Model fitting

author: stephane.ploix@grenoble-inp.fr

reviewer:
mircea_stefan.simoiu@upb.ro

Introduction

A model of the H358 office has been developed from knowledge based on physics but it appears that it does not match perfectly the measurements. It can be explained by different facts:

- some phenomena, like the contribution of the boundary layers, are approximated or even neglected, like radiation exchange with the celestial vault
- some parameters, like conductivity, are not known with precision
- the model structure has been simplified, like dimensions of the office.

A first idea is to adjust the parameters of the model to better match the measurements i.e. to minimize the error between the model predictions and the related measurements. However, this approach is not that easy because there are numerous parameters, the objective function is generally not convex and parameters might be impossible to identify: there are many local minima. Moreover, the knowledge-based model is nonlinear in the parameters. This approach will be discussed at the end of the tutorial.

Alternatively, regular models, like ARX models, can be used. They are linear in the parameters and the optimization problem is easy to solve. Nonlinear extensions can also be considered like recurrent artificial neural networks like NARX, which are relatively easy to tune.

We are going to study how to fit a linear regressor named ARX, and compare it with a physically inspired knowledge-based model andthen a nonlinear regressor named NARX (a Multi-Layer Recurrent Perceptron Artificial Neural Network) models, easy to tune.

A regular model uses to requires a larger dataset because the model structure can be deduced but little expertise of the domain.

A knowledge-based model requires expertise of the domain but a smaller dataset.

ARX model

The main issue with regular models is to find a good model structure. Indeed, these models have little connection with the physics of the system therefore their structure can't be deduced from the physics.

An ARX model with an output y (y means measured value and $\hat{y}$ the related simulated value) and the following inputs $u_1, \ldots, u_p$ can be written:

$$\begin{aligned}\hat{y}(k) = &-a_1\hat{y}(k-1) - \cdots - a_n\hat{y}(k-n_a) \\ &\cdots + b_{1,1}u_1(k-n_k) + \cdots + b_{1,n_b}u_1(k-n_k-n_b) \\ &\cdots + \ldots \\ &\cdots + b_{p,1}u_p(k-n_k) + \cdots + b_{p,n_b}u_p(k-n_k-n_b) + \Delta\end{aligned}$$

where:

- n_a is the maximum output delay
- n_k is the minimum input delay common to all inputs (it's modeling a time delay in the response)
- n_b are the maximum input delay for each input
- p is the number of inputs

The key idea is that an ARX model should capture the underlying dynamics of the system, but not the random fluctuations that happen to appear in one particular dataset.

1. Model complexity

For an ARX model, the hyper-parameters are n_a, n_b and n_k that define the model structure (or complexity). Once these are fixed, the coefficients are estimated from the data.

There are two extreme cases.

Case 1: Model too simple (under-fitting)

If n_a, n_b, or n_k are too small, the model does not have enough degrees of freedom to reproduce the system dynamics.

As a result:

- the prediction errors remain large,
- important dynamic behaviors are not represented,
- both the training and testing datasets are poorly predicted.

For example, if the real system behaves like a third-order system but we estimate a first-order ARX model, the model simply cannot reproduce the observed dynamics.

Case 2: Model too complex (over-fitting)

If n_a and n_b are unnecessarily large, the model has many parameters.

Some of these parameters begin to explain not only the true system dynamics but also accidental features of the training data, such as

- measurement noise,
- disturbances,
- sensor errors,
- random fluctuations.

These phenomena are not part of the physical system and will not appear in exactly the same way in another experiment.

Consequently,

- the model fits the training data extremely well,
- but its predictions on new data become worse.

This is called over-fitting because the model is fitting the entire training dataset, including its random imperfections.

The compromise

The objective is therefore not to obtain the smallest possible training error. Instead, we seek the model that best predicts unseen data.

A good model captures the true dynamics, ignores random noise, and produces similar prediction accuracy on both the training and testing datasets.

This is why hyper-parameter selection (choosing n_a, n_b, and n_k) is a trade-off between sufficient complexity to describe the system accurately and sufficient simplicity to ensure good generalization.

Parameter estimation

A rigorous ARX identification workflow should clearly separate the objectives of **model estimation**, **hyperparameter selection**, and **final performance evaluation**. A common mistake is to use the same data for all three tasks, which leads to optimistic performance estimates. The procedure below is widely used in system identification and predictive modeling.

Step 1. Acquire and preprocess the data

Collect synchronized measurements of

- input(s): $u(k)$,
- output(s): $y(k)$.

Then perform the usual preprocessing: remove missing values, detect outliers, synchronize timestamps, optionally filter measurement noise, detrend if necessary and meaningful, normalize variables (optional but often useful).

Step 2. Split the data into training and testing sets

The first split should always be chronological.

