SC42025 Filtering & Identification

MATLAB EXERCISE HW3

Instructions: Fill in the live script with your code and answers. Then export the live script as a pdf (in the 'Live Editor tab', click on the arrow under 'Save' and then 'Export to pdf').

General Hints:

- The live script is devided in several sections. You can **execute the sections seperately.**
- To reduce the runtime, you can comment the test-functions. Make sure, that all test-functions are uncommented when you export the live script and that their output appears on the pdf. **Missing test-function output will be treated as "wrong" regardless of the implementation.**
- Please **do not change given variable names or variables or any other code that you are asked to not change**. Otherwise the functionality of the live script and the testfunctions cannot be guaranteed.
- If you decide to implement your functions in seperate files, make sure to copy the code to the bottom of the script before you export it. **Missing code will be treated as "wrong" regardless of the result of the testfunction.**
- **Please read the cheat sheet carefully.**

Table of Contents

Consider the following Output Error system:

$$y(k) = \frac{b_2 q^{-2} + b_3 q^{-3} + b_4 q^{-4}}{1 + a_1 q^{-1} + a_2 q^{-2} + a_3 q^{-3} + a_4 q^{-4}} u(k) + e(k).$$

```
close all; clear; clc;
addpath('./function_folder')

load iodata.mat
```

Questions

a) Parameterize the system using the observable canonical form (according to the lecture slides!).

Answer:

$y(k) = H^{-1}\hat{G}u(k) + [1 - H^{-1}]e(k)$

$H = 1 \Rightarrow H^{-1} = 1 \Rightarrow \hat{G} = G, \hat{H} = 0$

As such the K matrix is 0, and the prediction is only based on u(k).

$$A = \begin{bmatrix} -a_1 & 1 & 0 & 0 \\ -a_2 & 0 & 1 & 0 \\ -a_3 & 0 & 0 & 1 \\ -a_4 & 0 & 0 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 0 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \qquad x_0 = \begin{bmatrix} x_{1,0} \\ x_{2,0} \\ x_{3,0} \\ x_{4,0} \end{bmatrix}$$

$$C = [1 \quad 0 \quad 0 \quad 0] \qquad D = [0]$$

$$\theta_A = \begin{bmatrix} -a_1 & -a_2 & -a_3 & -a_4 \end{bmatrix}$$

$$\theta_B = \begin{bmatrix} b_2 & b_3 & b_4 \end{bmatrix} \qquad \theta_{x_0} = \begin{bmatrix} x_{1,0} & x_{2,0} & x_{3,0} & x_{4,0} \end{bmatrix}$$

b) How many parameters do we have in part (a) ($\theta \in \mathbb{R}^p$)*?* Represent as $\theta = \begin{pmatrix} \theta_A \\ \theta_B \\ \theta_{x_0} \end{pmatrix}$.

Answer: From the above equations 11 unknowns can be counted.

c) We are now going to implement a Prediction Error method (`pem.m`) for the Output Error system. We will do this in the following four steps:

I. Derive expressions for $\frac{\partial \overline{A}(\theta)}{\partial \theta_p^{(i)}}, \frac{\partial \overline{B}(\theta)}{\partial \theta_p^{(i)}}, \frac{\partial K(\theta)}{\partial \theta_p^{(i)}}, \frac{\partial C(\theta)}{\partial \theta_p^{(i)}}, \frac{\partial D(\theta)}{\partial \theta_p^{(i)}}, \frac{\partial x_0(\theta)}{\partial \theta_p^{(i)}}$ (for all p).

$$\frac{\partial \overline{A}(\theta)}{\partial \theta_p^{(i)}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \cdots$$

$$\frac{\partial \overline{B}(\theta)}{\partial \theta_p^{(i)}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$\frac{\partial K(\theta)}{\partial \theta_p^{(i)}}$=zeros(4, p)

$\frac{\partial C(\theta)}{\partial \theta_p^{(i)}}$=zeros(1, 4*p)

$\frac{\partial D(\theta)}{\partial \theta_p^{(i)}}$ = zeros(1, p)

$$\frac{\partial x_0(\theta)}{\partial \theta_p^{(i)}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

II. on the result in part (a), implement a MATLAB function that computes the state-space matrices from the parameter vector θ. (*Look at* `theta2matrices.m` *in the provided template*).

```
[~] = test_theta2matrices;
```

```
Nice job! theta2matrices is correct!
```

III. Write a function `simsystem.m` that simulates a dynamic system given any input vector u and matrices A, B, C, D, as well as initial condition $x(0)$.

```
[~] = test_simsystem;
```

```
Nice job! simsystem is correct!
```

IV.

- Implement a function `jacobian.m` that calculates the Jacobian vector.

```
[~] = test_jacobian;
```

```
Nice job! jacobian is correct!
```

- Implement a function `hessian.m` that calculates the approximated Hessian matrix.

```
[~] = test_hessian;
```

```
Nice job! hessian is correct!
```

Note: The convergence loop for the regularized Gauss-Newton algorithm is already implemented in the `pem.m` function. There is no reason to edit this.

