Parameter Estimation - Computer Laboratory 4

Table of Contents

The Instrumental Variable (IV) method

Problem:

$$y(k) = \varphi^T(k) p_0 + \nu_0(k)$$

- correlated measurement ($\varphi(k)$) and noise ($\nu_0(k)$)
- LS method is not optimal

Solution: Instrumental variable method

- Change $\varphi(k)$ to a suitable $\xi(k)$ signal that is uncorrelated with $\nu_0(k)$ and correltated with $\varphi(k)$
- $\xi(k) = [-z(k-1) \dots -z(k-n_a)\ u(k) \dots u(k-n_b+1)]^T$
- $z(k)$ is generated as the output of a linear system with input u: $N(q^{-1})z(k) = M(q^{-1})u(k)$, where $N(q^{-1})$ and $M(q^{-1})$ define a stable filter
- Choosing $N(q^{-1})$ and $M(q^{-1})$ with a LS pre-estimation step
- The IV estimate:

$$\hat{p}_N^{IV} = \left[\frac{1}{N}\sum_{k=1}^{N} \xi(k)\varphi(k)^T\right]^{-1} \frac{1}{N}\sum_{k=1}^{N} \xi(k) y(k)$$

The iv4 function

The `iv4` function in MATLAB estimates the parameters of an ARX model using the four-step instrumental variable method.

Syntax: `sys=iv4(data,[na,nb,nk])`

- `sys` is a discrete time `idpoly` object representing the ARX model
- `na` is the order of the polynomial $A(q^{-1})$
- `nb` is the order of the polynomial $B(q^{-1})+1$
- `nk` is the input-output delay

Task

Consider the following ARX model

$$y(k) = y(k-1) + p_1 u(k-1) + p_2 u(k-2) + \nu(k)$$

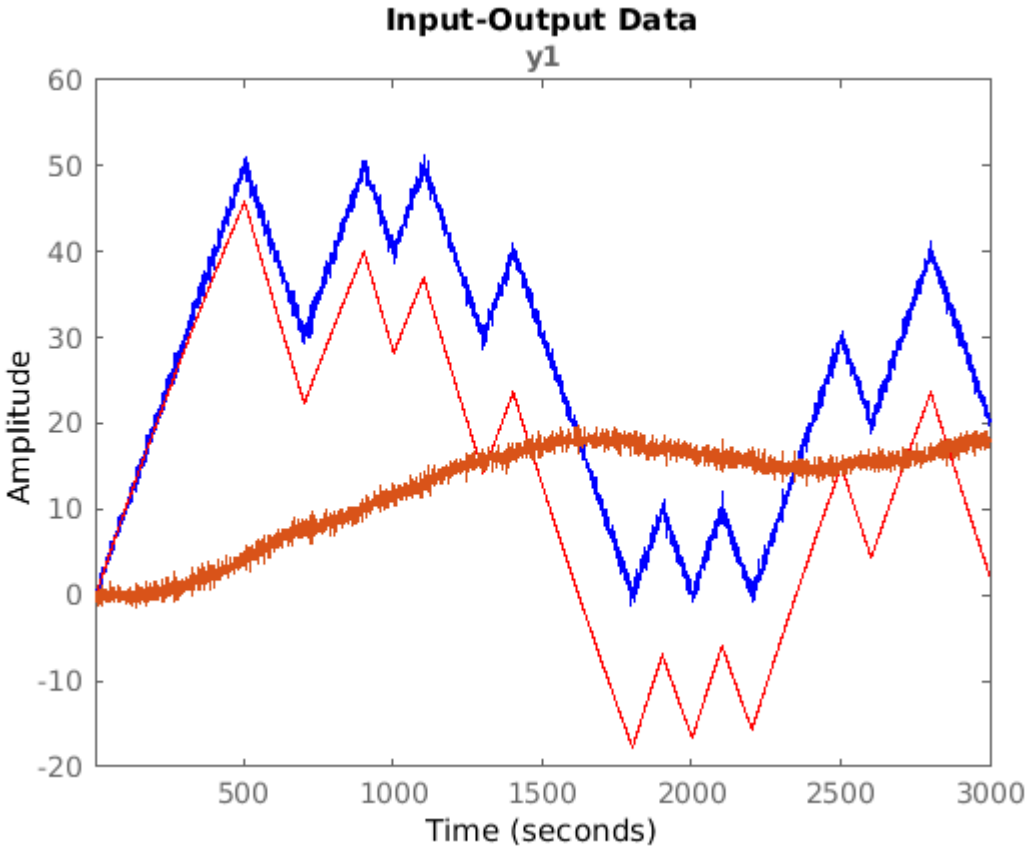
The measurement data can be found in D1.

