 14b integer computation, the green curves are not hidden under magenta curves as it was in the previous case (compare Figure 10 and Figure 11).

The last example shows comparison results of QRD filter with DP and lattice filter. Matlab file performing the comparison is **short_filter_compare_70_70_46_70_70.m**.

The reference algorithm, that is QRD filter with DP, is presented by magenta lines, while green curves show the outputs of lattice filter. The differences can be explained mainly by the initial conditions two algorithms have in the beginning of estimation and by different methods the estimation itself is performed. Thus, on the right graph in the bottom it is clearly seen, that accumulated numerical error is rapidly growing only in the beginning of estimation process. It is fast converging to the right results (see the left graph in the bottom) and becomes constant.

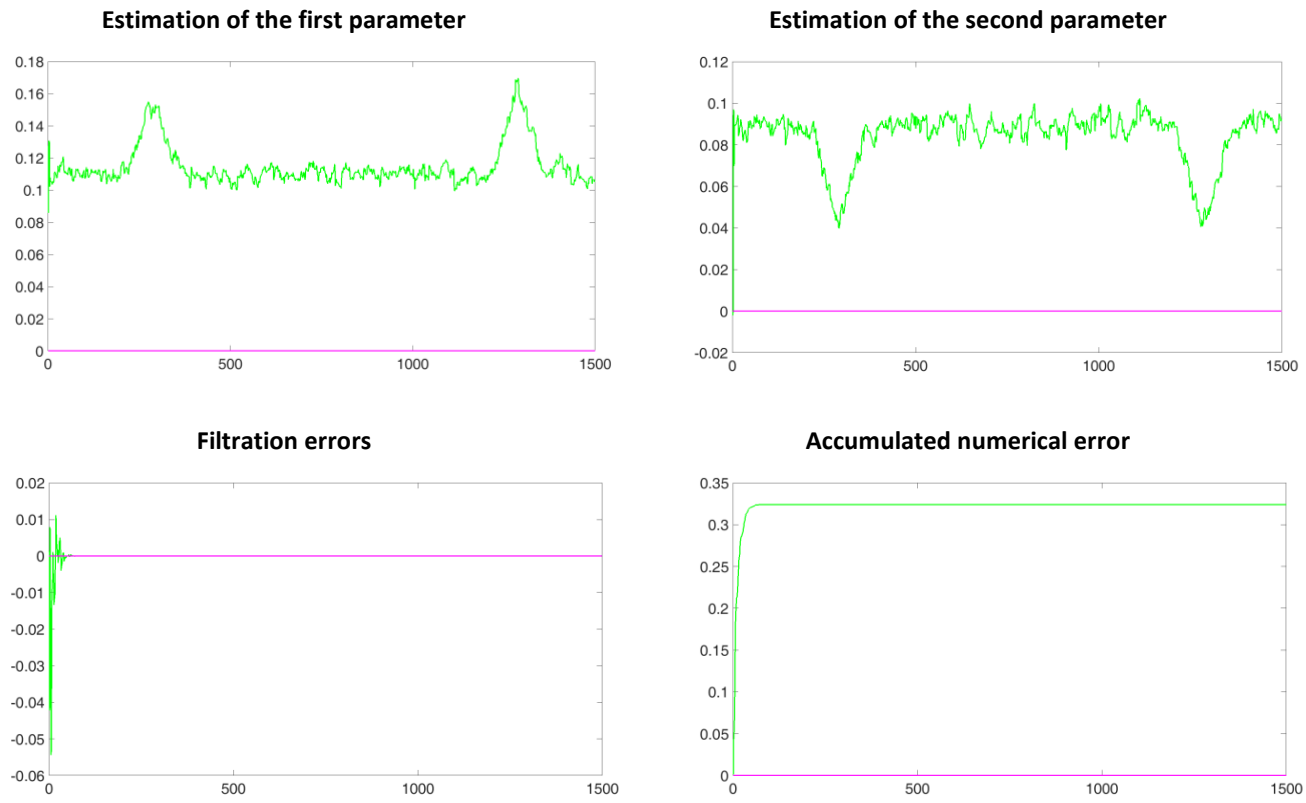


Figure 12: Comparison results for QRD filter with DP and lattice filter

For more details, please, refer to corresponding Matlab scripts.

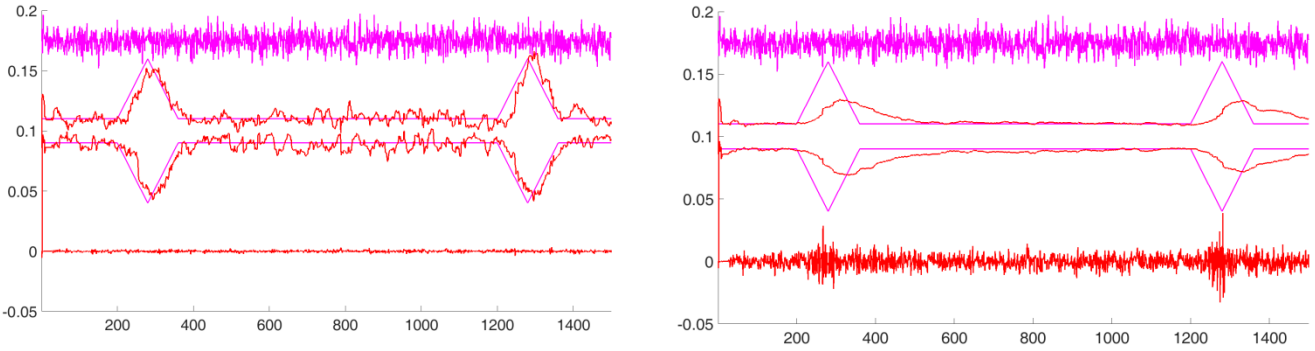
3.3 Using higher order regression model

The previous subchapters describe the examples of different algorithms, based on the third order regression model; thus, only three parameters have been estimated. In real situations the order of the model is much higher and the algorithms have to be able to estimate a score of the parameters correctly. To prove that proposed algorithms also function sufficiently accurately and are robust when using a regression model of higher order; several examples with the 23rd order regression model are presented in this subchapter. The parameters added to create the 23rd order regression model are set to be zeros, but all of them are evaluated during estimation process, though not depicted on the graph.

All algorithms listed in Table 1 and Table 2 have been tested using the 23rd order regression model. The filtering results of several algorithms are illustrated here.

The first example gives overlook, how estimation process is performed when using inverse QRD filter with double precision arithmetic both with EF and DF. Figure 13 shows the outputs for the case of persistent excitation as well as ill-defined excitation (refer to **long_filter_d1_1_51.m**, **long_filter_d1_2_51.m**, **long_filter_d1_3_51.m** and **long_filter_d1_4_51.m**).

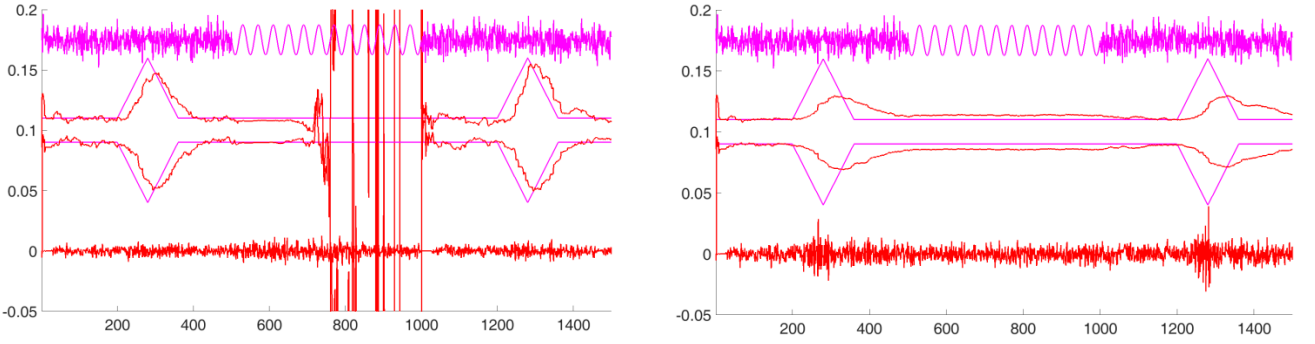
Persistent excitation



Inverse QRD filter, DP, EF

Inverse QRD filter, DP, DF

Ill-defined excitation



Inverse QRD filter, DP, EF

Inverse QRD filter, DP, DF

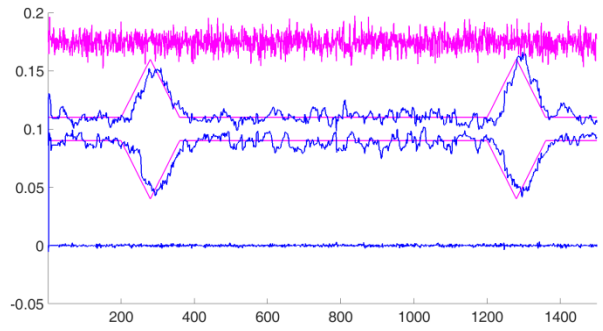
Figure 13: Inverse QRD filter with DP and EF/DF in case of persistent excitation resp. ill-defined excitation

From the graphs it is obvious, that the algorithms function well when a regression model with higher order is concerned. Estimation of only two parameters is depicted on the graph to make it more comprehensible. The upper left graph shows the outputs of inverse QRD filter with DP and EF. The estimation process is very similar to that for the third order regression model. However, it should be noted, that filtration errors are a little bit put down by the higher order of the model and its zero parameters.