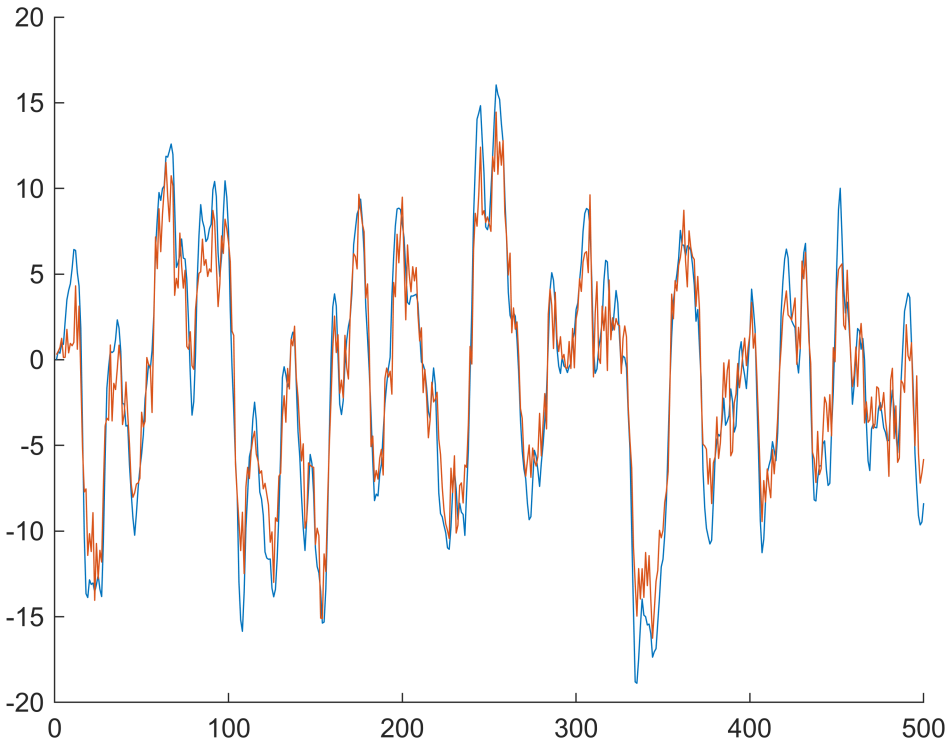
d) Choose two different initial guesses for θ. For each guess, train two models such that the first model is obtained from the first 500 samples and the second one is obtained from the first 1000 samples (from the given iodata.mat). Then, apply all four identified models on **the validation set** and plot the predicted output of all models (only for the validation set) in one figure. Add the numerical values of the VAF and the RMSE to the legend of this figures. (Do not forget to label the models properly!)

```
% split data

ut1 = u(1:500);            % input for train-set 1
ut2 = u(1:1000);           % input for train-set 2
uv = u(1001:end);          % input for validation-set

yt1 = ymeas(1:500);        % output for train-set 1
yt2 = ymeas(1:1000);       % output for train-set 2
yv = ymeas(1001:end);      % output for validation-set
```

```
theta1 = [1, 0, -0.1, 0, 1, 1, 1, 0, 0.5, 0, 0.5]';
y_ref = yt1;
u_ref = ut1;
% theta = [0 0 0 0 0 0 0 0 0 0 0 ]';
[A, B, C, D, x0] = theta2matrices(theta1);
[y_sim, x_sim] = simsystem(A, B, C, D, x0, u_ref);
theta2 = [0.9,-0.05, -0.5, 0.5, 1, 1, 1, 0.3, 0.2, 0.3, -0.01]';
y_ref = yt1;
u_ref = ut1;
% [y_sim, x_sim] = simsystem(A, B, C, D, x0, u_ref);
[A2, B2, C2, D2, x02] = theta2matrices(theta2);
% [y_sim, x_sim ] = simsystem(A2, B2, C2, D2, x02, u_ref);
figure;
hold on
plot(y_sim)
plot(y_ref)
hold off
```




```
[A_s1s, B_s1s, C_s1s, D_s1s, x0_s1s, J_s1s, H_s1s] = pem(theta1, A, B, C, D, x0,
yt1(), ut1(), 0.1, 500);
[A_s1l, B_s1l, C_s1l, D_s1l, x0_s1l, J_s1l, H_s1l] = pem(theta1, A, B, C, D, x0,
yt2(), ut2(), 0.1, 500);
```

```
Warning: Maximum iterations reached
```

```
[A_s2s, B_s2s, C_s2s, D_s2s, x0_s2s, J_s2s, H_s2s] = pem(theta2, A2, B2, C2, D2,
x02, yt1(), ut1(), 0.1, 500);
[A_s2l, B_s2l, C_s2l, D_s2l, x0_s2l, J_s2l, H_s2l] = pem(theta2, A2, B2, C2, D2,
x02, yt2(), ut2(), 0.1, 500);
```

```
Warning: Maximum iterations reached
```

```
VAF = @(y_true, y_hat) 1 - norm(y_true-y_hat)/norm(y_true);
RMSE = @(y_true, y_hat) sqrt(sum((y_true-y_hat) .* (y_true-y_hat)) / size(y_true,
1));


[y_s1s, x_s1s] = simsystem(A_s1s, B_s1s, C_s1s, D_s1s, x0_s1s, uv);
VAF_s1s = VAF(yv, y_s1s)
```

```
VAF_s1s = 0.7457
```

```
RMSE_s1s = RMSE(yv, y_s1s)
```

```
RMSE_s1s = 1.4992
```

```
[y_s1l, x_s1l] = simsystem(A_s1l, B_s1l, C_s1l, D_s1l, x0_s1l, uv);
VAF_s1l = VAF(yv, y_s1l)
```

```
VAF_s1l = 0.7457
```

```
RMSE_s1l = RMSE(yv, y_s1l)
```

```
RMSE_s1l = 1.4989
```

```
[y_s2s, x_s2s] = simsystem(A_s2s, B_s2s, C_s2s, D_s2s, x0_s2s, uv);
VAF_s2s = VAF(yv, y_s2s)
```

```
VAF_s2s = 0.7435
```

```
RMSE_s2s = RMSE(yv, y_s2s)
```

```
RMSE_s2s = 1.5122
```

```
[y_s2l, x_s2l] = simsystem(A_s2l, B_s2l, C_s2l, D_s2l, x0_s2l, uv);
VAF_s2l = VAF(yv, y_s2l)
```

```
VAF_s2l = 0.7457
```