- Estimate the parameters p_1 and p_2 with the LS method! Compute and plot the residual sequence!

```
sys1_LS=arx(D1,[1 2 1])
```

```
sys_LS =
Discrete-time ARX model: A(z)y(t) = B(z)u(t) + e(t)
  A(z) = 1 - z^-1

  B(z) = 0.02171 z^-1 - 0.07289 z^-2

Sample time: 1 seconds

Parameterization:
   Polynomial orders:   na=1   nb=2   nk=1
   Number of free coefficients: 3
   Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using ARX on time domain data "D1".
Fit to estimation data: 93.89% (prediction focus)
FPE: 0.8175, MSE: 0.8148
```

```
y1_LS=sim(sys1_LS,D1);
hold on
plot(D1.y,'b');
plot(y1_LS,'r');
plot(D1.y-y1_LS.y);
```

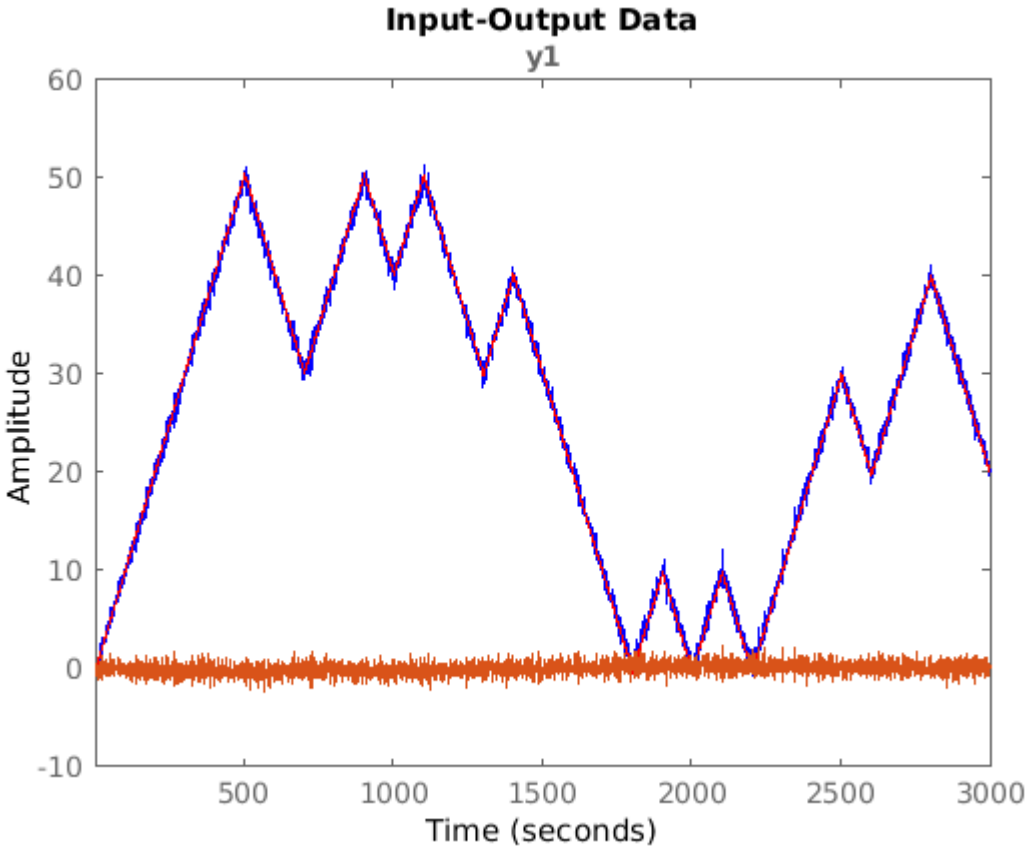


- The output noise is not white!
- Estimate the parameters p_1 and p_2 with the IV method! Compute and plot the residual sequence!

```
sys1_IV=iv4(D1,[1 2 1])
```

```
sys_IV =
Discrete-time ARX model: A(z)y(t) = B(z)u(t) + e(t)
  A(z) = 1 - z^-1

  B(z) = 0.0646 z^-1 - 0.115 z^-2

Sample time: 1 seconds

Parameterization:
   Polynomial orders:   na=1   nb=2   nk=1
   Number of free coefficients: 3
   Use "polydata", "getpvec", "getcov" for parameters and their uncertainties.

Status:
Estimated using IV4 on time domain data "D1".
Fit to estimation data: 93.89% (prediction focus)
FPE: 0.8178, MSE: 0.8151
```

```
y1_IV=sim(sys1_IV,D1);
figure
hold on
plot(D1.y,'b');
plot(y1_IV,'r');
