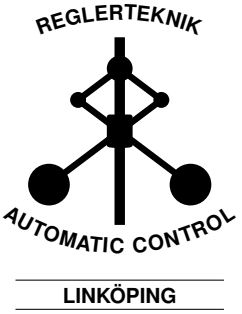
# Computer Exercises in System Identification
# Part 4

**This version: 2023-10-09**



# 1 Identification of Nonlinear Physical Models

All parametric models we have considered until now have been linear. Surprisingly often, these linear models are good descriptions of reality. One explanation of this is that systems, which in reality are nonlinear, often are well approximated by linear models when the inputs and outputs are within certain ranges (linearization).

However, sometimes linear models are insufficient. This might be because the model must be valid within such a large range that nonlinear phenomena such as input limits occur, or that the dynamics is nonlinear and *one* choice of linearization point is not enough for the required model range. The system can also be nonlinear even when within a small range of inputs. One example of the latter case is the static friction of a mass being pulled with force $F$ on a surface with friction. When $F$ is smaller than the static friction $F_s$ of the surface, the mass will remain stationary. However, when $F > F_s$, the mass will start moving, and is affected by a smaller kinetic friction $F_k$. If $F_s$ is large, the behavior of the system around the speed $v = 0$ cannot be well described using a linear model, even when we only want to model the system at low speed.

In this chapter we look at three types of nonlinear model structures:

- Nonlinear physically parameterized state-space models (`idnlgrey` in SITB)
- Nonlinear ARX models (`idnlarx` in SITB)
- Hammerstein and Wiener models (`idnlhw` in SITB)

We now give a brief introduction of these model structures and describe how to estimate them in SITB.

## 1.1 Nonlinear Physical State-space Models (`idnlgrey`)

Nonlinear state-space models in continuous time are described by

$$\dot{x}(t) = f(x(t), u(t), \theta), \tag{1a}$$
$$y(t) = h(x(t), u(t), \theta) + e(t), \tag{1b}$$

where $e(t)$ is white noise for simplicity. The prediction then is (cancel the noise!)

$$\dot{\hat{x}}(t) = f(\hat{x}(t), u(t), \theta), \tag{2a}$$
$$\hat{y}(t|\theta) = h(\hat{x}(t), u(t), \theta), \tag{2b}$$

which is a simulation of the model without noise (compare with the OE model predictor). This requires that the model is stable. The parameter $\theta$ can then be estimated by minimizing the prediction error using the loss function

$$V_N(\theta) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(y(t_i) - \hat{y}(t_i|\theta))^2$$

**Note**

- In most cases where the system is unstable, the predictor must be stable anyway, in order for the identification to be well-behaved. In the linear case this is easily handled by using a more complex noise model than the white noise model in OE (for example, ARMAX). For a nonlinear system, this is very complicated (nonlinear observers) and is not implemented in SITB.

  One way to remedy this is to include the controller used for the experimental data collection in the model, and identify the closed-loop system from reference to output instead.

In SITB, nonlinear state-space models are defined and estimated similarly to linear state-space models (`idgrey`) that we have considered previously. The model is defined as a function in a separate m file (or mex file) with the syntax:

```
function [dx,y] = MODFILENAME(t,x,u,p1,p2,...,pN,aux)
  dx = ...
  y = ...
end
```

(The code defining `dx` and `y` is omitted.) The main difference from before is that the return values of the function are `dx` ($\dot{x}(t)$ or $x(t+T)$) and `y` instead of the matrices of a linear state-space model. The arguments are the current time `t` (often unused), the state vector `x`, the input signal vector `u`, and all the parameters `p1,p2,...,pN`. Additional arguments can be included in `aux`. Thus, the function implements (2).

In order to identify the model, an `idnlgrey` model object must be created:

```
m0 = idnlgrey('MODFILENAME',[ny nu nx],par);
```

The second function argument `[ny nu nx]` defines the number of outputs, inputs, and states of the model, respectively. The third argument `par` is an object which contains information about the parameters `p1,p2,...,pN`. The simplest alternative is that `par` is a vector with initial guesses for the parameters `p1,p2,...,pN`, as before. More advanced alternatives are also available, including naming each parameter and defining minimum and maximum parameter values. Run `doc idnlgrey` in the MATLAB command prompt for more information.