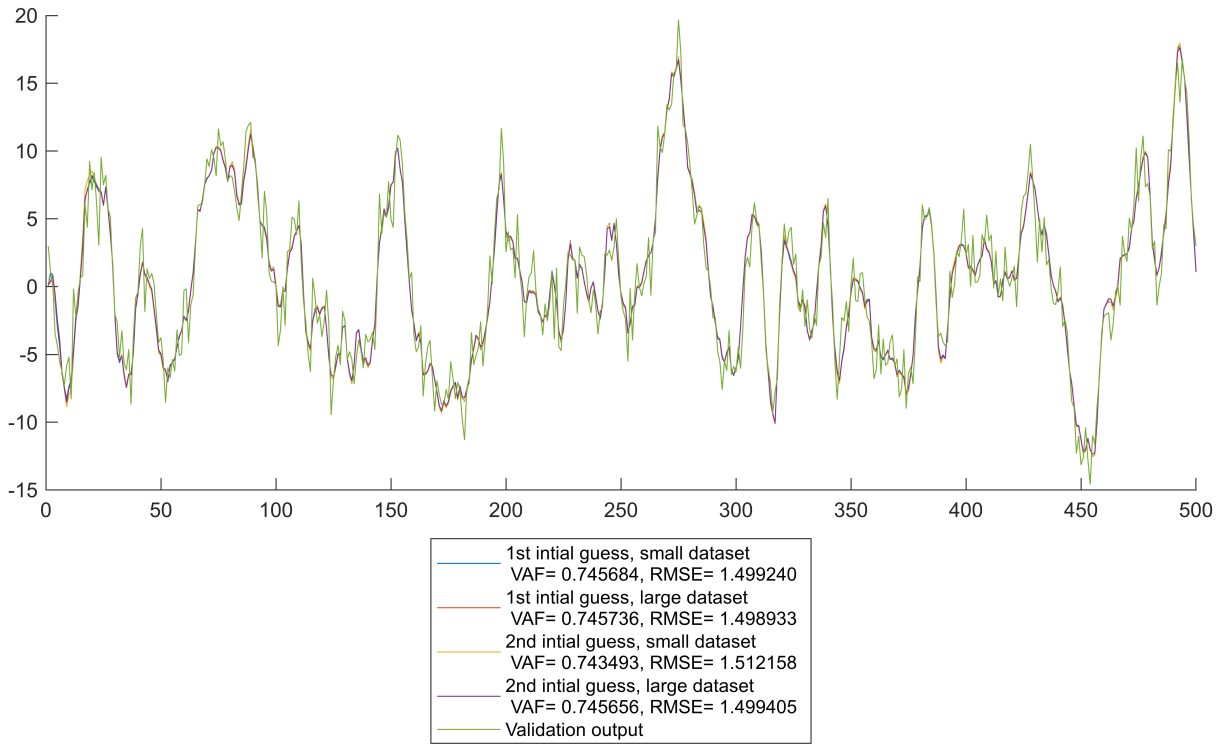
```
RMSE_s2l = RMSE(yv, y_s2l)
```

```
RMSE_s2l = 1.4994
```

```
txt_s1s = sprintf("1st intial guess, small dataset\n VAF= %f, RMSE= %f", VAF_s1s,
RMSE_s1s);
txt_s1l = sprintf("1st intial guess, large dataset\n VAF= %f, RMSE= %f", VAF_s1l,
RMSE_s1l);
```

```
txt_s2s = sprintf("2nd intial guess, small dataset\n VAF= %f, RMSE= %f", VAF_s2s,
RMSE_s2s);
txt_s2l = sprintf("2nd intial guess, large dataset\n VAF= %f, RMSE= %f", VAF_s2l,
RMSE_s2l);


hF = figure;
clf;
hold on
plot(y_s1s)
plot(y_s1l)
plot(y_s2s)
plot(y_s2l)
plot(yv)
legend(txt_s1s, txt_s1l, txt_s2s, txt_s2l, "Validation output", location =
"southoutside");
hF.Position(3:4) = [960 540];
hold off
```



