Local Model Network Toolbox for Nonlinear System Identification

Oliver Nelles,
Benjamin Hartmann, Tobias Ebert, Torsten Fischer,
Julian Belz, Geritt Kampmann

January 2013

Abstract

A new MATLAB toolbox for nonlinear system identification with local model networks is introduced. Its goal is to provide even unskilled users with an easy possibility to generate nonlinear models from data. It trains a number of different models of different complexity with different architectures and a polynomial for comparison. At the end it gives an overview on the achieved performances and makes a recommendation for the best model. In [**?**] a more detailed description of the validation methods used for this toolbox is given.

The toolbox can be used by the trivial function call `LMNTrain(data)` where the training data `data = [u_1 u_2 ... u_p y]` contains the inputs in the first columns and the output in the last column. The model with the best penalty loss function value ($\mathrm{AIC_c}$) is recommended [1, 2].

1 Introduction

This toolbox limits itself to nonlinear models with p inputs and 1 output. For several outputs the user has to generate separate models for each. In the static case, the generated models are of the type

$$\hat{y} = f(u_1, u_2, \ldots, u_p) \tag{1}$$

Dynamic models can be constructed by using tapped delay lines, i.e.:

$$\begin{aligned}\hat{y}(k+1) = f(&u_1(k), \ldots, u_1(k-n_{u1}), u_2(k), \ldots, u_2(k-n_{u2}), \ldots, \\ &u_p(k), \ldots, u_p(k-n_{up}), y(k), \ldots, y(k-n_y))\end{aligned} \tag{2}$$

However, such models performs only one-step-ahead predictions from $y(k)$ to $\hat{y}(k+1)$. For a simulation feedback is necessary.

This documentation concentrates on static models only. Dynamic models are not documented so far. They will be included in the documentation with future toolbox releases.

2 Toolbox Methods

The toolbox carries out several incremental learning algorithms for local model networks. It is build to run automatically and recommends the "best" model achieved. Here "best" stands for the least expected squared error on new data. Therefore, it tries to find a good bias/variance tradeoff and thus to avoid overfitting. Of course, each method can be called on each own and many parameters and options are available to fine tune the results. The primary goal of this toolbox, however, is to address the non-expert and deliver a very robust performance. It is highly encouraged and recommended to look into more details of the proposed methods. This can improve the performance even further.

The investigated methods are:

1. LOLIMOT with local linear models
2. LOLIMOT with local quadratic models
3. HILOMOT with local linear models
4. HILOMOT with local quadratic models

These methods are briefly discussed in the following. They will be extended and improved in the future. More methods will be added. Some methods can easily be commented out in order to save computation time.

2.1 LOLIMOT with Local Linear Models

LOLIMOT (LOcal LInear MOdel Tree) is extensively covered in [5]. It is an incremental tree-structured algorithm that starts with a simple (typically linear) global model and refines its performance in each iteration by splitting the input space in an axes-orthogonal manner. In each iteration the worst local model is split into two equal halves. The associated local linear models are estimated by a local least squares approach. The algorithm is very fast and robust. Its performance can deteriorate with increasing input dimensions due to the sub-optimality of the axes-orthogonal splits.

2.2 LOLIMOT with Local Quadratic Models

In this case, the LOLIMOT algorithm trains local models of full polynomial type. All quadratic local model terms including all cross-terms of type $u_1 u_2$ and similar are considered. It also is recommended for optimization purposes but becomes inefficient for high-dimensional input spaces due to the huge number of cross-terms. Then the sparse quadratic models might be a superior choice.

2.3 HILOMOT with Local Linear Models

HILOMOT and its original ideas are discussed in e.g. [3, 4]. It realizes two major improvements over the LOLIMOT algorithm. a) The strong limitations of axes-orthongonal splits fall and b) the flat (parallel) model structure is transferred into a hierarchical model structure. Advantage a) means that axis-oblique splits are carried out which overcomes the key

weakness of LOLIMOT. The price to be paid is that the split has to be optimized which necessarily is a nonlinear optimization problem. This makes HILOMOT one or two orders of magnitude slower than LOLIMOT. However, it results in much better and more compact models. Advantage b) means that re-activation effects [6] due to the normalization can be completely avoided and only involves drawbacks on truly parallel hardware.