For example,

The **training set** is used to estimate the model and tune the hyper-parameters.

The **test set** is never used during model selection. It is kept aside until the very end to estimate the true generalization performance.

For building data, typical splits are 70%/30% or 80%/20%.

Step 3. Define the hyper-parameter search space

Choose candidate values for

- autoregressive order n_a
- input order n_b
- input delay: n_k

Example: $n_a \in 1, \ldots, 8, n_b \in 1, \ldots, 8, n_k \in 1, \ldots, 6$

The total number of candidate models is $N_{\text{models}} = |n_a||n_b||n_k|$

Step 4. Perform rolling-origin validation on the training set

This is where the hyperparameters are selected.

Suppose the training set contains

```
January ========================================== October
```

Use several rolling validation windows. Only past data are used to predict future data.

```
Fold 1

Train
=====================
```

Step 5. Estimate each candidate ARX model

For every candidate (n_a, n_b, n_k), repeat the following steps:

For each rolling-origin split:

1. estimate the ARX coefficients using least squares,
   $\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T y$,
2. predict the validation period,
3. compute prediction metrics.

Typical metrics are (see hereafter)

- $\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum(y-\hat{y})^2}$
- $\mathrm{MAE} = \frac{1}{N}\sum|y-\hat{y}|$
- $\mathrm{FIT} = 100\left(1-\frac{|y-\hat{y}|}{|y-\bar{y}|}\right)$
- $\mathrm{NRMSE} = \frac{\mathrm{RMSE}}{\mathrm{std}(y)}$
- $\mathrm{MSE}_{\text{multi-step}} = \frac{1}{N}\sum_k \sum_{h=1}^{H} w_h(y(k+h)-\hat{y}(k+h))^2$ (multi-step prediction error: see hereafter) typically for $H \in \{1h, \cdots, 24h\}$

Step 6. Select the optimal hyper-parameters

Compute $J(n_a, n_b, n_k) = \text{Average validation error}$

Choose $(n_a^*, n_b^*, n_k^*) = \arg\min J$

However, do not automatically choose the most complex model.

For example

Model	Validation RMSE
(3,3,1)	0.84
(4,4,1)	0.83
(7,7,1)	0.82

The improvement from 0.83 to 0.82 may not justify doubling the model complexity.

A common engineering practice is to choose the **simplest model within one standard deviation of the minimum validation error** (the “one-standard-error rule”), or another parsimonious criterion if appropriate.

Step 7. Retrain the selected model

Once the hyperparameters are fixed,
estimate the ARX model again,
but now using the **entire training dataset**.

This provides the best estimate of the parameters because more data are available.

Step 8. Evaluate on the test set

Now, and only now,

apply the final model to the unseen test data.

```
Training data                          Test data

Estimate model  ------------------->  Evaluate once
```

Compute

- RMSE,
- MAE,
- FIT,
- residual analysis,
- prediction plots.

This is the only unbiased estimate of future performance.

For an ARX model, once the parameters have been estimated, the model is evaluated on the **test set** (or on each validation window during rolling-origin validation). Let

- measured output: $y(k)$,
- predicted output: $\hat{y}(k)$,
- number of samples: N.

The prediction error (or residual) is defined as

$$e(k) = y(k) - \hat{y}(k).$$

All the indicators below are derived from this error sequence.

Evaluation metrics

1. Root Mean Squared Error (RMSE)

The RMSE measures the average magnitude of the prediction error while giving more weight to large errors.

It is defined as

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left(y(k) - \hat{y}(k)\right)^2}$$

or equivalently

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} e(k)^2}$$

Interpretation

- Perfect prediction:

$$\mathrm{RMSE} = 0$$

- Smaller values are better.

Because the error is squared before averaging, RMSE penalizes large prediction errors more strongly than MAE.

Example

Error	Squared error
1	1
2	4
5	25

A few large errors can therefore significantly increase the RMSE.

2. Mean Absolute Error (MAE)

MAE measures the average absolute prediction error.

It is defined as

$$\mathrm{MAE} = \frac{1}{N} \sum_{k=1}^{N} |y(k) - \hat{y}(k)|$$

or

$$\mathrm{MAE} = \frac{1}{N} \sum_{k=1}^{N} |e(k)|.$$

3. FIT (Fit Percentage)

In system identification, MATLAB's System Identification Toolbox reports the FIT index.

It is defined as

$$\text{FIT} = 100\left(1 - \frac{|y - \hat{y}|}{|y - \bar{y}|}\right)$$

where the mean output is:

$$\bar{y} = \frac{1}{N}\sum y(k)$$

The norm is usually the Euclidean norm

$$|x| = \sqrt{\sum x(k)^2}$$

Equivalent expression

$$\mathrm{FIT} = 100 \left(1 - \sqrt{\frac{\sum e(k)^2}{\sum (y(k) - \bar{y})^2}} \right).$$

Interpretation

FIT	Meaning
100 %	Perfect prediction
90 %	Excellent
80 %	Very good
60 %	Moderate
0 %	No better than using the mean value
<0 %	Worse than predicting the mean

Unlike RMSE, FIT is normalized by the variability of the measured signal, making it easier to compare models across datasets with different scales.