plot(D1.y-y1_IV.y);
```



Parameter estimation of nonlinear models

Linear-in-parameters models

A special case of nonlinear models is that when the model is linear in parameters:

$$\hat{y}(k|p) = p^T \cdot g^*(k, D[1, k-1])$$

It can be written in a linear model form using auxiliary variables, then the least-squares parameter estimation method can be applied **without the guarantee of asymptotic unbiasedness**!

Example 1:

$$y(k) = a_1 y^2(k-1) + b_0 u^3(k) + e(k)$$

- the parameter vector: $p = [a_1\ b_0]^T$,
- the uxiliary variables: $y^2(k-1) = z(k-1)$, $u^3(k) = w(k)$
- the regressor: $\varphi(k) = [z(k-1)\ w(k)]^T$
- the linear model: $\hat{y}(k|p) = a_1 z(k-1) + b_0 w(k) = p^T \varphi(k)$

Example 2:

$$y(k) = p_1^2 y(k-1) + \frac{p_1}{p_2} y(k-2) + p_3 u(k) + e(k)$$

- the auxiliary parameters and the parameter vector: $\theta_1 = p_1^2, \quad \theta_2 = \frac{p_1}{p_2}, \quad \theta_3 = p_3, \; \theta = [\theta_1 \; \theta_2 \; \theta_3]^T$
- the regressor: $\varphi(k) = [y(k-1) \; y(k-2) \; u(k)]^T$
- the linear model: $\hat{y}(k|p) = \theta^T \varphi(k)$

Task

Estimate the parameters of Example 1. The data can be found in `D2.`

- First you need to crearte the new regressor vector $\varphi(k) = [y^2(k-1) \; u^3(k)]^T$, let $\varphi(1) = [0 \; 0]^T$.
- Then compute the LS estimate with the new regressor, using the formula:

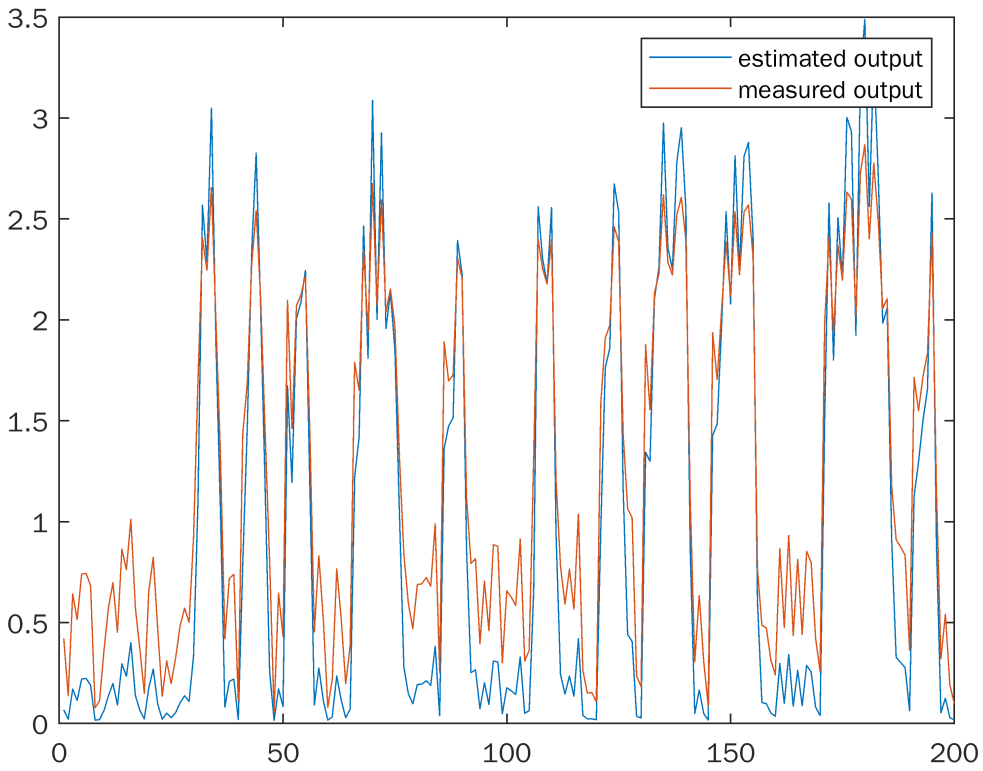
$$\hat{p}_{LS} = \left[\frac{1}{N}\sum_{k=1}^{N} \varphi(k)\varphi^T(k)\right]^{-1} \frac{1}{N}\varphi(k)y(k)$$

- Compute and plot the residuals.

```
phi=zeros(length(D2.u),2);
phi(1,:)=[(D2.y(1))^2,0];
phi(2:end,:)=[(D2.y(2:end)).^2,(D2.u(1:end-1)).^3];
P=LS_est([D2.y,phi]);
```


```
yest=phi*P;
figure
```

```
plot(yest)
hold on
plot(D2.y)
legend('estimated output','measured output')
```




```
% phi(:,1)=[0;0];
% for k=1:length(u4)-1
%     phi(:,k+1)=[D2.y(k).^2 ;D2.u(k+1).^3];
% end
%
% R=zeros(2);
% Y=[0;0];
% N=length(phi);
% for i=1:length(phi)
%     R=R+phi(:,i)*phi(:,i)';
%     Y=Y+phi(:,i)*y(i);
% end
% P2=inv(1/N*R)*1/N*Y;
% y2=P2'*phi;
% figure
% plot(y2)
% hold on
% plot(D2.y)
% figure
% plot(D2.y-yest);
```

Nonlinear models

$$\hat{y}(k|p) = g(k, D[1, k-1]; p)$$

Measured values: $D[1, N] = D^N = \{(y(k), u(k))|k = 1, \dots, N\}$

Prediction error: $\varepsilon(k, p) = y(k) - \hat{y}(k|p)$

Basic idea: minimizing the prediciton error

- Norm of the prediciton error: $V_N(p, D^N) = \frac{1}{N}\sum_{k=1}^{N} \ell(\varepsilon(k, p))$
- Find: $\hat{p}_N = \hat{p}_{D^N} = arg\min_p V_n(p, D^N)$
- Nonlinear optimization problem

Solutions:

- solving a set of nonliear equations
- minimizing the loss function (e.g. gradient method)

Implementing the gradient method

Inputs:

- initial value of the parameter x_0
- accuracy limit ε
- step size δ

Algorithm:

1. Initialization: $i = 0; \quad x_i = x_0;$
2. Compute the gradient vector $V_x(x_i) = \nabla V^T(x_i)$ in the point x_i
3. If $||V_x(x_i)|| < \varepsilon$ then $x_{min} = x_i$
4. Else $x_{i+1} = x_i - \delta V_x x_i, \quad i = i + 1;$
5. Continue with step 2.

