Submission of the Programming assignment of Estimation and Identification(EE551)

submitted

by

Yogesh Kumar Chauhan

Roll No 234102508

Dated:19-11-2023

Submitted to:

Prof. Indrani Kar

Department of Electronics and Electrical Engineering
Indian Institute of Technology Guwahati
November 2023

1 Assignment: Question 1

Kalman Filter Estimation Report

The provided MATLAB code implements a Kalman filter for state estimation. This Kalman filter aims to estimate the states of a dynamic system given noisy measurements.

System Equation Being Estimated

The system equation being estimated in the code is described by the following dynamics:

$$\begin{aligned} x_1(k+1) &= \frac{x_1(k)\cdot x_2(k)}{1+x_1^2(k)} + w_1(k) \\ x_2(k+1) &= \frac{x_2(k)}{1+x_1^2(k)} + w_2(k) \\ y(k) &= x_2(k) + v(k) \end{aligned}$$

where:
x$_1(k)$ and $x_2(k)$ represent the states of the dynamic system,
$w_1(k)$ and $w_2(k)$ are process noise terms with covariance Q,
$v(k)$ is measurement noise with covariance R,
$y(k)$ is the measurement obtained from $x_2(k)$ with added noise $v(k)$.

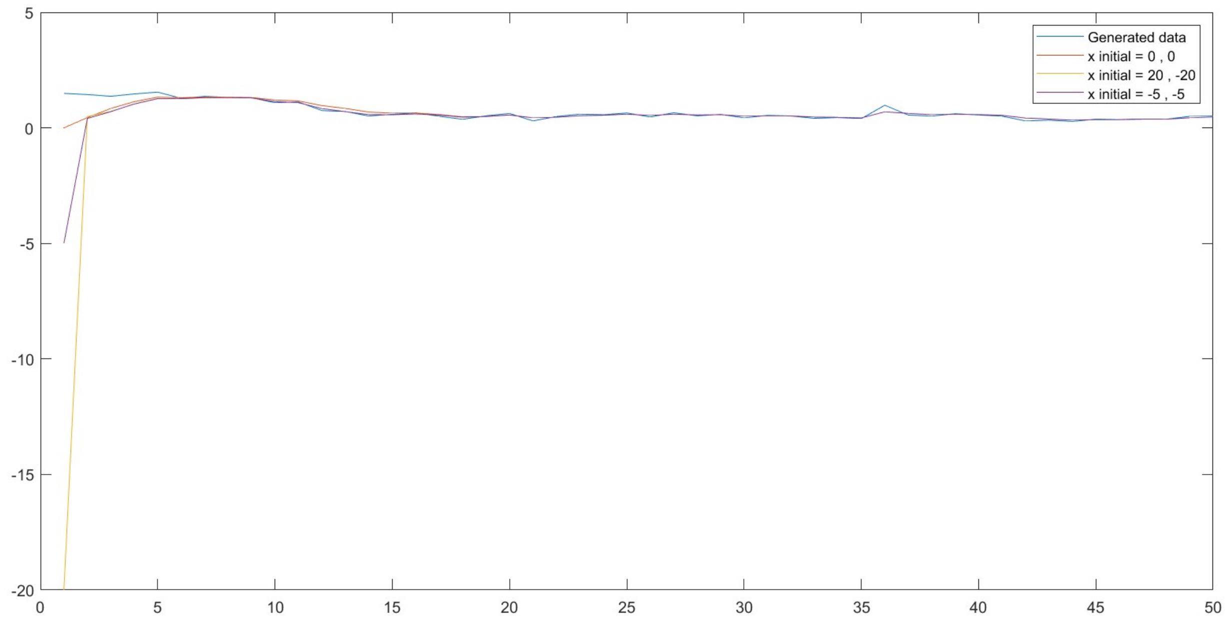
MATLAB Code

a) For different initial condition:

```
%-- .- -.. . / -... -.-- / -.-- --- --. . ... ....
%Date:19_11_2023
clc;clear all;close all;
N=50;   %number of time steps;
C=[0 1];
Q=[0.04 0;
    0 0.04];
R=0.09;
P0=1.5;
V0=0;
%initialise covariance matrix
p=[0 0;
    0 0];
y=[];
x=[V0,P0]';
w1=Q(1,1)*randn(N,1);
w2=Q(2,2)*randn(N,1);
v=R*randn(N, 1);
```

```
%input generation
x1=x(1);
