System Identification and Modeling Exercises*

Philippe Dreesen Oliver Lauwers Bob Vergauwen

philippe.dreesen@vub.ac.be
oliver.lauwers@kuleuven.be
bob.vergauwen@kuleuven.be

Academic year 2016–2017

1 Introduction and Practicalities

This series of exercises was designed to train the theoretical knowledge gathered during the course on System Identification. The exercises will enable you to try out the theory on a computer and some of the exercises will confront you with real-life problems. The idea behind these exercises is applying knowledge instead of memorizing knowledge. The exam will take this strategy to heart and emphasize the insight you gathered during the exercises.

The course consists of two parts, and is taught by two professors. The exercise sessions consist of two parts as well. Each part focusses on different aspects of system identification problems. In the following table, an overview of the exercise sessions is given, together with the corresponding parts from the course lectures involved.

Part	Exercise Description	Session	Report
Part I	Noise on Input and Output	Session 1	Yes
	Model Selection using AIC	Session 2	Yes
Part II	Getting Acquainted with the SYSID Toolbox	Session 3	No
	A Real-life Identification Problem	Sessions 4+5	Yes

Since this is a different form of exercises, there is also a different form of examination. You should **write down your results in two reports**, one for each part. During the exam, you will defend your results and motivate the design decisions you made. You will be judged on comprehension, clarity and also inventiveness, since these are all qualities that are highly appreciated in the industry.

We encourage you to form groups of two people, so you can discuss the problems and thus enlarge your insight in the material. The reports can also be written in groups of two people. If you really prefer to work alone, this is also possible.

*Acknowledgments: Sincerest thanks to Antoine Vandermeersch, Niels Haverbeke, Stan De Schepper and Peter Van Overschee, among others, for earlier versions this booklet and (MATLAB) code.

It should be emphasized that the idea behind the reports is not to let you write a first thesis: the reports should be concise. Conciseness is mostly in your own benefit since it will limit the time you spend on writing the report. The reports should contain explanatory figures and their discussion (figures without explanation will be ignored). Additional practical information concerning the reports is summarized at the end of each part.

Keep in mind that the reports are corrected by **two professors**, so you should write and hand in **two separate reports**!

You can work on the computers at ESAT whenever you have the time. If you want to work at night or during the weekends, you should ask at the departmental secretary for an entry badge. If you have any questions, there are different possibilities:

- ask one of the professors during or after one of the lectures,
- come to one of the exercise sessions,
- send an e-mail to one of the teaching assistants.

You should hand the reports to Mrs. Ida Tassens (room ESAT B00.10). The dates that the reports are due and the exam dates will be communicated during during the course lectures and via toledo. A schedule indicating the exact time and place of the oral discussions (exam) will be distributed in due time via toledo and email.

2 Exercises Part Prof. J. Schoukens / P. Dreesen

2.1 Noise on Input and Output

2.1.1 Introduction

In the course we have seen that the presence of disturbing noise on the input measurements creates a systematic error in the least squares (LS) estimates. In this exercise more advanced identification methods are illustrated that can deal with this situation. Two methods are studied, the first is called the instrumental variables method (IV), the second is the errors-in-variables (EIV) method.

The major advantage of the IV-methods is its simplicity. No additional information is required from the user. The disadvantage is that this method does not always perform well. Both situations are illustrated in this exercise. The EIV performs well in many cases, but in general additional information of the user is required. The covariance matrix of the input-output noise should be known.

All methods are illustrated on the example of an electrical resistor with measured current and voltage $i(k), u(k), k = 1, 2, \ldots, N$. Consider a set of repeated measurements

$$u_0(k) = R_0\, i_0(k), \quad k = 1, 2, \ldots, N, \tag{1}$$

with R_0 the (unknown) exact resistance value of an electrical resistor, say, $R_0 = 1000\Omega$, and u_0 and i_0 the exact values of the voltage and the current, respectively. The task is to estimate

the value of R_0 from noisy measurements of voltage and current. Both measurements of u_0 and i_0 are disturbed by mutually uncorrelated Gaussian noise (n_i and n_u, respectively):

$$\begin{aligned} i(k) &= i_0(k) + n_i(k) \\ u(k) &= u_0(k) + n_u(k). \end{aligned} \tag{2}$$

The least squares, instrumental variables and the errors-in-variables estimators are given below:

- The LS estimate of the resistance is given by

$$\hat{R}_{\mathrm{LS}} = \frac{\sum_{k=1}^{N} u(k)i(k)}{\sum_{k=1}^{N} i(k)^2}. \tag{3}$$

- The IV estimate is given by

$$\hat{R}_{\mathrm{IV}} = \frac{\sum_{k=1}^{N} u(k)i(k+s)}{\sum_{k=1}^{N} i(k)i(k+s)}, \tag{4}$$

  where s is a user specified shift variable. Note that the IV estimator equals the LS estimator for $s = 0$.

- The EIV estimate is given by

$$\hat{R}_{\mathrm{EIV}} = \frac{\frac{\sum u(k)^2}{\sigma_{n_u}^2} - \frac{\sum i(k)^2}{\sigma_{n_i}^2} + \sqrt{\left(\frac{\sum u(k)^2}{\sigma_{n_u}^2} - \frac{\sum i(k)^2}{\sigma_{n_i}^2}\right)^2 + 4\frac{(\sum u(k)i(k))^2}{\sigma_{n_u}^2 \sigma_{n_i}^2}}}{2\frac{\sum u(k)i(k)}{\sigma_{n_u}^2}}, \tag{5}$$

  where $\sigma_{n_u}^2$ and $\sigma_{n_i}^2$ the variance of the voltage and current noise, respectively; the covariance is assumed to be zero in this expression: $\sigma_{ui}^2 = 0$.

2.1.2 The Instrumental Variables method

Generate the current i_0 from a Gaussian white noise source e_1, filtered by a first order Butterworth filter with cut-off frequency f_{gen}, using the MATLAB instructions

```
>> e1=randn(N,1);
>> [bgen,agen] = butter(1,fgen);
>> i0 = filter(bgen,agen,e1);
```

Generate the measured current and voltage

$$\begin{aligned} i(k) &= i_0(k) + n_i(k) \\ u(k) &= u_0(k) + n_u(k), \end{aligned} \tag{6}$$

where n_u is zero mean white Gaussian noise with standard deviation σ_{n_u} and n_i is white Gaussian noise filtered by a second order Butterworth filter with cut-off frequency f_{noise}, scaled to σ_{n_i}:

```
>> e2=randn(N,1);
>> [bnoise,anoise] = butter(2,fnoise);
>> ni = filter(bnoise,anoise,e2);
```

After filtering, all generated signals are scaled such that they match their requested standard deviations.