Implementation

- Input arguments: `fun` - function to minimize, `p0` - initial value, `D` - measured data, `tol` - tolerance, accuracy limit, `step` - initial step size, `max_iter` - maximum number of iterations
- Output arguments: `p` - estimated parameter, `V` - value of the loss function at `p`, `p_hist` - history of parameter values where the loss function was evaluated, `V_hist` - history of loss function values during the algorithm, `grad_hist` - history of loss function gradients, `iter` - number of iterations at the end of the algorithm

1. First initialize the variables `p0`, `iter`, `V`, `dV`, `p_hist`, `fun_hist`, `grad_hist`. `feval(fun,p)` is used to evaluate the function `fun` at the point `p`. The result is the function value `V` and its gradient `dV` at point `p`. The function `fun` will be implemented later.
2. Create a `while` loop in wihich the iteration steps are executed. The iteration continues until the norm of the gradient is greater than the tolerance and the iteration number is smaller than the predefined maximum.
3. In the loop the following operations are performed:

- The value of the loss function and its gradient is calculated at the current point `p`.
- The new parameter value is calculated by taking a step in the direction of the gradient.
- The next step is choosing a new step size: in this example the step size is either duplicated or halved. The new step size is chosen in a way, that the new loss function value be smaller.
- After that the history of the parameters, loss function values and the gradients are updated.
- Finally the iteration variable is increased.

```
function [p,V,p_hist,fun_hist,grad_hist,iter]=gradient(fun,p0,D,tol,step, max_iter)
    p=p0;
    iter=0;
    [V,dV]=feval(fun,p,D);
    p_hist=p;
    fun_hist=V;
    grad_hist=dV;
    while(norm(dV,2)>tol)&&(iter<max_iter)
        [V,dV]=feval(fun,p,D);
        p=p-step*dV;
        if feval(fun,p-0.5*step*dV,D)<feval(fun,p-2*step*dV,D)
            step=0.5*step;
        else
            step=2*step;
        end
        p_hist=[p_hist;p];
        fun_hist=[fun_hist;V];
        grad_hist=[grad_hist;dV];
        iter=iter+1;
    end
end
```

Task

Let the nonlinear model given in the following form:

$$y(k) = p_1^2 \cdot 0.1 \cdot y(k-1) + p_2^2 \cdot 0.2 \cdot y(k-2) + u(k-1) + e(k)$$

The measured y and u data can be found in D3.

In order to estimate the parameters using the previously created gradient function, the loss function need to be implemented.

$$V(p_1,p_2)=\frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(y(k)-p_1^2\cdot 0.1\cdot y(k-1)-p_2^2\cdot 0.2\cdot y(k-2)-u(k-1))^2$$

The gradient of the loss function:

$$\frac{dV}{dp_1}=\frac{1}{N}\sum_{k=1}^{N}(y(k)-p_1^2\cdot 0.1\cdot y(k-1)-p_2^2\cdot 0.2\cdot y(k-2)-u(k-1))(-2p_1\cdot 0.1\cdot y(k-1))$$

$$\frac{dV}{dp_2}=\frac{1}{N}\sum_{k=1}^{N}(y(k)-p_1^2\cdot 0.1\cdot y(k-1)-p_2^2\cdot 0.2\cdot y(k-2)-u(k-1))(-2p_2\cdot 0.2\cdot y(k-2))$$

Implemeting the loss function:

```
function [f,df]=V2(P,D)
%y(k+1)=p1^2*0.1*y(k)+p2^2*0.2*y(k-1)+u(k)+e(k)
y=D(:,1);
u=D(:,2);
p1=P(1);
p2=P(2);
    N=length(y);
    tmp=0;
    tmp2=[0 0];
    for k=1:N-2
        tmp=tmp+(y(k+2)-p1^2*0.1*y(k+1)-p2^2*0.2*y(k)-u(k+1))^2;
        %gradients
        dVp1=2*(y(k+2)-p1^2*0.1*y(k+1)-p2^2*0.2*y(k)-u(k+1))*(-2*p1*0.1*y(k+1));
        dVp2=2*(y(k+2)-p1^2*0.1*y(k+1)-p2^2*0.2*y(k)-u(k+1))*(-2*p2*0.2*y(k));
        tmp2=tmp2+[dVp1 dVp2];
    end
    f=1/(2*(N))*tmp;
    df=1/(2*(N))*tmp2;
end
```