e) According to the result in (d), report the system parameters a_1, a_2, a_3, a_4, b_2, b_3, b_4 for the best identified model. Explain how you chose the best model.

Answer:

The starting points were selected based on trial and error and only this combination yielded convergence, where $a_1 \approx 1, a_2 \approx 0, -a_1 < a_3 < 0, a_4 \approx 0$, these starting points all converged to the same final solution, with getting into a local minima. As no relaxation was applied during the newton method the solution got into an oscillating part in some of the runs. The solution selected finally is the first initial guess trained on the large dataset, as that gives the lowest RMSE out of the four. For this $a_1 = -2.3770, a_2 = 2.0548, a_3 = -0.7654, a_4 = 0.1042, b_2 = 0.9966, b_3 = -0.8053, b_4 = 0.2154$

Functions

Implement your functions in the templates below.

You may implement additional functions for your convenience.

```
function [Abar,Bbar,C,D,x0] = theta2matrices(theta)
%%
% Function INPUT
% theta     Parameter vector (vector of size: according to the realization)
%
%
% Function OUTPUT
% Abar      System matrix A (matrix of size n x n)
% Bbar      System matrix B (matrix of size n x m)
% C         System matrix C (matrix of size l x n)
% D         System matrix D (matrix of size l x m)
% x0        Initial state (vector of size n x one)




%%%%%% YOUR CODE HERE %%%%%%
theta_a = theta(1:4);
theta_b = theta(5:7);
theta_x0 = theta(8:11);

Abar=[theta_a(1) 1 0 0;
      theta_a(2) 0 1 0;
      theta_a(3) 0 0 1;
      theta_a(4) 0 0 0];
Bbar=[ 0; theta_b(1); theta_b(2); theta_b(3)];
C=[1 0 0 0];
D=[0];
x0=[theta_x0(1); theta_x0(2); theta_x0(3); theta_x0(4)];
end
```

```
function [y, x] = simsystem(A, B, C, D, x0, u)
% Instructions:
% Simulating a linear dynamic system given input u, matrices A,B,C,D ,and
% initial condition x(0)
%
 n = size(A, 1);
 m = size(B, 2);
 l = size(C, 1);
 N = size(u, 1);
%
% Function INPUT
% A system matrix (matrix of size n x n)
% B system matrix (matrix of size n x m)
% C system matrix (matrix of size l x n)
% D system matrix (matrix of size l x m)
% x0 initial state (vector of size n x one)
% u system input (matrix of size N x m)
%
% Function OUTPUT
% x state of system (matrix of size N x n)
% y system output (matrix of size N x l)




%%%%%% YOUR CODE HERE %%%%%%
y = zeros(l, N);
x = zeros(n, N+1);

x(:, 1) = x0;

for i = 1:size(u, 1)
    x(:, i+1) = (A * x(:, i) + B * u(i, :));
    y(:, i) = (C * x(:, i) + D * u(i, :));
end

x = x(:, 1:N)';
y = y';
end
```

```
function J = jacobian(psi,E)
% Instructions:
% This function calculates the Jacobian vector in the Gauss-Newton
```

```
% algorithm
%
%  p = size(theta);
%  N = size(y, 1);
%  l = size(C, 1);
%
% Function INPUT
% psi derivative of estimation error wrt theta (matrix of size l*N x p)
% E estimation error vector (vector of size l*N x one)
%
% Function OUTPUT
% J Jacobian vector (vector of size p x one)




%%%%%% YOUR CODE HERE %%%%%%
N = size(psi, 1);
J = 2/N * psi' * E;
end
```

```
function H = hessian(psi)
% Instructions:
% This function calculates the approximated Hessian matrix in the Gauss-Newton
% algorithm
%
%  p = size(theta);
%  N = size(y, 1);
%  l = size(C, 1);
%
% Function INPUT
% psi derivative of estimation error wrt theta (matrix of size l*N x p)
%
% Function OUTPUT
% H Hessian matrix (matrix of size p x p)




%%%%%% YOUR CODE HERE %%%%%%
N = size(psi, 1);
H = 2/N * (psi' * psi);
end
```

```
function [Abar,Bbar,C,D,x0, J, H] = pem(theta,A0,B0,C0,D0,x00,y,u,lambda,maxiter)
% Instructions:
% Implement your Prediction Error Method for the Output Error system here.
% Use the following function inputs and outputs.
%
% Function INPUT