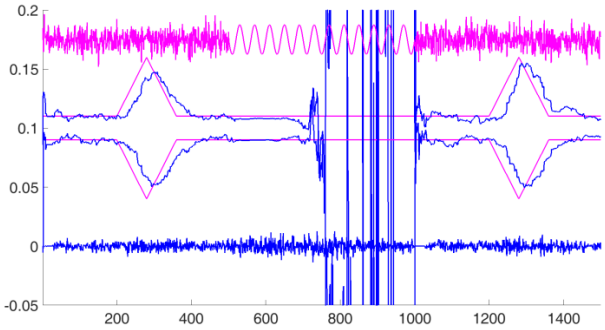
Parameter estimation in case of DF and persistent excitation (see the upper right graph) is converging slower to the right values of the parameters, but the shape of the estimation curve is smooth. The filtration errors are higher there than in the case of EF.

Two graphs in the bottom part of Figure 13 illustrate the case of ill-defined excitation. Inverse QRD filter with DP and EF has a problem to estimate parameters in the area of ill-excited signal. Therefore, parameter estimation in this part is so scattered. However, if we look at the results of inverse QRD filter with DF, it is obvious that it performs extremely well in this case and its estimation is very similar to that when the signal was persistently excited (the upper right graph).

Very similar results are obtained while evaluating algorithms working with single precision arithmetic and logarithmic numbering system (32b). The following example depicts the filtering process for normalized QR algorithms with LNS (32b).



Normalized QR with 32b LNS, EF, persistent excitation

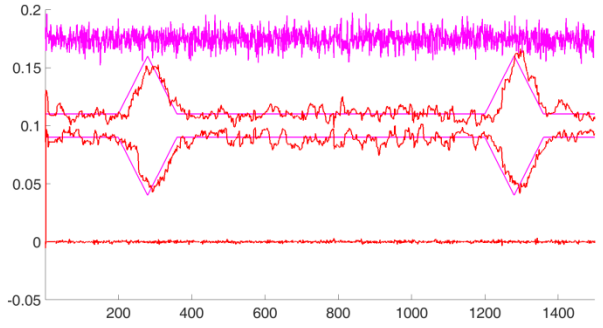


Normalized QR with 32b LNS, EF, ill-defined excitation

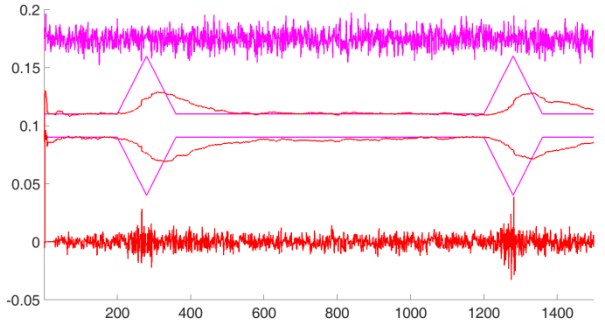
Figure 14: Normalized QR algorithm with 32b LNS

The algorithms using 19b logarithmic numbering system naturally give less precise results, particularly when ill-excited signal is concerned. Figure 15 and Figure 16 illustrate examples of filtering results for inverse QRD filter and normalized QR filter with 19b LNS. The estimation as in previous examples in this subchapter is performed using the 23rd order regression model.

Persistent excitation

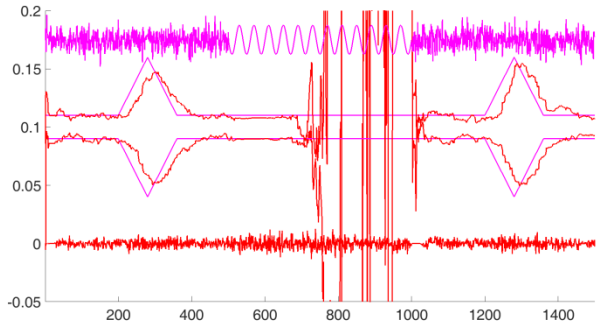


Inverse QRD filter with 19b LNS, EF

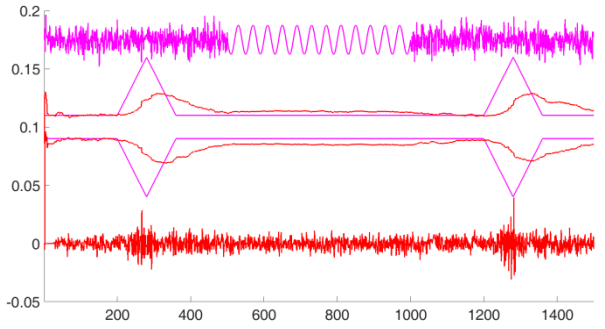


Inverse QRD filter with 19b LNS, DF

Ill-defined exitation

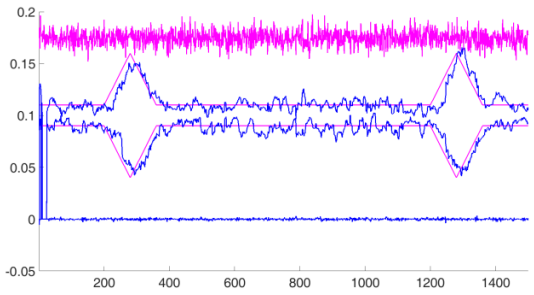


Inverse QRD filter with 19b LNS, EF

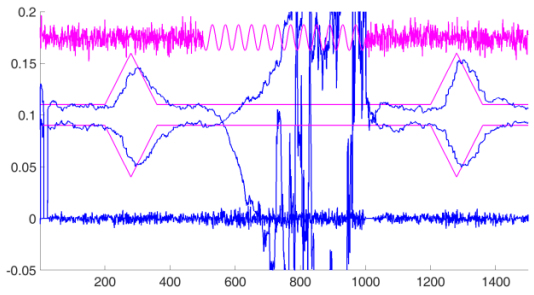


Inverse QRD filter with 19b LNS, DF

Figure 15: Inverse QRD filter using 19 b LNS



Normalized QR filter with 19b LNS, EF, persistent excitation



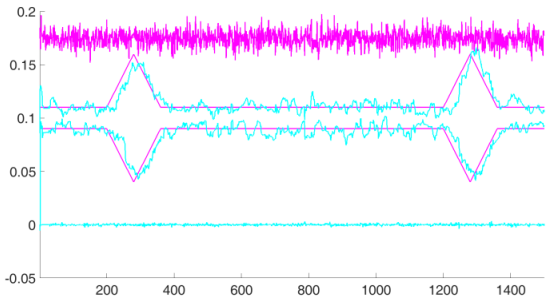
Normalized QR filter with 19b LNS, EF, ill-defined excitation

Figure 16: Normalized QR filter with 19b LNS

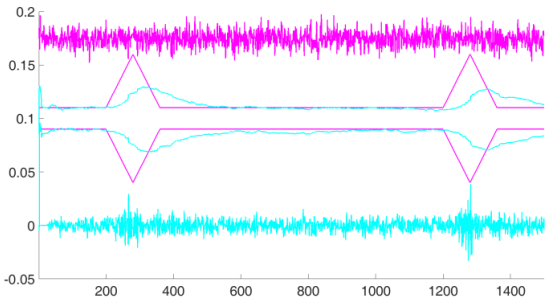
From Figure 16 (the right graph) it is obvious that algorithm has problems when the input signal is ill-excited. However, the positive definiteness of the matrix is preserved and, when the signal is again persistently excited, the algorithm begins functioning correctly.

The least precise results are obtained for normalized QR algorithms working with 14b integer computation (see Figure 17). From the bottom left picture it is seen, that in the case of ill-defined excitation the parameter estimation is scattered and gives good results only when the signal is persistently excited again. It is valid for the algorithm using EF. On the other hand, DF can be a benefit for estimation process, allowing to perform estimation more precisely (see the right bottom graph).