In SITB, `idnlgrey` models can be estimated in the GUI by first importing the initial model using `Import models`, and then estimate the parameters using `Estimate` and `Refine Existing Models`. It can also be done through code:

```
m = pem(ze,m0);
```

The command computes an estimate of the parameters using a prediction error method on the estimation data set `ze`, where the model structure is defined by `m0`. The return value is the estimated model `m`. Note that SITB is object-oriented, meaning that the same functions (for example `pem` and `plot`) can be applied to many objects, and may give different results depending on the object.

The estimated model `m` can now be analyzed, either in the GUI or it can be done in code. For example the `Model output` plot can be created through:

```
figure; compare(zv,m);
```

where `zv` is the validation data. A summary of the estimated model (including the estimated parameter values) can be shown by running

```
present(m);
```

## 1.2 Nonlinear ARX Models (`idnlarx`)

A nonlinear ARX model differs from a linear ARX model by allowing the predictor $\hat{y}(t|\theta)$ to be a *nonlinear function of old input and output signals*:

$$\begin{aligned}\hat{y}(t|\theta) &= g(\varphi(t), \theta)\\ \varphi(t) &= (y(t-1), \ldots, y(t-n), u(t-1), \ldots, u(t-m))\end{aligned}$$

The nonlinear function $g(\varphi(t), \theta)$ can be defined in many ways, see [1, Section 15.1]. One of the most common ways is to use a neural network with a sigmoidal basis function [1, Eqs. (15.13) and (15.15)].

In the SITB GUI, the models are estimated by clicking `Estimate` and then `Nonlinear ARX models`. This yields the dialogue shown in Figure 1. The

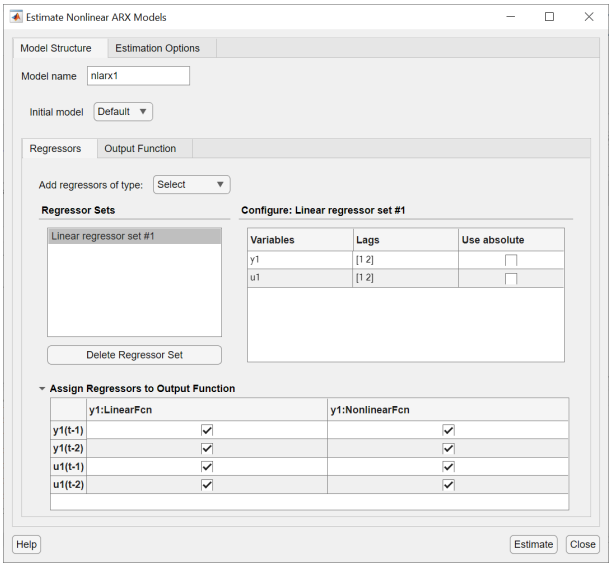



Figure 1: Prompt for estimation of nonlinear ARX-models.

model order is chosen under `Regressors`, and the nonlinearity can be set in `Output Funtion`.

Custom regressors can be added by clicking `Add regressor of type:`, then choosing `Custom`. These regressors can for instance be formed from physical reasoning.

When a nonlinear ARX model has been estimated, the option `Nonlinear ARX` appears in `Model views` in the main GUI window. This plot shows the nonlinear function $g$. The function surface may be viewed from other angles by selecting `Style` and then `Rotate 3D`.

## 1.3 Hammerstein and Wiener models (`idnlhw`)

Hammerstein and Wiener models are nonlinear model structures, which are often perceived as somewhat simpler than nonlinear ARX models. They are gotten by *combining a linear dynamical system* $G(q)$ *and a static nonlinearity* $f(\cdot)$:

- Hammerstein model (nonlinearity at the input):
$$y(t) = G(q)z(t), \quad z(t) = f(u(t))$$
- Wiener model (nonlinearity at the output):
$$y(t) = f(z(t)), \quad z(t) = G(q)u(t)$$

The structures can be combined, which yields a nonlinearity at both the input and output, and a linear dynamical system in between; this is called a Hammerstein-Wiener model. The models occur naturally when the input behaves nonlinearly (like valves or saturations) or if the measurement is nonlinear.

In the GUI, these models are estimated by clicking `Estimate` and then `Hammerstein-Wiener Models`. This yields the dialogue shown in Figure 2. The model order of the linear system is set in the tab `Linear Block` and the nonlinearities in the input and output are set in the tab `Input Nonlinearity` and `Output Nonlinearity`, respectively.

When a nonlinear model has been estimated, the option `Hamm-Wiener` appears in `Model views` in the main GUI windows. In this plot, the nonlinear functions and properties of the linear system (Bode plot, step response, ...) can be studied.

