EE 551 Estimation and Identification

Assignment 1

Nikhil N (194102506)

19 Nov 2019

1 Extended Kalman Filter

Given the discrete time system,

$$x(k+1) = \begin{bmatrix} x_1(k+1) \\ \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} \frac{x_1(k)}{1+x_2^2(k)} \\ \\ x_2(k+1) \end{bmatrix} + w(k)$$

$$y(k) = x_1(k) + v(k)$$

Where, w(k) and v(k) are independent Gaussian noises with covariance matrices Q_k and R_k.The given model is of the form,

$$x(k+1) = f(k, x_k) + w_k$$

$$y(k) = h(k, x_k) + v_k$$

The functional $f(k, x_k)$ denotes a non-linear transition matrix function that is possibly time-variant. Similarly, the functional $h(k, x_k)$ denotes a non-linear measurement matrix that may be time-variant too. The basic idea of the extended Kalman filter is to linearise the state-space model of above equations at each time instant around the most recent state estimate, which is taken to be either $\hat{x_k}$ or $\bar{\hat{x_k}}$, depending on which particular functional is being considered. The approximation is done by considering the Jacobian of two function $f(k, x_k)$ and $h(k, x_k)$ as,

$$A_{k+1} = \frac{\partial f(k,x)}{\partial x} \mid_{x=\hat{x_k}}$$

$$C_k = \frac{\partial h(k,x_k)}{\partial x} \mid_{x=\bar{\hat{x_k}}}$$

Now approximated state equations are given by,

$$X_{k+1} \approx A_{k+1,k} X_k + w_k$$

$$Y_k \approx C_k X_k + v_k$$

1.1 Algorithm

Initialization: For k=0, set $\hat{x_0}$ and P_0 with some initial values
Computation: For k=1,2,... compute

State estimate propagation

$$\bar{\hat{X}}_k = f(k, x_{k-1})$$

Error covariance propagation

$$\bar{P}_k = A_{k,k-1} P_{k-1} A_{k,k-1}^T + Q_{k-1}$$

Kalman gain matrix

$$K_k = \bar{P}_k C_k^T [C_k \bar{P}_k C_k^T + R_k]^{-1}$$

State estimate update

$$\hat{X}_k = \bar{\hat{X}}_k + K_k (Y_k - C_k \bar{\hat{X}}_k)$$

Error covariance update

$$P_k = \bar{P}_k - K_k C_k \bar{P}_k$$

1.2 MATLAB Code

```
% EKF   Extended Kalman Filter for nonlinear dynamic systems
% For nonlinear dynamic system:
%           x_k+1 = f(x_k) + w_k
%           y_k    = h(x_k) + v_k
% where w ¬ N(0,Q) w is gaussian noise with covariance Q
%       v ¬ N(0,R) v is gaussian noise with covariance R
% Inputs:   f: function for f(x)
%           x: "a priori" state estimate
%           P: "a priori" estimated state covariance
%           h: function for h(x)
%           z: current measurement
%           Q: process noise covariance
%           R: measurement noise covariance
% Output:   x: "a posteriori" state estimate
%           P: "a posteriori" state covariance
%%
clear all;
clc;
n = 2;                                                     % number of state
q = 0.1;                                                   % standard deviation of process
r = 0.1;                                                   % standard deviation of measurement
Q = q*eye(n);                                              % covariance of process
R = r^2;                                                   % covariance of measurement
f = @(x)[x(1)/(1+x(2)^2);(x(1)*x(2))/(1+x(2)^2)];          % nonlinear state equations
h = @(x)x(1);                                              % measurement equation
s = [1;1];                                                 % initial state
x = s+q*randn(n,1);                                        % initial state with noise
P = 0.5*eye(n);                                            % initial state covraiance
N = 25;                                                    % total dynamic steps
xV = zeros(n,N);                                           % estmate
sV = zeros(n,N);                                           % actual
yV = zeros(1,N);                                           % output measurement
err= zeros(1,N);
for k=1:N
y = h(s)+r*randn;                                          % measurments
sV(:,k) = s;                                               % store actual state
yV(k) = y;                                                 % store measurement
[x,P] = extendedKF(f,x,P,h,y,Q,R);                         % ekf
xV(:,k) = x;                                               % save estimate
s = f(s)+ q*randn(n,1);                                    % update process
end
for k=1:n                                                  % plot results
  subplot(n,1,k)
  str = sprintf('x(%d)',k);
  err=abs(sV(k,:)-xV(k,:));
  plot(1:N, sV(k,:), '-', 1:N, xV(k,:), '--'),grid on,
  xlabel('Iteration'),ylabel(str),title('Extended KF'),legend('Actual','Predicted')
end
%% METHODS
% Extended Kalman Filter
function [x P]= extendedKF(f,x,P,h,y,Q,R)
[x1,A] = jaccsd(f,x);                           % nonlinear update and linearization at current state
P1 = A*P*A'+Q;                                             % apriori error covariance
[y1,C] = jaccsd(h,x1);                                     % nonlinear measurement and linearization
P12 = P1*C';                                               % cross covariance
K = P12*inv(C*P12+R);                                      % Kalman filter gain
x = x1+K*(y-y1);                                           % state estimate
P = P1-K*P12';                                             % state covariance matrix
end
% JACCSD Jacobian through complex step differentiation
function [z,A]=jaccsd(fun,x)
z = fun(x);                                                % evaluation of x(k+1)
n = numel(x);
m = numel(z);
A = zeros(m,n);
h = n*eps;
for k=1:n
    x1=x;
    x1(k)=x1(k)+h*1i;
    A(:,k)=imag(fun(x1))/h;
end
end
```