Persistent excitation

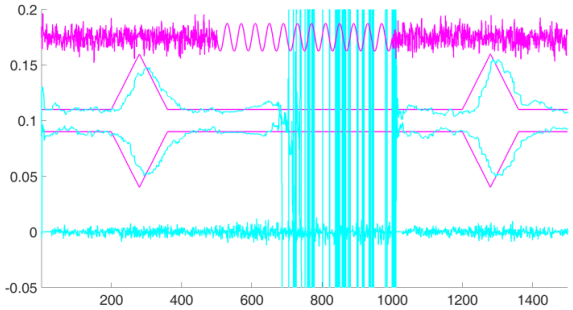


Normalized QR filter, 14b integer computation, EF

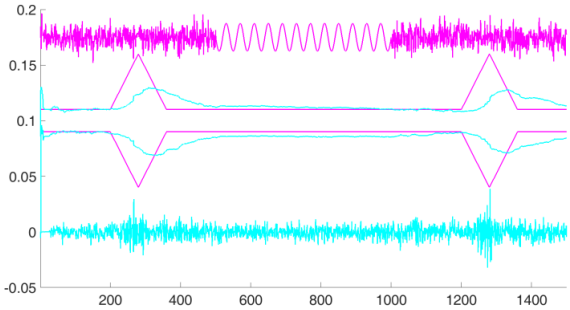


Normalized QR filter, 14b integer computation, DF

Ill-defined excitation



Normalized QR filter, 14b integer computation, EF

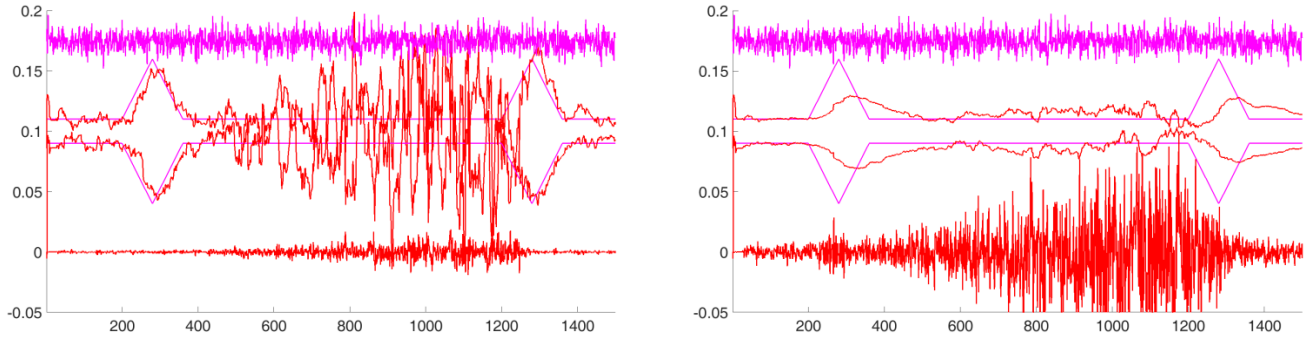


Normalized QR filter, 14b integer computation, DF

Figure 17: Normalized QR filter in fixed point representation

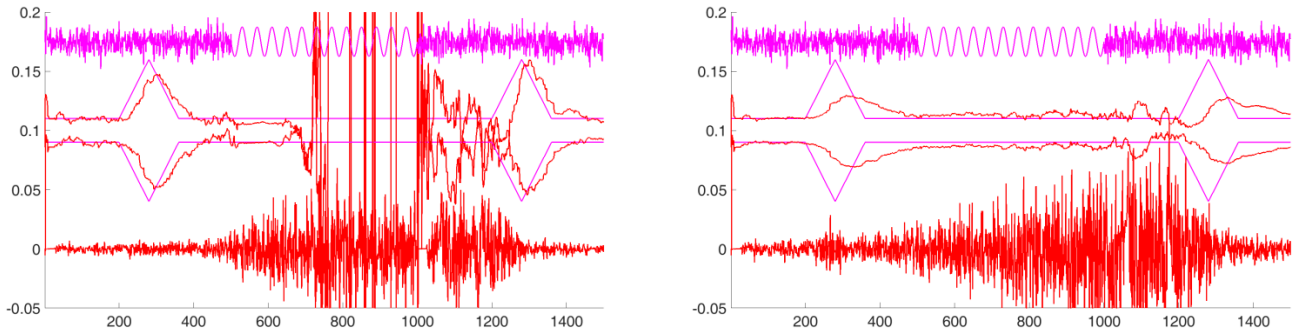
All examples described in this subchapter as well as in the previous subchapters have supposed stationary noise. In real life the noise should not be always time-invariant and often is not. Thus, the algorithms have to be able to function sufficiently precisely in case of non-stationary noise as well. The following example presents the outputs of inverse QRD filter with DP and EF/DF (both for persistently excited signal and ill-defined excitation), when some time-varying, i.e. non-stationary, noise is added to the signal (see Figure 18).

Persistent excitation



Inverse QRD filter, DP, EF **Inverse QRD filter, DP, DF**

Ill-defined excitation



Inverse QRD filter, DP, EF **Inverse QRD filter, DP, DF**

Figure 18: Inverse QRD filter performance in case of non-stationary noise

From the graphs above it is obvious, that the algorithms also manage to perform estimation in case of non-stationary noise. It should be noted again that the algorithms with DF give a little bit better results in comparison with the same algorithms with EF.

To conclude, the described examples prove that proposed algorithms are also numerically robust and sufficiently precise when a regression model of higher order is used. Besides, they prove to function when non-stationary noise enters the system. Thus, they are reliable to be implemented in real-life applications.

As far as comparison of performance of different algorithms for the 23rd order regression model is concerned, it is omitted here, because the obtained results while comparing different algorithms are very similar to those presented and described in Subchapter 1.7.

4. Applications for Evaluation

4.1 Compiled scripts

The algorithms described in previous subchapters are created in Matlab 2015b (32b) and are available in two forms:

- as **.m** files with .**mexw32** (Matlab 2015b 32b has to be installed to use the files),
- as application with installation package for Win7 (32b or 64b) PC or Win10 (32b or 64b) PC. Matlab is not necessary to be installed on the computer in this case.

This subchapter describes the way **.exe applications** can be used on Win7 (32b or 64b) PC or Win10 (32b or 64b) PC.

There are two **.exe applications** available:

- **test_algorithms.exe** showing the performance of the algorithms as it has been described in Chapters 1.4, 1.5 and 1.8,
- **test_algorithm_comparison.exe** consisting of examples described in Chapter 1.7.

In a folder downloaded from web there are **test_algorithms** and **test_algorithm_comparison** subfolders, which obtain **.exe applications**.

To be able to run **.exe applications** compiled in Matlab you need first to install **MATLAB Runtime**.

If you do not have MATLAB Runtime, in directory ***test_algorithms -> for_redistribution*** click on ***MyAppInstaller_mcr***. A window will appear:

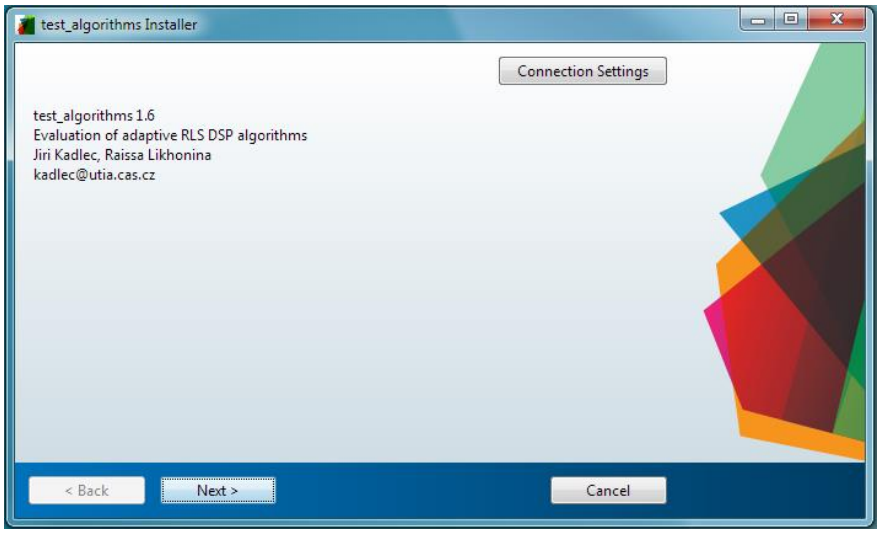


Figure 19: Installation of .exe application

Click ***Next*** and choose installation folder, where you want to install the application.