Average these metrics over all rolling windows.

**4. Multi-step prediction error ($\mathrm{MSE}_{\text{multi-step}}$)

Instead of minimizing only one-step prediction error, one may minimize the multi-step prediction error:

$$\mathrm{MSE}_{\text{multi-step}} = \frac{1}{N} \sum_{k} \sum_{h=1}^{H} w_h (y(k+h) - \hat{y}(k+h))^2$$

where H is the prediction horizon and w_h is a weighting factor.

This criterion encourages models that remain accurate over longer horizons rather than only one-step ahead.

5. Residual Analysis

Residual analysis is often more important than RMSE or FIT because it evaluates whether the model has captured all the predictable dynamics.

A good model should leave only random noise in the residuals.

The residual sequence is

$$e(k) = y(k) - \hat{y}(k).$$

Several statistical tests are commonly performed.

(a) Mean of residuals

Compute

$$\bar{e} = \frac{1}{N} \sum e(k).$$

A good model should satisfy

$$\bar{e} \approx 0.$$

A nonzero mean indicates a systematic bias.

(b) Residual variance

Estimate

$$\sigma_e^2 = \frac{1}{N-1}\sum (e(k) - \bar{e})^2.$$

Lower variance generally indicates better predictive performance.

(c) Residual autocorrelation

The sample autocorrelation at lag τ is

$$R_e(\tau) = \frac{1}{N} \sum e(k)e(k-\tau).$$

It is often normalized as

$$\rho_e(\tau) = \frac{R_e(\tau)}{R_e(0)}.$$

A good ARX model should satisfy:

$$\rho_e(\tau) \approx 0 \quad (\tau > 0).$$

Significant autocorrelation suggests that some dynamics remain unmodeled.

A residual autocorrelation plot typically shows the autocorrelation coefficients for different lags together with 95% confidence bounds (approximately $\pm 1.96/\sqrt{N}$). Ideally, all coefficients except at lag 0 lie within these bounds.

(d) Cross-correlation between residuals and inputs

Compute

$$R_{ue}(\tau) = \frac{1}{N} \sum u(k-\tau)e(k).$$

A good model should satisfy:

$$R_{ue}(\tau) \approx 0$$

If residuals are correlated with past inputs, it means the inputs still contain information that the model has failed to exploit.

(e) Distribution of residuals

Plot histograms

Ideally,

- zero mean,
- symmetric,
- approximately Gaussian (although Gaussianity is not strictly required for least-squares estimation).

6. Prediction Plots

Prediction plots compare measured and predicted outputs over time.

The simplest plot displays

- measured output: $y(k)$
- predicted output: $\hat{y}(k)$

A good model should exhibit:

- close overlap between the two curves,
- no long-term drift,
- no systematic lag,
- accurate tracking of peaks and transients.

Prediction plots are often produced for:

- one-step-ahead predictions,
- h-step-ahead predictions (e.g., 6-hour or 24-hour forecasts in building applications),
- free-run simulations (where the model predicts recursively without using measured outputs).

Comparing these plots helps distinguish models that perform well only over short horizons from those that remain accurate over extended simulations.

Typical evaluation workflow

For each candidate ARX model:

1. Estimate the parameters using the training data.
2. Generate predictions on the validation or test data.
3. Compute the residuals:

$$e(k) = y(k) - \hat{y}(k).$$

4. Calculate quantitative metrics:
    - RMSE,
    - MAE,
    - FIT,
    - $\mathrm{MSE}_{\text{multi-step}}$
5. Perform residual diagnostics:
    - mean close to zero,
    - low variance,
    - residual autocorrelation within confidence bounds,
    - negligible cross-correlation with past inputs.
6. Inspect graphical diagnostics:

Step 9. Validate the residuals

A good ARX model should produce residuals that resemble white noise.

Typical checks include:

- zero mean,
- autocorrelation close to zero (except at lag 0),
- no significant cross-correlation between residuals and past inputs,
- approximately constant variance.

These diagnostics assess whether the model has captured the available dynamics.

This procedure cleanly separates three distinct objectives:

- **Training set:** estimate the model parameters.
- **Rolling-origin validation (within the training set):** select the hyperparameters (n_a, n_b, n_k) while respecting temporal causality.
- **Independent test set:** obtain an unbiased estimate of how well the final model will perform on future, unseen data.