% theta  Paramter vector (size: depends on your mapping choice)
% A0 Initial guess for system matrix A (matrix of size n x n)
% B0 Initial guess for system matrix B (matrix of size n x m)
% C0 Initial guess for system matrix C (matrix of size l x n)
% D0 Initial guess for system matrix D (matrix of size l x m)
% x00 Initial guess for initial state (vector of size n x one)
% u System input (matrix of size N x m)
% y System output (matrix of size N x l)
% lambda regularization parameter (scalar)
% maxiter Maximum number of iterations (scalar)
%
%
% Function OUTPUT
% Abar Estimate of system matrix A (matrix of size n x n)
% Bbar Estimate of system matrix B (matrix of size n x m)
% C Estimate of system matrix C (matrix of size l x n)
% D Estimate of system matrix D (matrix of size l x m)
% x0 Estimate of initial state (vector of size n x one)




%%%%%% YOUR CODE HERE %%%%%%
% check the given inputs for consistency
[A, B, C, D, x0] = theta2matrices(theta);
assert(isequal(A, A0))
assert(isequal(B, B0))
assert(isequal(C, C0))
assert(isequal(D, D0))
assert(isequal(x0, x00))
% traj = simsystem(A, B, C, D, x0);


p = size(theta, 1);
N = size(y, 1);
n = size(A, 1);


% Gauss-Newton
```

```
% initialize
converged = false;
maxiter_reached = false;
iter = 1;
e_hist = [];
while ~converged && ~maxiter_reached
    [A, B, C, D, x0] = theta2matrices(theta);
    [y_sim, x_sim] = simsystem(A, B, C, D, x0, u);
    E = y-y_sim;
    e_hist = cat(1, e_hist, norm(E));
    psi = setup_psi(A, C, x_sim, u);
    jaco = jacobian(psi, E);
    hess = hessian(psi);
    % size(jaco)
    % size(theta)
    % size(theta - inv(hess + lambda * eye(p)) * jaco)
    % size(inv(hess + lambda * eye(p)) * jaco)
    theta_new = theta + (hess + lambda * eye(p)) \ jaco;
    % theta_new'
    if anynan(psi)
        % psi
        disp('nans in psi')
        break
    end
    if anynan(theta_new)
        disp('nans in theta:')
        % theta_new
        % psi
        % hess
        % A
        % (hess + lambda * eye(p))
        break
    end
    % Check convergence
    if norm(theta_new - theta) < 1e-3
        converged = true;
    elseif iter == maxiter
        maxiter_reached = true;
        warning('Maximum iterations reached');
    end
    theta = theta_new;
    iter = iter + 1;
end
e_hist';
theta;

%%%%%% YOUR CODE HERE %%%%%%
[Abar, Bbar, C, D, x0] = theta2matrices(theta);
J = jaco;
```

```
H = hess;

end

function [psi] = setup_psi(A, C, x, u)
    N = size(x, 1);
    p = 11;
    n = 4;

    psi = zeros(N, p);
    dA_dtheta = zeros(n, n, p);
    dA_dtheta(1:4, 1, 1:4) = eye(4);

    dB_dtheta = zeros(n, 1, p);
    dB_dtheta(2:4, 5:7) = eye(3);

    dx0_dtheta = zeros(n, 1, p);
    dx0_dtheta(:, 1, 8:11) = eye(4);

    for p = 1:4
        dx_dtheta = dx0_dtheta(:, 1, p);
        for j = 1:N
            psi(j, p) = C * dx_dtheta;
            p1 = A * dx_dtheta;
            p2 = dA_dtheta(:, :, p) * x(j, :)';
            dx_dtheta = p1 + p2;
            if anynan(psi)
                disp('nans in A part {i}')
            end
        end
    end
    for p = 5:7
        dx_dtheta = dx0_dtheta(:, 1, p);
        for j = 1:N
            psi(j, p) = C * dx_dtheta;
            p1 = A * dx_dtheta;
            p3 = dB_dtheta(:, :, p)*u(j);
            dx_dtheta = p1 +p3;
        end
        if anynan(psi)
            disp('nans in B part {i}')
        end
    end
    for p = 8:11
        dx_dtheta = dx0_dtheta(:, 1, p);
        for j = 1:N
            psi(j, p) = C * dx_dtheta;
            p1 = A * dx_dtheta;
            dx_dtheta = p1;
        end
```

```
        if anynan(psi)
            disp('nans in x0 part {i}')
            dx0_dtheta(:, 1, p)
            A
        end
    end
end
```