Experiments:

- Experiment 1: Generate three sets of 1000 experiments with $N = 5000$ measurements each, and the following parameter settings:

$$\begin{array}{lll} R_0 = 1000, & f_{\text{gen}} = 0.1, & f_{\text{noise}} = \{0.999, 0.95, 0.6\}, \\ \sigma_{i_0} = 0.1, & \sigma_{n_i} = 0.1, & \sigma_{n_u} = 1. \end{array} \tag{7}$$

  Recall that the filtered signals are to be rescaled *after* filtering, not before filtering!

- Process these measurements with the LS-estimator, and with the IV-estimator with the shift parameter $s = 1$.

- Experiment 2: Generate a set of 1000 experiments with $N = 5000$ measurements each, and the following parameter settings:

$$\begin{array}{lll} f_{\text{gen}} = 0.1, & f_{\text{noise}} = 0.6, & \\ \sigma_{i_0} = 0.1, & \sigma_{n_i} = 0.1, & \sigma_{n_u} = 1. \end{array} \tag{8}$$

- Process these measurements with the LS-estimator, and with the IV-estimator with the shift parameter values $s = \{1, 2, 5\}$.

- Plot for both experiments:
  - the probability density functions (PDF) of $\hat{R}_{\text{LS}}$ and $\hat{R}_{\text{IV}}$,
  - the autocorrelation functions of i_0 and n_i (hint: MATLAB instruction `xcorr`),
  - the frequency response functions (FRF) of the generator and the noise filter.

Observations and questions:
The results are shown in Figure 1 and Figure 2 below.

In Figure 1, the results are shown for a fixed generator filter and a varying noise filter. The shift parameter for the IV is kept constant to 1. It is clearly seen that the LS-estimates are strongly biased.

- Try to explain what causes the bias, and find an expression of the bias as a function of $\sigma^2_{n_i}$ and $\sigma^2_{i_0}$.

For the IV-results, the situation is more complicated.

- Is there a bias present for the white noise case? Why (not)?
- Is there a bias present for the case the output noise is filtered?
- Find an expression of the bias using the auto-correlation functions of the noise and the current $R_{n_i n_i}(s)$ and $R_{i_0 i_0}(s)$.

Similar observations can also be made for Figure 2. In this figure, the shift parameter s is changed while the filters are kept constant.

- What happens to the bias as the shift variable s increases (while the filters are kept constant)?
- Can you relate this to the auto-correlation functions of the noise and the current $R_{n_i n_i}(s)$ and $R_{i_0 i_0}$?
- Can you explain the sign of the bias (over-estimate vs. under-estimate)?
- What happens to the dispersion (variance) of the estimator, and why?
- Complete the following statement: The IV-method works well if the bandwidth of the generator signal is much smaller/larger than that of the noise disturbances. ($f_{\text{gen}} \ll f_{\text{noise}}$ or $f_{\text{gen}} \gg f_{\text{noise}}$)?.

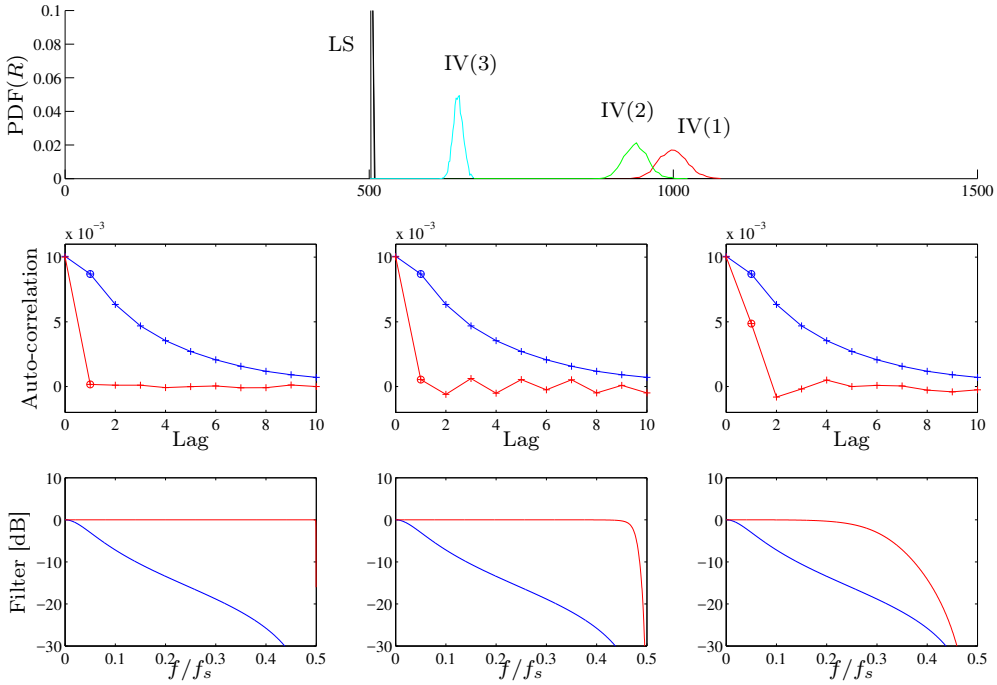

Figure 1: Study of the LS and IV estimates for a varying noise filter bandwidth and fixed shift parameter $s = 1$. Top: the LS (black) and IV estimates. IV(1) (red), IV(2) (green) and IV(3) (blue) correspond to the noise filters with cut-off frequencies 0.999, 0.95 and 0.6 respectively. Middle: the auto-correlation of i_0 (blue) and n_i (red) for the different noise filters. Bottom: filter characteristics of i_0 (blue) and of n_i (red).