This separation minimizes information leakage, reduces the risk of overfitting, and reflects best practices in both system identification (e.g., Ljung, 1999) and modern time-series forecasting.

nonlinear regression with recurrent NARX Artifical Neural Network

Recurrent nonlinear auto-regressive model with exogenous inputs (NARX), often called dynamic NARX or MLP-NARX (Multi-Layer Perceptron) is used in nonlinear system identification.

It is depicted by:

$$y(t) = f\Big(u(t), u(t-1), \ldots, u(t-n_u),\ y(t-1), \ldots, y(t-n_y)\Big)$$

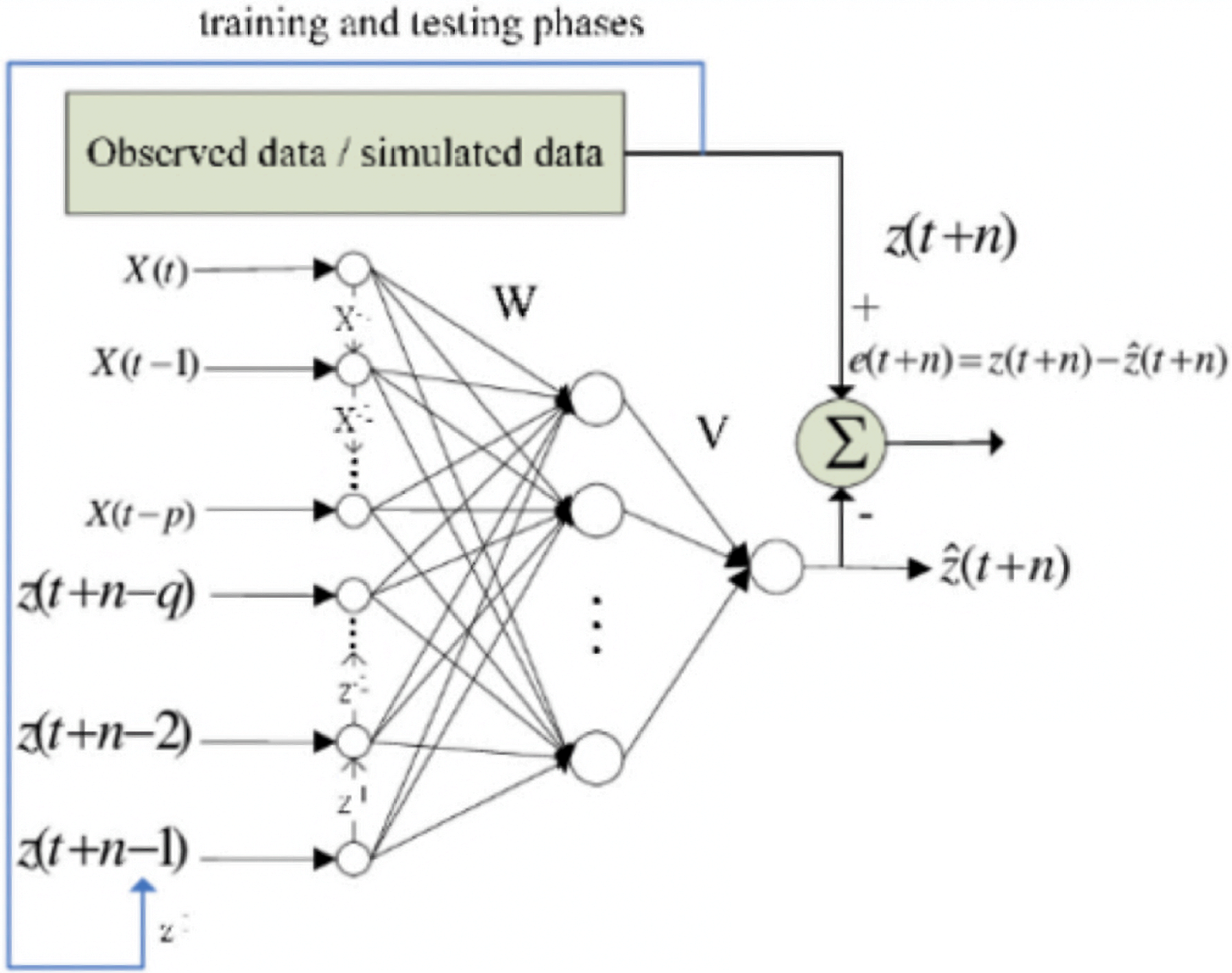
$u(.)$ and $y(.)$ can be vectors, for us, $u(.)$ will contain the explaining variables and $y(.)$, the indoor temperature and concentration in CO_2

The network reintroduces its previous outputs (predicted or measured) as inputs, giving it a dynamic behavior.