Minimizing functions (fminsearch - Optimization toolbox)

The `fminsearch` function in the Optimization Toolbox can be used to find the minimum value of a function, using a simplex method. The `fminsearch` can minimize functions defined as either a Matlab function or an anonymus function.

- `x=fminsearch(fun,x0)` tries to find the local minimum of the function `fun` starting from `x0`
- `x=fminsearch(fun,x0,options)` additional options can be defined in the options field, using Name - Value pair arguments.
- `[x,fval]=fminsearch(fun,x0,_)` returns the function value at the local minimum, too.
- some `options` and their possible values: `MaxFunEvals` - number of maximum function evaluations (positive integer), `MaxIter` - maximum number of iterations (positive integer), `TolFun` - termination tolerance on the function value (positive scalar), `TolX` - termination tolerance on `x` (positive scalar)

Example 1

Estimate the parameter of the following model:

$$y(k) = \frac{1}{exp(a)} y(k-1) + 5u(k-1) + e(k)$$

Measured input-output data can be found in `D4`. The first column contains the measured `y`, and the second column contains the measured `u`.

Steps of solution:

1. Create the quadratic loss function. Global variables are needed to access the measured data outside the function.
2. Minimize it using `fminsearch`.

```
function [f]=V(a)
%y(k+1)=exp(-a)*y(k)-5*u(k)+e(k)
global y u;
    N=length(y);
    tmp=0;
    for k=1:N-1;
        tmp=tmp+(y(k+1)-exp(-a)*y(k)+5*u(k))^2;
    end
    f=1/(2*(N-1))*tmp;
end
```

```
y4=D4(:,1);
u4=D4(:,2);
a0=1;
a=fminsearch('V',a0);
```

Example 2

Functions can be defined as anonymus functions. It is recommended when the model is not dynamic. An anonymous function is a function that is *not* stored in a program file, but is associated with a variable whose data type is `function_handle`. Anonymous functions can accept multiple inputs and return one output. They can contain only a single executable statement.For example

$$y = \sqrt{p_1 x_1 - p_2 x_1 x_2}$$

```
x1=D5(:,2);
x2=D5(:,3);
y5=D5(:,1);
modelfun=@(p)sqrt(p(1)*x1-p(2)*x1.*x2);
costfun=@(p)1/2*sum(y5-modelfun(p)).^2; %sum of squares cost function
p0=[1;1];
P2=fminsearch(costfun,p0);
```

Estimating nonlinear ARX models (nlarx - System Identification toolbox)

Nonlinear ARX models can be estimated with the nlarx functions from the System Identification toolbox. Different types of regressors can be used, for example polynomial, or custom regressors.

The

- `sys=nlarx(Data, Orders)`: estimates a nonlinear ARX model to fit the given estimation data using the specified orders and a default wavelet network nonlinearity estimator. With the model orders and delay the set of standard regressors are created.
- `sys=nlarx(Data, Orders,Nonlinearity)`: specifies the nonlinearity to use for model estimation. (E.g. `'wavenet'` (default) | `'sigmoidnet'` | `'treepartition'` | `'linear'` | nonlinearity estimator object | array of nonlinearity estimator objects)
- `sys=nlarx(_,options)`: specifies additional configuration options for the model estimation, e.g. define custom regressors.

- `options: 'CustomRegressors':` cell array of character vectors | array of `customreg` objects. Regressors constructed from combinations of inputs and outputs, specified as the comma-separated pair consisting of `'CustomRegressors'` and one of the following for single-output systems:
- Cell array of character vectors. For example: `{'y1(t-3)^3','y2(t-1)*u1(t-3)','sin(u3(t-2))'}.` Each character vector must represent a valid formula for a regressor contributing towards the prediction of the model output. The formula must be written using the input and output names and the time variable name as variables.
- Array of custom regressor objects, created using `customreg` or `polyreg`.