1.3 Output Plots

1.3.1 Different Initial condition

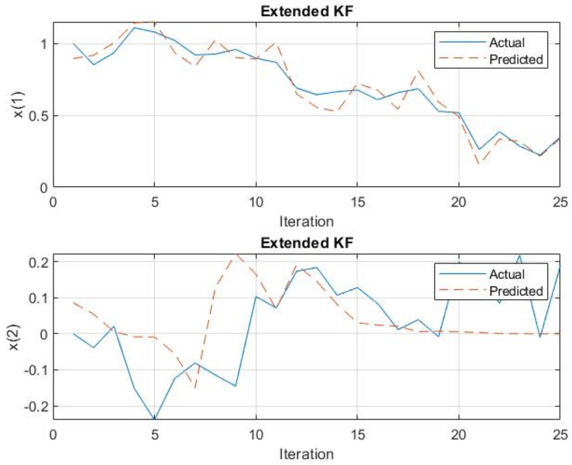

Figure 1: Initial state [1 0]

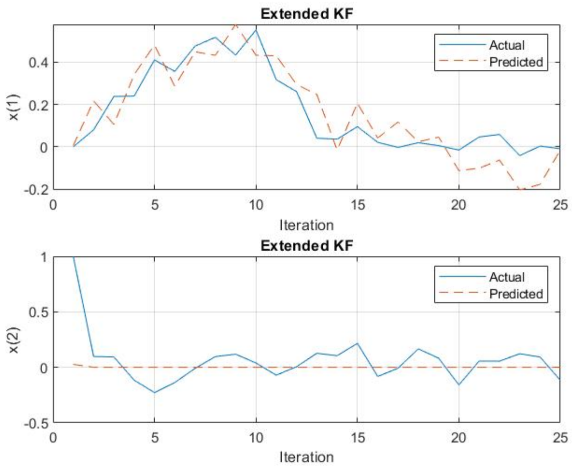

Figure 2: Initial state [0 1]

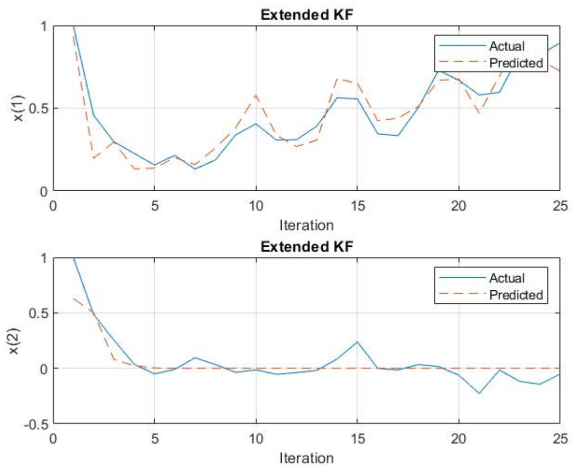

Figure 3: Initial state [1 1]