(1) a)

$$x_1(k+1) = -a_1 x_1(k) + x_2(k) + b_1 u(k)$$

$$x_2(k+1) = -a_2 x_1(k) + b_2 u(k)$$

$$x_3(k+1) = -(a_1+d_1) x_3(k) + x_4(k) - (a_1+d_1) e(k)$$

$$x_4(k+1) = -(a_2+d_2+a_1 d_1) x_3(k) + x_5(k) - (a_2 d_1 + a_1 d_2) e(k)$$

$$x_5(k+1) = -(a_1 d_2 + a_2 d_1) x_3(k) + x_6(k) - (a_1 d_2 + a_2 d_1) e(k)$$

$$x_6(k+1) = -(a_2 d_2) x_3(k) - (a_2 d_2) e(k)$$

$y(k) = x_1(k) + x_3(k) + e(k)$, so y is related to (x_1, x_3)

the equations for x_1, x_2 form an observable canonical form for a transfer fn. (ARMAX)

$$\Downarrow$$

$$A = \begin{bmatrix} -a_1 & 1 \\ -a_2 & 0 \end{bmatrix} \quad B = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \Rightarrow X_1(k) = \frac{b_1 q^{-1} + b_2 q^{-2}}{1 + a_1 q^{-1} + a_2 q^{-2}} u(k)$$

same way for x_3 : $A = \begin{bmatrix} -(a_1+d_1) & 1 & 0 & 0 \\ -(a_2+d_2+a_1 d_1) & 0 & 1 & 0 \\ -(a_1 d_2 + a_2 d_1) & 0 & 0 & 1 \\ -a_2 d_2 & 0 & 0 & 1 \end{bmatrix}$ $B = \begin{bmatrix} -(a_1+d_1) \\ -(a_2+d_2+a_1 d_1) \\ -a_1 d_2 + a_2 d_1 \\ -(a_2 d_2) \end{bmatrix}$

$$F^e(q^{-1}) = \frac{-(a_1+d_1) q^{-1} - (a_2+d_2+a_1 d_1) q^{-2} + (a_1 d_2 + a_2 d_1) q^{-3} - a_2 d_2 q^{-4}}{1 + (a_1+d_1) q^{-1} + (a_2+d_2+a_1 d_1) q^{-2} + (a_1 d_2 + a_2 d_1) q^{-3} + (a_2 d_2) q^{-4}} =$$

$$= \frac{-(a_1 q^{-1} + a_2 q^{-2} + d_1 q^{-1} + d_2 q^{-2} + (a_1 q^{-1} \cdot b_1 q^{-1}) + (a_1 q^{-1} \cdot d_2 q^{-2} + a_2 q^{-2} \cdot d_1 q^{-1}) + a_2 q^{-2} d_2 q^{-2})}{1 + a_1 q^{-1} + a_2 q^{-2} + d_1 q^{-1} \cdot (1 + a_1 q^{-1} + a_2 q^{-2}) + d_2 q^{-2} (1 + a_1 q^{-1} + a_2 q^{-2})}$$

$$= \frac{-(1 + a_1 q^{-1} + a_2 q^{-2})(1 + d_1 q^{-1} + d_2 q^{-2})}{(1 + a_1 q^{-1} + a_2 q^{-2})(1 + d_1 q^{-1} + d_2 q^{-2})} = \frac{-A(q^{-1}) \cdot D(q^{-1})}{A(q^{-1}) \cdot D(q^{-1})}$$

$$F^u(q^{-1}) = \frac{B(q^{-1})}{A(q^{-1})} \cdot u(k) \quad F^e = \frac{1 - A(q^{-1})\,B(q^{-1})}{A(q^{-1})\,B(q^{-1})} \cdot e(k)$$

$$y(k) = F^u \cdot u(k) + F^e\, e(k) + e(k) = \frac{B(q^{-1})}{A(q^{-1})} u(k) - \frac{A(q^{-1})\,D(q^{-1})}{A(q^{-1})\,D(q^{-1})} e(k) +$$

$$+ \frac{1}{A(q^{-1})D(q^{-1})} e(k) + e(k) = \frac{B(q^{-1})}{A(q^{-1})} u(k) + \frac{1}{A(q^{-1})\,D(q^{-1})} e(k)$$

$$\Downarrow$$

$$A(q^{-1})\, y(k) = B(q^{-1})\, u(k) + \frac{1}{D(q^{-1})} e(k), \text{ where}$$