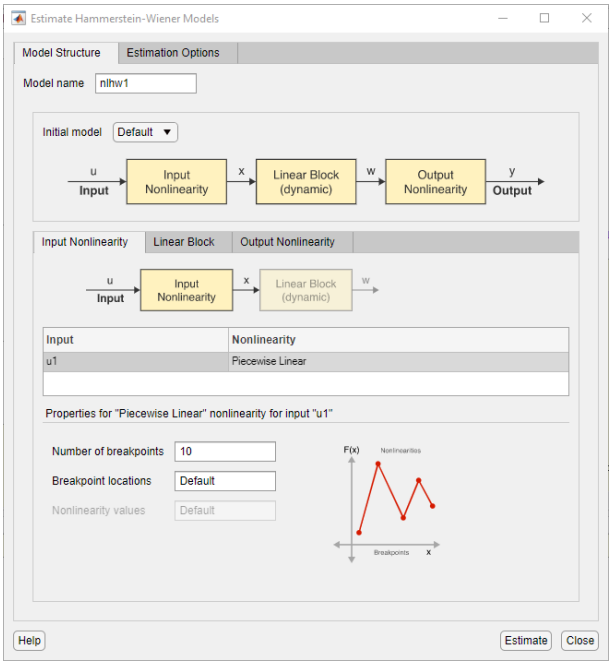



Figure 2: Prompt for estimation of Hammerstein-Wiener models.

## 1.4 Exercises

1. Consider a tank system, where the inflow $q_{\text{in}}(t)$ is generated by an electric pump controlled by the voltage $u(t)$. The measured output signal is the water level $h(t)$ of the tank, and the maximum water level $H$ is 35 cm. An overview of the tank system can be seen in Figure 3.

   Using Bernoulli's law, a simple physical model of the water tank can be derived as

   $$\frac{d}{dt}h(t) = \frac{k}{A}u(t) - \frac{a}{A}\sqrt{2gh(t)} \tag{3}$$

   The cross-section area $A$ of the tank is assumed to be known, but the cross-section area $a$ of the outlet pipe cannot be determined with sufficient precision using a measuring tape. For simplicity, we have assumed that the pump has a linear behavior $q_{\text{in}}(t) = k \cdot u(t)$, where $k$ is an unknown gain.

   In `tankdata.mat` (enter `load tankdata`), data from the tank system is available. The file contains two data sets, `ze` and `zv`, each of which is 17 minutes long and contains 1000 samples with a sample time of 1 second.

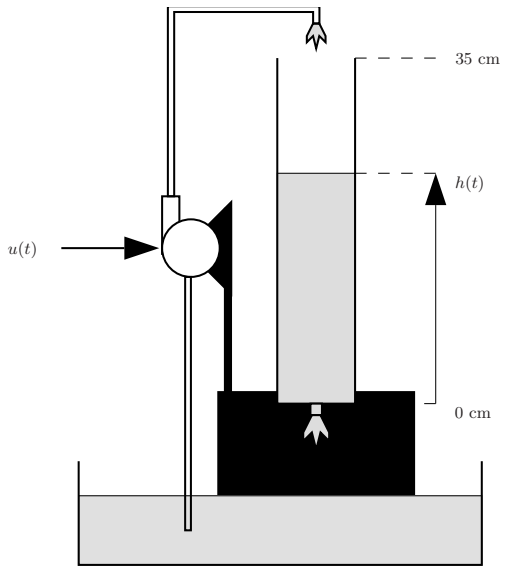



Figure 3: A tank system.

(a) Import and inspect the data sets. Do not remove the means of the signals, because we later wish to estimate a model describing the static levels accurately.

(b) Use the estimation data set `ze` and estimate a few linear discrete-time models (for example ARX and state-space models) for the tank system.

Simulate the models with the validation data set `zv`. Do the models always yield physically reasonable water levels (the water level should always be between 0 and 35 cm)?

(c) Estimate the unknown parameters $k$ and $a$ in the model (3) with $A = 20\,\mathrm{cm}^2$ by first writing an idnlgrey file like in the previous example. As initial values for $k$ and $a$, pick for instance 10.

Simulate and compare the model with the result in (b). What is the reason for the difference?

