Parameter Estimation - Computer Laboratory 2

Table of Contents

Least squares estimation of static linear models

Model types

The simplest model is the **linear scalar** model

$$y^{(M)} = p \cdot x$$

where the dependent variable y can be written as the product of the parameter p and the independent variable x.

- We can set the value of the input variable and can measure the output. Measurement error (ε) may be present in the output, where $\varepsilon_j, j = 1, \ldots, m$ are independent identically distributed random variables, with p.d.f. $f_\varepsilon(z)$

$$y_j = p \cdot x_j + \varepsilon_j$$

- The *m* independent measurements of the input and output variables are included in the measurement vectors:

$$X = [x_1, x_2, \ldots, x_m]^T,\ Y = [y_1, y_2, \ldots, y_m]^T$$

- The data matrix is composed of the measured output and intput pairs:

$$D = [Y, X] = \begin{bmatrix} y_1 & x_1 \\ \vdots & \vdots \\ y_m & x_m \end{bmatrix}$$

In the **static linear** model, there can be more than one independent variables and parameters. The dependent variable can be written as the product of the **parameter vector** and the **independent variable vector**:

$$y^{(M)} = x^T p = \sum_{i=1}^{n} x_i p_i$$

- The measured outputs in case of fixed independent variables (x_{ji}) and with measurement errors (ε_j - independent identically distributed random variables) can be written as:

$$y_j = \sum_{i=1}^{n} x_{ji} p_i + \varepsilon_j, \; j = 1, \ldots, m$$

- The measured inputs are included in the *X* input matrix, where one row corresponds to the *j*th measurement. The measured outputs can be found in the *Y* output vector.

$$X = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m1} & x_{m2} & \ldots & x_{mn} \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

- The data matrix *D* is composed of the measured output-input values:

$$D = [Y, X] = \begin{bmatrix} y_1 & x_{11} & x_{12} & \ldots & x_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_m & x_{m1} & x_{m2} & \ldots & x_{mn} \end{bmatrix}$$

Linear regression

residuals (r_j): difference between the measured output y_j and the model predicted output $y_j^{(M)}$:

$$r_j = y_j - y_j^{(M)}, \; j = 1, \ldots, m$$

where m is the number of measurements.

A loss function of the residuals can be defined, that depends on the parameter to be estimated and the fixed input measurements. This loss function can be used to evaluate the quality of the model with the estimated parameter, i.e."how good" is our estimated model, "how far" is the estimated model from the real measurements.

The loss function is usually a quadratic function of the residuals:

$$V(p;X) = \sum_{j=1}^{m} r_j^2 = \sum_{j=1}^{m} (y_j - y_j^{(M)})^2$$

The aim of the **least squares estimation** is to find the parameter(s) p which **minimizes the loss function**! In case of linear scalar and static linear models this problem can be solved analytically, and there exist ecplicit formulas for the estimated parameter values.

Solution of the LS estimation in case of **linear scalar** models:

$$\widehat{p} = \frac{1}{\sum_{j=1}^{m} x_j x_j} \sum_{j=1}^{m} x_j y_j$$

- variance of the estimation: $\sigma_{\widehat{p}}^2 = \frac{1}{x^T x}\sigma_\epsilon^2$ is the variance of the measurement errors (= the variance of the residuals)

Solution of the LS estimation in case of of **static linear** models:

$$\widehat{p} = (X^T X)^{-1} X^T y$$

- **Important**: the solution exists only if the inverse of $X^T X$ exists!
- covariance of the estimation: $COV_{\widehat{p}} = (X^T X)^{-1}\sigma_\epsilon^2$

Note: The linear scalar model is the special static linear model with 1 input, 1 output and 1 parameter. The formula used to estimate the parameters of static linear models can be used for the linear scalar case, too. In this case *X* becomes a vector instead of a matrix.

Creating functions in MATLAB

General function declaration

In MATLAB, the functions can be created in separate files, or in anonymus way. The anonymus function is associated with a function handle, and it can execute only one statement and return only one output. **Functions created in files** have more potential, because they can contain several statements, and multiple inputs and outputs are possible.

