1. LMN Toolbox

This chapter briefly introduces a Local Model Network (LMN) toolbox in Matlab. It can be downloaded from

`http://www.mb.uni-siegen.de/mrt/lmn-tool/`

for non-commercial use. The two algorithms for building local model networks with axis-orthogonal partitioning (LOLIMOT) and with axis-oblique partitioning (HILOMOT) introduced in Chapters **??** and **??** are implemented.

Both algorithms incrementally construct a tree with local models as their leaf nodes. However, the obtained model structures are flat with LOLIMOT and hierarchical with HILOMOT. While LOLIMOT ensures the partition of unity by normalizing the membership functions (defuzzification step), HILOMOT hierarchically ensures that each split maintains the partition of unity by utilizing complementary splitting functions Ψ and $1-\Psi$. LOLIMOT uses local membership functions (Gaussians) while HILOMOT uses S-shaped splitting functions (sigmoids). Note that the exact functional from of theses functions is not very relevant; just their principal shape and their smoothness is important.

The toolbox is implemented in MATLAB in an object-oriented manner. First an object corresponding to a tree of local model networks needs to be established for either axis-orthogonal or -oblique partitioning by

```
lmn = lolimot
```

or

```
lmn = hilomot .
```

Then the networks can be trained with the default settings by

```
lmnTrained = lmn.train
```

resulting in a tree of trained networks `lmnTrained`. Afterwards, the output of the trained model can be calculated by

```
ModelOutput = lmnTrained.calculateModelOutput(input)
```

or

$$\texttt{ModelOutput} = \texttt{lmnTrained.calculateModelOutput}(\texttt{input}, \texttt{output})$$

where `output` is optional and only used for one-step prediction in dynamic models (if `kStepPrediction=0`) and initialization in case of simulation (if `kStepPrediction=inf`).

In the following, the most important settings to influence the training are explained. These settings can be altered from their default value by

$$\texttt{lmn.property} = \texttt{new value}$$

before training.

1.1 Termination Criteria

One decisive factor for the quality of a model is an appropriate bias/variance tradeoff. The toolbox offers a variety of criteria for finding a good model complexity. In particular, these are information criteria ($\mathrm{AIC_C}$ and $\mathrm{BIC_C}$) and performance on validation data. In addition, the model complexity can be directly controlled by the maximal number of local models or effective number of parameters or indirectly controlled by choosing a maximal training time.

1.1.1 Corrected AIC

The Akaike information criterion (AIC) corrected for finite data sets called $\mathrm{AIC_C}$ is the most popular choice, compare Section **??** and [1, 2]. For the number of parameters the *effective* not the nominal number is used, compare Section 1.1.5. The goal is finding the network associated with the global minimum of the information criterion. This model is suggested by the toolbox.

At some point the incremental training has to be terminated to keep the computational demand low. Sometimes random fluctuations on the convergence curve can be observed that are due to noise and suboptimalities in the training algorithm. Therefore it is advisable to NOT terminate training after the *first* deterioration of the criterion. Rather a couple of subsequent deteriorations are required for termination, see Fig. 1.1. The default number is

$$\texttt{maxValidationDeterioration} = \texttt{2}.$$

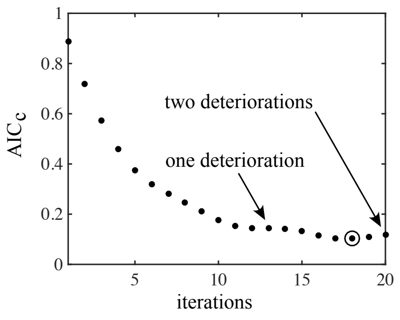

Fig. 1.1. Convergence of the $\mathrm{AIC_C}$ criterion. The model corresponding to the global optimum (here at $M = 18$) shall be selected. Training is terminated after the criterion increases 2 times (by default)

1.1.2 Corrected BIC

The Bayesian information criterion (BIC) corrected for finite data sets called $\mathrm{BIC_C}$ exhibits a larger parameter penalty than the AIC, compare Section **??** and [1, 2]. Therefore it yields simpler models. This may be advisable for scenarios with huge training data sets or whenever the motivation for simple models is higher than normal. These circumstances are often fulfilled for dynamic models since typically much more data can be gathered in the same amount of time than with static measurements.

Note that many motivations exist for simpler models than the best bias/variance tradeoff which can be seen as the upper threshold for model complexity. These additional motivations include:

- Computational demand (computing time and memory) during training and use.
- Interpretability.
- Robustness.