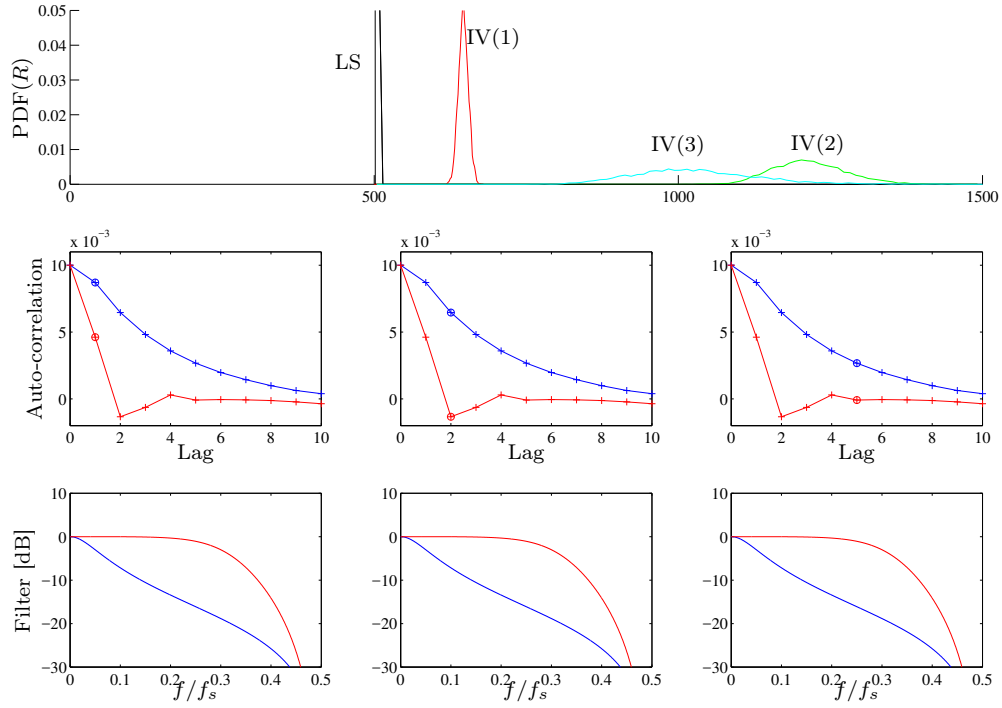

Figure 2: Study of the LS and IV estimates for a fixed noise filter bandwidth and varying shift parameter $s = \{1, 2, 5\}$. Top: the LS (black) and IV estimates. IV(1) (red), IV(2) (green) and IV(3) (blue) correspond to a shift of 1, 2 and 5 respectively. Middle: the auto-correlation of i_0 (blue) and n_i (red). Bottom: filter characteristics of i_0 (blue) and of n_i (red).

2.1.3 The Errors-In-Variables method

In the next part of this exercise, the EIV method is used as an alternative for the IV method to reduce/eliminate the bias of the LS estimate for the resistor example. This time no constraint is put on the power spectra (bandwidth) of the excitation and disturbing noise, but instead, the variance of the input and output disturbing noise should be priorly known.

Experiments:

- Generate the current $i_0(k)$ from a Gaussian white noise source with standard deviation σ_{i_0} (not filtered this time) and generate the measured current and voltage signals as before:

$$\begin{aligned} i(k) &= i_0(k) + n_i(k) \\ u(k) &= u_0(k) + n_u(k), \end{aligned} \tag{9}$$

  with $n_u(k)$ and $n_i(k)$ zero mean white Gaussian noise with standard deviation σ_{n_u} and σ_{n_i} respectively (again not filtered this time).

- Experiment: Generate a set of 1000 experiments with $N = 5000$ measurements each, and the following parameter settings:

$$R_0 = 1000, \quad \sigma_{i_0} = 0.01, \quad \sigma_{n_i} = 0.001, \quad \sigma_{n_u} = 1. \tag{10}$$

- Process these measurements with the LS-estimator, and with the EIV-estimator.
- Plot the histograms for $\hat{R}_{\text{LS}}$ and $\hat{R}_{\text{EIV}}$.

Observations and questions:
The results are shown in Figure 3.

- Is the LS estimator biased?
- How big is the relative bias?
- Try to express this as a function of the given parameter settings.
- Is the EIV estimator biased?
- Would the IV estimator work here? Why (not)?

2.2 Report

For this part you should hand in a report covering the two exercises. The report should be at most **10 pages** long.

2.3 Model selection using AIC

2.3.1 Introduction

The goal of this exercise is to study the optimal model order selection for a simple modeling problem. Too simple models will fail to capture all important aspects of the output, and this

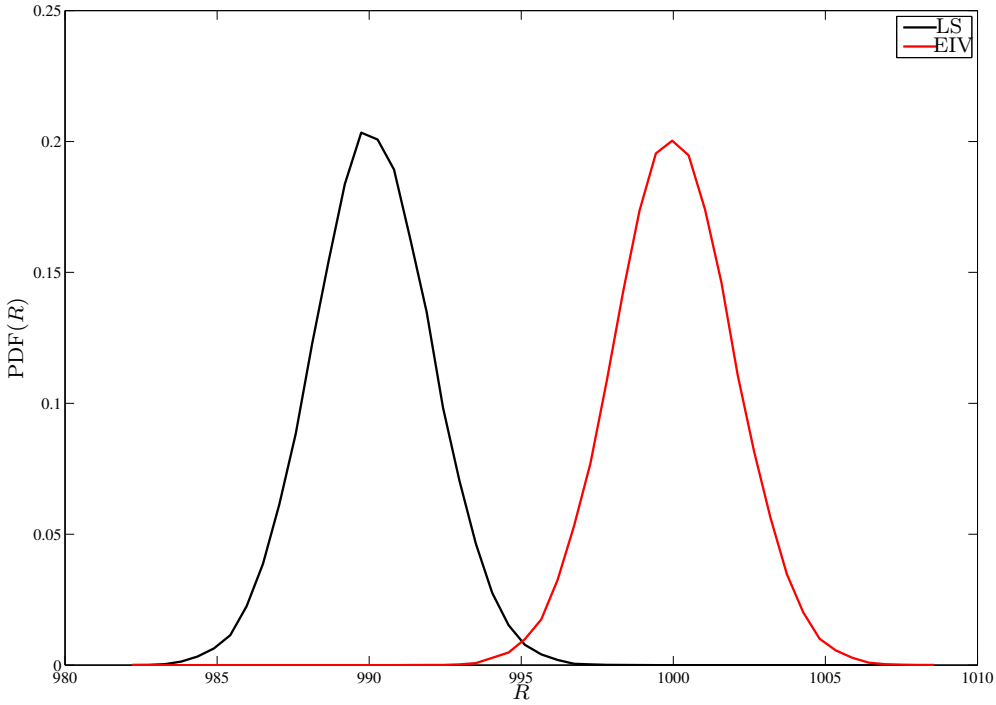

Figure 3: Comparison of the PDF of the LS (black) and EIV (red) estimate, calculated with prior known variances.

will result in too large errors in most cases. Too complex models use too many parameters and may result in a poor behavior of the modeled output because the model becomes too sensitive to the noise. Hence we need a tool that helps us to select the optimal complexity that balances the model errors against the sensitivity to the noise disturbances. It is clear that this choice will depend on the quality of the data. All these aspects are illustrated in this exercise where we propose the Akaike information criterion (AIC) as a tool for model selection.

Consider a single input single output (SISO) linear dynamic system, excited with an input $u_0(k)$ and output $y_0(k) = g_0(k) * u_0(k)$. The system has (in general) an impulse response $g_0(k)$ which is infinitely long (infinite impulse response or IIR-system). However, for a stable system $g_0(k)$ decays exponentially fast to zero, so that the IIR system can be approximated sufficiently well by a system with a finite length impulse response $g(k), k = 0, 1, \ldots, I$ (finite impulse response or FIR system):

$$\hat{y}(k) = \sum_{t=0}^{I} \hat{g}(t) u_0(k-t), \quad \text{with} \quad u_0(k) = 0 \text{ for } k < 0. \tag{11}$$

For $k > I$, the remaining contribution $g(k)$ can be considered negligible. The choice of I will depend not only on g_0, but also on the SNR (signal-to-noise ratio) of the measurements.