It is composed of:

1. A static nonlinear function $f(\cdot)$ implemented by a multilayer perceptron (MLP)
2. A feedback loop that feeds past outputs into the input at each time step

This combination creates a recurrent structure, even though the core neural network is feedforward.


training and testing phases
Observed data / simulated data
X(t)
X(t−1)
X(t−p)
z(t+n−q)
z(t+n−2)
z(t+n−1)
W
V
Σ
z(t+n)
+
e(t+n)=z(t+n)−ẑ(t+n)
-
ẑ(t+n)

At each time step t, $u(t), \ldots, u(t - n_u)$ and predicted past outputs $y(t-1), \ldots, y(t - n_y)$ are concatenated to form the NARX regressor vector:

$$\phi(t) = \begin{bmatrix} u(t) \\ u(t-1) \\ \vdots \\ u(t-n_u) \\ y(t-1) \\ y(t-2) \\ \vdots \\ y(t-n_y) \end{bmatrix}$$

This vector feeds a standard Multi-Layer Perceptron.

$$\hat{y}(t) = f_\theta(\phi(t))$$

with:

- 1 or 2 hidden layers (tanh, ReLU (ReLU(x)=max(0,x)))
- a linear output layer
- and parameters θ trained by gradient descent

It is recurrent because at time t, the input of the network depends on:

- $y(t-1)$, which itself depends on $y(t-2)$
- and so on...

Therefore, unfolding in time gives a computation graph with temporal recurrence:

$$\hat{y}(t) = f_\theta(u(t), \ldots, u(t - n_u), \hat{y}(t-1), \ldots, \hat{y}(t - n_y))$$

This is exactly a recurrent neural network, even though the recurrence is implemented externally through delayed inputs rather than through a hidden state.

Different training modes can be used. Firstly, the Series-Parallel (Teacher Forcing) training. During training:

$$\phi(t) = [u(t), \ldots, u(t - n_u),\ y_{\text{true}}(t - 1), \ldots]$$

This avoids instability and makes training easier.

It is stable, fast, convex-like behavior but it does not reflect rollout behavior

Second, the Parallel (Closed-Loop) Training. During training:

$$\phi(t) = [u(t), \ldots, u(t - n_u),\ \hat{y}(t-1), \ldots]$$

Network uses its own predictions as feedback.It is a faithful representation of operational behaviour but unstable gradients might occur leading to a harder training

It is used to:

1. train in series-parallel mode
2. fine-tune in parallel mode using BPTT

Recurrent NARX should be used when

- you want a nonlinear extension of ARX/ARMAX
- you know the input/output lag structure
- you want a simple dynamic nonlinear model without complex RNNs
- system dynamics are moderately nonlinear
- interpretability and stability matter (e.g. control applications)

It is extensively used in:

- system identification (Ljung, Suykens, Narendra)
- nonlinear model predictive control
- fault detection
- robotics and mechatronics
- building/HVAC models

Tuning a knowledge model

The process is similar to the one of the ARX model but it is simplified because the model structure is already known:

- take a dataset of measurements and split it into a training set and a validation set (50%/50% or 70%/30%)
- select the most impacting parameters for adjustment and determine a realistic range of variation for each parameter

- Adjust the parameters of the structure by minimizing the MSE of the training data set between the model output and the measurements, thanks to a least squares algorithm (a greedy algorithm searching around calculated parameter values use to operate better than an exploratory search)
- check if bounds are reached for some parameters: if yes, the parameter range should be increased or the parameter should be fixed because it is not identifiable.
- Note down the precision and check whether the precision is about the same with the validation data set
- If the precision is good enough, save the parameters

Determining most important parameters with Morris method

In order to determine the most important parameters, we are going to use the global sensitivity analysis method named: the Morris method, also known as the Elementary Effects Method. It is a fast global sensitivity analysis technique (Sobol approach is a more computationally demanding approach) used to evaluate the influence of parameter on the output of a model.

It is a screening Method that identifies key parameters (factors) that significantly affect the output and those that can be considered negligible. It’s computationally efficient and suitable for high-dimensional models with many parameters.

It can capture both nonlinear relationships between parameters and outputs and interactions among input variables.

parameters are varied one at a time while keeping others fixed to assess their individual effects on the output.

The steps in the Morris Method are:

- Input Space Sampling
  - Define the parameters ($X_1, X_2, \ldots, X_k$) and their ranges.
  - Divide each parameter range into discrete levels (e.g., 5–10 levels).
- Generation of Trajectories
  - Create a number of random paths (or trajectories) through the input space.
  - In each trajectory, vary one input variable at a time while keeping the others fixed.

- Compute Elementary Effects (EE):
  - For each input variable in a trajectory, calculate its elementary effect:

$$EE_i = \frac{f(X_1, \ldots, X_i + \Delta, \ldots, X_k) - f(X_1, \ldots, X_i, \ldots, X_k)}{\Delta}$$

    where:
    - $f(\cdot)$: the model output.
    - Δ: a small step size (change in parameters).