In order to force even simpler models than those suggested by the $\mathrm{BIC_C}$ the following property allows to increase a factor multiplied with the parameter penalty of the $\mathrm{AIC_C}$ or $\mathrm{BIC_C}$ criterions, i.e.,

`complexityPenalty` $= 1$.

1.1.3 Validation

If plenty of data is available it may be the best choice to generate a separate representative validation data set. Instead of the information criteria discussed above then the performance on validation data is monitored in exactly the same manner.

Note that the user possesses two alternative procedures

1. Validation data: Splitting the data into training and validation data. Both should be representative. Determining the model complexity on the validation data performance.
2. No validation data: Using all data for training. Determining the model complexity on an information criterion.

Procedure 2 offers the potential for better models (more training data) but this comes with a worse choice of model complexity (no validation data).

1.1.4 Maximum Number of Local Models

The number of local models or neurons or rules M can be used as termination criterion. Note that a minimum number of data is requested for each local model. By default it is required that the number of data points within each local models is at least as big as the number of parameters to be estimated. Because the validity functions overlap this requirement is formulated in a fuzzy way

$$n_i \leq \sum_{j=1}^{N} \Phi_i(j) \tag{1.1}$$

where n_i is the number of parameters in local model i and $\Phi_i(j)$ is the validity of data point j in the local model i under investigation. If (1.1) is violated then this split of the local model is not allowed.

In case of LOLIMOT, other splits of the same model may be possible. If none of the splits of the worst local model is possible then the next worse local model is considered for splitting. Therefore a reasonable limit on the choice of $M_{\max}$ is given by the data density and distribution.

In case of HILOMOT, the split optimization is constrained with (1.1); thus it is always met. However, local models which violate (1.1) for all initial splits even *before* splitting optimization starts, are locked and the next worse local model is split.

1.1.5 Effective Number of Parameters

The effective number of parameters gives a good indication of the flexibility of the model. For LOLIMOT just the effective number of parameters of the local models are summed up, see also (**??**) in Section **??**

$$n_{\text{eff}} = \sum_{i=1}^{M} n_{\text{eff},i} \tag{1.2}$$

with

$$n_{\text{eff},i} = \text{tr}\{\underline{S}_i\} \tag{1.3}$$

where $\underline{S}_i$ is the smoothness matrix of local model i. There is some discussion whether the effective number of parameter should be calculated as $\mathrm{tr}\{\underline{S}_i\}$ or $\mathrm{tr}\{\underline{S}_i^T\underline{S}_i\}$ but the statistics literature seems to agree on $\mathrm{tr}\{\underline{S}_i\}$.

The degrees of freedom contained in the flexibility of the partitioning determined by the centers and standard deviations of the Gaussians are neglected. This can be justified by the coarse LOLIMOT algorithm which delivers far from optimal partitioning.

For HILOMOT the partitioning is much more flexible and the splitting parameters are numerically optimized. Thus, in addition to the local model parameters in (1.2), the number of splitting parameters needs to be considered. For each split there is one splitting parameter per dimension (in the space of the validity functions spanned by $\underline{z}$) plus the offset. However, the smoothness of the sigmoid is not optimized but rather normalized and determined by a heuristic smoothness adjustment. Thus each split possesses n_z independent splitting parameters, compare Section **??**. Since for an LMN with M local models there are $M-1$ splits this yields

$$n_{\mathrm{split}} = n_z(M-1)\,. \tag{1.4}$$

Thus a HILOMOT trained LMN with M local models is considered more flexible than a LOLIMOT trained one. This discrepancy increases if the validity function input space dimensionality n_z grows.

These effective number of parameters also enter the $\mathrm{AIC_C}$ and $\mathrm{BIC_C}$ criteria mentioned above. Note that these are very rough estimates. The splitting parameters are not optimized concurrently which would make the model much more flexible. Thus, (1.2)+(1.4) is an overestimation of HILOMOTs flexibility.

1.1.6 Maximum Training Time

For huge models that may be generated for large training data sets it can be reasonable to specify the maximum training time [in seconds] which can be utilized as termination criterion. In practice, this feature can be used for generating the best possible results over night, for example.

