Assignment 11 Extended Kalman filter for navigation and tracking

Group 6:

- **Andrei Shumeiko**
- **Ayush Gupta**
- **Olga Klyagina**
- **Simona Nitti**


```
clc
clear
close all
```

Task 1: Generate a true trajectory X

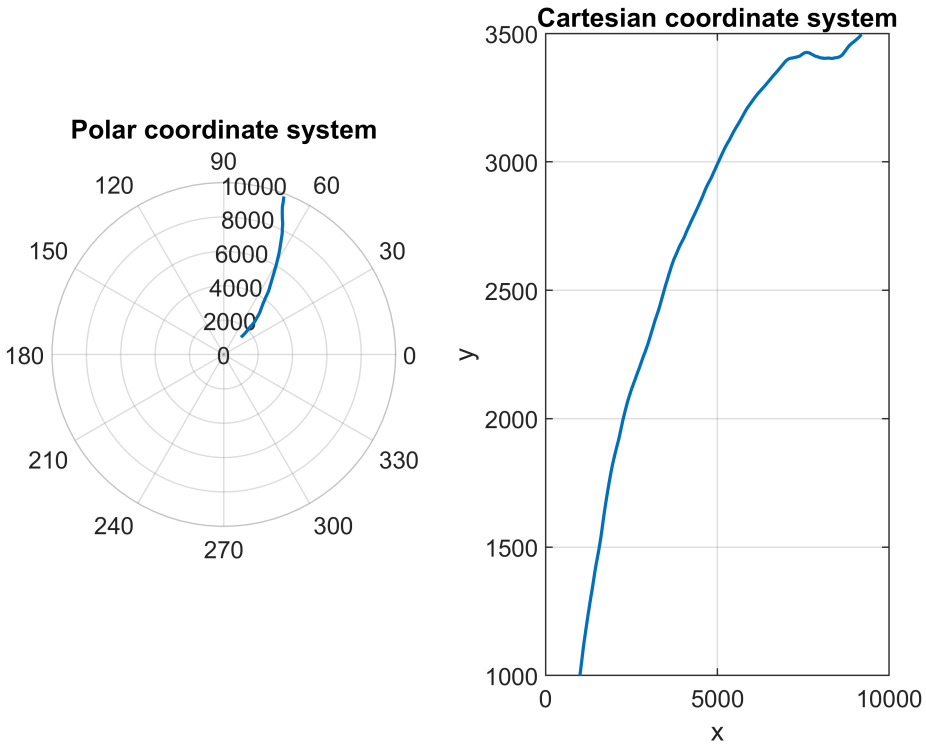
```
n = 500; % Size of trajectory
T = 1;   % Interval between measurements

%Initial components of velocity
V = zeros(2,n);
V(1,1)= 10; % V_x
V(2,1)= 10; % V_y

% Initial coordinates
x = zeros(2,n);
x(1,1)= 1e3; % x
x(2,1)= 1e3; % y

% Random acceleration
var_a = 0.3^2;
a = sqrt(var_a)*randn(2,n);

for i = 2:n
    x(1,i) = x(1,i-1) + V(1,i-1)*T + a(1,i-1)*T^2/2;
    x(2,i) = x(2,i-1) + V(2,i-1)*T + a(2,i-1)*T^2/2;
    V(1,i) = V(1,i-1) + a(1,i-1)*T;
    V(2,i) = V(2,i-1) + a(2,i-1)*T;
end
```

Task 2: Generate also true values of range and azimuth

```
D = sqrt(x(1,:).^2 + x(2,:).^2);
beta = atan(x(1,:)./x(2,:));

figure
sgtitle('Fig 1: True trajectory')
subplot(1,2,1)
```

```
polarplot(beta,D,'linewidth',1.2)
title('Polar coordinate system')

subplot(1,2,2)
plot(x(1,:),x(2,:),'linewidth',1.2)
title('Cartesian coordinate system')
xlabel('x')
ylabel('y')
grid on
```

Fig 1: True trajectory



Task 3: Generate measurements

```
var_D = 50^2;
var_beta = 0.004^2;

eta = randn(2,n);
eta(1,:) = eta(1,:)*sqrt(var_D);    % eta_D
eta(2,:) = eta(2,:)*sqrt(var_beta); % eta_beta

D_meas = D + eta(1,:);
beta_meas = beta + eta(2,:);
z = [D_meas; beta_meas];
```