- Statistical Aggregation:
  - For each input variable, compute:
    - Mean (μ) of the absolute values of the EEs:
      - Indicates the overall influence of the variable on the output.
    - Standard Deviation (σ) of the EEs:
      - Captures the variability of the effect, indicating nonlinearities or interactions.
    - Mean of Signed Effects (μ^*):
      - Indicates whether the effect is consistently positive or negative.

Interpretation of the Results:

- High μ: The parameter has a strong overall effect on the model output.
- High σ: It indicates significant nonlinear effects or interactions with other variables.
- Low mu and σ: The parameter has negligible influence on the output.

Self-check questionnaire

12 questions — model fitting (ARX, NARX, knowledge models)

Choose the best answer each time.

(In class: discuss in pairs; answers on the following slides.)

Q1 — Why tune the H358 model?

The introduction explains that the physics-based office model may not match measurements mainly because:

- A. Approximations/neglected phenomena, uncertain parameters, and simplified structure (not only measurement noise)
- B. ARX models are illegal in buildings
- C. All conductivity values are known exactly from regulation
- D. State-space models cannot represent temperature

Q2 — ARX vs knowledge-based models

Compared with a knowledge-based model, a regular model (e.g. ARX) typically:

- A. Needs a large dataset and little domain expertise; KB models need expertise and less data
- B. Needs no data; KB models need infinite data only
- C. Is always nonlinear in its parameters
- D. Cannot be fitted by least squares

Q3 — Over-fitting

Over-fitting in the lecture means:

- A. The model fits noise and non-reproducible details, so it performs poorly on another dataset
- B. The training MSE is exactly zero in all cases
- C. The model order is always too low
- D. Validation MSE is always lower than training MSE

Q4 — ARX structure

An ARX model combines:

- A. Auto-regressive terms (past outputs $\hat{y}(k-i)$) and exogenous terms (delayed inputs $u_j(k-\cdot)$)
- B. Only future outputs $y(k+1)$
- C. Only steady-state U-values with no dynamics
- D. PMV and PPD as states

Q5 — Parameter estimation (ARX)

ARX parameters are attractive to estimate because the model is:

- A. Linear in the parameters → least squares / MSE minimization is straightforward
- B. Always convex in every physical parameter of the wall
- C. Identified only by hand without optimization
- D. Independent of input delays Δ and orders n, m_i

Q6 — Train / validation split

The recommended fitting workflow includes:

- A. Split data into training and validation sets; fit on training, check similar error on validation
- B. Fit and test on the same samples only
- C. Never change model order n or m_i
- D. Ignore residuals entirely

Q7 — Residual diagnostics

If residuals are not like white noise, the slides suggest:

- A. Structure may be wrong: missing input, or increase orders n / m_i (e.g. spike at lag l)
- B. The model is certainly perfect
- C. Only increase the offset term
- D. Delete the validation set

Q8 — Akaike criterion (AIC)

AIC is used to:

- A. Balance fit and complexity ($\mathrm{AIC} = 2k - 2\ln L$); lower AIC is preferred when comparing models on the same data
- B. Maximize the number of parameters always
- C. Replace MSE with COP
- D. Measure only wall thickness in metres

Q9 — NARX definition

A NARX model on the slides has the form:

- A. $y(t) = f\big(u(t), \ldots, u(t-n_u),\, y(t-1), \ldots, y(t-n_y)\big)$ with nonlinear f (often an MLP)
- B. $y(t) = a_1 y(t-1)$ only, with no inputs
- C. A static map $y = U_{\text{bat}} T_{\text{out}}$
- D. A heat-pump Carnot cycle only

Q10 — Series-parallel vs parallel training

Series-parallel (teacher forcing) training uses:

- A. Measured past outputs $y_{\text{true}}(t-1), \ldots$ in the regressor → easier/stable training, but rollout may differ
- B. Only predicted $\hat{y}(t-1)$ during training, never true past outputs
- C. No exogenous inputs $u(t)$
- D. Only validation data during gradient descent

Q11 — Knowledge-model tuning

Tuning a knowledge-based model differs from ARX mainly because:

- A. The structure is fixed; you select impactful parameters and ranges, then minimize MSE (optimization harder / non-convex)
- B. Structure and all parameters are re-discovered from AIC only
- C. No validation set is used
- D. Morris method replaces all measurements

Q12 — Morris method

The Morris (elementary effects) method is used to:

- A. Screen which parameters strongly affect outputs (μ, σ of elementary effects) before detailed tuning
- B. Compute HDD/CDD for a city
- C. Size the expansion valve
- D. Prove residuals are always Gaussian without plotting

Answer key (1/2)

Q	Answer	Short rationale
1	A	Approximations, uncertain params, simplified structure.
2	A	Data-driven vs expertise trade-off.
3	A	Fits noise; poor generalization.
4	A	AR lags + exogenous input lags.
5	A	Linear in params → least squares.
6	A	Train/val split; similar val error.