1.2 Polynominal Degree of Local Models

The local models in the LMN toolbox can be polynomials of any order. If the order of the polynomials and/or input space dimensionality nx grows then the number of parameters increases rapidly. Therefore, good trade-offs typically are low order polynomials. The most frequently chosen local model types certainly are

- *Linear* models: The default choice. The number of parameters per local model is $nx + 1$. They offer particular advantages for dynamic models due to their extrapolation behavior which maintains the dynamics in extrapolation.
- *Sparse quadratic* models: This refers to quadratic models without cross-product (or interaction) terms, i.e., in the two-dimensional case
$$y = w_0 + w_1u_1 + w_2u_2 + w_3u_1^2 + w_4u_2^2 \tag{1.5}$$
without $w_5u_1u_2$. The number of parameters per local model is $2nx + 1$, i.e., grows only linearly with nx. Nevertheless they are able to describe a minimum or maximum without the support of neighboring local models which is a clear benefit for non-monotonous models. In contrast, local linear models need to change sign in each dimension to describe a minimum or maximum. This is difficult to achieve, particularly in high dimensions. Furthermore, it is sensitive with respect to the exact shape of the validity functions which makes the model less robust. Therefore sparse quadratic models are a good choice in theses cases. Contrary to full quadratic models they can be used even for large nx.
- *Full quadratic* models: The number of parameters per local model is $(nx + 2)(nx + 1)/2$. Certainly the best choice if optima of the model are of interest. They can be interpreted as a nonlinear extension of Newton's method for optimization where local quadratic models are estimated in certain areas of the input space, compare Section **??**. However, the number of parameters grows quadratically with nx which limits their applicability to low-dimensional problems.

Higher order polynomials usually are recommended only if motivated by prior knowledge about the process. Of course, subset selection techniques like orthogonal least squares or lasso allow to extend the limits by local regressor selection, compare Section **??**.

1.3 Dynamic Models

In order to build dynamic models the delays of the inputs and possibly outputs must be specified. To allow for a maximum of flexibility not just the dynamic order (and dead time(s)) can be chosen but specifically every delay for every network input. This also be done individually for the regressors of the local models (or rule consequents) gathered in $\underline{x}$ and those for the validity functions (or rule premises) gathered in $\underline{z}$, compare Section **??**.

Pure feedforward structures of NFIR type just require delays of inputs. They are specified in the properties `xInputDelay` for $\underline{x}$ and `zInputDelay` for $\underline{z}$.

Example 1.3.1. NFIR of Fourth Order
The model is

$$\hat{y}(k) = f\left(u(k-1), u(k-2), u(k-3), u(k-4)\right) . \tag{1.6}$$

This require in the general form $\underline{x} = \underline{z}$ and thus

$$\texttt{xInputDelay} = \texttt{zInputDelay} = \{[1\ 2\ 3\ 4]\} .$$

The "{}" indicates a cell array.

The delayed output does not enter either $\underline{x}$ or $\underline{z}$ and thus

$$\texttt{xOutputDelay} = \texttt{zOutputDelay} = \{[\ ]\} .$$

It may be sufficient to provide the validity functions (rule premises) just with the level of $u(k)$, not with any dynamics (like first and/or second derivate of $u(k)$). Then it would be sufficient to choose

$$\texttt{zInputDelay} = \{[1]\}$$

which makes the whole problem significantly simpler going from 4D to 1D still maintaining the fourth order in the local models.

□

Example 1.3.2. Dead Time
A fourth order model with dead time d is

$$\hat{y}(k) = f\left(u(k-1-d), u(k-2-d), u(k-3-d), u(k-4-d)\right) \tag{1.7}$$

which translates in the following delays

$$\texttt{xInputDelay} = \texttt{zInputDelay} = \{[1+d \quad 2+d \quad 3+d \quad 4+d]\} .$$

□

Example 1.3.3. MISO NFIR
For system with multiple inputs it is possible to specify the delays individually because the properties are cell arrays and thus can have individual lengths for each input. Assume the following model:

$$\hat{y}(k) = f\left(u_1(k-1), u_1(k-2), u_1(k-3),\ u_2(k-1), u_2(k-2)\right) . \tag{1.8}$$

For $\underline{x} = \underline{z}$ this is specified by

$$\texttt{xInputDelay} = \texttt{zInputDelay} = \left\{ \begin{array}{l} [1\ 2\ 3] \\ [1\ 2] \end{array} \right\} .$$

□

Example 1.3.4. MISO NARX
For NARX models the delayed outputs also enter the network. Assume

$$\hat{y}(k) = f\left(u_1(k),\ u_2(k-1), u_2(k-2),\ y(k-1), y(k-2)\right) . \tag{1.9}$$