Example

Estimate the parameters of the following model:

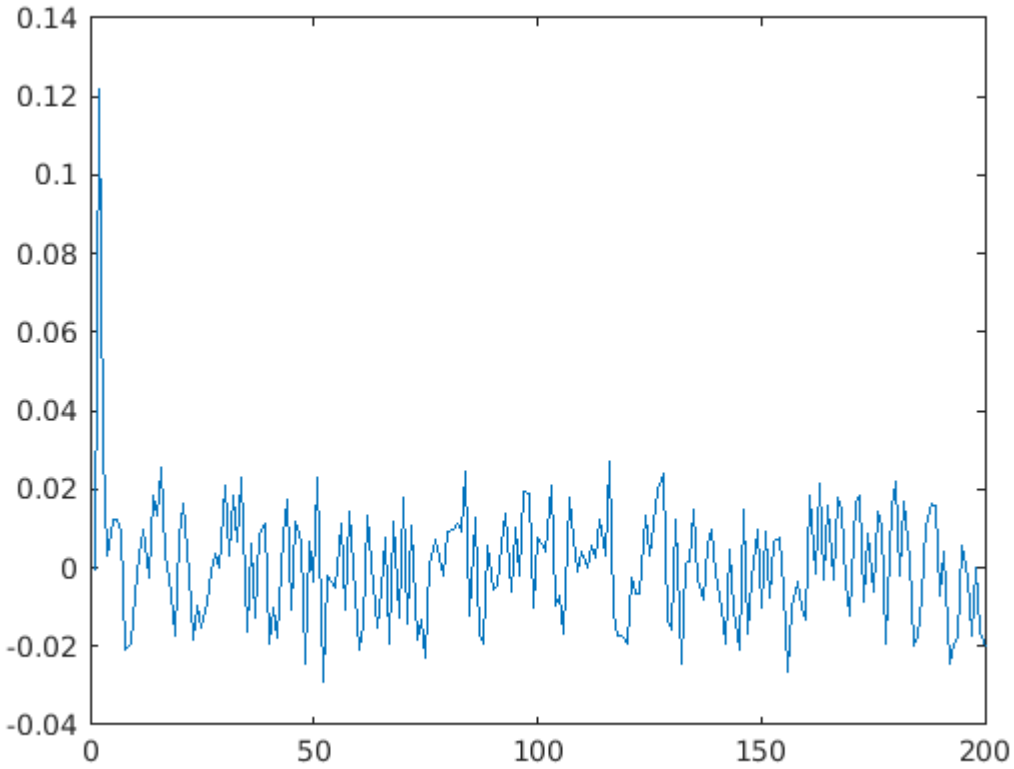
$$y(k) = p_1 y^2(k-1) + p_2 y(k-1)u(k-1) + p_3 u^2(k-2)$$

The measurement data can be found in `D6.`

```
C={'y1(t-1)^2','y1(t-1)*u1(t-1)','u1(t-2)^2'}
```

```
C = 1×3 cell
'y1(t-1)^2' 'y1(t-1)*u1(t-1)''u1(t-2)^2'
```

```
sys6=nlarx(D6,[0 0 0],'linear','CustomRegressors',C);
P6=getpvec(sys6);
y6=sim(sys6,D6(:,2));
figure
plot(D6(:,1)-y6)
```



HOMEWORK (Deadline 2020. November 18. 10:00)

Estimate the parameters of the following model.

$$y(k) = p_1 \cdot (x_1(k) - x_2(k))^3 + p_2 \cdot (x_1(k) \cdot x_2(k))$$

- Create a quadratic cost function, and minimize it with `fminsearch`. The measurement data can be found in `D7`. (1st column: y, 2nd column: x1, 3rd column: x2)

Send the created script file (NEPTUNKOD_HW4.m) to pozna.anna@virt.uni-pannon.hu!