Task 4: Initial conditions for Kalman filter algorithm

```
X = zeros(4,n);
X(:,1) = [D_meas(1)*sin(beta_meas(1)) 0 D_meas(1)*cos(beta_meas(1)) 0]'; % Initial filtered est
```

```
P = 1e10*eye(4);                          % Initial filtration error covariance matrix

polar_f(1,1) = sqrt(X(1,1)^2+X(3,1)^2); % Initial filtered estimate of D
polar_f(2,1) = atan(X(1,1)/X(3,1));     % Initial filtered estimate of beta
```

Task 5: Create the transition matrix

```
phi = eye(4)+ T*diag(ones(3,1),1);
phi(2,3)=0;
```

Task 6: Calculate state noise covariance matrix Q

```
% Input matrix
G = [T^2/2 0; T 0; 0 T^2/2;0 T];

% State noise covariance matrix
Q = G*G'*var_a;
```

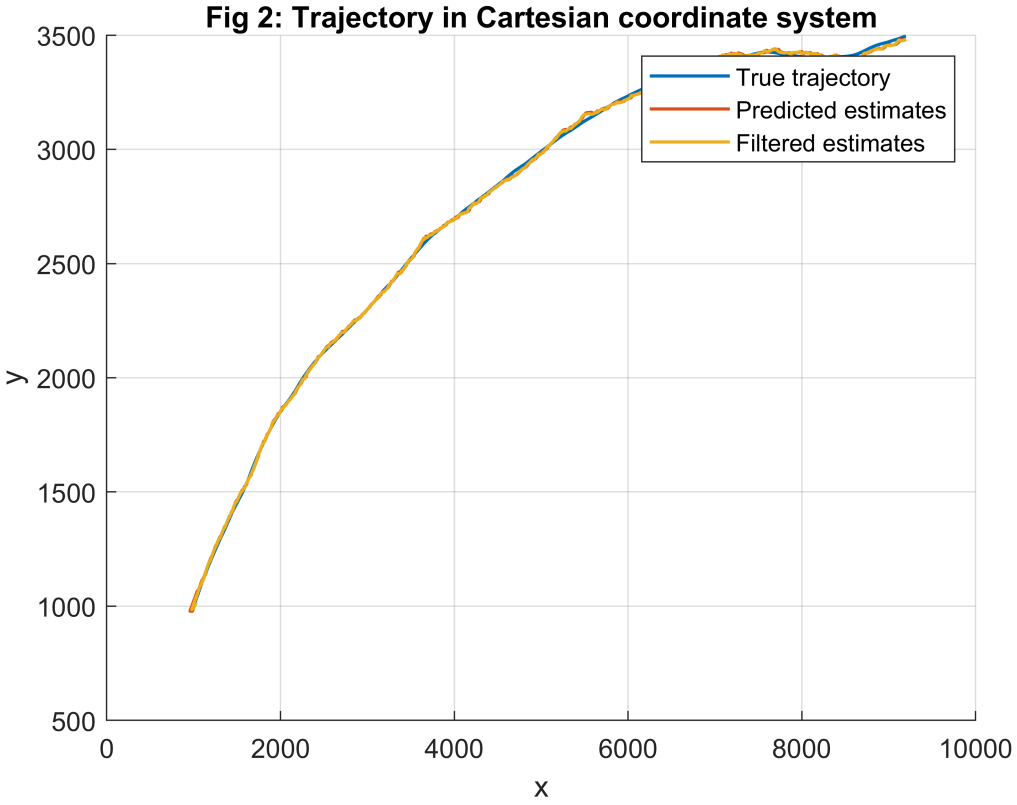
Task 7: Create the measurement noise covariance matrix R

```
R = [var_D 0; 0 var_beta];
```