For $\underline{x} = \underline{z}$ this is specified by the following input delays

$$\texttt{xInputDelay} = \texttt{zInputDelay} = \left\{ \begin{matrix} [0] \\ [1\ \ 2] \end{matrix} \right\}$$

and the following output delays

$$\texttt{xOutputDelay} = \texttt{zOutputDelay} = \{[1\ 2]\} .$$

□

Example 1.3.5. MIMO NARX
For a first order NARX model with two inputs and two outputs different possibilities exist. Two separate models of the following type can be used

$$\hat{y}_1(k) = f_1\left(u_1(k-1),\ u_2(k-1),\ y_1(k-1)\right) \tag{1.10}$$
$$\hat{y}_2(k) = f_2\left(u_1(k-1),\ u_2(k-1),\ y_2(k-1)\right) \tag{1.11}$$

where each model describes one output without relating to the other output. Here two models – one for $y_1(k)$, the other for $y_2(k)$ – are set up. For $\underline{x} = \underline{z}$ with

$$\texttt{xInputDelay} = \texttt{zInputDelay} = \left\{ \begin{matrix} [1] \\ [1] \end{matrix} \right\}$$

and

$$\texttt{xOutputDelay} = \texttt{zOutputDelay} = \{[1]\} .$$

Alternatively, both delayed outputs can be used in both models

$$\hat{y}_1(k) = f_1\left(u_1(k-1),\ u_2(k-1),\ y_1(k-1),\ y_2(k-1)\right) \tag{1.12}$$
$$\hat{y}_2(k) = f_2\left(u_1(k-1),\ u_2(k-1),\ y_1(k-1),\ y_2(k-1)\right) \tag{1.13}$$

which not only requires feedback from each model's output to its input in simulation but also from each model's output to the other model's input. This certainly will increase potential stability problems.

WARNING: Such an approach can only be used for one-step prediction with the discussed toolbox. For simulation it would require two simulate two models in parallel and feed back the outputs in a crosswise manner.

For prediction the “other” output can be treated as an additional (third) input which could be realized by

$$\texttt{xInputDelay} = \texttt{zInputDelay} = \left\{ \begin{matrix} [1] \\ [1] \\ [1] \end{matrix} \right\} .$$

and

$$\texttt{xOutputDelay} = \texttt{zOutputDelay} = \{[1]\}$$

with $u_3(k) = y_2(k)$ for model 1 and $u_3(k) = y_1(k)$ for model 2.

Finally, a real MIMO model can be established:

$$\begin{bmatrix} \hat{y}_1(k) \\ \hat{y}_1(k) \end{bmatrix} = \underline{f}\,(u_1(k-1),\ u_2(k-1),\ y_1(k-1),\ y_2(k-1))\ . \tag{1.14}$$

□

Often significant improvements can be achieved by keeping $\underline{z}$ low-dimensional. Especially for high-order dynamics it is very beneficial to choose only $\underline{x}$ high-dimensional where it only increases the estimation variance and computational demand moderately; while keeping the dimensionality of the $\underline{z}$ input space to a minimum which is essential for combating the curse of dimensionality.

Nonlinear Orthonormal Basis Function Models. This dynamic realization is not specifically supported. It can be realized by filtering the inputs appropriately and utilizing them with a "static" model.

1.4 Different Input Spaces $\underline{\mathrm{x}}$ and $\underline{\mathrm{z}}$

Static models do not require any delays. Therefore always

$$\texttt{xOutputDelay} = \texttt{zOutputDelay} = \{[\ ]\}\,.$$

The entries for `xInputDelay` and `zInputDelay` can either be 0 or empty dependent on whether the inputs do exist in $\underline{x}$ and $\underline{z}$ or not.

Example 1.4.1. Scheduling
The following relationship shall be modeled where the parameters depend on u_3. Such models are called *linear parameter varying (LPV)*:

$$\hat{y}(k) = w_0(u_3) + w_1(u_3)u_1 + w_2(u_3)u_2\,. \tag{1.15}$$

The local model network describing this relationship has three inputs: u_1 and u_2 enter $\underline{x}$ and u_3 enters $\underline{z}$. Thus

$$\texttt{xInputDelay} = \left\{ \begin{array}{c} [0] \\ [0] \\ [\ ] \end{array} \right\}$$

and

$$\texttt{zInputDelay} = \left\{ \begin{array}{c} [\ ] \\ [\ ] \\ [0] \end{array} \right\}\,.$$

□

1.5 Smoothness

The smoothness of the validity functions is determined by the standard deviations of the Gaussians in the LOLIMOT case and by the absolute values of the direction weights of the sigmoids in the HILOMOT case, respectively. The property `smoothness` determines the proportionality factor between the partitions' extensions and the standard deviations in the LOLIMOT case and determines a minimum validity value for data points close to the split or close to the center of the local model in the HILOMOT case.