Answer key (2/2)

Q	Answer	Short rationale
7	A	Non-white residuals → structure/order issues.
8	A	AIC penalizes complexity; lower is better.
9	A	Nonlinear f on delayed u, y.
10	A	Teacher forcing with true past y.
11	A	Known structure; tune selected parameters.
12	A	Global sensitivity screening (Morris).

ARX model: a particular Artificial Neural Network

An ARX (Auto-Regressive with eXogenous input) model can be transformed into a state-space representation by rewriting its difference equation into a state-space format.

As seen before, a state-space model consists of:

$$x(k+1) = Ax(k) + Bu(k)$$

$$y(k) = Cx(k) + Du(k)$$

Let's choose the state vector. The state vector $x(k)$ typically contains

For simplicity, consider the following state vector:

$$x(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ \vdots \\ y(k-n) \\ u(k-1) \\ \vdots \\ u(k-m) \end{bmatrix}$$

Using the ARX model structure, the next state $X(k+1)$ is a linear combination of the current states $X(k)$ and the current inputs $U(k)$.

$$X(k+1) = AX(k) + BU(k)$$

The matrix A will capture the shift dynamics and the AR coefficients ($-a_i$), while B will include the exogenous input dynamics (b_i).

The output $Y(k)$ depends on the state vector $X(k)$ and the current input $U(k)$:

$$Y(k) = CX(k) + DU(k)$$

- C extracts the current output from the state vector and D accounts for direct input-output relationship (b_0).

Let's exemplify this by an example. For an ARX model of order $n = 2$, $m = 1$:

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) + b_0 u(k) + b_1 u(k-1)$$

To gather terms in k, let $z(k) = y(k) - b_0 u(k)$. A canonical state vector is given by:

$$X(k) = \begin{bmatrix} z(k) \\ z(k-1) \end{bmatrix}$$

The related input vector is:

$$U(k) = \begin{bmatrix} u(k) \\ u(k-1) \end{bmatrix}$$

The recurrent state equation is:

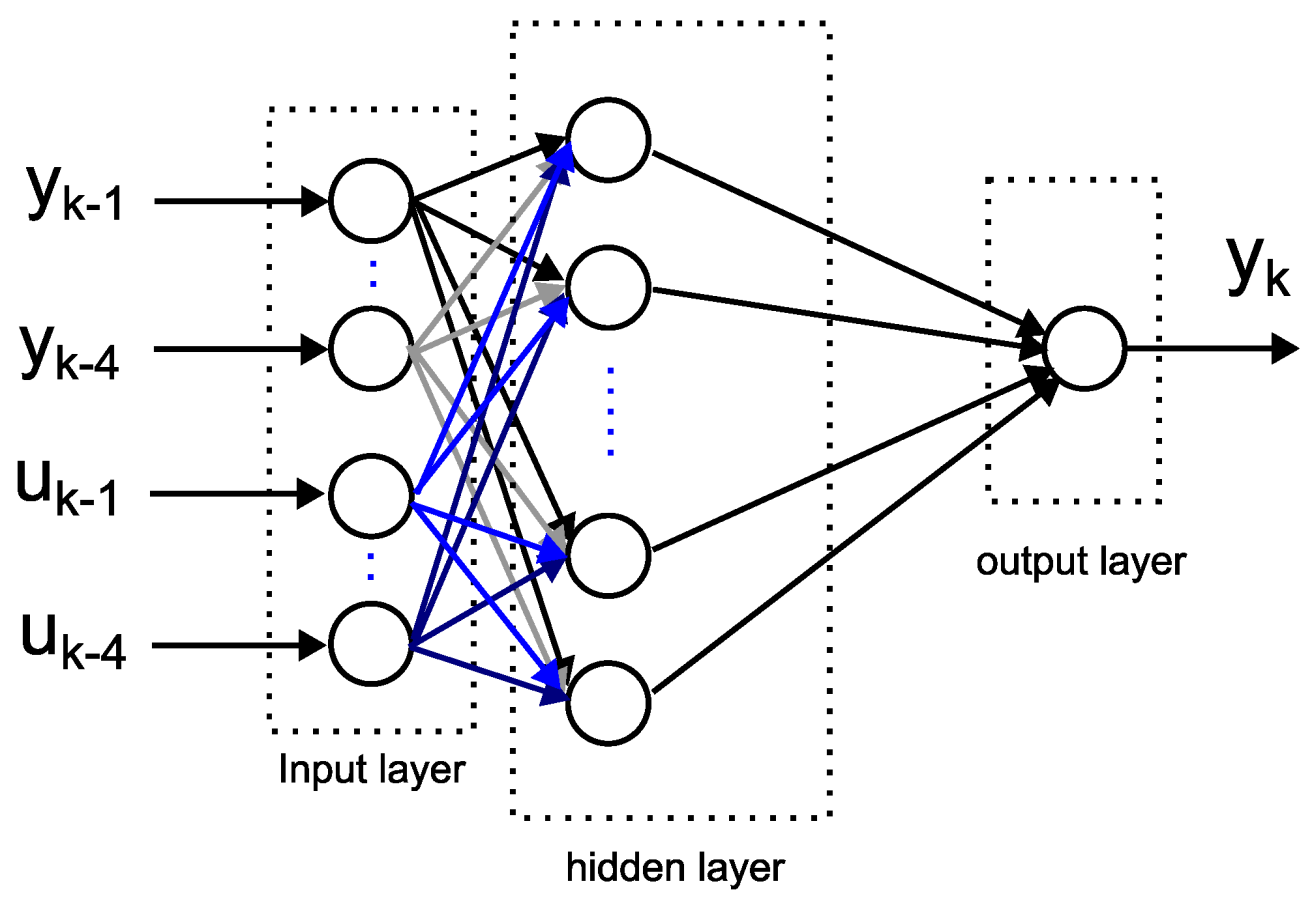
$$\begin{bmatrix} z(k+1) \\ z(k) \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} z(k) \\ z(k-1) \end{bmatrix} + \begin{bmatrix} b_1 - a_1 b_0 & a_2 b_0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u(k) \\ u(k-1) \end{bmatrix}$$