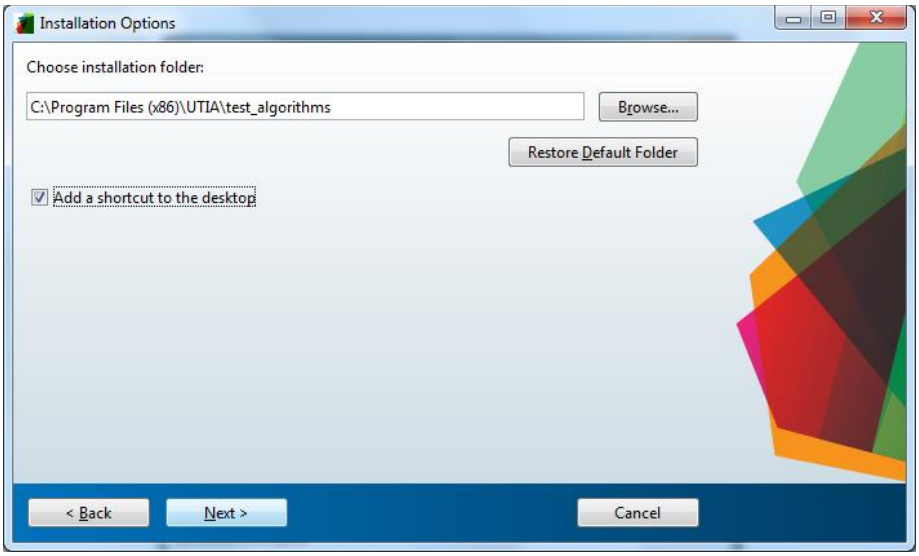


Figure 20: Installation options

Click ***Next*** again and choose installation folder for MATLAB Runtime.

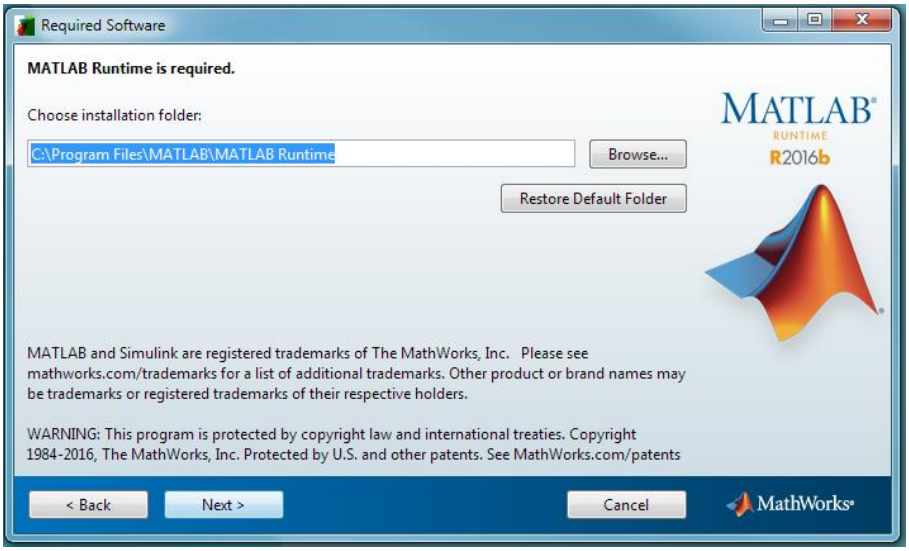


Figure 21: MATLAB Runtime installation

Click ***Next***.

The License Agreement will appear. Read it carefully; click ***Yes***, if you accept the terms of the license agreement, and press ***Next***.

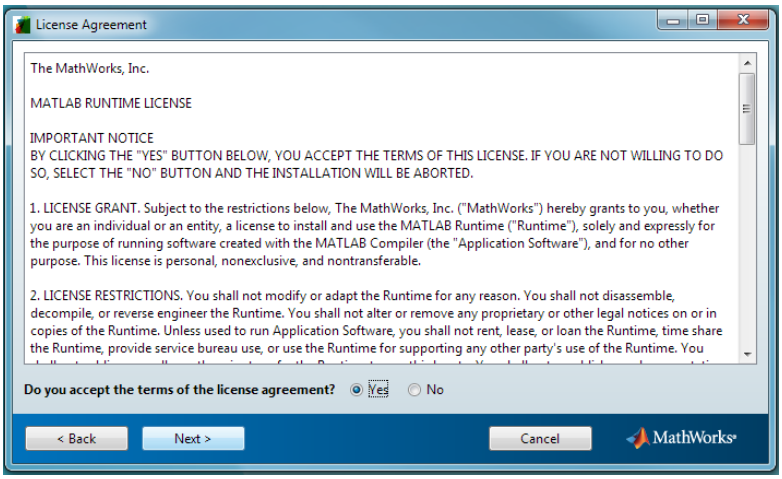


Figure 22: License Agreement

The confirmation window will appear.

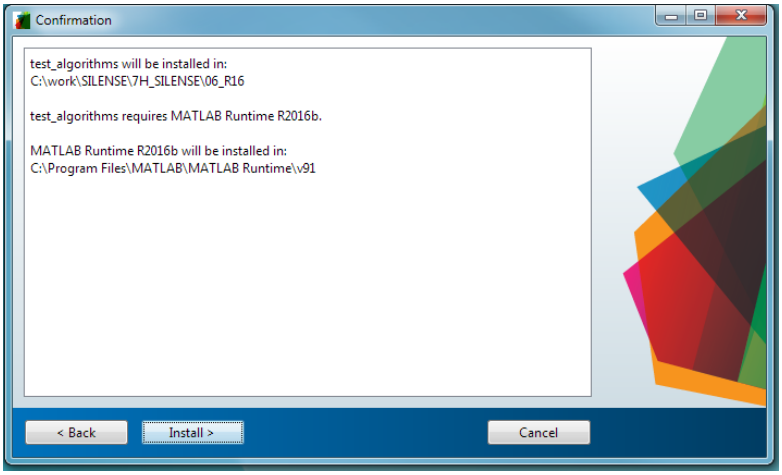


Figure 23: Confirmation window

Click ***Install***.

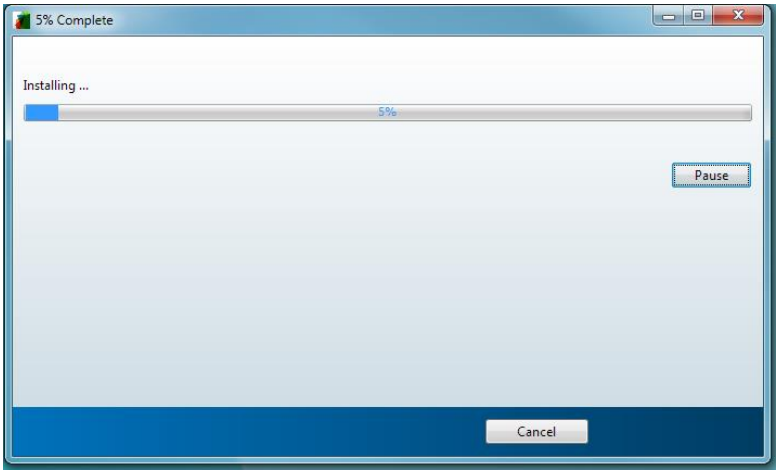


Figure 24: Installation progress

After installation being completed press ***Finish***.

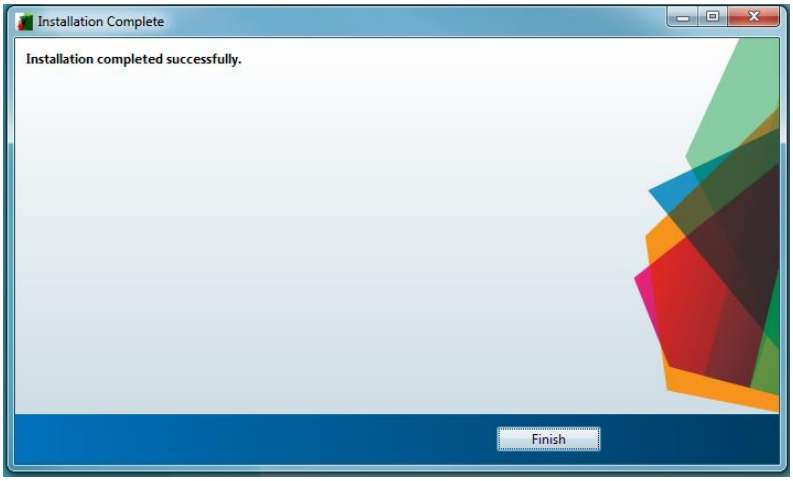


Figure 25: Installation completed

Now you can use **test_algorithm.exe** to see the performance of algorithms.