x2=x(2);
%x1_vec=[x1];
%x2_vec=[x2];
for i=1:N
    x1new=(x1*x2)/(1+x1*x1)+w1(i);
    x2new=x2/(1+x1*x1)+w2(i);
    x1=x1new;
    x2=x2new;
%     x1_vec=[x1_vec;x1];
%     x2_vec=[x2_vec;x2];
    y=[y, x2+v(i)];
end
%y(k) to be arranged for each step
t=1:N;
plot(t, y, 'DisplayName',"Generated data");
hold on
clear x1 x2 x1new x2new w1 w2 v;
xt_initial={[0,0]', [20,-20]', [-5,-5]'};
    for f=1:length(xt_initial)
        xt=xt_initial{f};
        %Estimation_begin
        for k=2:N
            xt_ap=[(xt(1,k-1)*xt(2,k-1))/(1+xt(1,k-1)*xt(1,k-1)) xt(2,k-1)/(1+xt(1,k-1)*xt(1,k-
            A=[((1+3*xt_ap(1)*xt_ap(1))*xt_ap(2))/((1+xt_ap(1)*xt_ap(1))^2) xt_ap(1)/(1+xt_ap(1
                -2*xt_ap(1)*xt_ap(2)/((1+xt_ap(1)*xt_ap(1))^2)           1/(1+xt_ap(1)*xt_ap(1)
            %error c0variance propagation
            p_bar=A*p*A'+Q;

            %find kalman gain
            K=p_bar*C'*inv(C*p_bar*C'+R);

            %update state estimate
            xt(:,k)=xt_ap+K*[y(k)-C*xt_ap];

            %update covariance matrix
            p=(eye(2,2)-K*C)*p_bar;
            p_chol=chol(p);
            p=p_chol*p_chol';
        end
        t=1:N;
        %print_xt=num2str(xt_initial{f})
        plot(t,xt(2,:), 'DisplayName', ['x initial = ', num2str(xt(1,1)),' , ' num2str(xt(2,1))
        legend; %("Generated data","Predicted data")
        % Clearing all variables except f and xt_initial at the end
        vars = who;
        clearvars('-except', 'f', 'xt_initial', vars{:});
    end
```

OUTPUT:

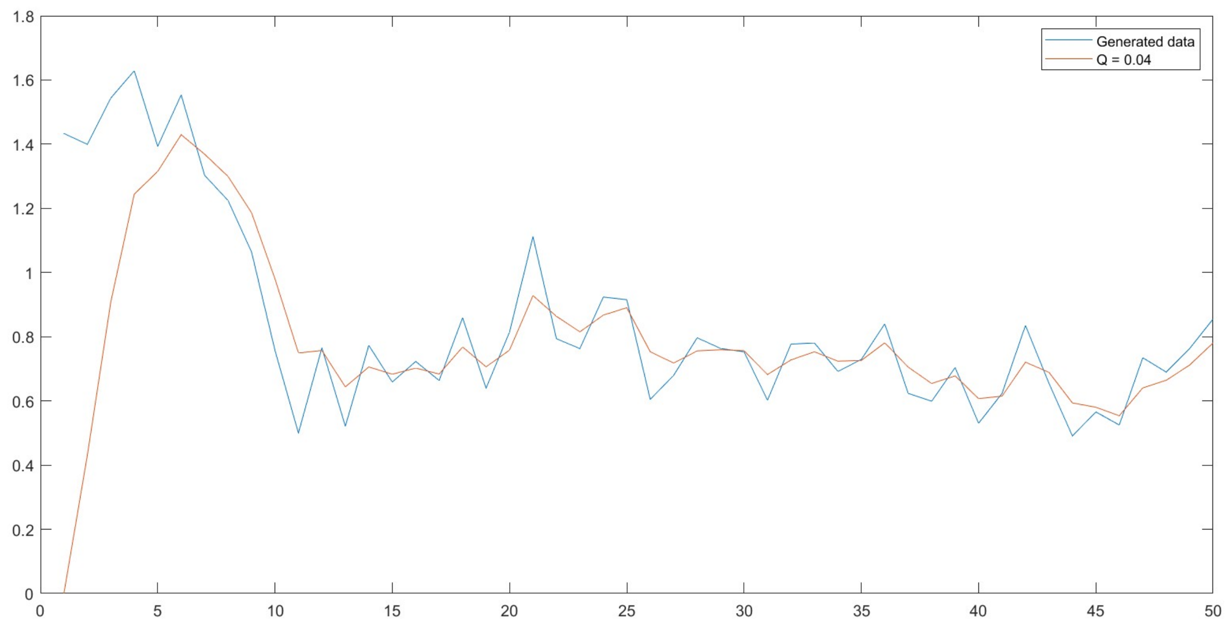


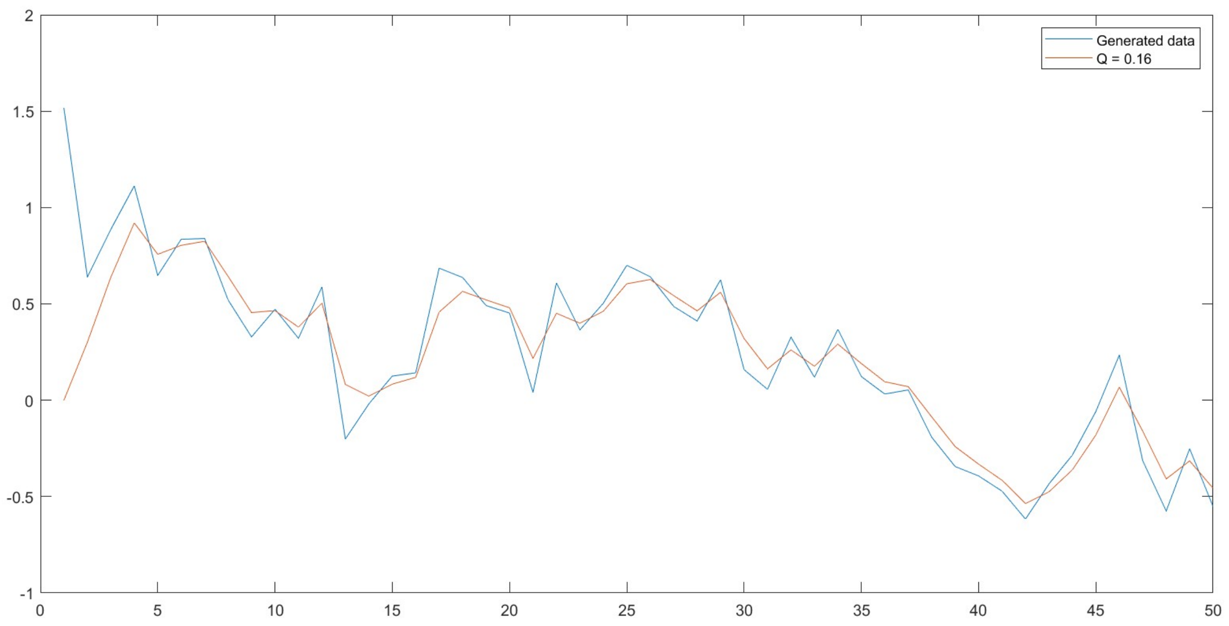
a) For different Q and R:

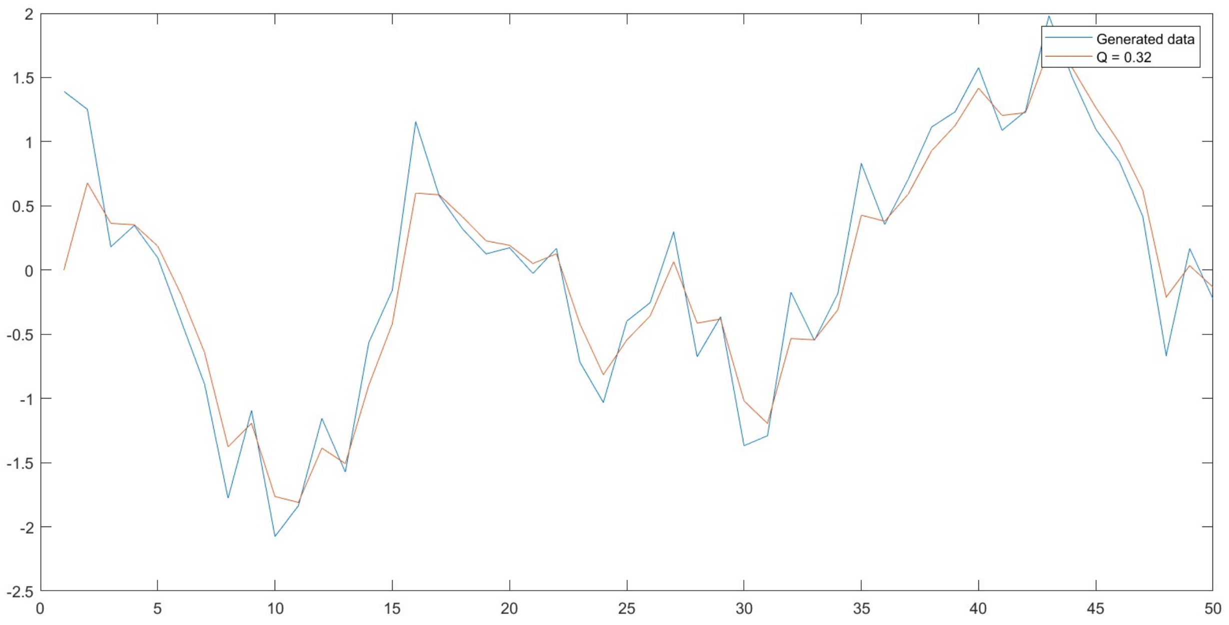
```
%-- .- -.. . / -... -.-- / -.-- --- --. . ... ....
%Date:19_11_2023
clc;clear all;close all;
N=50;    %number of time steps;
Q_vec={[0.04 0; 0 0.04],[0.16 0; 0 0.16],[0.32 0; 0 0.32]};
R_vec={0.09, 0.18, 0.27};
for f=1:length(Q_vec)
    C=[0 1];
    Q=Q_vec{f};
    R=R_vec{f};
    xt=[0,0]';
```

```
P0=1.5;
V0=0;
%initialise covariance matrix
p=[0 0;
    0 0];
y=[];
x=[V0,P0]';
w1=Q(1,1)*randn(N,1);
w2=Q(2,2)*randn(N,1);
v=R*randn(N, 1);
%input generation
x1=x(1);
x2=x(2);
%x1_vec=[x1];
%x2_vec=[x2];
for i=1:N
    x1new=(x1*x2)/(1+x1*x1)+w1(i);
    x2new=x2/(1+x1*x1)+w2(i);
    x1=x1new;
    x2=x2new;
%     x1_vec=[x1_vec;x1];
%     x2_vec=[x2_vec;x2];
    y=[y, x2+v(i)];
end
%y(k) to be arranged for each step
t=1:N;
figure;
plot(t, y, 'DisplayName',"Generated data");
hold on
clear x1 x2 x1new x2new w1 w2 v;
xt_initial={[0,0]', [20,-20]', [-5,-5]'};
    %Estimation_begin
    for k=2:N
        xt_ap=[(xt(1,k-1)*xt(2,k-1))/(1+xt(1,k-1)*xt(1,k-1)) xt(2,k-1)/(1+xt(1,k-1)*xt(1,k-
        A=[((1+3*xt_ap(1)*xt_ap(1))*xt_ap(2))/((1+xt_ap(1)*xt_ap(1))^2) xt_ap(1)/(1+xt_ap(1
            -2*xt_ap(1)*xt_ap(2)/((1+xt_ap(1)*xt_ap(1))^2)          1/(1+xt_ap(1)*xt_ap(1)
        %error c0variance propagation
        p_bar=A*p*A'+Q;

        %find kalman gain
        K=p_bar*C'*inv(C*p_bar*C'+R);
        %update state estimate
        xt(:,k)=xt_ap+K*[y(k)-C*xt_ap];

        %update covariance matrix
        p=(eye(2,2)-K*C)*p_bar;
        p_chol=chol(p);
        p=p_chol*p_chol';
    end
    t=1:N;
    %print_xt=num2str(xt_initial{f})
    plot(t,xt(2,:), 'DisplayName', ['Q = ', num2str(Q(1, 1))])
    legend; %("Generated data","Predicted data")
    % Clearing all variables except f and xt_initial at the end
    vars = who;
    clearvars('-except', 'f', 'xt_initial', vars{:});
end
```

OUTPUT:







Working of the Code Explained

The provided MATLAB code aims to implement a Kalman filter for state estimation in a dynamic system. The code is structured into two subsections:

For Different Initial Conditions

The code first generates noisy measurements considering different initial conditions for the states of the dynamic system. It plots the generated data and then applies the Kalman filter algorithm for state estimation.