Since the smoothness is not optimized but heuristically determined it can be increased to make the model smoother or decreased to make the model crisper. Its default value is 1.

$$\texttt{smoothness} = \texttt{value}\,.$$

Even after training this value can be changed. Although, strictly speaking, it changes the validity values and thus the outcome of the weighted LS estimations of the local models' parameters, this effect typically is small. If the smoothness is altered significantly a re-estimation of the local models' parameters is recommended. Slight suboptimalities with respect to the partitioning usually can be tolerated.

1.6 Data Weighting

By default all data points are weighted equally with weight 1. It can be set by the property

$$\texttt{dataWeighting} = [w(1)\ w(2)\ \cdots\ w(N)]'\,.$$

By choosing an appropriate weighting "badly" distributed data which is undesirably dense in some regions and sparse in others can be "normalized" in its importance on the model fit. Alternatively, sometimes it is desirable to over-pronounce certain regions compared to others because their performance it more important. Data weighting is an easy and effective method to deal with these issues.

1.7 Visualization and Simplified Tool

For first steps that do not require (and allow) delving into the details of the toolbox and dealing with the settings/properties explained above a very simplified approach is as follows. Local model networks can be trained by the trivial function call

```
LMNTrain(data)
```

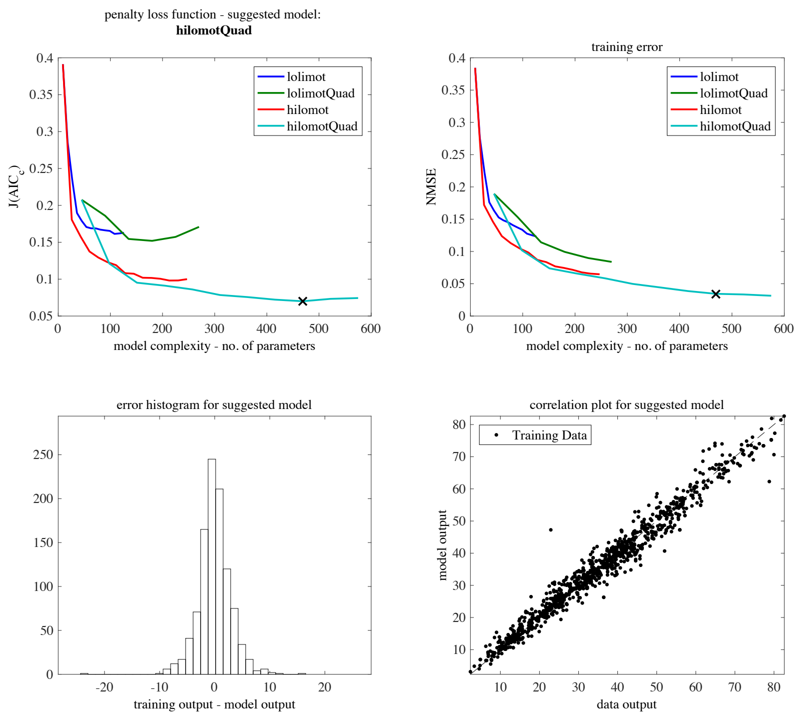

Fig. 1.2. Typical outcome of the LMN toolbox. Default setting is the training of four networks with local linear and quadratic models with axis–orthogonal (LOLIMOT) and -oblique (HILOMOT) partitioning

where the training data

$$\texttt{data} = [\texttt{u\_1 u\_2 ... u\_p y}]$$

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]. Figure 1.2 shows the plot that is generated by this function call. The data set is the benchmark “Concrete Compressive Strength” from the UCI data repository[1].

In extension, this simplified tool can be called with two or three inputs `traindata` and *optionally* `valdata` and/or `testdata`, respectively, and delivers two outputs `LMNBest` and `AllLMN`:

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

[1] `https://archive.ics.uci.edu/ml/datasets.html`

Then the model is assessed and the complexity recommendation is made according to this separate validation data set which is more reliable (if representative).

Commonly, no separate validation data set is available. Then the function is just called as `LMNTrain(traindata)` without test data or `LMNTrain(traindata, [], testdata)` with test data. The complexity then is determined with the help of the $\mathrm{AIC_C}$ criterion [1, 2].

If the user provides a separate test data set then it is used purely for testing the model. This allows comparability to other models. It is important that no actions are based on the test data performance, e.g. any further subsequent selection step by comparing different model architectures. Then the test data would be used as some kind of validation data whose performance is over-optimistic.

References

1. H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.
2. Kenneth P Burnham and David R Anderson. *Model selection and multimodel inference: a practical information-theoretic approach.* Springer Science & Business Media, 2002.