2.4 HILOMOT with Local Quadratic Models

Axes-oblique version of the method from Section 2.2; see Section 2.3.

3 Toolbox Inputs and Outputs

The toolbox can be called with two inputs `traindata` and *optionally* `valdata` and/or `testdata`, respectively, and delivers two outputs `LMNBest` and `AllLMN` :

$$[\texttt{LMNBest},\ \texttt{AllLMN}] = \texttt{LMNTrain}(\texttt{traindata}, \texttt{valdata}, \texttt{testdata})$$

This is the help of the toolbox:

Each method builds its models incrementally. The choice for a good model complexity is of paramount importance.

Commonly, no separate validation data set is available. Then the function is just called as `LMNTrain(data)`. The complexity is determined for each method individually with the help of the AICc criterion [1, 2]. It is not aimed for the optimal bias/variance trade-off but other benefits for parsimonious models are also considered such as: computational demand, interpretability, and ease of handling. Furthermore, a too complex model with significant overfitting is considered more dangerous than a too simple model, because it causes the illusion of a good model fit to the unexperienced user.

If validation data is available, the toolbox shall be called by `LMNTrain(traindata, valdata)`. Then the model is assessed and the recommendation is made according to this separate validation data set which is more reliable. Additionally, the user can provide a separate test data set by calling `LMNTrain(traindata, [ ], testdata)`, which is then used only for testing the model. This ensures comparability to other models.

4 Model Use

The model considered best is characterized by the structure `LMNBest`. The model output can be evaluated for a single data point or a whole data set by

```
output = LMNBest.calculateModelOutput(input)
```

with `input` as $1 \times p$ vector (p = number of inputs) or $N \times p$ matrix (N = number of data points), respectively.

For a one- or two-dimensional visualization of the model the user can type `LMNBest.plotModel` into the MATLAB command window. For further information type `help plotModel`. Furthermore, the partitioning can be plotted by typing `LMNBest.plotPartition`.

If there are more than two inputs, a graphical user interface (GUI) can be used to visualize the model at a demanded operating point. The user is free to choose the two dependent variables, that span the input space. While changing the operating point the change in the model output is simultaneously calculated and visualized. Furthermore, different models or different complexities of one model can be compared. There are two different possibilities to load a model to the GUI. Either you use the following syntax to import the trained local model networks `lmn1` up to `lmnN` from the MATLAB workspace:

$$\texttt{GUIvisualize}(\texttt{lmn1}, \texttt{lmn2}, ..., \texttt{lmnN}),$$

or you save a trained local model network to a variable named 'model' and save this variable as a mat-file. The mat-file can be imported within the GUI.

5 Toolbox Examples

The toolbox functionality is demonstrated on four static example data sets. The user just has to run the function `LMNTrainDemo`. After choosing the example the training procedure starts and after completion the user gets informed about the modeling via the LMNTool output screen (see Figs. 1 and 2).

A Installation Instructions

This short installation guide should help the user to install the Local Model Network (LMN-) Toolbox and get started with the help of the implemented demonstration examples.

1. Navigate to the folder, where all files and sub folders of the toolbox are contained.
2. Open and run the `installLMNTool.m` file in MATLAB.

After the execution of the `installLMNTool.m` file, the LMN-Toolbox has been added to the MATLAB path. Furthermore the documentation of the toolbox is added to the MATLAB help.

To get started with HILOMOT, LOLIMOT and the LMNTool please refer to the m-files located in the folders `hilomot/demo`, `lolimot/demo` and `LMN_Tool/LMNTrainExamples`. For further information just type in one of the following commands in the MATLAB command window:

1. `doc hilomot`,
2. `doc lolimot` or
3. `doc LMNTrain`.

B Implementation Notes

Here one will find additional notes regarding specific MATLAB code implementations. These notes should help to understand some mathematical background as well as weird looking respectively confusing code lines.