In this exercise, it is assumed that the system is initially in rest ($u_0(k) = 0$ for $k < 0$). If this is not the case, transient errors will appear, but these disappear in this model for $t > I$ (why?). The data acquired from the system are $Z^N = \{(u_0(k), y(k))\}_{k=1}^N$, where u_0 represents the undisturbed input signal and y represents the measurement of output data,

$$y(k) = y_0(k) + n_y(k), \tag{12}$$

that is, the true output $y_0(k)$ is disturbed with i.i.d. noise with zero mean and variance σ_{n_y}. The model parameters θ are estimated by minimizing the least squares (LS) cost function

$$V_{\text{LS}}(\theta, Z^N) = \frac{1}{N} \sum_{k=1}^{N} (y(k) - \hat{y}(k))^2, \tag{13}$$

with $y_0(k) = g_0(k) * u_0(k)$ and $\hat{y}(k) = g(k) * u_0(k)$. Notice that for the FIR model the parameters are simply the FIR coefficients $g(0), \ldots, g(I)$! Moreover, this model is linear-in-the-parameters: the parameters θ enter the equation through $\hat{y}(k)$, which means that the linear least-squares method can be used (cf. course notes Section 2.2.2 pp. 93). Verify the fact that the FIR model is linear-in-the-parameters for a very simple FIR system, e.g., by setting $I = 3$ and writing out $\hat{y}(k) = g(k) * u(k)$ for $k = 0, 1, 2, 3$.

In order to find the *best* model, a trade-off should be made between model errors and (output) noise errors using a modified cost function that accounts for the model complexity (number of parameters). Here we propose to use the Akaike Information Criterion (AIC)

$$V_{\text{AIC}}(\theta, Z^N) = V_{\text{LS}}(\theta, Z^N)(1 + 2\frac{\dim(\theta)}{N}), \tag{14}$$

where $\dim(\theta)$ depicts the number of identifiable (free) model parameters.

Experiments: Consider the discrete time system $g_0(k)$ given by its transfer function:

$$G_0(z) = \frac{\sum_{k=0}^{n_b} b_k z^{-k}}{\sum_{k=0}^{n_a} a_k z^{-k}}, \tag{15}$$

Generate the filter coefficients a_k, b_k using the MATLAB instruction

```
>> [b,a]=cheby1(3,0.5,[2*0.15 2*0.3]);
```

This is a band pass system with a ripple of 0.5dB in the pass band (check this by plotting its transfer function). Generate two data sets D_{est} and D_{val}, the former with length N_{est} being used to identify the model, the latter with length N_{val} to validate the estimated model. Recall that the initial conditions for both sets are zero! Use the MATLAB instructions

```
>> u0=randn(N,1);
>> y0=filter(b,a,u0);
>> y=y0+ny;
```

with u_0 zero mean normally distributed noise with standard deviation $\sigma_{u_0} = 1$, and n_y zero mean white Gaussian noise with σ_{n_y} equal to 0.5 for a first experiment, and 0.05 for a second experiment. Take $N_{\text{est}} = 1000$, and $N_{\text{val}} = 10000$ in both experiments.

Perform the following tasks:

- Use the linear least squares procedure to estimate the model parameters, and this for varying orders from 0 to 100;
- Calculate for each of the models the simulated output `y = filter(g,1,u0)`, and calculate the cost function V_{LS} on D_{est} and on D_{val};
- Calculate V_{AIC};
- Plot V_{est}, V_{AIC} and V_{val} as a function of the model order. Normalize the value of the cost function by $\sigma_{n_y}^2$ to make an easier comparison of the behavior for different noise levels;
- Calculate $V_0 = \frac{1}{N}\sum_{k=1}^{N}(y(k) - \hat{y}(k))^2$ on the undisturbed output of the validation set;
- Plot $\sqrt{\frac{V_0}{\sigma_{n_y}^2}}$ as a function of the model order.

Observations and questions:

The results are shown in Figure 4.

- Increasing the model order results in a monotonic decreasing cost function V_{est}. Can you explain this?
- Under what assumptions does an increase of complexity reduce the cost function V_{ext}?
- Do we observe the same monotonic decrease in the cost function V_{val}?
- Try to give an intuitive explanation of the cost function V_{val} for 1) low orders, 2) the 'optimal' order, and 3) high model orders.

- Does the Akaike cost function provide an alternative for using a validation data set? Do we need a validation data set if we employ the Akaike criterion?
- What happens to the optimal model order if the variance of the disturbing noise increases? Can you give an intuitive explanation for this effect? (Try to reason by considering the 'optimal' model order for an undisturbed data set: What is the optimal model order if the disturbing noise is absent?)
- Let us look at the normalized cost function $\sqrt{V_0/\sigma_{n_y}^2}$. This function gives us a fair idea about the quality of the model. The figure shows that a wrong selection of the model can result in much larger simulation errors. What is the advantage of looking at $\sqrt{V_0/\sigma_{n_y}^2}$ rather than V_{val}?
- Is the model selection very critical (is the minimum flat or well-pronounced)?
- What is the effect of choosing the maximal model order $I = 100$; compare for this the situation in both V_{val} and $\sqrt{V_0/\sigma_{n_y}^2}$?