The syntax of the function declaration is

```
function [y1,...,yN]=myfun(x1,...,xM)
```

`where`

- `function` is the keyword defining a function.
- the output arguments `y1,...,yN` are listed between square brackets `[ ]`

- `myfun` is the function name. Valid function names begin with an alphabetic character, and can contain letters, numbers, or underscores.
- the input arguments `x1,...,xM` are listed between round brackets `( )`

The function body contains the executable statements. The end of the function can be denoted by the `end` word, but is in only required in case of nested and local functions.

The most important **control statements**, that can be used in functions:

- **if-else**

```
if expression
    statements
elseif expression
    statements
else
    statements
end
```

- **switch-case**

```
switch switch_expression
   case case_expression
      statements
   case case_expression
      statements
    ...
   otherwise
      statements
end
```

- **`for` loop**: the values can be defined in three ways
- *`initVal`*:*`endVal`* — Increment the *`index`* variable from *`initVal`* to *`endVal`* by `1`, and repeat execution of *`statements`* until *`index`* is greater than *`endVal`*.
- *`initVal`*:*`step`*:*`endVal`* — Increment *`index`* by the value *`step`* on each iteration, or decrements *`index`* when *`step`* is negative.
- *`valArray`* — Create a column vector, *`index`*, from subsequent columns of array *`valArray`* on each iteration.

```
for index = values
   statements
end
```

- **`while` loop**

```
while expression
    statements
end
```

- **`break`**: Terminates execution of for or while loops.
- **`return`**: Return control to invoking script or function.
- **`continue`**: Pass control to next iteration of `for` or `while` loop.

Example

The following function computes the autocovariance of `x` at a given time delay `s`. It is similar to the `xcov` function, but it returns only the specific autocovariance value instead of the whole sequence. Copy the following code into a file and save it.

```
function y=autocov(x,s)
   %compute the autocovariance at a given time delay s
   N=length(x);
   y=0;
   m=mean(x);
   for i=1:N-s
       y=y+(x(i)-m)*(x(i+s)-m);
   end
   y=1/(N-s)*y;
end
```

```
x=rand(1,10);
autocov(x,2)
```

```
ans = -0.0351
```

Individual task

Create a function that computes the autocorrelation of x at a given time delay.

$$r_{xx}(s) = \frac{1}{n-s}\sum_{i=1}^{n-s} x_i \cdot x_{i+s}$$

Data import

If we have measurement data in files (txt, csv), we can easily import them into MATLAB.

1. On the **Home** tab find the **Import Data**
2. Set the variable name where you want to import the data in the **Name** field.
3. Set the **Type** of the variable

- Table
- Column vectors
- **Numeric matrix**
- String array
- Cell array

Click on the **Import Selection** button on the right. Now the data is imported.

Command line option:

```
D=importdata('data.txt')
```

Imports the content of

into array `D`.

Individual task

Import data from *dataset_4.txt* and *dataset_5.txt* from the *Lab-2_data* folder into separate variables!

Creating data for parameter estimation

At first we need to create some data, that can be used in the parameter estimation examples later.

Linear scalar model

- create vector of independent variables: different methods introduced in the previous labratory can be used, e.g. manually enumerating the elements, using the : operator, using `linspace`, or using random number functions e.g:

```
x=0:0.1:100;
```

- define a parameter: create a variable with the value of the parameter, e.g:

```
p=3;
```

- compute the output: create a new variable y whose value is `p*x`

```
y=p*x;
```

- add white noise: you can use the different random number generators, e.g. rand, `randn` or `normrnd`. In parameter estimation we usually use zero mean gaussian white noise (because of its good properties), therefore the use of `randn / normrnd` functions are preferred. You can modify the variance if the distribution.

```
y=y+5*randn(size(y));
```

- create the data matrix:

```
D=[y',x'];
```

Static linear model

- create vectors of independent variables: be careful that all vectors must be the same length! The vectors should be column vectors

```
x1=linspace(-5,5,200);
x2=2+4*rand(size(x1));
```