B.1 WLSBackwardElimination.m

WLSBACKWARDELIMINATION selects a subset of regressors using the method of backward elimination. The algorithm starts with the whole set of regressors and then iteratively eliminates unimportant regressors.

In every iteration a weighted least squares problem has to be solved with an other subset of regressors contained in the regression matrix. At first the (unweighted) regression matrix $\vec{X}_{org}$ as well as the matrix $\vec{y}_{org}$ (containing the measured output values) are weighted as follows:

$$\vec{X}_i = \sqrt{\vec{W}_i} \cdot \vec{X}_{org} \ , \tag{3}$$

$$\vec{y}_i = \sqrt{\vec{W}_i} \cdot \vec{y}_{org} \ . \tag{4}$$

The weighting matrix $\vec{W}_i$ contains the validity values of each data point in the i-th local model on its diagonal and hence the transposed weighting matrix equals the weighting matrix itself:

$$\vec{W}_i = \vec{W}_i^T \ . \tag{5}$$

Because the weighting matrices are diagonal, the square root of these matrices can be calculated by taking the square root of all entries. Through the weighting in equation 3 and 4, the parameters that minimize the weighted sum of squared errors can be obtained by the following calculation:

$$\hat{\vec{\theta}}_{opti} = (\vec{X}_i^T \vec{X}_i)^{-1} \vec{X}_i^T \vec{y}_i \ . \tag{6}$$

In fact the optimal parameter vector $\hat{\vec{\theta}}_{opti}$ is calculated with the *economy size*[1] QR-decomposition of $\vec{X}_i$:

$$\vec{X}_i = \vec{Q}\vec{R} \ . \tag{7}$$

with the orthogonal matrix $\vec{Q}$ (its columns are orthogonal unit vectors meaning $\vec{Q}^T\vec{Q} = \vec{I}$) and the upper triangle matrix $\vec{R}$. With the QR-decomposition of $\vec{X}_i$ equation 6 turns into:

$$\begin{aligned} \hat{\vec{\theta}}_{opti} &= (\vec{R}^T \underbrace{\vec{Q}^T\vec{Q}}_{=\vec{I}} \vec{R})^{-1} \vec{R}^T \underbrace{\vec{Q}^T\vec{y}_i}_{=\vec{z}_i} \\ &= (\vec{R}^T\vec{R})^{-1}\vec{R}^T\vec{z}_i \ . \end{aligned} \tag{8}$$

Now we look at the weighted error

$$\sqrt{\vec{W}_i}\vec{e} = \vec{y}_i - \vec{X}_i \cdot \hat{\vec{\theta}}_{opti} \tag{9}$$

[1]Note that because of the economy size of the matrices $\vec{Q}$ and $\vec{R}$, $\vec{Q}\vec{Q}^T \neq \vec{I}$

and replace $\vec{X}_i$ with equation 7 as well as $\hat{\vec{\theta}}_{opti}$ with equation 8:

$$\begin{aligned}\sqrt{\vec{W}_i}\vec{e} &= \vec{y}_i - \vec{Q}\underbrace{\vec{R}(\vec{R}^T\vec{R})^{-1}\vec{R}^T}_{=\vec{I}}\vec{z}_i \\ &= \vec{y}_i - \vec{Q}\vec{z}_i \ . \end{aligned} \tag{10}$$

The weighted squared sum of errors is given by:

$$\begin{aligned}\vec{e}^T\sqrt{\vec{W}_i}\sqrt{\vec{W}_i}\vec{e} &= (\vec{y}_i - \vec{Q}\vec{z}_i)^T(\vec{y}_i - \vec{Q}\vec{z}_i) \\ \vec{e}^T\vec{W}_i\vec{e} &= \vec{y}_i^T\vec{y}_i - \underbrace{\vec{y}_i^T\vec{Q}}_{=(\vec{Q}^T\vec{y}_i)^T=\vec{z}_i^T}\vec{z}_i - \vec{z}_i^T\underbrace{\vec{Q}^T\vec{y}_i}_{=\vec{z}_i} + \vec{z}_i^T\underbrace{\vec{Q}^T\vec{Q}}_{=\vec{I}}\vec{z}_i \\ &= \vec{y}_i^T\vec{y}_i - \vec{z}_i^T\vec{z}_i - \vec{z}_i^T\vec{z}_i + \vec{z}_i^T\vec{z}_i \\ &= \vec{y}_i^T\vec{y}_i - \vec{z}_i^T\vec{z}_i \ . \end{aligned} \tag{11}$$

As can be seen in equation 11, the calculation of the weighted MSE on the training data can be computed very efficiently.