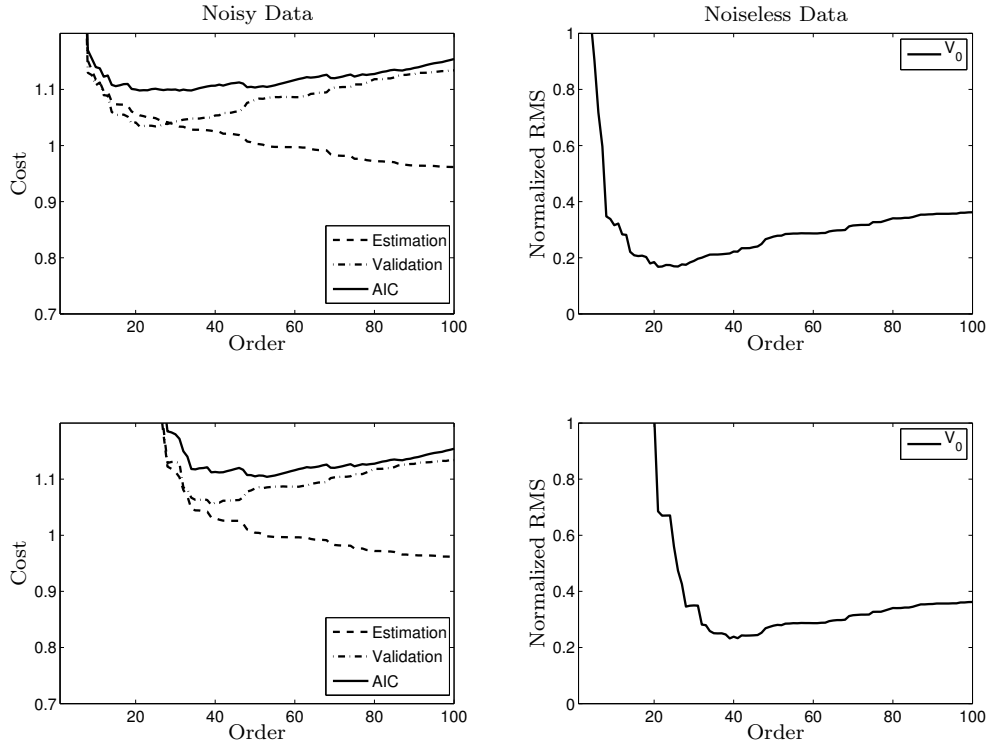

Figure 4: Comparison of the normalized cost functions for estimation data, validation data and AIC-criterion (left) and plot of V_0 (right) for $\sigma_{n_y} = 0.5$ (top) and $\sigma_{n_y} = 0.05$ (bottom).

3 Exercises Part Prof. B. De Moor

3.1 Introduction and General Information

3.1.1 Preparing your workspace

Boot one of the computers in the computer rooms in Linux, if the computer is in Windows you should reboot to Linux. All of the instructions below are given for Linux. Once you have a username and password, you can log in. Open a terminal with a command line. You can find it under `Applications -> Accessories -> Terminal`. There are three ways to prepare your workspace:

1. When you start up matlab enter the command

   ```
   addpath('~freeware/matlab/hb81');
   ```

2. If you use Bash as your Unix shell (default for standard user) you can add the following command to your `.bashrc`[1] file in your home folder

   ```
   source ~freeware/bash/bash_profile.matlab_hb81
   ```

3. If your Unix shell is csh (C shell) you can issue the following commands

   ```
   cp ~freeware/.login ~/.login
   ```

   and remove the hash symbol `#` in front of

   ```
   source ~freeware/login_matlab_hb81
   ```

   in the `.login` file. You can switch to C shell using `exec /bin/csh`.

Any of the last two options will require you to logout and login again for the changes to take effect.

3.1.2 The System Identification Toolbox

You will be using a collection of 'identification functions'. This collection is called the *System Identification Toolbox*. The toolbox comes with a graphical user interface which can be called from the MATLAB prompt by

```
ident
```

You are free to use and experiment with the grapical user interface, however, in these exercise sessions we will use the command line instructions (functions) from the toolbox instead. The System Identification Toolbox manual describes all functions in this toolbox.

It is important to note that the toolbox uses models in the so called `idpoly` format. As long as you keep on using functions of the toolbox, you can use this format. To extract the coefficients of a model, you can utilise the `get` function

[1] A file preceded by a period is a hidden file in Linux. If you're using a file manager, check that hidden files are also shown.

```
get(syspoly,'a')
get(syspoly,'b')
get(syspoly,'c')
get(syspoly,'nk')
```

where in this case we've extracted the coefficients of $A(q), B(q), C(q)$ and the delay n_k. Don't forget to consult the help files to brush up on conventions!

3.1.3 Extra Functions

To avoid losing time writing sophisticated functions, we supplied some functions that can be useful when solving the exercises (feel free to write your own modified functions). The functions are listed below. For more help, use the help files

```
help <function_name>
```

E.g.,

- `pkshave`: removes outliers
- `diffth`: plots the difference between the transfer functions of two theta models

3.1.4 Working at home

Working further at home requires you to do two things. First of all, you have to save the data corresponding to your exercise. In order to do so, type in MATLAB:

```
clear all
exercise1
save oef1
```

This saves all the variables in a `.mat` file. Secondly, you will have to copy the extra `.m` files to your computer. To copy them to the current directory type (in a terminal, from the desired working directory):

```
cp  ~freeware/matlab/hb81/pchome/*.m
```

Now you can either transport these files with a usb-stick or access them at home via SSH File Transfer Client (you can use both `ssh` or `scp`, but the latter is recommended). At home you can load all variables into the workspace by typing `load oef1` at the MATLAB prompt. You can view the workspace in the frame at the left (switch between *Current Directory* and *Workspace*) or by typing `whos` at the MATLAB prompt.

Since the real-life identification exercise from section 3.3 requires some executables, you will have to work on the computers of ESAT. There are two options: work at ESAT or work at home by logging in on the ESAT network with an SSH-client. Of course you can also generate the necessary data at an ESAT computer and save it to continue working on the assignments at home.

3.1.5 Secure Shell (SSH)

You can find a free and recent version of SSH for Windows
`http://www.filehippo.com/download_putty/`.

Many Linux distributions have `openssh` installed by default. To connect to the ESAT network using the Linux option and a console, use

```
ssh <username>@ssh.esat.kuleuven.be
```

and password (or upload your public SSH key from your computer to the `.ssh` folder of your ESAT account). If you want to transfer files or some folder you can use the following command for downloading:

```
scp -r <username>@ssh.esat.kuleuven.be:<target_esat> <target_home>
```

where the ESAT and home targets can be files, a folder or some expression. For uploading use:

```
scp -r <target_home> <username>@ssh.esat.kuleuven.be:<target_esat>
```

If you do not want to include subdirectories, leave out the `-r` flag. Don't hesitate to ask for more information.

3.2 Getting Acquainted with the System Identification Toolbox

These exercises are intended to make you familiar with the MATLAB System Identification Toolbox and with the identification theory of the course notes. You do not have to write a report for this exercise.

Before starting it is advised to have a look on the documentation of the toolbox and its functions and data structures. It is also helpful to run through the `iddemo` in MATLAB.Just type:

```
iddemo
```

To load your system, type at the MATLAB prompt:

```
exercise1
```

This will load a system of your choice (pick a number between 1-30) into the MATLAB environment, represented by `syspoly`, which is an `idpoly` object. The system is represented as a discrete-time polynomial model with identifiable (estimable) coefficients

$$A(q)y_k = \frac{B(q)}{F(q)}u_k + \frac{C(q)}{D(q)}e_k$$

For the first exercise, you will be using ARX (AutoRegressive with eXogeneous input) models. The coefficients in the `idpoly` are arranged in descending orders of q. More information about the model at hand can be obtained using the Matlab command

```
present(syspoly)
```

While for models directly constructed from coefficients the information is minimal, it is more abundant for estimated models (e.g. variance on coefficients).

3.2.1 Delay estimation

First determine the delay in the system. This can be easily done by looking at the impulse response. You can plot it directly using the model or by simulation and using input-output data. Some useful commands are

```
impulse(syspoly)
u = iddata([],randn(1000,1),1);
y = sim(syspoly,u);
cra([y,u])
```

Plot the impulse response. What is the delay of the system? In this case the system is already known, so compare with the system characteristics.