For Different Q and R Values

Next, the code varies the process noise covariance matrix (Q) and measurement noise covariance (R) to observe their impact on state estimation.

Conclusion

The provided MATLAB code demonstrates the application of a Kalman filter for estimating the states of a dynamic system from noisy measurements. For each different initial condition, the Kalman filter converges quickly to the generated data. The plotted results demonstrate the convergence of the estimated states to the actual states of the system. And in the second, we are observing that with the increase in the variance of the noise the smoothness of the plot is decreasing and deviating from the true considered model. But still the algorithm is converging to the generated data.

2 Assignment: Question 2

2.1 2) Frols algorithm question:

2.1.1 Function Description

The function being estimated is defined by the equation:

$$f(Y, U, \text{noise}) = -0.6 \cdot Y(k-1) - 0.18 \cdot Y(k-1) \cdot Y(k-2) + 0.58 \cdot U(k-1) - 0.24 \cdot U(k-2) + \text{noise}(k)$$

where Y_{k-1} and Y_{k-2} represent the lagged values of the observed output `Y`, U_{k-1} and U_{k-2} denote the lagged values of the input `U`$, and$`noise`$_k$ is Gaussian noise.

MATLAB Code

```
clc; clear all; close all;

N = 300;    % number of time steps
nu = 2; ny = 2; ne = 0;   % As per the problem being solved
deg = 3;    % Nonlinear degree 3
U = 2 * rand(N, 1) - 1;    % Acknowledged that U mean is not zero out of this way
noise = 0.04 * randn(N, 1);
Y = [0 + noise(1); 0 + noise(2)];    % Y vector initialized with the value of y(1) and y(0)

% Data generation
for i = 3:N
    Y = [Y; -0.6 * Y(i - 1) - 0.18 * Y(i - 1) * Y(i - 2) + 0.58 * U(i - 1) - 0.24 * U(i - 2) +
end

% Dictionary generation
[DICT, DICT_HEADER] = dict_element({}, Y, U, noise, 1, deg, nu, ny, ne);
fprintf("length of dict=%d", length(DICT(1,:)));
fprintf("length of Y=%d", length(Y(3:N)));

[Y(1:N - 2) U(1:N - 2) noise(1:N - 2) DICT];

% Initialization
max_terms = size(DICT, 2);  % Maximum number of terms in the dictionary
FROLS_tol = 1e-6;  % Tolerance for the FROLS algorithm
max_iter = 100;    % Maximum iterations for FROLS

% Initialize variables for FROLS algorithm
selected_terms = [];  % List to store the selected regressor indices
B = zeros(max_terms, max_terms);  % Matrix to store coefficients
```

```
residuals = Y;  % Initialize residuals with output

for iter = 1:max_iter
    min_err = inf;  % Initialize minimum error

    % Loop through each term in the dictionary to find the best regressor
    for i = 1:max_terms
        if ~ismember(i, selected_terms)
            % Include the term and solve for coefficients
            curr_terms = [selected_terms i];
            B(curr_terms, curr_terms) = pinv(DICT(:, curr_terms)' * DICT(:, curr_terms)) * DICT

            % Calculate residuals
            y_pred = DICT(:, curr_terms) * B(curr_terms, curr_terms);
            residuals_new = Y - y_pred;

            % Calculate error
            err = norm(residuals_new);

            % Check if error is reduced
            if err < min_err
                min_err = err;
                best_term = i;
            end
        end
    end

    % Update selected terms and residuals
    selected_terms = [selected_terms best_term];
    residuals = residuals_new;

    % Check convergence
    if min_err < FROLS_tol
        break;
    end
end

% The selected_terms now contain the indices of the significant regressors
% The B matrix holds the estimated coefficients

function [dict, dict_header] = dict_element(vec, Y, U, noise, ip, deg, nu, ny, ne)
```

```
N = length(Y);
max_delay = max([nu, ny, ne]);
max_N = N - max_delay;
fprintf("Max delay found= %d Max_N= %d\n", max_delay, max_N);
dict_header = [];
index = 0;
for t = 1:deg
    l = length(vec);
    size_vec = size(vec);
    if l == 0