References

[1] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Second international symposium on information theory*, volume 1, pages 267–281. Springer Verlag, 1973.

[2] K.P. Burnham and D.R. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer Verlag, 2002.

[3] S. Ernst. Hinging hyperplane trees for approximation and identification. In *IEEE Conference on Decision and Control (CDC)*, pages 1261–1277, Tampa, USA, 1998.

[4] B. Hartmann and O. Nelles. Automatic adjustment of the transition between local models in a hierarchical structure identification algorithm. In *European Control Conference (ECC)*, Budapest, Hungary, August 2009.

[5] O. Nelles. *Nonlinear System Identification*. Springer, Berlin, Germany, 2001.

[6] R. Shorten and R. Murray-Smith. Side-effects of normalising basis functions in local model networks. In *Multiple Model Approaches to Modelling and Control*, chapter 8, pages 211–229. Taylor & Francis, London, 1997.

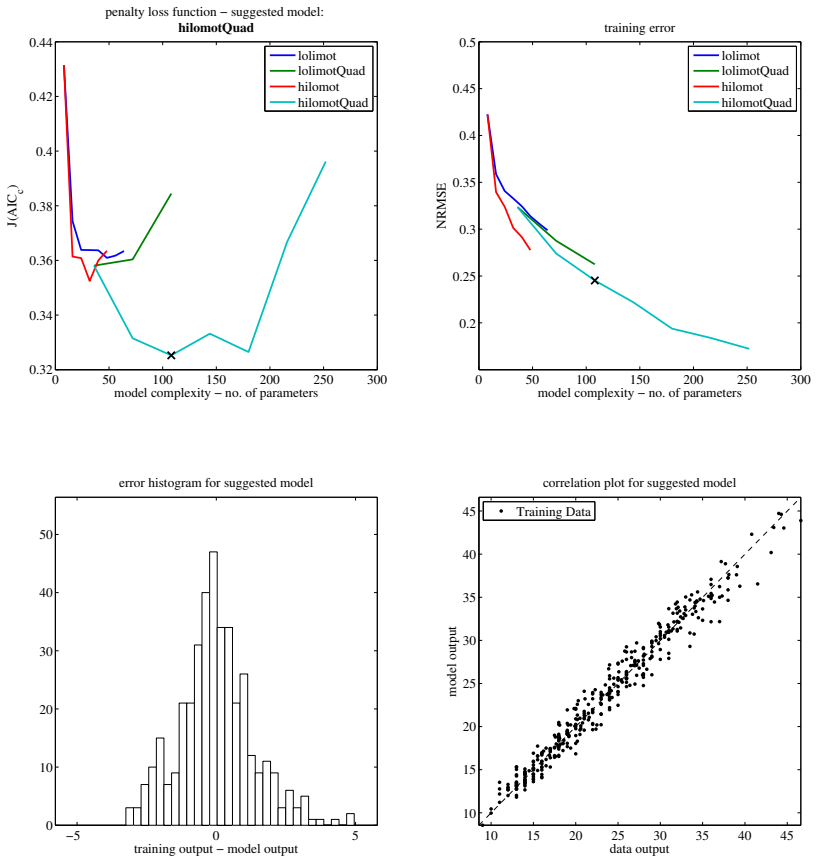

Figure 1: Output screen of the LMNTool for modeling *without* test and validation data. In contrast to modeling just using the training data now the model complexity is selected on the validation data (top right). The selected model is marked with the black cross and highlighted in the title of the figure for the penalty loss function. As additional plots the model vs. test data correlation (bottom middle) and the model vs. validation data correlation (bottom rigth) are drawn.