(d) By looking at the inputs and outputs, it can be seen that the pump appears to have a dead zone (nothing happens when $u$ is small). Thus, a better pump model than (c) could be $q_{\mathrm{in}}(t) = k{\cdot}(u(t){-}u_0)$, which yields

$$\frac{d}{dt}h(t) = \frac{k}{A}(u(t) - u_0) - \frac{a}{A}\sqrt{2gh(t)}$$

where $u_0$ is another parameter that must be estimated (a reasonable initial guess is 0). Write a new m file (so that you still have

the first tank model to compare with) and estimate this extended model. Compare with the earlier models.

(e) It appears that the dead zone in the input is the dominant nonlinearity. Thus, also try estimating a Hammerstein model, where the linear model has order $n_b = n_f = n_k = 1$ (we get these model orders if (3) is linearized and discretized). The static nonlinearity in the input can be set to `Dead Zone` or `Piecewise Linear`. Set the output nonlinearity to `None`.

Simulate and compare the result to the earlier models. Also try an OE model of the same model order in order to see the effect of the nonlinearity in the Hammerstein model.

You can also plot the estimated nonlinearity by clicking `Hamm-Wiener` in the GUI.

(f) Let us now apply a nonlinear ARX model to the system. If (3) is discretized using Euler forward, we get:

$$h(t) = h(t-1) + T\Big(\frac{k}{A}u(t-1) - \frac{a}{A}\sqrt{2gh(t-1)}\Big)$$

and with $y(t) = h(t)$ we have the model

$$\hat{y}(t|\theta) = g(y(t-1), u(t-1), \theta)$$

Estimate a nonlinear ARX model of this order.

Simulate the estimated model and compare the result to the earlier models. Also plot the estimated model by clicking `Nonlinear ARX` in the GUI.

(g) The nonlinear ARX model in (f) can be further improved by adding the information about the nonlinearity $\sqrt{y(t-1)}$. Custom regressors can be added by clicking `Add regressor of type:`, and then choosing `Custom`. Depending on the choice of these regressors, we can get closer to a physical model. Add the regressor `sqrt(abs(y1(t-1)))` and compare with the other models.

Also try estimating an ARX model where $g$ is linear by choosing `Nonlinear Function: None` in `Output Function`.

2. Solve exercise 10.7 from the exercise book.

3. Solve exercise 10.5 from the exercise book.

# Solutions with Discussion

The solutions below are not necessarily the only correct solution. Depending on choices during data processing and estimation, your results may be perfectly correct even if they do not exactly agree with the discussion below.

1. (a) Load and inspect the data:

   ```
   load tankdata
   plot(ze,zv);
   ```

   The data sets appear similar; the validation data has somewhat lower average water level than the estimation data (13 vs. 17 cm). Also note that the input has an average higher than 0.

   (b) We estimate ARX and state-space models using `Order Selection`. The ARX model of orders `[5 7 1]` and the state-space model of order 3 have a model fit to validation data of 81% and 86% respectively. With an OE `[5 5 1]` model, a fit of 87% is achieved. This is hard to beat with linear models.

   The code for estimating the models in code is (try right-clicking a model created in the GUI):

   ```
   marx = arx(ze,[5 7 1]);
   mss = n4sid(ze,3,'N4W','CVA','N4H',[15  39  39]);
   moe = oe(ze,[5 5 1]);
   figure; compare(zv,marx,mss,moe);
   ```

   These models do not appear to fare well at modelling low water levels. They sometimes give water levels below 0, which is not physically reasonable. This is an indication that the tank system contains nonlinear phenomena. This can also be seen by running:

   ```
   advice(ze,'nonlinear');
   ```

   (c) Identification through `idnlgrey` is done by first defining the model structure in the m file `tank.m`:

   ```
   function [dx,y]=tank(t,x,u,k,a,aux)
   A=20; g=9.81;
   y=x;
   dx=k/A*u-a/A*sqrt(2*g*x);
   end
   ```

To estimate the parameters of the continuous-time nonlinear model, run the following code in a MATLAB script or in the command prompt:

```
mgr1=idnlgrey('tank',[1 1 1],[10 10]);
mgr1=pem(ze,mgr1);
```

The model parameters and a simulation of the model on validation data is shown by:

```
present(mgr1);
figure; compare(zv,mgr1)
```

The estimated parameter values are $k = 20.4$, $a = 6.1$. The continuous-time model is simple (only two estimated parameters) and always yields water levels between 0 and 35 cm, but the validation data model fit is 41.2% (and not more than 50% for estimation data).