        for j = ip:(ny + 1 + nu + 1 + ne + 1)
            dict_header = [dict_header j];
            if j <= ny + 1
                vec = [vec; Y(max_delay + 2 - j : max_N + (max_delay + 1 - j))'];
            elseif and(j > ny + 1, j <= ny + 1 + nu + 1)
                vec = [vec; U(max_delay + 2 - (j - ny - 1) : max_N + (max_delay + 1 - (j - 
            else and(j > ny + 1 + nu + 1, j <= ny + 1 + nu + 1 + ne + 1)
                vec = [vec; noise(max_delay + 2 - (j - ny - 1 - nu - 1) : max_N + (max_dela
            end
        end
    else
        new_vec = {};
        fprintf("Deg = %d Length of vec=%d and size=", deg, length(vec));
        index1 = 0;
        new_dict_header = [];
        for i = 1:(ny + 1 + nu + 1 + ne + 1)        % i corresponds to top layer
            new_subvec = [];
            sz = size(vec{i});
            for j = i:(ny + 1 + nu + 1 + ne + 1)    % j corresponds to sub layer
                fprintf("j= %d, length(vec(j))=%d, size=\n", j, length(vec{j}));
                sz = size(vec{j});
                index2 = 0;
                for t1 = 1:j - 1 % Just for header purpose
                    sztemp = size(vec{t1});
                    index2 = index2 + sztemp(1);
                end
                for k = 1:sz(1)       % k corresponds to element in sublayer
                    index1 = index1 + 1;
                    fprintf("index1=%d, index2=%d, k=%d", index1, index2, k);
                    new_dict_header(index1) = (10^(t - 1)) * i + dict_header(index2 + k);
                    if t == 2
                temp = vec{j};
            else
                temp = vec{j}(k, :);
            end
            if i <= ny + 1
                new_subvec = [new_subvec; temp .* Y(max_delay + 2 - i : max_N + (ma
            elseif and(i > ny + 1, i <= ny + 1 + nu + 1)
                new_subvec = [new_subvec; temp .* U(max_delay + 2 - (i - ny - 1) : 
            else and(i > ny + 1 + nu + 1, i <= ny + 1 + nu + 1 + ne + 1)
                new_subvec = [new_subvec; temp .* noise(max_delay + 2 - (i - ny - 1
            end
        end
```

OUTPUT: Dictionary is successfully generated but the code is still failing to estimate the model.

Working of the Code Explained

FROLS is an iterative method used for model selection and parameter estimation in regression analysis. It aims to find the most significant terms (regressors) in the dictionary by iteratively selecting terms that minimize the residual error. The algorithm proceeds as follows:

2.2 Algorithm Description

The code implements an algorithm called FROLS (Forward Regression Orthogonal Least Squares) for identifying significant terms in a dictionary and estimating their coefficients to model a given function.

1. Initialization: Initialize variables such as maximum iterations, tolerance, and matrices to store coefficients.
2. Iterate through terms:
   (a) For each term in the dictionary, calculate the coefficients using the least squares method (pinv function).
   (b) Compute predicted values using the selected terms and their coefficients.
   (c) Update the residuals and calculate the error.
   (d) Check if the error is reduced and update the selected terms accordingly.
3. Convergence: Check for convergence by evaluating the error against a predefined tolerance.

2.3 Dictionary Generation (`dict_element`)

The `dict_element` function constructs a dictionary of potential regressors. It includes a variety of terms involving different lags of `Y`, `U`, and `noise`, up to a specified degree (`deg`). The function aims to create a matrix representation of possible interactions and lags between these variables.

2.4 Conclusion

: Although the code is able to generate the dictionary for any degree required and specified nu, ny and, ne but corrections are required in the model prediction part which may be improved afterwards.

Thank You.