1.3.2 Different Q and R

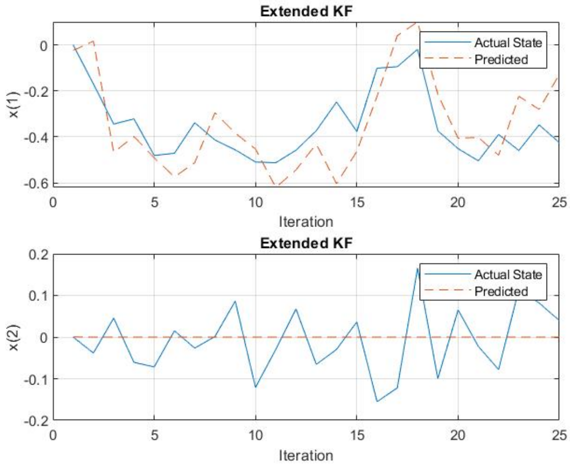

Figure 4: q=0.1 and r=0.2

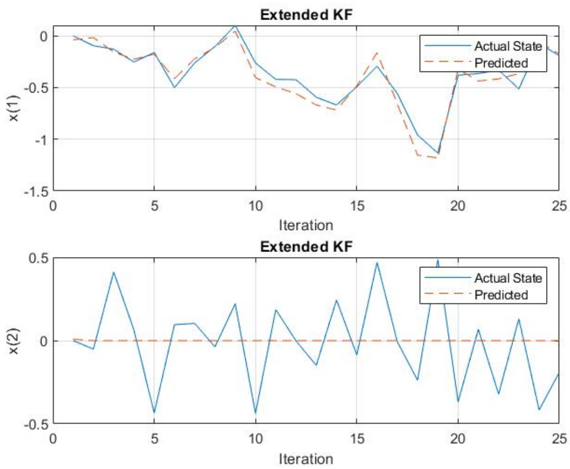

Figure 5: q=0.2 and r=0.1

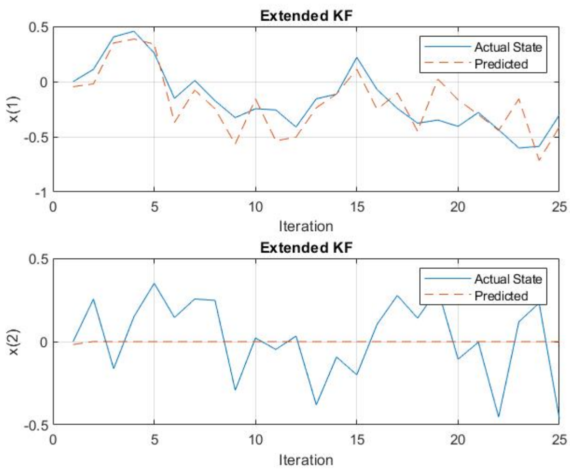

Figure 6: q=0.2 and r=0.2

2 FROLS Algorithm to fit an NARX Model for the Data

Given model

$$y(k) = -0.605y(k-1) - 0.163y^2(k-2) + 0.588u(k-1) - 0.240u(k-2) + \xi(k)$$

where $\xi(k)$ is white Gaussian noise sequence with zero mean and standard deviation 0.2. The input $u(k)$ is a uniformly random sequence between [-1 1], and the other parameters are non-linear degree $l = 3$, output delay terms $n_y = 2$, input delay terms $n_u = 2$, error terms $n_e = 0$, total time instants N=200 and ESR $\rho = 0.05$.
Let $y = [y(1), \ldots, y(N)]^T$ be a vector of measured outputs at N time instants, and $P_m = [p_m(1), p_m(2), \ldots, p_m(N)]^T$ be a vector formed by the m^{th} candidate model term,where $m = 1, 2, \ldots, M$. Let $D = \{P_1, P_2, \ldots, P_M\}$ be a dictionary composed of the M candidate bases.The model term selection through FROLS algorithm is equivalent to finding a full dimensional subset $D_{M_0} = \{\alpha_1, \alpha_2, ..., \alpha_{M_0}\} = \{P_1, P_2, \ldots, P_{M_0}\}$ of M_0 ($M_0 \le M$) bases, from the dictionary D. Now the final Y can be approximated as,