After clicking on **test_algorithms.exe** a window appears:

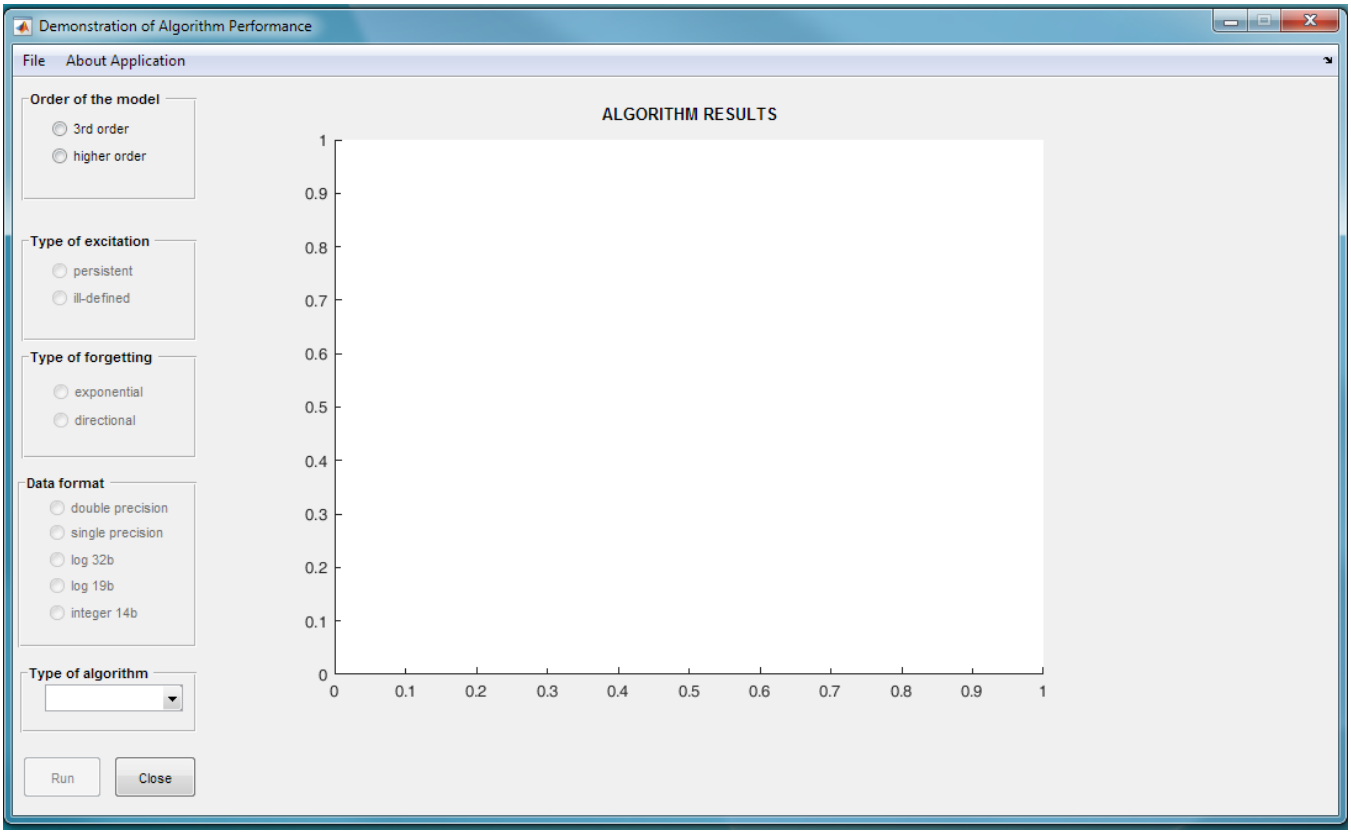


Figure 26: Application test_algorithms.exe

On the left side there are a number of options you can choose. The steps are the following:

1. Select between the third and higher order of the model used in the algorithms.
2. Select whether you want a persistently excited system or a system with a period of ill-defined excitation.

3. Select a type of forgetting: exponential or directional forgetting.
4. Select data format. After selecting data format a list of algorithms will appear in a popup menu.
5. Choose the algorithm you want to execute.
6. Press ***Run*** button.

It will result in performing the corresponding algorithm and presenting its outputs in a figure in the center of the window. The example can be seen from Figure 27.

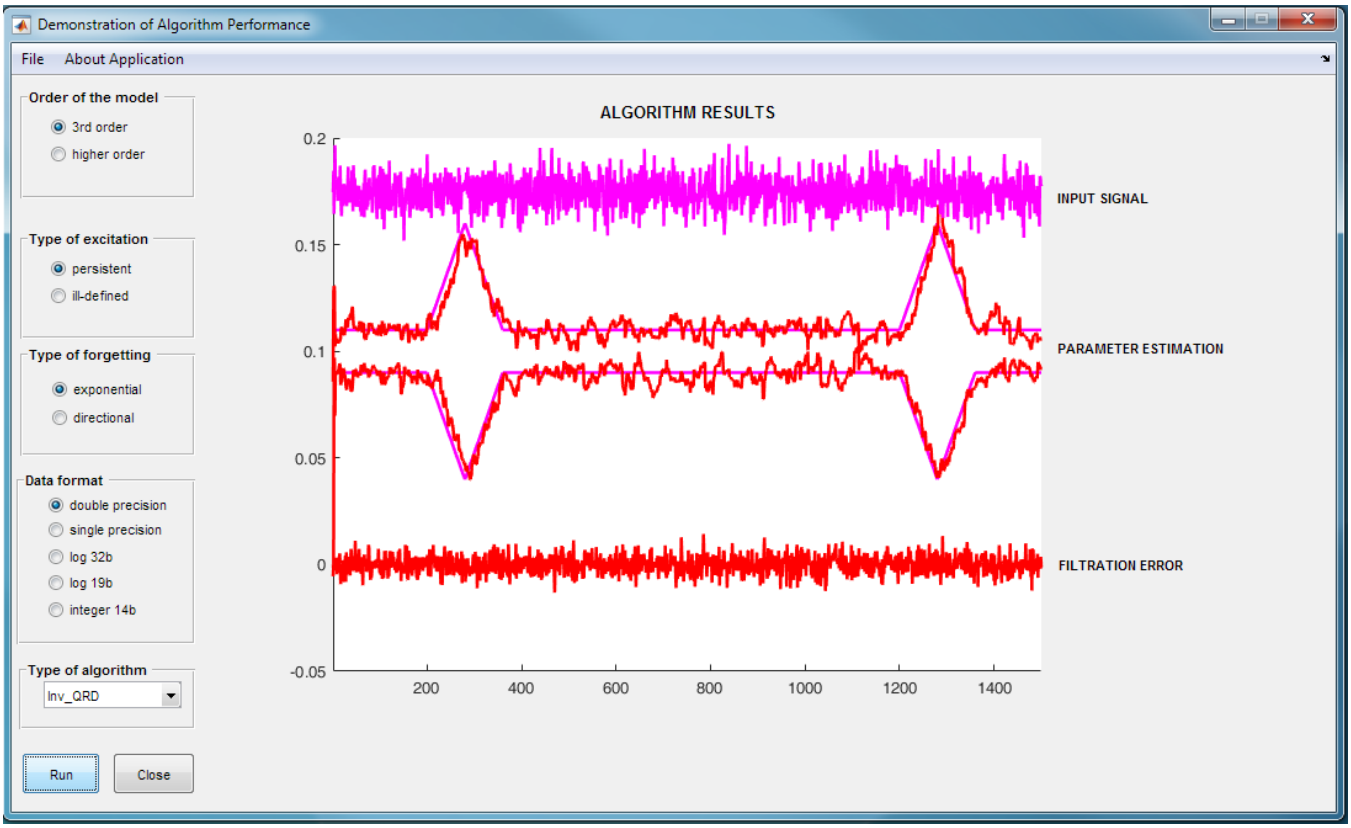


Figure 27: Example of application performance

You can then choose another algorithm for seeing the results or close the application by pressing ***Close*** button or ***File->Close*** on a menu bar.

There is ***About Application*** on a menu bar. Clicking on it you can read short information about **test_algorithm.exe**.