- define a parameter vector: the number of parameters must be the same as the number of inputs! The parameter vector is a column vector.

```
p1=[-1;2];
```

- compute the output: you can either compose a matrix from the inputs (be careful with the dimensions), or use the sum of the proper input x parameter elements

```
%input data, x1 and x2 are row vectors
X=[x1',x2'];
y1=X*p1;
%sum of products
y11=x1*p1(1)+x2*p1(2);
```

- add zero-mean gaussian white noise, with some variance

```
y1=y1+3*randn(size(y1));
y11=y11+3*randn(size(y11));
```

- construct the data matrix

```
D1=[y1,X];
D11=[y11',x1',x2'];
```

Individual task

1. Create two input vectors (both including 500 elements), where the first one contains normally distributed random numbers with 1 mean and 2 variance, and the second one is composed of 250 ones in the first half and 250 zeros in the second half.
2. Create a parameter vector with two elements.
3. Compute the output vector.
4. Add zero-mean gaussian white noise with 4 variance.
5. Construct the data matrix.

Least squares estimation using custom functions

We can easily create a function to compute the least squares estimate of linear scalar and static linear model. Because of the scalar model (1 parameter) can be viewed as a special static linear model, one function is enough to solve both cases.

The function should calculate the estimated value of the parameters using the least squares formula. It should return the (co)variance of the estiamate and the residual sequence in order to evaluate the quality of the estimate.

Function name:

- `LS_est`

The input of the function is

- The data matrix, whose first column always contains the measured outputs, and the other columns contain the measured input(s).

The outputs of the function are the

- estimated value of the parameter(s)
- variance or covariance of the estimation
- residual sequence

Solution steps

- Extract the inputs and outputs from the data matrix
- Check wether the solution exists. (Is `X'*X` invertable? - determinant is nonzero, or `X` has full rank)
- Compute the estimated parameter.
- Compute the model predicted output using the estimated parameter.
- Compute the residual sequence.
- Compute the (co)variance of the residuals.
- Compute the (co)variance of the estimate.

Task

1. Call the created function with the previously created data matrices (`D`, `D1`).
2. What are the estimated parameters?
3. What is the variance of the estimate? Is it good?
4. Evaluate the estimation by plotting the residuals in both cases. What can you say about the residuals?

Least squares estimation using the Curve Fitting Toolbox

The Curve Fitting Toolbox is a MATLAB application that can be used to fit different types of curves (e.g. polynomials, splines, exponential...) or surfaces to 1D and 2D data. It also provides tools for the analysis of the fit (e.g. confidence intervals and residual plots). The drawback of this app is that it is limited to models with 1 or 2 input variables.

You can start the Curve Fitting tool by selecting it from the APPS bar, or by typing `cftool` in the command prompt. It opens the graphical user interface of the Curve Fitting Tool.

Using the App

- On the left hand side, you can select the X, Y (and Z) data for the fit. In case of one input, only X and Y sould be used. In case of two inputs the X and Y data are the input and Z is the output.
- Set the previously used `x` and `y` for the X and Y data.
- You can select the type of the curve from the drop-down menu. In our case, the **Polynomial** option with **degree 1** should be used.
- Below the selected data, now you can see the results of the fit. Compare it with the reults of your `LS_est` custom function! Is there a difference?
- It can be noticed, that the linear model has an additive constant (p2). To eliminate this constant we have to modify the Fit Options.

- In the **Fit Options...** window, you can set the upper and lower bounds of the parameters. By setting the upper and lower bounds of `p2` to `0`, the results are now the same as the `LS_est` function.
- The residuals can be displayed by clicking on the **Residuals plot** icon (second in the upper left corner).

Task

1. Estiamate the parameters of the static linear model, using the data in `x1`, `x2` and `y1`. Compare the results with the results of the `LS_est` function.

Command prompt

The curve fitting task can be also carried out in the command prompt, using functions of the CF Tool.