$$A(q^{-1}) = 1 + a_1 q^{-1} + a_2 q^{-2}$$

$$B(q^{-1}) = b_1 q^{-1} + b_2 q^{-2}$$

$$D(q^{-1}) = 1 + d_1 q^{-1} + d_2 q^{-2}$$

b) transfer fn predictor-for model $y(k) = G(q^{-1})\,u(k) + H(q^{-1})\,e(k)$

$$\hat{y} = H^{-1}(q^{-1})\,G(q^{-1})\,u(k) + \left(1 - H^{-1}(q^{-1})\right) e(k)$$

$$\hat{y}(k) = \underbrace{\frac{D(q^{-1})\,A(q^{-1})\,B(q^{-1})}{A(q^{-1})}}_{\hat{G}}\, u(k) + \underbrace{\left(1 - D(q^{-1})\,A(q^{-1})\right)}_{\hat{H}} e(k)$$

$$\hat{y}(k) = \left(1 + d_1 q^{-1} + d_2 q^{-2}\right)\left(b_1 q^{-1} + b_2 q^{-2}\right) u(k) + \left(1 - \left(1 + d_1 q^{-1} + d_2 q^{-2}\right)\right) \ldots$$

$\hat{G}$: no poles; 2 zeros for D, B each $\Rightarrow$ 4 zeros

$\hat{H}$: no poles; 2 zeros

predictor for state space form:

$$\hat{x}(k+1) = A\hat{x}(k) + Bu(k) + K\left(y_k - C\hat{x}(k) - Du(k)\right)$$

$$\hat{y}(k) = C\hat{x}(k) + Du(k)$$

0, 9

$$y(k) = \frac{b_1 q^{-1}}{1 + a_1 q^{-1}} u(k) + \frac{1 - c_1 q^{-1}}{1 + a_1 q^{-1}} e(k)$$

predictor: $\hat{y}_k = H^T \theta \, u(k) + (1 - H^T) y(k) =$

$$= \frac{(1 + a_1 q^{-1}) b_1 q^{-1}}{(1 + a_1 q^{-1})(1 - c_1 q^{-1})} u(k) + \left(1 - \left(\frac{1 + a_1 q^{-1}}{1 - c_1 q^{-1}}\right)\right) y(k) =$$

$$= \frac{b_1 q^{-1}}{1 - c_1 q^{-1}} u(k) + \frac{1 - c_1 q^{-1} - 1 - a_1 q^{-1}}{1 - c_1 q^{-1}} y(k) = \frac{b_1 q^{-1}}{1 - c_1 q^{-1}} u(k) + \frac{c_1 q^{-1} - a_1 q^{-1}}{1 - c_1 q^{-1}} y(k)$$

$$(1 - c_1 q^{-1}) \hat{y}_k = b_1 q^{-1} u(k) + (-c_1 q^{-1} - a_1 q^{-1}) y(k)$$

$$\hat{y}(k) - c_1 \hat{y}(k-1) = b_1 u(k-1) + (-c_1 - a_1) y(k-1)$$

$$\hat{y}(k) = c_1 \hat{y}(k-1) + b_1 u(k-1) + (-c_1 - a_1) y(k-1)$$

given $\Theta = (b_1 \quad -(c_1 + a_1) \quad c_1)^T$

$$\hat{y}(k) = (u(k-1) \; y(k-1) \; \hat{y}(k-1)) \cdot \Theta$$

if sufficiently many measurements $y(k)$ exist, then Φ has full rank, so the problem is well defined least-squares

b, $\frac{1}{1-c_1 q^{-1}}$ let $q^{-1} = x$

Taylor expansion: $f(x) = f(x_0) + \frac{f'(x_0)}{1!}(x-a) + \frac{f''(x_0)}{2!}(x-a)^2 + \dots$

$f(x) = \frac{1}{1-c_1 x}$ $f'(x) = \frac{c_1}{(1-c_1 x)^2}$ $f''(x) = c_1 \frac{-2\cdot(1-c_1 x)\cdot(-c_1)}{(1-c_1 x)^4} = \frac{2c_1^2}{(1-c_1 x)^3}$

suppose $f^n(x) = a \cdot \frac{1}{(1-c_1 x)^{n+1}}$, then

$f^{k+1}(x) = a \cdot \frac{-(1-c_1 x)^{n+1}}{(1-c_1 x)^{2(n+1)}} = a \cdot \frac{-(n+1)(1-c_1 x)^n \cdot (-c_1)}{(1-c_1 x)^{2(n+1)}} =$