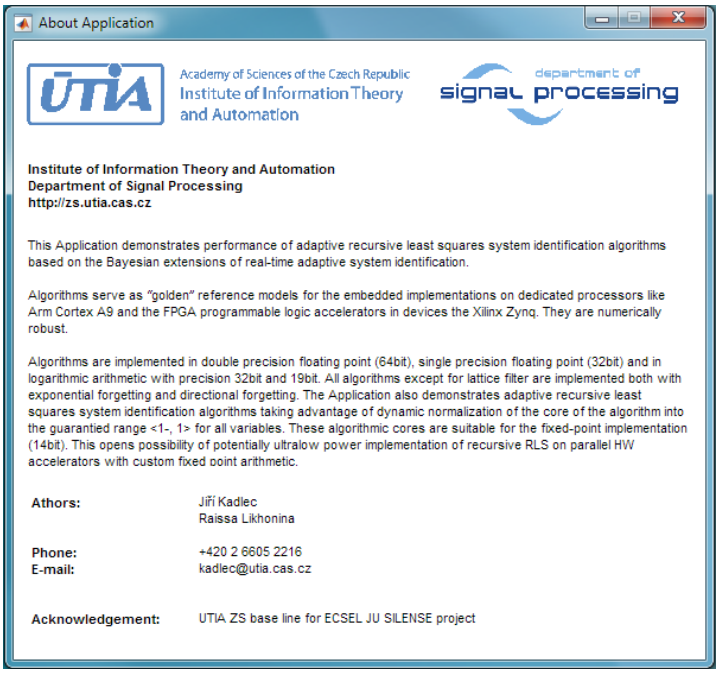


Figure 28: Short information about application test_algorithms.exe

To see the results of comparison of different algorithms you need to run **test_algorithm_comparison.exe**.

A window will appear as shown in Figure 29.

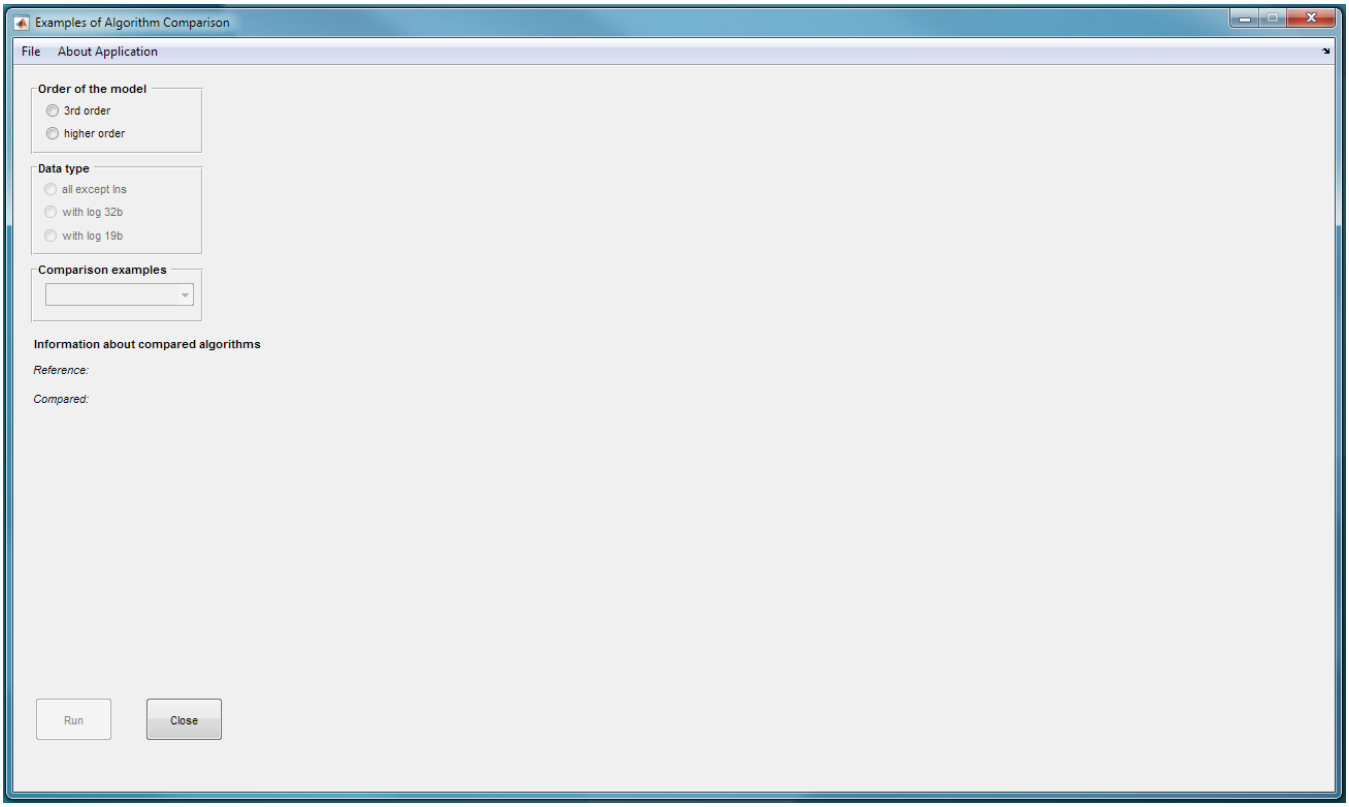


Figure 29: Application test_algorithm_comparison.exe

There you should:

1. select the order of the model,
2. select whether for comparison with other algorithms you would use algorithms with logarithmic numbering system (lns32 or lns19) or you would like to compare algorithms using all other data formats except logarithmic numbering system.

After selecting these options the comparison examples popup menu will be available, where you can choose the examples of algorithm comparison explained in Chapter 1.7.

After selecting some comparison example from a popup menu the information about compared algorithms will appear below. It obtains information about the reference algorithm and about the algorithms compared with the reference one.

Pressing ***Run*** button the outputs of comparison will appear in the window.

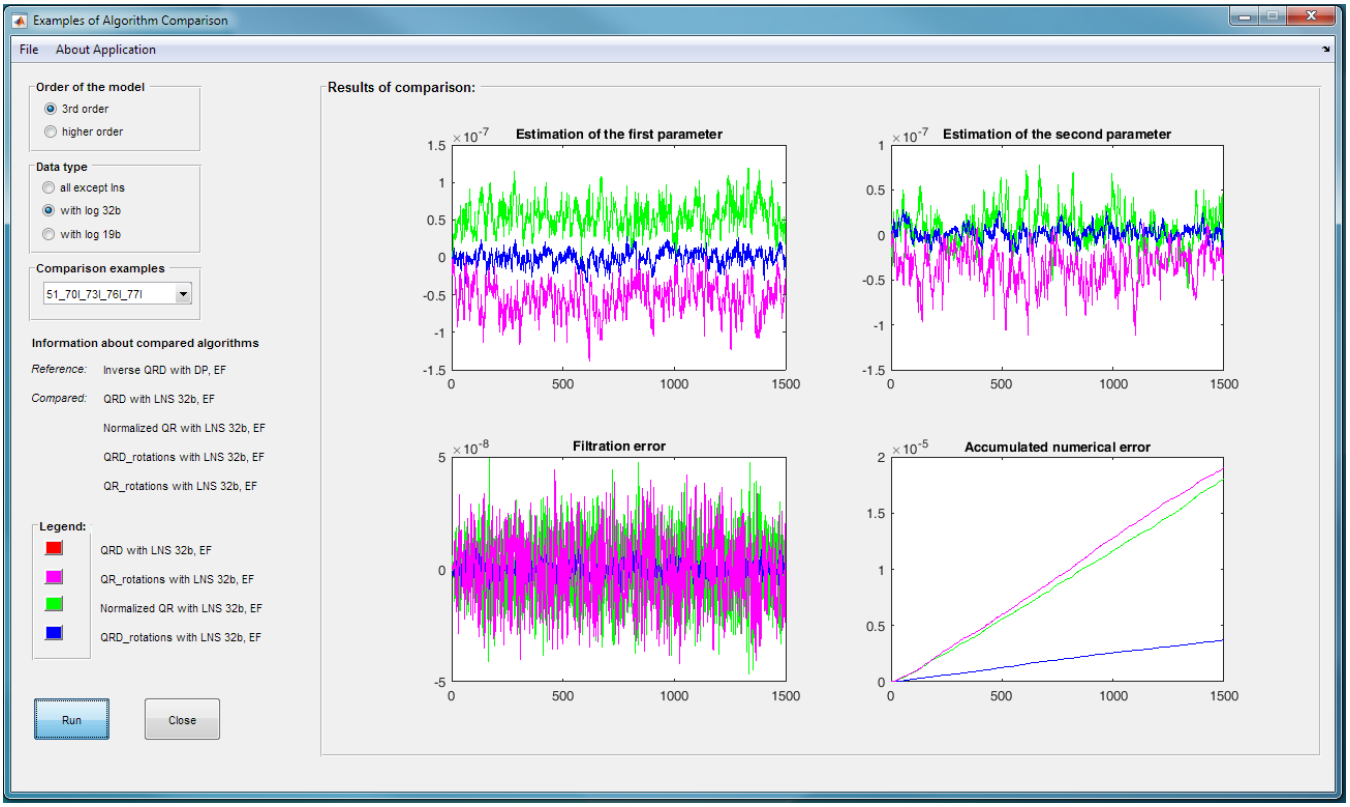


Figure 30: Performance of application for comparing algorithms

You can choose another example to see the results or close the program by pressing ***Close*** button or ***File->Close*** on a menu bar.