Try adding some noise, you can accomplish this by setting the noise variance property of the system as follows

```
set(syspoly,'NoiseVariance',1e-2);
u = iddata([],randn(1000,1),1);
opt = simOptions('AddNoise',true,'NoiseData',randn(1000,1));
y = sim(syspoly,u,opt);
cra([y,u])
```

What are your observations? Try playing with the variance. Also keep in mind that the noise variance field in your system changes that of your noise input relative to the variance of the supplied noise. If you were to supply a noise sequence in `simOptions` with a variance not equal to 1, it multiplies with the noise variance of the system. Try it out.

3.2.2 Transfer function

Since the $C(q)$ and $D(q)$ polynomials are primarily used to model the noise, you'll find that the transfer function is the same as extracting the coefficients of $A(q)$, $B(q)$ and $F(q)$ (in this case $F(q)$ equals 1). For example, using one of the samples gives:

```
>> tf(syspoly)

ans =

                    0.0804 z^-1 - 0.05523 z^-2 + 0.001418 z^-3
  z^(-2) * ------------------------------------------------------
           1 - 0.7983 z^-1 + 1.127 z^-2 - 0.7565 z^-3 + 0.3388 z^-4
```

Plot the transfer function using the `bode` function. Alternatively, use `bode_wrapper` from the extra functions if you desire to modify certain units or scales. Note that we only work with discrete time systems and assume a sampling time of 1. The maximum frequency in the Bode plot is therefore equal to π, as it represents an alternating series of 1 and -1 in discrete time.

Look to the poles and zeros of the system using `pzmap`. How do they correspond to the roots of the $A(q)$ and $B(q)$ polynomials? Starting from the poles and zeros, explain the shape of the transfer function and impulse response. Use

```
[p,z] = pzmap(syspoly)
```

to get numerical values of the poles and zeros. To relate the poles to the shape of the transfer function, consider the following reasoning:

- The transfer function is calculated by evaluating $G(z)$ at complex points on the unit circle: $z = e^{j\theta}$.
- The magnitude in a point is equal to the product of the lengths of the vectors between each of the zeros and the point on the unit circle, divided by the product of the lengths between each of the poles and the point on the unit circle:

$$|G(e^{j\theta})| = K\frac{|z_1 - e^{j\theta}| \cdot |z_2 - e^{j\theta}| \cdot \ldots \cdot |z_n - e^{j\theta}|}{|p_1 - e^{j\theta}| \cdot |p_2 - e^{j\theta}| \cdot \ldots \cdot |p_n - e^{j\theta}|}$$

- From this formula it is clear that poles p_i that lie close to the unit circle (dominant poles, i.e., $|p_i - e^{j\theta}|$ small for some θ) will induce a resonance (the denominator becomes small for that θ, thus the function becomes big).
- The radial frequency at which this resonance will occur (for a sampling time equal to 1), is exactly equal to the angle (use `angle`) of this pole (p_i).
- We can thus expect to see resonances at the radial frequencies equal to the angles of the complex poles. Also, the closer the poles are to the unit circle, the higher the resonance peak.

3.2.3 Estimating an ARX model

As a first exercise (to get acquainted with the toolbox) we will use the function `arx` to estimate ARX models. Generate a white noise input u_k ($\sigma = 1, N = 500$) using the function `randn`. Now apply this input to the system to find the output y_k, in order to simulate the system, you must wrap your input in an `iddata` object and use

```
u = iddata([],randn(1000,1),1);
y = sim(syspoly,u);
```

From this input-output pair, you can identify the system using the function `arx` and the exact model structure, which can be retrieved from the system itself:

```
na = get(syspoly,'na')
nb = get(syspoly,'nb')
nk = get(syspoly,'nk')
model = arx([y,u],[na nb nk]);
```

Use `present` to check that the system and the model identified with `arx` are identical.

3.2.4 Estimating an ARX model corrupted with colored noise

Now we will use the function `arx` to compute different ARX models:

1. Generate u_k as a white noise sequence ($\sigma = 1, N = 1000$, use `randn`).
2. Generate the output with the model:

$$G(z) = \frac{B(q)}{A(q)}$$
$$H(z) = \frac{C(q)}{D(q)}$$

   where $C(q)$ and $D(q)$ are the numerator and denominator of a high pass Butterworth filter (order 4, cutoff 0.2). The noise e_k has a standard deviation equal to 0.01 ($\sigma = 0.01$). To avoid any unnecessary loss of time, we present the simplest way to generate these data:

```
u = iddata([],randn(1000,1),1);
[b_butter, a_butter] = butter(4,0.2,'high');
std_butter = b_butter(1);
set(syspoly,'c',b_butter/std_butter);
set(syspoly,'d',a_butter);
set(syspoly,'NoiseVariance',(0.01*std_butter)^2);
e = randn(1000,1);
opt = simOptions('AddNoise',true,'NoiseData',e);
y = sim(syspoly,u,opt);
```

   or if you do not want to manipulate your system:

```
u = iddata([],randn(1000,1),1);
[b_butter, a_butter] = butter(4,0.2,'high');
e = randn(1000,1);
v = filter(b_butter,a_butter,e);
opt = simOptions('AddNoise',true,'NoiseData',v);
set(syspoly,'NoiseVariance',0.01*0.01);
y2 = sim(syspoly,u,opt);
```

3. Using the data generated in the previous point, identify an ARX model with n_a, n_b and n_k equal to the real n_a, n_b, n_k (use `get(syspoly,'na')`). Note that even though $G(z)$ of the ARX model is in the same model class as $G(z)$ of the data generating model, this is not the case for the noise model $H(z)$. Compare the transfer functions. Are they close? If not, try to explain.

Hint: `bode` allows plotting of more than one transfer function on top of each other. For instance:

```
bode(syspoly,modelpoly);
```

3.2.5 Estimating an ARX model corrupted with colored noise and unknown order

The previous question was easy since we started from a given model structure of order `[na nb nk]`. In principle this structure is unknown. A more realistic way to go about this is thus by trying out different model structures and checking which one gives the 'best' model. To generate all the possible structures in the vicinity of the true order (or some initial guess) use:

```
na = get(syspoly,'na')
nb = get(syspoly,'nb')
nk = get(syspoly,'nk')
search_region = struc([na-2:na+2],[nb-2:nb+2],[nk-1:nk+1]);
```

Now, generate a new data set : u_{val}, y_{val} in the same way you generated u and y at the beginning of this question:

```
e_val = randn(500,1);
opt = simOptions('AddNoise',true,'NoiseData',e_val);
set(syspoly,'NoiseVariance',0.01*0.01);
u_val = iddata([],randn(500,1),1);
y_val = sim(syspoly,u_val,opt);
```

Use this new data as a validation set in `arxstruc`. Afterwards `selstruc` enables you to choose the 'best' model. Does this structure correspond to the real one? If not, try to explain. Check the models that come close in terms of fitness. Compare the transfer function of the model obtained in the previous step with the real one. If there is a difference, try to explain it.