The `fit` function with the proper options has the same capabilities as the app. Syntax:

- `fitobject=fit(x,y,fitType)` creates the fit to the data in `x` and `y` with the model specified by `fitType`.
- `fitobject=fit([x,y],z,fitType)` creates a surface fit to the data in vectors `x`, `y`, and `z`.
- `fitobject=fit(x,y,fitType,fitOptions)` creates a fit to the data using the algorithm options specified by the `fitOptions` object.
- `[fitobject,gof,output]=fit(x,y,fitType)` returns goodness of fit statistics and fitting algorithm information in the structure `output`.
- `fitType` is used to define the type of he curve: `'poly1'` - linear polynomial curve, `'poly11'` - linear polynomial surface
- `fitOptions` is used to define the fitting method, algorithm, normalization, weights, bounds etc, depending on the seleted algorithm.
- `output` is a structure containing information about the fitting, e.g. number of parameters, residuals, number of iterations, etc.

Example

```
options = fitoptions('Method', 'LinearLeastSquares')
```

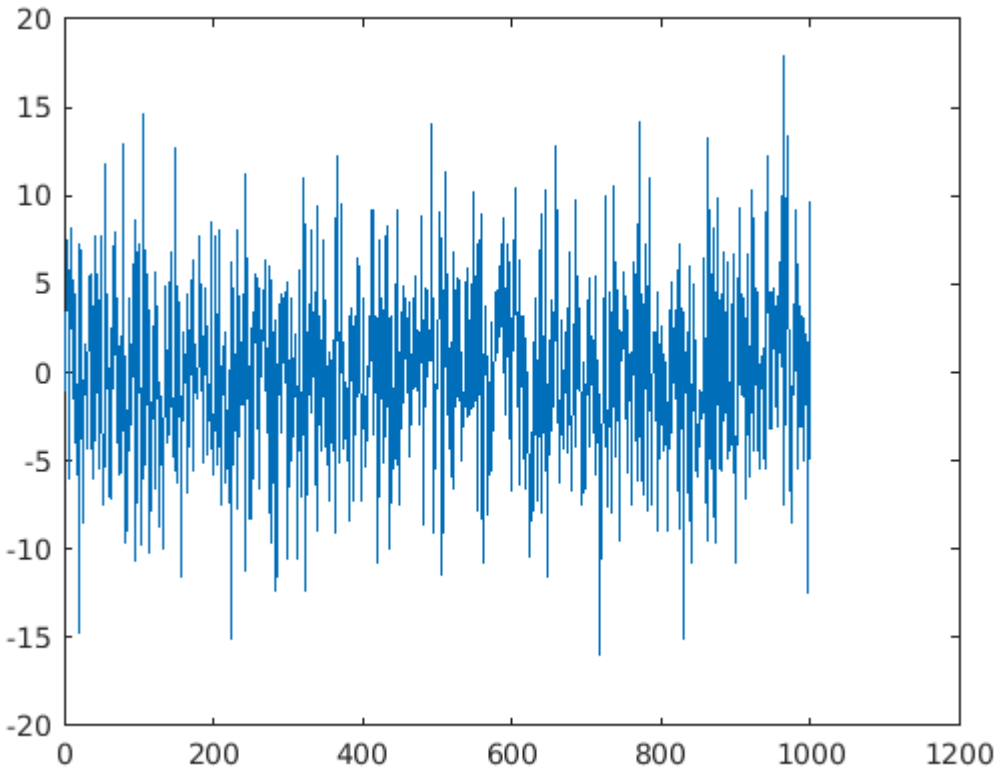
```
options =

    Normalize: 'off'
      Exclude: []
      Weights: []
       Method: 'LinearLeastSquares'
       Robust: 'Off'
        Lower: [1×0 double]
        Upper: [1×0 double]
```

```
f=fit(x',y','poly1',options)
```

```
f =
     Linear model Poly1:
     f(x) = p1*x + p2
     Coefficients (with 95% confidence bounds):
       p1 =           3  (2.989, 3.01)
       p2 =     -0.1975  (-0.8075, 0.4125)
```

```
options2=fitoptions('Method','LinearLeastSquares','Lower',[-inf, 0],'Upper',[inf,0]);
[f2,gof,out]=fit(x',y','poly1',options2);
```

```
res=out.residuals;  %residuals
plot(res)
```