Press ***About Application*** on a menu bar to see short information about **test_algorithm_comparison.exe**.

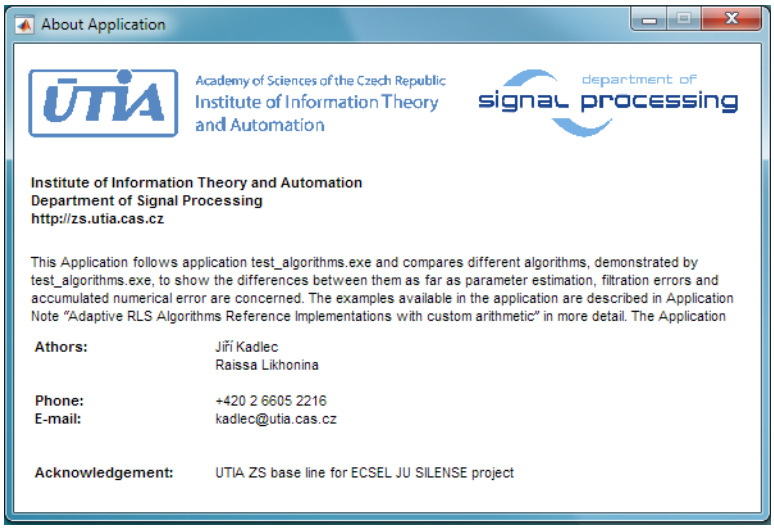


Figure 31: Short information about application test_algorithm_comparison

4.2 Availability and Licensing

The evaluation package includes .m scripts with DSP algorithms pre-compiled as .mexw32 files for MATLAB R2015.b (32b) and two standalone applications for Win7 (32b or 64b) PC or Win10 (32b or 64b) PC (for users without MATLAB).

- These included DSP algorithms pre-compiled as .mexw32 files have no time restriction.
- The evaluation package can be downloaded and used free of charge.
- Source code of these DSP algorithms is not provided in this evaluation package.

4.3 References

- Systolic arrays for implementation of QR, QRD and inverse QR adaptive RLS filters with exponential forgetting for FIR regression models are described in [1].
- Directional forgetting concept and implementation for the inverse QR adaptive RLS filter is described in [2].
- Bayesian structure estimation is described in [3].
- Dynamic normalization of QR, QRD adaptive RLS filters with exponential and directional forgetting and concept of fixed-point implementation of filters is described in [4] with concrete applications in [5] and [6].
- UTIA EdkDSP reprogrammable floating point HW accelerators are described in [7] and [8].
- Implementation of logarithmic arithmetic is described in [9] – [17].
- UTIA simulation environment based on MATLAB supporting simulation, identification and control of stochastic systems modelled by adaptive linear regression filters is described in [18] – [24]. Package described in this application note is utilizing from the UTIA packages [18] – [24] these building blocks :
    - Simulation of time variable systems modelled as time variable MIMO linear regression models.
    - Double precision implementation of the Inverse QRD RLS identification algorithm with exponential and directional forgetting (md51).
    - Double precision implementation of the QRD RLS identification algorithm with exponential forgetting (md70).
- Applications of UTIA DSP algorithms are described in [25] – [42].

[1] **Marc Moonen: Introduction to Adaptive Signal Processing.** K.U. Leuven, Leuven, Belgium. October 20, 1998 [online]. Available at http://homes.esat.kuleuven.be/~moonen/asp_course.html

[2] **Kulhavý Rudolf, Zarrop M. B. :** On a General Concept of Forgetting **,** *International Journal of Control vol.58, 4 (1993), p. 905-924* [1993]

[3] **Kadlec Jiří** : Systolic Arrays for Identification of Systems with Variable Structure, ÚTIA AV ČR, (Praha 1994) Research Report 1817 [1994]

[4] **Kadlec Jiří** : Numerical Analysis of a Normalized RLS Filter Using a Probability Description of Propagated Data, ÚTIA AV ČR, (Praha 1994) Research Report 1818 [1994]

[5] **Kadlec Jiří, Gaston F. M. F., Irwin G. W.** : A parallel fixed-point predictive controller , *International Journal of Adaptive Control and Signal Processing vol.11, 5 (1997), p. 415-430* [1997] Download

[6] **Kadlec Jiří, Gaston F. M. F., Irwin G. W.** : The block regularised parameter estimator and its parallelisation , *Automatica vol.31, 8 (1995), p. 1125-1136* [1995] Download

[7] **Daněk Martin, Kadlec Jiří, Bartosinski Roman, Kohout Lukáš** : Increasing the Level of Abstraction in FPGA-based Designes , *International Conference on Field Programmable Logic and Applications, p. 5-10 , Eds: Kebschull Udo*, International Conference on Field Programmable Logic and Applications, (Heidelberg, DE, 08.09.2008-10.09.2008) [2008] Download

[8] **Kadlec Jiří, Daněk Martin, Kohout Lukáš :** Proposed architecture of configurable, adaptable SoC , *The IET Irish Signals and Systems Conference ISSC 2008, p. 368-373 , Eds: Morgan Fearghal, Glavin Martin, Jones Edward*, The Institution of Engineering and Technology Irish Signals and Systems Conference, ISSC 2008, (Galway, IE, 18.06.2008-19.06.2008) [2008]

[9] **Coleman J. N., Chester E. I., Softley C. I., Kadlec Jiří** : Arithmetic on the European Logarithmic Microprocessor , *IEEE Transactions on Computers vol.49, 7 (2000), p. 702-715* [2000]

[10] **Tichý Milan, Schier Jan, Gregg D.** : FPGA Implementation of Adaptive Filters based on GSFAP using Log Arithmetic , *Proceedings of The 2006 IEEE Workshop on Signal Processing Systems Design and Implementation, p. 342-347 , Eds: Badawy W., Boumaiza S.*, IEEE Workshop on Signal Processing Systems Design and Implementation. 2006, (Banff, CA, 02.10.2006-04.10.2006) [2006]

[11] **Tichý Milan, Schier Jan, Gregg D.** : Efficient Floating-Point Implementation of High-Order (N)LMS Adaptive Filters in FPGA , *Reconfigurable Computing: Architecures and Applications. Proceedings of the Second International Workshop ARC, p. 311-316 , Eds: Bertels K., Cardoso J. M. P., Vassiliadis S.*, The Second International Workshop on Reconfigurable Computing ARC 2006, (Delft, NL, 01.03.2006-03.03.2006) [2006]

[12] **Tichý Milan, Nisbet A., Gregg D.** : GSFAP adaptive filtering using log arithmetic for rescouse-constrained embedded systems , *FPGA 2006. Fourteenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, p. 236-236 , Eds: Wilton S., DeHon A.*, FPGA 2006. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays /14./, (Monterey, US, 22.02.2006-24.02.2006) [2006]

[13] **Tichý Milan** : Multi-Floating-Point-Arithmetic Fast Adaptive Filtering Algorithm Library for Matlab. (program), ÚTIA AV ČR, (Praha 2006) [2006]

[14] **Tichý Milan** : Log Units, (N)LMS and GSFAP Hardware Macros and the Hardware Simulation Infrastructure. (program), ÚTIA AV ČR, (Praha 2006) [2006]

[15] **Tichý Milan** : Adaptive Filtering Algorithms Implementation and Evaluation of their Filtering Properties, ÚTIA AV ČR, (Praha 2006) Research Report 2164 [2006]

[16] **Tichý Milan** : Review and Classification of Adaptive Filtering Algorithms for the LNS Arithmetic, ÚTIA AV ČR, (Praha 2006) Research Report 2162 [2006]

[17] **Tichý Milan** : Efficient Floating-point-like Implementation of the (N)LMS and GSFAP Algorithms in FPGA, ÚTIA AV ČR, (Praha 2006) Research Report 2165 [2006]

[18] **Warwick K., Kárný Miroslav, Halousková Alena** : Advanced Methods in Adaptive Control for Industrial Applications, Springer, (Berlin 1991) [1991]

[19] **Hrudková Dagmar, Kulhavá Lenka** : ÚTIAlab - Knihovna funkcí, ÚTIA ČSAV, (Praha 1991) [1991]

[20] **Nedoma Petr, Hrudková Dagmar** : UTIAlab - Referenční příručka, ÚTIA ČSAV, (Praha 1991) [1991]

[21] **Nedoma Petr, Hrudková Dagmar** : ÚTIAlab - Uživatelská příručka, ÚTIA ČSAV, (Praha 1991) [1991]