An alternative way of estimating a model with unknown order is to overestimate and apply model reduction methods. A final model with this input-output data can be obtained as follows:

1. Estimate a high order ARX model ($n_a = n_b = 10$, and the delay equal to the given delay or the one estimated from the impulse response).
2. Convert the ARX model to a state space model using `ss`.
3. Inspect the Hankel singular values using `hsvd`. What is the order?
4. Reduce the state space model using the balanced model reduction techniques of the course notes (you only need `balred`). Plot the frequency response of the reduced model using `bode`.
5. Convert the result back to a polynomial (ARMAX) model using `idpoly`. Compare the coefficients of $A(q)$ and $F(q)$, as well as the $B(q)$ coefficients in both models. Try to list some advantages of this method. You can also go straight for the transfer function using `idtf` on the reduced state space system.

3.2.6 Effects of a wrong model class

Take u_k a white noise sequence ($\sigma = 1, N = 1000$) and generate an output using an OE (output error) model[2]

$$G(z) = \frac{B(q)}{A(q)}$$
$$H(z) = 1$$

and with the standard deviation of the noise sequence e_k equal to 0.05 ($\sigma = 0.05$):

```
b = get(syspoly,'b');
f = get(syspoly,'a');
oepoly = idpoly(1,b,1,1,f,0.05*0.05,1);
opt = simOptions('AddNoise',true,'NoiseData',randn(1000,1));
u = iddata([],randn(1000,1),1);
y = sim(oepoly,u,opt);
```

Determine an OE model in the exact model class (use `oe`). Note that the order of the model class parameters for `arx` was `[na,nb,nk]`, while for `oe` the order is `[nb,nf,nk]`). Plot the identified transfer function and the real transfer function.

Now determine (with the data you just generated) an ARX model as follows:

```
arxmodel = arx([y,u],[nf nb nk]);
```

Compare the identified transfer function with the real one. What goes wrong? Try to solve this problem by low pass filtering the input output data (use `idfilt`). Does the transfer function agree better with the real transfer function now? If you have the time you can try other filters to solve the problem.

Tip: If you find it hard to improve the ARX model even with low pass filtering, try to notch up the model orders, then retry estimating an ARX model.

3.2.7 Asymptotic properties and overmodeling

Take u_k a white noise sequence ($\sigma = 1, N = 1000$) passed through a fourth order Butterworth filter:

```
>> [bf,af] = butter(4,0.15);
>> u = filter(bf,af,randn(1000,1));
```

Now generate the output with the OE model:

$$G(z) = \frac{B(q)}{A(q)}$$
$$H(z) = 1$$

within Matlab:

[2]Note that we are using $A(q)$ from the original ARX system as $F(q)$ in our OE model.

```
b = get(syspoly,'b');
f = get(syspoly,'a');
oepoly = idpoly(1,b,1,1,f,0.01*0.01,1);
opt = simOptions('AddNoise',true,'NoiseData',randn(1000,1));
u = iddata([],randn(1000,1),1);
y = sim(oepoly,u,opt);
```

Plot the input and output using `plot`. Determine an OE model in the exact model class (meaning the model class of the data generating model). Plot the transfer function and the real transfer function. Plot on the same plot the uncertainty of the transfer function (3σ, use `bode`, check the documentation). Is the real transfer function within the uncertainty band? If not, try to explain.

Make a pole-zero plot with 3σ error bounds. Are there poles and zeros that are likely to cancel? You can plot the confidence intervals using

```
h = iopzplot(oepoly,arxmodel);
showConfidence(h,3);
```

Overmodel the transfer function by taking n_b and n_f two times the exact value. Repeat the plotting of transfer function and pole-zeros with their error bounds. What is the difference with before? Are there clear pole-zero cancellations?

For the two OE models computed in this exercise, also compute the prediction error using `compare` (always with an independent validation set!). Which model predicts the data the best?

3.2.8 Model assessment

Throughout this question, you used `syspoly` as a reference model to assess the model quality. In reality (as a preparation for the last question), the real model is not known, and you will need to resort to other means of validation. In this question you will explore the possibilities.

Remark: A small remark on the use of `compare` (and `predict`) is in order. These functions calculate 'predicted' outputs. These 'predictions' can be done over different horizons though (cf. M, the prediction horizon in `compare`). The two extreme (and most important) cases are:

- $M = \infty$ (which is the default). This is called 'pure simulation', and the errors (the difference between the real output and the simulated output) are called simulation errors. The simulation mechanism is simply:

  $$y_k^{\text{sim}} = G(q)u_k$$

  The simulation error can then be computed as:

  $$\begin{aligned} e_k^{\text{sim}} &= y_k - y_k^{\text{sim}} \\ &= [G(q)u_k + H(q)e_k] - G(q)u_k \\ &= H(q)e_k \end{aligned}$$

  It should thus be clear that these 'simulation errors' are NOT white.

- $M = 1$. This is called 'prediction', and the errors (the difference between the real output and the predicted output) are called prediction errors. The simulation mechanism is:

$$y_k^{\text{pred}} = H^{-1}(q)G(q)u_k + [1 - H^{-1}(q)]y_k$$

The prediction error can then be computed as:

$$\begin{aligned} e_k^{\text{pred}} &= y_k - y_k^{\text{pred}} \\ &= y_k - H^{-1}(q)G(q)u_k - [1 - H^{-1}(q)]y_k \\ &= -H^{-1}(q)G(q)u_k + H^{-1}(q)y_k \\ &= -H^{-1}(q)G(q)u_k + H^{-1}(q)[G(q)u_k + H(q)e_k] \\ &= e_k \end{aligned}$$

The prediction errors are thus indeed white. This is the reason why `resid` uses 'prediction errors' (and not simulation errors).

Take two models you computed in the previous steps (the models should be taken from the same step, since they should be comparable). Validate both models by looking at:

1. The simulation errors (use `compare`).
2. The prediction errors (use `compare` with the prediction horizon set to 1).
3. The whiteness of the errors (use `resid`)
4. The cross-correlation between errors and inputs (use `resid`)

Which of the two models do you prefer?

3.3 A Real-Life Identification Problem

In this exercise, you are confronted with a real life identification problem. You will need to

- design input-signals;
- do an experiment (software);
- preprocess the data;
- identify different models;
- validate the model.

3.3.1 Introduction

When you have designed an input u_k, type:[3]

```
>> y = exercise2(u);
```

[3]Don't forget to set the path as `path(path,'/freeware/env/matlab/hb81/hp/')` after starting MATLAB.