$$Y = \theta_1\alpha_1 + \theta_2\alpha_2 + ... + \theta_{M_0}\alpha_{M_0} + e$$

or, in compact matrix form,

$$Y = X\theta + e$$

Here the parameter θ can be estimated by using least square method as,

$$\theta = (X^TX)^{-1}X^TY$$

2.1 Algorithm

The first-step in FROLS starts with the initial full model, in our case it is,

$$Y = c_0 + \sum_{i_1=1}^{n} c_{i_1}x_{i_1}(k) + \sum_{i_1=1}^{n}\sum_{i_2=i_1}^{n} c_{i_1i_2}x_{i_1}(k)x_{i_2}(k) + \sum_{i_1=1}^{n}\sum_{i_2=i_1}^{n}\sum_{i_3=i_2}^{n} c_{i_1i_2i_3}x_{i_1}(k)x_{i_2}(k)x_{i_3}(k)$$

where $n = n_u + n_y = 4$ and the initial full dictionary $D = \{P_1, P_2, \ldots, P_M\}$. For $m = 1, 2, \ldots, M(\frac{(n+l)!}{n!\,l!})$ let $q_m = p_m$ and $\sigma = Y^TY$, calculate

$$g_m^{(1)} = \frac{Y^Tq_m}{q_m^Tq_m}$$

$$ERR^{(1)}[m] = (g_m^{(1)})^2(q_m^Tq_m)/\sigma$$

Let

$$ERR[\ l_1\ ] = \max\{ERR^{(1)}[m] : 1 \le m \le M\}$$

$$l_1 = arg \max_{1\le m\le M}\ \{ERR^{(1)}\}$$

The first significant basis can then be selected as $\alpha_1 = P_{l_1}$, and the first associated orthogonal vector can be chosen as $q_1 = P_{l_1}$ also set $g_1 = g_{l_1}^{(1)}$ and $err[1] = ERR^{(1)}[\ l_1\ ]$. Now from the second step onwards till s^{th} step, $m \ne l_1, m \ne l_2, ..., m \ne l_{s-1}$. For $m = 1, 2, \ldots, M$, calculate

$$q_m^{(s)} = P_m - \sum_{r=1}^{s-1}\frac{P_m^Tq_r}{q_r^Tq_r}q_r,\ P_m \in D - D_{s-1}$$

$$g_m^{(s)} = \frac{Y^Tq_m}{q_m^Tq_m}$$

$$ERR^{(s)}[m] = (g_m^{(s)})^2(q_m^Tq_m)/\sigma$$

Let

$$ERR[\ l_s\ ] = \max\{ERR^{(s)}[m] : 1 \le m \le M\}$$

$$l_s = arg \max_{1\le m\le M}\ \{ERR^{(s)}\}$$

and set $q_s = q_{l_s}^{(s)}$, $g_s = g_{l_s}^{(s)}$ and $err[s] = ERR^{(s)}[\ l_s\ ]$

The search is terminated at the M_0 step when the ESR is less than a pre-specified threshold

$$ESR = 1 - \sum_{s=1}^{M_0} err(s) \le \rho \text{ where } \rho \text{ is } 0.05$$