Special cases

Model with static constant

Model form:

$$y^{(M)} = p \cdot x + c \quad \text{or} \quad y^{(M)} = x^T \cdot p + c$$

It can be converted to static linear model form, introducing a constant auxiliary input x_z, whose value is 1 at every measurement point.

$$y^{(M)} = x'^T \cdot p',$$

where $x'^T = [x_1, x_2, \ldots, x_n, x_z]$ and $p^T = [p_1, p_2, \ldots, p_n, c]$.

Solution of these models:

1. Parameters of models with maximum 2 original inputs can be estimated with the Curve Fitting Toolbox. In this case the parameter bounds have to be set to [-inf, inf] for all parameters.
2. Using the `LS_est` function, one should examine the data first. The constant auxiliary input need to be added to the measurement data matrix if it is not present in it. Then the `LS_est` function can be invoked.

Estimate the parameters of the static linear model using the data that was imported from *dataset_4.txt*!

Degenerate data

If the inputs are not independent, then the LS estimation cannot be performed. The independence of the inputs means that the input matrix X has full rank, i.e. all rows/columns are linearly independent from each other.

The independence can be checked with several tests:

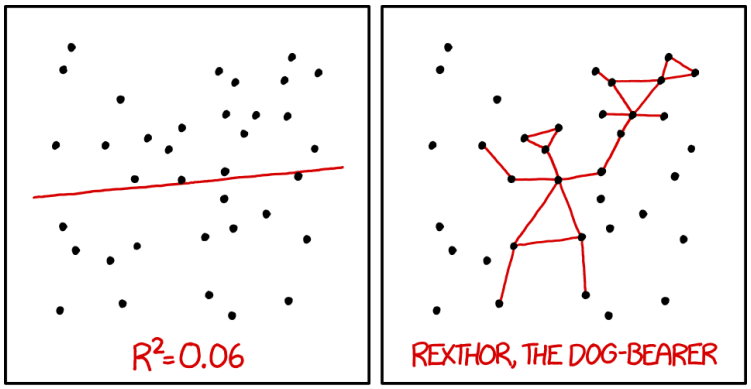
- The matrix has full rank, i.e. the rank of the matrix equals to the minimum number of rows/columns. For example a 2x3 matrix with rank 2 is a full rank matrix.
- The determinant of the matrix is nonzero. The `det` function can be used to compute the determinant in MATLAB.

Call the `LS_est` function with the with the data that was imported from *dataset_5.txt.*

HOMEWORK (Deadline 2020. October 14. 10:30)

1. Modify the LS_est function in such a way, that it represents the residual plot and the mean of the residuals (as a constant line).
2. Test the modified function with the data from *dataset_0.txt*!

Send the created script file (NEPTUNKOD-HW2.m) to pozna.anna@virt.uni-pannon.hu!



Solutions (functions)

```
function y=autocov(x,s)
   %compute the autocovariance at a given time delay s
   N=length(x);
   y=0;
   m=mean(x);
   for i=1:N-s
       y=y+(x(i)-m)*(x(i+s)-m);
   end
   y=1/(N-s)*y;
end
```

```
function y=autocorr(x,s)
   %compute the autocovariance at a given time delay s
   N=length(x);
   y=0;
   for i=1:N-s
       y=y+(x(i)*x(i+s));
   end
   y=1/(N-s)*y;
end

function [p_est,sigma_est,residual]=LS_est(D)
  y=D(:,1);
  X=D(:,2:end);
  if(det(X'*X)~=0)
      p_est=inv(X'*X)*X'*y;
      y_M=X*p_est;
      residual=y-y_M;
      sigma_r=var(residual);
      sigma_est=inv(X'*X)*sigma_r;
  else
      error('Error: matrix is rank deficient, solution does not exist.')
  end
end
```