The output of your system will be returned in the vector `y`.

When the program returns with an error message, and you can not figure out what you did wrong, please try the following: instead of running your m-file at once evaluate all preceding steps, then execute `exercise2` seperately and then evaluate the remaining code. If the problem persists, do not hesitate to contact us.

3.3.2 Questions

The question for this exercise is simple. The answer is harder. The basic question is: try to generate a 'good' model for the black box system. Everything is left to your creativity and inventiveness (which will be validated at the exam). You can use everything from the course notes and much more if you want to try some things out.

As stated before, the basic steps are :

1. Input design: step, white noise, colored noise, impulse, sine waves. You should also bear in mind that the input will saturate at a certain level (just like in reality the actuators will saturate). The saturation level is at 3. So, if you put an input in the simulator that has an amplitude bigger than 3, it will be clipped to 3. No warning will be given (this is your own responsibility).
2. Preprocessing : peak shaving (try `spline`), detrending (try `filtfilt` or `detrend`), delay correction (try `cra`). For the purpose of simplicity (not too much programming), we have written a peak shaving utility function for you (see `help pkshave`).
3. Identification
4. Validation

You are allowed to do as many experiments as you want on the system (using `exercise2`). But keep in mind that in industry experiments are costly. So, think before you do an experiment. Take note this is an open question. You are encouraged to apply the knowledge you gathered from the previous question as much as possible to this question.

Hints:

- The function `cra` allows you to get an idea of the time delay in the system, in the same way as described in the course notes. It will compute an estimate of the impulse response and plot this estimate together with the confidence bounds. Another way to determine the time delay was indicated in the previous exercise (using `arxstruc`). When you are confronted with large time delays, it is advised to shift your output, so that the time delay is reduced (this makes identification easier).
- One way to estimate a reference transfer function is as follows (stepped sine):
  - Apply a sine wave input at a certain frequency.
  - Measure and pre-process the output. Check (through an `fft` for instance) if the output still has an energy peak at the frequency used for the input (if not, the signal to noise ratio is too bad).

  - The ratio of (output magnitude)/(input magnitude) is an indication for the transfer function magnitude at that frequency. Taking the maximum value might not be the most robust approach, try using the standard deviation ratio instead.

- You can get an idea of the noise spectrum by applying a zero input to the system. The output then measured is purely due to the disturbances.
- You could also use the functions `bj`, `armax` and `arma`.

3.4 Report

For this part you should hand in a report for the real-life identification problem only. The report should be at most **15 pages** long. It should contain (at least) the following sections: Input design, Preprocessing, Identification, Validation, Conclusion (including choice of best models).

A On MATLAB

This Section contains some practical tricks and hints to use MATLAB. Even though these are helpful, it does not exclude the need for reading the MATLAB manual.

Some good advice to stay away from time pitfalls: don't try to go over all the questions, to find out at the end (when you write your report) that you have to go back and solve the questions again. Instead, try to write your report while you're solving the question. You should thus think about the question, solve it in MATLAB (possibly write a small .m file - see below), make a plot (see below) and discuss it. A little self-discipline during the exercises is required, but this will save you quite some time at the end.

A.1 MATLAB, Editing & Printing

- MATLAB can be started by typing (in any terminal window):

  ```
  matlab
  ```

  There are four sources where you can get more information about matlab:

  - From this appendix.
  - From the demos. Type :

    ```
    demo
    ```

    to start up matlab demonstration window. You can now browse through the different demos of MATLAB. Look also at the System Identification demo under Toolboxes. You can also start a more extensive identification demo by typing:

    ```
    iddemo
    ```

  - The on-line help of MATLAB. Just type `help command-name`, and you will get a page full of information.

- You can print on the printer in room 91.94 (ulstudent).
- Any text editor can be used to create so called m-files:

  `<filename>.m`

  These files contain a sequence of MATLAB commands (written one after the other). If you type (in MATLAB):

  `<filename>`

  then the commands in filename will be executed. See also the next paragraph or the MATLAB manual.

A.2 .m Files

For every exercise, you can set up a .m (*dot m*) file. This file contains MATLAB commands you would normally execute at the MATLAB prompt. The .m files enable you to bundle these commands into a "program". Once the "program" is written, you can execute the file by typing the file name at the MATLAB prompt. MATLAB will then execute the file just as when you had typed all the commands one by one at the MATLAB prompt. In a similar way you can generate MATLAB functions (see MATLAB manual).

A.3 Plotting

The MATLAB plot command is fairly powerful. Typing:

```
help plot
```

will give you a short overview of the possibilities. More information can be found in the manual. Consider the following basic plot:

```
u=randn(100,1);
plot(u);
```

This plot can now be further labeled by typing:

```
grid;
title('My first plot');
xlabel('Time Units');
ylabel('Volt');
```

If you want to plot another curve on top, do:

```
y=randn(100,1);
hold on;
plot(y);
hold off;
```

Another (and clearer) way to plot these two data sets is as follows (see also the help on subplot):

```
subplot;
time=[0:2:199];
subplot(211);
plot(time,u);
grid;
xlabel('Time Units');
ylabel('Volt')
title('Subplot 1');
subplot(212);
plot(time,y);
grid;
xlabel('Time Units');
ylabel('Volt')
title('Subplot 2');
```

A second graphical window can be opened by typing:

```
figure(2)
```

All subsequent plots will now be sent to this second window. If you want to plot on the first window again, do:

```
figure(1)
```

A.4 List of Useful MATLAB Commands

A.4.1 General

- `help <function>`
- `helpdesk`
- `doc`
- `lookfor`
- `clear`
- `close`

A.4.2 Matrices, Vectors, Scalars

- `x = 1`
- `A = [1 2 3; 4 5 6; 7 8 9];`
- `b = [1;5];`
- `A(1:end,2)`

- `A'`
- `zeros`, `ones`, `eye`, `rand`, `randn`
- `pi`, `i`, `j`, `Inf`, `NaN`
- `sin`, `cos`, `exp`, `log`

A.4.3 Workspace

- `whos`
- `save`
- `load`
- `clear`
- `close`

A.4.4 Matrix functions

- `A'`
- `inv(A)`
- `pinv(A)`
- `eig(A)`
- `svd(A)`
- `diag(A)`
- `fliplr(A)`, `flipud(A)`

A.4.5 Graphics

- `figure`, `plot`
- Plotting a sine function $y = sin(x)$ for $x = [0, 2\pi]$:
```
x = 0:0.1:2*pi
y = sin(x)
plot(x,y)
```
- `xlabel`, `ylabel`, `title`, `legend`
- `stem`
- `axis`
- `grid`
- `subplot`

- `plot3`
- `meshgrid`, `mesh`

A.4.6 Program flow

- `for`
- `if, then, else`
- `while`
- `switch, case`