and the output equation is:

$$[y(k)] = [1 \quad 0] \begin{bmatrix} z(k) \\ z(k-1) \end{bmatrix} + [b_0 \quad 0] \begin{bmatrix} u(k) \\ u(k-1) \end{bmatrix}$$

Note that the state model is nonlinear in the parameters: the ARX model is more interesting because the parameters can be obtained from least squares inversion.

This approach generalizes to higher orders by expanding the state vector and appropriately defining A, B, C and D.



An ARX model corresponds to a particular **recurrent linear neural network (RLNN)**. Since the ARX model is fundamentally a linear system, an RLNN can represent it exactly by using a linear activation function and appropriately structuring its weights and biases to match the ARX model's dynamics.

An RLNN corresponding to the ARX model would involve a single layer of recurrence with linear activation functions. It maps to the ARX structure as follows:

Let's define a state vector $x(k)$ to hold the lagged values of $y(k)$ (output) and $u(k)$ (input):

$$x(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ \vdots \\ y(k-n) \\ u(k-1) \\ u(k-2) \\ \vdots \\ u(k-m) \end{bmatrix}$$

The recurrence of the RLNN is linear and can be written as:

$$x(k+1) = Ax(k) + Bu(k)$$

where A and B are weight matrices designed to encode the ARX dynamics.

The output $y(k)$ is obtained as a linear combination of the state:

$$y(k) = Cx(k) + Du(k)$$

where C and D are weight matrices representing the coefficients of the ARX model.

Let's now construct the RLNN that matches the ARX model.

Matrices are weight matrices:

- The A matrix captures the auto-regressive part (a_i coefficients).
- The B matrix captures the exogenous input dynamics (b_i coefficients).
- The C matrix extracts the lagged outputs to produce $y(k)$.
- The D matrix maps the direct influence of the current input $u(k)$.

For an ARX model of order $n = 2, m = 1$:

$$y(k) = -a_1 y(k-1) - a_2 y(k-2) + b_0 u(k) + b_1 u(k-1)$$

The RLNN parameters are:

$$x(k) = \begin{bmatrix} y(k-1) \\ y(k-2) \\ u(k-1) \end{bmatrix}$$

$$A = \begin{bmatrix} -a_1 & -a_2 & b_1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} b_0 \\ 0 \\ 0 \end{bmatrix}, \quad C = [1 \quad 0 \quad 0], \quad D = [b_0].$$

The RLNN uses a linear activation function (identity), which ensures the system behaves like an ARX model.

The RLNN exactly replicates the ARX dynamics when its weights are set to the ARX coefficients.

The recurrence is purely linear, corresponding to the lagged variables in the ARX model.

Remarks

- Translating ARX models into neural network frameworks for compatibility with machine learning systems.
- Combining with nonlinear extensions to create hybrid models (e.g., combining ARX and nonlinear RNNs).
- The RLNN is limited to linear dynamics, like the ARX model.
- It lacks flexibility for capturing nonlinear or unknown dynamics unless extended.