The reason for the poor model fit must be that the model structure is not accurate/good enough. The linear models are more flexible and can fit the data better. Some of the assumptions we made when forming the physical model were wrong. What assumptions? Look at the input and output signals more closely. The input varies between 4 and 8 V, and when the input is at 4 V, the output is almost 0. Thus, the main candidate for the error is the assumption $q_{\text{in}}(t) = k \cdot u(t)$ because there is a dead zone in the input where $q_{\text{in}}(t)$ is 0 even when $u > 0$ (maybe up to 3–4 V).

(d) Now, write a new m file, `tank2.m`:

```
function [dx,y]=tank2(t,x,u,k,a,u0,aux)
  A=20; g=9.81;
  y=x;
  dx=k/A*(u-u0)-a/A*sqrt(2*g*x);
end
```

Estimate the model and display the result through:

```
mgr2=idnlgrey('tank2',[1 1 1],[10 10 0]);
mgr2=pem(ze,mgr2);
present(mgr2);
figure; compare(zv,mgr1,mgr2)
```

The model now gives a much better result. The parameter values are $k = 26.8, a = 3.78, u_0 = 3.13$ and the fit is 89.7%. This is a

better fit than the best linear models and in particular the water levels are mostly physically reasonable. At one point in time, the level $h > 35$ cm, but otherwise the levels are within the physical limits 0–35 cm.

(e) in the GUI, the models are estimated by clicking `Estimate` and then choose `Hammerstein-Wiener models`. The order of the linear system is chosen in `Linear Block` and the nonlinearities are chosen in `Input Nonlinearity` and `Output Nonlinearity`. The models can also be estimated in code:

```
mhw1=nlhw(ze,[1 1 1],idDeadZone,idUnitGain);
mhw2=nlhw(ze,[1 1 1],idPiecewiseLinear,idUnitGain);
oe111 = oe(ze,[1 1 1]);
figure; compare(zv,mhw1,mhw2,oe111);
figure; plot(mhw1,mhw2);
```

With this nonlinearity on the input, we get a model fit to validation data of 89.3% (dead zone) or 88.6% (piecewise linear with 10 terms). Both models keep the tank level within 0–35 cm. A simple linear OE model has a model fit of 16.8%. The high model orders in (b) thus appear to be used to model the nonlinearity in the input.

When the nonlinearity is plotted, the dead zone is at roughly 4 V. The differing slopes of the curves are compensated by the gain of the linear system; thus, we should focus on the shape of the curve rather than the slope.

(f) ARX models are estimated in the GUI by clicking `Estimate` and then `Nonlinear ARX models`. The model order is chosen in `Regressors` and the nonlinearity is specified in `Output Function`. Choose, for example, `Sigmoid Network` (other choices give similar results). The model can also be estimated in code:

```
nlarx1 = nlarx(ze,[1 1 1],'idSigmoidNetwork');
figure; compare(zv,nlarx1);
figure; plot(nlarx1);
```

We get a model fit to validation data of 92.4%, but $h > 35$ at one point in time for the validation data (like the second nonlinear grey-box model `mgr2`).

When the nonlinearity is plotted, we can see a flat function surface shifted vertically in order to handle the input dead zone. Thus, there doesn't appear to be any other notable nonlinearity.

(g) By adding the custom regressor, we get a model with fit 93.8%. This is the best model thus far, and the levels stay within the limits 0–35 cm. The models can also be estimated in code:

```
C = 'sqrt(abs(y1(t-1)))';
nlarx2 = nlarx(ze,[1 1 1],'idSigmoidNetwork','Customreg',C);
nlarx3 = nlarx(ze,[1 1 1],'idLinear','Customreg',C);
figure; compare(zv,nlarx2,nlarx3);
```

We also try estimating a linear regression `nlarx3`, which is much less complex than `nlarx2`. This model has 4 parameters:

$$\hat{y}(t|\theta) = \theta_0 + \theta_1 y(t-1) + \theta_2 u(t-1) + \theta_3 \sqrt{|y(t-1)|},$$

where the constant term $\theta_0$ is automatically added for `nlarx`. This term makes a notable difference because it corrects for the dead zone. With this model, we get a model fit of 91.9%. The model is slightly worse at high tank levels in comparison to `nlarx2`.

2. See textbook, use `'idSigmoidNetwork'` instead of `'sigmoid'` and `'idPiecewiseLinear'` instead of `'pwlinear'`.

3. See textbook, use `'idLinear'` instead of `'linear'`.

# References


[1] L. Ljung, T. Glad and A. Hansson. *Modeling and identification of Dynamic Systems*. Studentlitteratur, 2021.