2.2 MATLAB Code

```
%% FROLS ALGORITHM TO FIT AN NARX MODEL
% Non Linear system
%           y_k :-0.605*y(k-1)-0.163*y^2(k-1)+0.588*u(k-1)-0.240*u(k-2)+e(k)
% where       e ¬ Gaussian white noise
% Inputs:   ny : number of output delay terms
%            nu : number of input delay terms
%            ne : number of error terms
%             l : max degree
%             u : unifromily distribute input btw [-1 1]
% Outputs     c : selected terms (specified vector format)
%           not : number of selected terms
%           phi : parameter values
%
%
%% Preliminaries
clear all;
clc;
ny=2                                                 % nummber of y terms
nu=2                                                 % nummber of u terms
ne=0;                                                % nummber of e terms
n=ny+nu+ne;                                          % total terms
l=3;                                                 % max degree
ly=200;                                              % length
M= factorial(n+l)/(factorial(n)*factorial(l))        % Total number of possible terms
mu=0;                                                % noise average
sigma=0.1;                                           % noise standard deviation
e=sigma*randn(200,1)+mu;                             % Gaussian white noise
u=-1+2*rand(200,1);                                  % uniformly distributed input
y=[0.1;0.5];                                         % initial output

for k=3:ly
    y(k)=(-0.605*y(k-1))-(0.163*y(k-2)^2)+(0.588*u(k-1))-(0.240*u(k-2))+e(k);%generating outputs
end

%% Generating all possible term sets
% C-stores all possible terms sets. --each row indicate a term
% and column values represent the powers corresponfing delay terms as u(k-1),u(k-2),y(k-1),y(k-2)
% eg: [1 0 0 2] indicate term = u(k-1)^1*y(k-2)^2
b=2;
C=zeros(M,n);                                        % dimension (35,4)
for p=1:l
    if p==1                                          % for power=1
      for i_1=1:n
          C(b,i_1)=C(b,i_1)+1;
          b=b+1;
      end
    end
    if p==2                                          % for power=2
        for i_1=1:n
            for i_2=i_1:n
                C(b,i_1)=C(b,i_1)+1;
                C(b,i_2)=C(b,i_2)+1;
                b=b+1;
            end
        end
    end

    if p==3                                          % for power=3
        for i_1=1:n
            for i_2=i_1:n
                for i_3=i_2:n
                    C(b,i_1)=C(b,i_1)+1;
                    C(b,i_2)=C(b,i_2)+1;
                    C(b,i_3)=C(b,i_3)+1;
                     b=b+1;
                end
            end
        end
    end

end

%% creating D matrix -- Holds the values of all possible terms for each output
```

```
nm=max(nu,ny);                                     % picking max delay
D=ones(ly-nm,M);
%size(D)
for i=nm+1:ly                                      % Iteraing y
    for j=1:M                                      % Iterating C
       k=1;
       for l=1:nu
       D(i-nm,j)=(D(i-nm,j))*(u(i-l)^(C(j,k)));      % combining delay terms of input u
       k=k+1;
       end
       for m=1:ny
       D(i-nm,j)=(D(i-nm,j))*(y(i-m)^(C(j,k)));      % combining delay terms of output y
       k=k+1;
       end
    end
end
size(D);

%% FROLS to Select the terms
Y=y(nm+1:ly,:);
sig=Y'*Y;                                          % sigma
sg=zeros(M,1);                                     % vector to hold selected g_m
serr=zeros(M,1);                                   % vector to hold selected err
%sl=zeros(M,1);
q=zeros(ly-nm,M);                        % vector to hold evaluated orthogonal vectors

for j=1:M
    err=zeros(M,1);
    g=zeros(M,1);
    if j==1                                        % loop to find first prominant term
        for i=1:M
            q(:,i)=D(:,i);
            qq=q(:,i)'*q(:,i);
            g(i)=(Y'*q(:,i))/qq;
            err(i)=(g(i)^2*qq)/sig;
        end
    else
        for i=1:M                      % loop to find second and remaining prominant terms
            if ¬ismember(i,sl)
                p=D(:,i);
                pd=zeros(ly-nm,1);

                for r=1:size(sq,2)    % evaluating the subtracting term in orthogonalisation
                    qr=sq(:,r);
                    pd=pd+((p'*qr)/(qr'*qr))*qr;
                end

                q(:,i)=p-pd;                       % orthogonal vecctor q_m
                qq=q(:,i)'*q(:,i);
                g(i)=(Y'*q(:,i))/qq;
                err(i)=(g(i)^2*qq)/sig;            % evaluating err_m
            end
        end
    end
[ERR,l]=max(err);                        % picking the term with max err and its index
serr(j)=ERR;                                       % store selected err value
sl(j)=l;                                           % store selected index of term
sg(j)=g(l);                                        % store selected g_m
sq(:,j)=q(:,l);                                    % store selected orthogonal vector

ESR=1-sum(serr);                                   % termination parameter
if ESR≤0.05                                       % check termination condition
    break                                          % end the search if condition meets
end
end
sl;                                                % selected term index
c=C(sl,:)                                          % selected terms

%% Calculating the coefficients(parameters) using LS
for i=1:ly-nm
    X(i,:)=D(i,sl);
end
X;
not=size(c,1)                                      % number of selected terms
phi=inv(X'*X)*X'*Y                                 % parameter values
```