$= \frac{a\cdot(n+1)(c_1)}{(1-c_1 x)^{n+2}} = f^n(x) \cdot \frac{-(n+1)(-c_1)}{(1-c_1 x)}$

since 1° it's supposition, by induction all subsequent f^n will be $\frac{(n!)\cdot(c_1)^n}{(1-c_1 x)^{n+1}}$

centering the series expansion at $x=0$, $\frac{f^n(0)}{n!}$ simplifies to c_1^n, so $m_n = c_1^n$ $m_0 = c_1$ $m_2 = c_1^2$ $m_3 = c_1^3$

if $|c_1 x| < 1$ the series converges, otherwise diverges to infinity, so $|x| \le \frac{1}{|c_1|}$

c)

$$\frac{1}{1+c_1 q^{-1}} = \lim_{p\to\infty} \frac{1}{\sum_{n=0}^{p} c_1^n q^{-n}}$$

$$y^{ARX}(k) = \frac{B(q^{-1})}{A(q^{-1})} u(k) + \frac{1}{(1+a_1 q^{-1}) \sum_{n=0}^{p} c_1^n q^{-n}} \qquad e(k), \text{ pero } \lim_{p\to\infty} y^{ARX} = y_k$$

$$\lim_{p\to\infty} \frac{1}{(1+a_1 q^{-1}) \sum_{n=0}^{p} c_1^n q^{-n}} = \frac{1}{A(q^{-1})}$$

$$A(q^{-1}) = \lim_{p\to\infty} (1+a_1 q^{-1}) \cdot \sum_{n=0}^{p} c_1^n q^{-n}$$

$$\frac{B(q^{-1})}{A(q^{-1})} = \frac{b_1 q^{-1}}{(1+a_1 q^{-1})} \Rightarrow B(q^{-1}) = \lim_{p\to\infty} b_1 q^{-1} \left(\sum_{n=0}^{p} c_1^n q^{-n} \right)$$

$\frac{B(q^{-1})}{A(q^{-1})}$ los p polos and zeros ato...

p cancellations $(p \to \infty)$

d) $$\lim_{p\to\infty} \hat{y}^{ARX}(k) = A(q^{-1}) \cdot \frac{B(q^{-1})}{A(q^{-1})} u(k) + \left(1 - A(q^{-1})\right) \cdot y^{ARX}(k)$$

$$\lim_{p\to\infty} \hat{y}^{ARX}(k) = \sum_{n=1}^{p} c_1^{n-1} b_1 u(k-n) + y^{ARX}(k) - \sum_{n=0}^{p} c_1^n (1+a_1 q^{-1}) q^{-n} =$$

$$= \sum_{n=1}^{p} c_1^{n-1} b_1 u(k-n) + y^{ARX}(k) - \sum_{n=0}^{p} c_1^n y^{ARX}(k-n) - \sum_{n=1}^{p} c_1^{n-1} a_1 y^{ARX}(k-n) =$$

$$= \sum_{n=1}^{p} c_1^{n-1} b_1 u(k-n) + \sum_{n=1}^{p} c_1^n y^{ARX}(k-n) - \sum_{n=1}^{p} c_1^{n-1} a_1 y^{ARX}(k-n)$$

$$\Theta = \begin{pmatrix} b_1 & c_1 - a_1 & c_1 b_1 & c_1 c_1 - c_1 a_1 & c_1^2 b_1 & c_1^2 c_1 - c_1^2 a_1 & \cdots \end{pmatrix}^T$$

$$\varphi = \begin{pmatrix} u(k-1) & y^{ARX}(k-1) & y^{ARX}(k-1) & \cdots \end{pmatrix}$$

both Θ and φ are infinite

2.1

$\lim_{p\to\infty}\left(y_k - y_k^{ARX}\right)$ the values of $B(q^{-1})$ and $A(q^{-1})$ were derived such, that $\lim_{p\to\infty} y_k^{ARX} = y_k$, what to prove here?

$$E(y_k) = E\left(\frac{b q^{-1}}{1+a_1 q^{-1}} u(k) + \frac{1-c_1 q^{-1}}{1+a_1 q^{-1}} e(k)\right)$$

$$(1+a_1 q^{-1})\, y_k = b q^{-1} u(k) + (1-c_1 q^{-1}) e(k)$$

$$y_k + a_1 y_{k-1} = b u(k-1) + e(k) - c_1 e(k-1)$$

$$y_{k-1} = b u(k-2) + e(k-1) - c_1 e(k-2) - a_1 y_{k-2}$$

$$y_k = b_1 u(k-1) + e(k) - c_1 e(k-1) - a_1\left(b_1 u(k-2) + e(k-1) - c_1 e(k-2) - a_1 (b_1 u(k-3)\ldots\right.$$

$$y_k = \sum_{n=1}^{p} b_1 \cdot a_1^{n-1} u(k-n) + \sum_{n=0}^{p} a_1^n e(k-n) - \sum_{n=1}^{p} a_1^{n-1} e(k-n) - a^p y_{k-p}$$

$$E[y_k] = b_1 \cdot \frac{1}{1-a_1} \cdot E[u(k-n)] - a^p \cdot E[y_{k-p}] = \frac{b_1 \bar{u}}{1-a_1}$$

$p\to\infty$: $E[u(k-n)] \to \bar{u}$, $a^p \to 0$