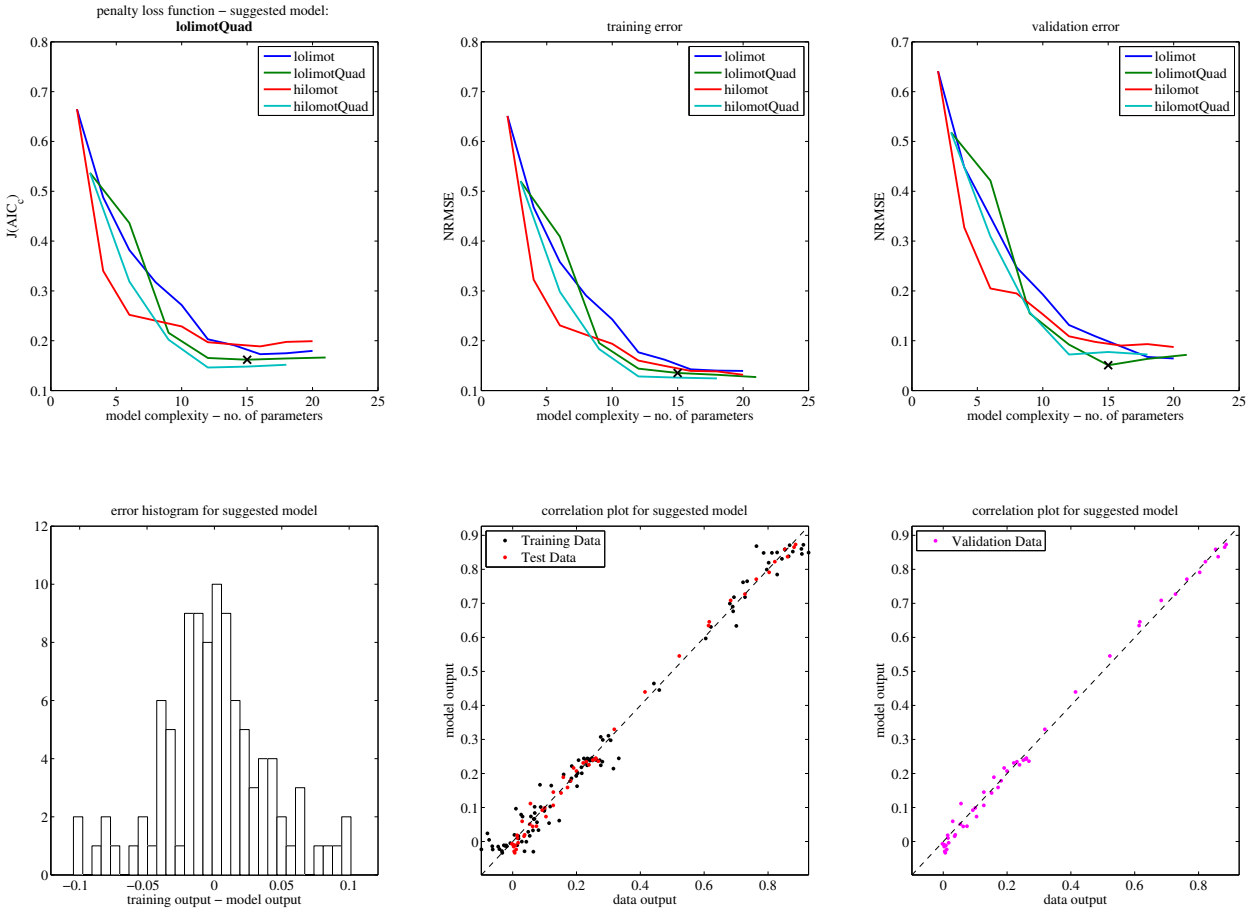

Figure 2: Output screen of the LMNTool for modeling *with* test or validation data. The model complexity is chosen with respect to the penalty loss function value (top left). Furthermore, training errors of each modeling approach (top right), error histogram (bottom left) and data vs. model correlation (bottom right) are illustrated.