2.2.1 Strategy used for storing all possible terms

In the program the all possible terms of the full NARX model is stored as a matrix C. In which the columns represent the degree of a particular delay term i.e. [u(k-1) u(k-2) y(k-1) y(k-2)], and rows represents a whole term. For e.g. Row [1 0 0 0] represents the term $u(k-1)$ and [1 0 2 0] represents the term $u(k-1)y(k-1)^2$.

2.3 Output results

Sl No	C Matrix Entry				Term	Parameter value
1	0	0	1	0	$y(k-1)$	-0.6415
2	1	0	0	0	$u(k-1)$	0.5872
3	0	1	0	0	$u(k-2)$	-0.1585
4	0	0	0	2	$y(k-2)^2$	-0.0110
5	1	0	0	2	$u(k-1)y(k-2)^2$	0.1362
6	0	0	1	2	$y(k-1)y(k-2)^2$	1.6401
7	0	0	0	1	$y(k-2)$	-0.0809
8	2	1	0	0	$u(k-1)^2u(k-1)$	-0.0917
9	0	0	0	3	$y(k-2)^3$	0.3299
10	1	1	0	1	$u(k-1)u(k-2)y(k-2)$	-0.2946
11	3	0	0	0	$u(k-1)^3$	0.0637
12	0	2	0	0	$u(k-2)^2$	0.0523
13	1	1	1	0	$u(k-1)u(k-2)y(k-1)$	-0.1195
14	1	2	0	0	$u(k-1)u(k-2)^2$	0.1303
15	1	0	0	1	$u(k-1)y(k-2)$	-0.0309
16	0	1	0	1	$u(k-2)y(k-2)$	-0.1646
17	0	0	1	1	$y(k-1)y(k-2)$	0.2626
18	0	3	0	0	$u(k-2)^3$	-0.0917
19	0	2	1	0	$u(k-2)^2y(k-1)$	0.2682
20	1	0	1	1	$u(k-1)y(k-1)y(k-2)$	-0.0696
21	0	2	0	1	$u(k-2)^2y(k-2)$	0.6765
22	2	0	1	0	$u(k-1)^2y(k-1)$	0.2648
23	2	0	0	1	$u(k-1)^2y(k-2)$	0.2110
24	1	0	2	0	$u(k-1)y(k-1)^2$	-0.2215
25	0	0	2	0	$y(k-1)^2$	0.0238
26	0	1	1	0	$u(k-2)y(k-1)$	-0.0351
27	0	0	0	0	$constant$	-0.0351
28	0	1	0	2	$u(k-2)y(k-2)^2$	-1.1112
29	0	1	1	1	$u(k-2)y(k-2)y(k-2)$	-2.6937
30	0	0	3	0	$y(k-1)^3$	0.8689
31	0	0	2	1	$y(k-1)^2y(k-2)$	2.2821
32	0	1	2	0	$u(k-2)y(k-1)^2$	-1.2493
33	2	0	0	0	$u(k-1)^2$	0.1082
34	1	1	0	0	$u(k-1)u(k-2)$	-0.0161
35	1	0	1	0	$u(k-1)y(k-1)$	0.0089

Table 1: $\sigma = 0.2$ and ESR = 0.05

Sl No	C Matrix Entry				Term	Parameter value	Error
1	0	0	1	0	$y(k-1)$	-0.6099	0.0049
2	1	0	0	0	$u(k-1)$	0.5848	0.0032
3	0	1	0	0	$u(k-2)$	-0.2392	-0.0008
4	0	0	0	2	$y(k-2)^2$	-0.1663	0.0033

Table 2: $\sigma = 0.1$ and ESR = 0.05

References

[1] Stephen A Billings. *NonLinear System Identification*. Wiley, 2013.

[2] Simon Haykin. *Kalman Filtering And Neural Networks*. Awiley-Interscience Publication, 2001.

[3] mathworks.com. *Kalman Filter*. https://in.mathworks.com/, 2019.