[22] **Nedoma Petr, Kadlec Jiří, Schier Jan** : Tools for Implementation of Parallel Algorithms for Adaptive Control and Signal Processing , *4th IFAC International Symposium on Adaptive Systems in Control and Signal Processing. ACASP '92, p. 727-730 , Eds: Landau I. D., Dugard L., M'Saad M.*, Laboratoire d'Automatique, (Grenoble 1992) , IFAC International Symposium on Adaptive Systems in Control and Signal Processing. ACASP '92 /4./, (Grenoble, FR, 01.07.1992-03.07.1992) [1992]

[23] **Kadlec Jiří** : Transputer Implementation of Block Regularised Filtering , *Progress in Transputer Computing Technology, p. 1-15 , Eds: Kulhavá L., Schier J., Kárný M.*, ÚTIA AV ČR, (Prague 1993) , TCT'93 International Workshop, (Prague, CZ, 04.05.1993-05.05.1993) [1993]

[24] **Kadlec Jiří** : Direct Software Bridge MATLAB-Transputer Boards, ÚTIA AV ČR, (Praha 1994) Research Report 1819 [1994]

[25] **Kadlec Jiří, Gaston F. M. F., Irwin G. W. :** Parallel implementation of restricted parameter tracking **,** *Mathematics in Signal Processing, p. 315-325 , Eds: McWhirter J. G.*, Clarendon Press, (Oxford 1994) Institute of Mathematics and its Applications Conference Series. vol.49 [1994]

[26] **Kadlec Jiří** : A Recursive Modified Gram-Schmidt Identification with Directional Tracking of Parameters , *Preprints of the 9th IFAC/IFORS Symposium on Identification and System Parameter Estimation, p. 1707-1712*, AKA PRINT Nyomdaipari, (Budapest 1991) , IFAC/IFORS Symposium on Identification and System Parameter Estimation /9./, (Budapest, HU, 08.07.1991-12.07.1991) [1991]

[27] **Kadlec Jiří** : Identification Algorithms for Parallel Computing Networks with Fixed Point Arithmetic , *Neural Nets for System Applications, p. -*, IEE/ÚTIA, (Prague 1991) , Neural Nets for System Applications, (Prague, CS, 20.05.1990-21.05.19

[28] **Kadlec Jiří** : Fast and Adaptive Identification Algorithms Suitable for Neural Network Applications , *Neural Nets for System Applications, p. -*, IEE/ÚTIA, (Prague 1991) , Neural Nets for System Applications, (Prague, CS, 20.05.1990-21.05.1990) [1991] 90) [1991]

[29] **Kadlec Jiří** : A Joint Criterion for Exponential Directional and Mixed Parameter Tracking , *4th IFAC International Symposium on Adaptive Systems in Control and Signal Processing. ACASP '92, p. 687-692 , Eds: Landau I. D., Dugard L., M'Saad M.*, Laboratoire d'Automatique, (Grenoble 1992) , IFAC International Symposium on Adaptive Systems in Control and Signal Processing. ACASP '92 /4./, (Grenoble, FR, 01.07.1992-03.07.1992) [1992]

[30] **Kadlec Jiří, Gaston F. M. F., Irwin G. W.** : Regularised Lattice-Ladder Adaptive Filter , *IFAC Workshop on Mutual Impact of Computing Power and Control Theory. MICC '92, p. 143-150 , Eds: Kárný M., Warwick K.*, ÚTIA ČSAV, (Prague 1992) , IFAC Workshop on Mutual Impact of Computing Power and Control Theory. MICC '92, (Praha, CS, 01.09.1992-02.09.1992) [1992]

[31] **Kadlec Jiří** : Fast Ladder-Lattice Identification Architecture with Numerically Robust Tracking of Parameters , *Algorithms and Architectures for Real-Time Control, p. 105-111 , Eds: Fleming P. O., Jones D. I.*, Pergamon Press, (Oxford 1992) IFAC Workshop Series. vol.4 , IFAC Workshop on Algorithms and Architectures for Real-Time Control, (Bangor, GB, 11.09.1991-13.09.1991) [1992]

[32] **Kadlec Jiří, Gaston F. M. F., Irwin G. W.** : Systolic Implementation of the Regularized Parameter Estimator , *VLSI Signal Processing 6, p. 520-529 , Eds: Yao K., Jain R., Przytula W., Rabaey J.*, IEEE, (New York 1992) , Workshop on VLSI Signal Processing, (Napa, US, 28.10.1992-30.10.1992) [1992]

[33] **Kadlec Jiří** : Unified Design of Fast Array Estimators, Queen's University, (Belfast 1992) Research Report 1/6 [1992]

[34] **Kadlec Jiří, Gaston F. M. F., Irwin G. W.** : Parallel Implementation of Restricted Parameter Tracking , *Mathematics in Signal Processing, p. 86-88*, University of Warwick, (SouthendonSea 1992) , IMA Conference on Mathematics in Signal Processing /3./, (Warwick, GB, 15.12.1992-17.12.1992) [1992]

[35] **Kadlec Jiří** : The Lattice-Ladder with Generalized Forgetting , *Linear Algebra for Large Scale and Real-Time Applications, p. 397-398 , Eds: Moonen M. S., Golub G. H., De Moor B. L. R.*, Kluwer Academic, (Leuven 1993) NATO ASI Series. Series E: Applied Sciences. vol.232 , Proceedings of the NATO Advanced Study Institute on Linear Algebra for Large Scale and Real-Time Applications, (Leuven, BE, 03.08.1992-14.08.1992) [1993]

[36] **Kadlec Jiří, Gaston F. M. F., Irwin G. W.** : Parallel Implementation of Restricted Parameter Tracking, Queen's University, (Belfast 1993) Research Report 1/2 [1993]

[37] **Kadlec Jiří, Gaston F. M. F., Irwin G. W.** : Regularised Lattice-Ladder Adaptive Filter , *Mutual Impact of Computing Power and Control Theory, p. 245-257 , Eds: Kárný M., Warwick K.*, Plenum Press, (New York 1993) , IFAC Workshop on the Mutual Impact of Computing Power and Control Theory, (Prague, CZ, 01.09.1992-02.09.1992) [1993]

[38] **Kadlec Jiří** : Structure Determination and Tracking for Parallel Radial Basis Function Based Nonlinear Networks , *Innovative Approaches to Modelling and Optimal Control of Large Scale Pipeline Networks, p. 75-84 , Eds: Králik J.*, ÚTIA AV ČR, (Prague 1993) , International Workshop SIMONE /2./, (Prague, CZ, 27.09.1993-30.09.1993) [1993]

[39] **Kadlec Jiří** : Lattice feedback regularised identification , *10th IFAC Symposium on System Identification. Preprints, p. 277-282 , Eds: Blanke M., Söderström T.*, IFAC, (Copenhagen 1994) , IFAC SYSID '94 /10./, (Copenhagen, DK, 04.07.1994-06.07.1994) [1994]

[40] **Kadlec Jiří** : Systolic arrays for identification of systems with variable structure , *Computer-Intensive Methods in Control and Signal Processing, p. 123-132 , Eds: Kulhavá L., Kárný M., Warwick K.*, ÚTIA AV ČR, (Praha 1994) , IEEE Workshop CMP '94, (Praha, CZ, 07.09.1994-09.09.1994) [1994]

[41] **Nedoma Petr, Kárný Miroslav, Halousková Alena** : Recursive Approximation by ARX Model. Simulation Verification of Adaptive Controller Initialization, ÚTIA AV ČR, (Praha 1994) Research Report 1816 [1994]

[42] **Kadlec Jiří** : Parallel Normalized Identification Algorithm with Lattice Feedback Regularization, ÚTIA AV ČR, (Praha 1994) Research Report 1820 [1994]

Disclaimer

This disclaimer is not a license and does not grant any rights to the materials distributed herewith. Except as otherwise provided in a valid license issued to you by UTIA AV CR v.v.i., and to the maximum extent permitted by applicable law:

(1) THIS APPLICATION NOTE AND RELATED MATERIALS LISTED IN THIS PACKAGE CONTENT ARE MADE AVAILABLE "AS IS" AND WITH ALL FAULTS, AND UTIA AV CR V.V.I. HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
(2) UTIA AV CR v.v.i. shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under or in connection with these materials, including for any direct, or any indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or UTIA AV CR v.v.i. had been advised of the possibility of the same.

Critical Applications:
UTIA AV CR v.v.i. products are not designed or intended to be fail-safe, or for use in any application requiring fail-safe performance, such as life-support or safety devices or systems, Class III medical devices, nuclear facilities, applications related to the deployment of airbags, or any other applications that could lead to death, personal injury, or severe property or environmental damage (individually and collectively, "Critical Applications"). Customer assumes the sole risk and liability of any use of UTIA AV CR v.v.i. products in Critical Applications, subject only to applicable laws and regulations governing limitations on product liability.