Task 8-9: Develop Kalman filter algorithm

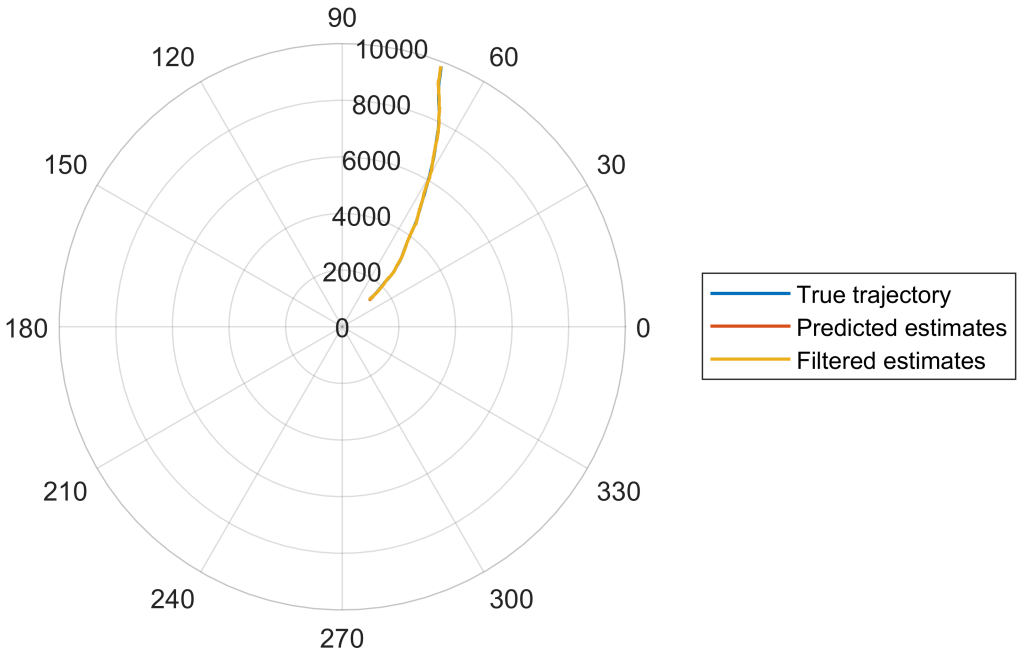
```
polar_e = zeros(2,n);
X_pred = zeros(4,n);
for i = 2:n
    % Prediction
    X_pred(:,i) = phi*X(:,i-1);
    P_pred = phi*P*phi' + Q;
    polar_e(1,i) = sqrt(X_pred(1,i)^2+X_pred(3,i)^2);  % extrapolated estimate of D
    polar_e(2,i) = atan(X_pred(1,i)/X_pred(3,i));      % extrapolated estimate of beta

    % dh/dX
    dh = zeros(2,4);
    dh(1,1) = X_pred(1,i)/sqrt(X_pred(1,i)^2 + X_pred(3,i)^2);
    dh(1,3) = X_pred(3,i)/sqrt(X_pred(1,i)^2 + X_pred(3,i)^2);
    dh(2,1) = X_pred(3,i)/(X_pred(1,i)^2 + X_pred(3,i)^2);
    dh(2,3) = -X_pred(1,i)/(X_pred(1,i)^2 + X_pred(3,i)^2);

    % Filtration
    K = P_pred*dh'*inv(dh*P_pred*dh'+R);
    X(:,i) = X_pred(:,i) + K*(z(:,i)-polar_e(:,i));
    P = (eye(4)-K*dh)*P_pred;

    % Change of coordinates
    polar_f(1,i) = sqrt(X(1,i)^2+X(3,i)^2); %D
    polar_f(2,i) = atan(X(1,i)/X(3,i));     %beta
end
polar_e(:,1) = NaN(2,1);
X_pred(:,1) = NaN(4,1);
```

```
figure
hold on
plot(x(1,:),x(2,:),'linewidth',1.2)
plot(X_pred(1,:),X_pred(3,:),'linewidth',1.2)
plot(X(1,:),X(3,:),'linewidth',1.2)
xlabel('x')
ylabel('y')
legend('True trajectory','Predicted estimates','Filtered estimates')
title('Fig 2: Trajectory in Cartesian coordinate system')
grid on
```




```
figure
polarplot(beta,D,'linewidth',1.2)
hold on
polarplot(polar_e(2,:),polar_e(1,:),'linewidth',1.2)
polarplot(polar_f(2,:),polar_f(1,:),'linewidth',1.2)
legend('True trajectory','Predicted estimates','Filtered estimates')
title('Fig 3: Trajectory in Polar coordinate system')
grid on
```

Fig 3: Trajectory in Polar coordinate system



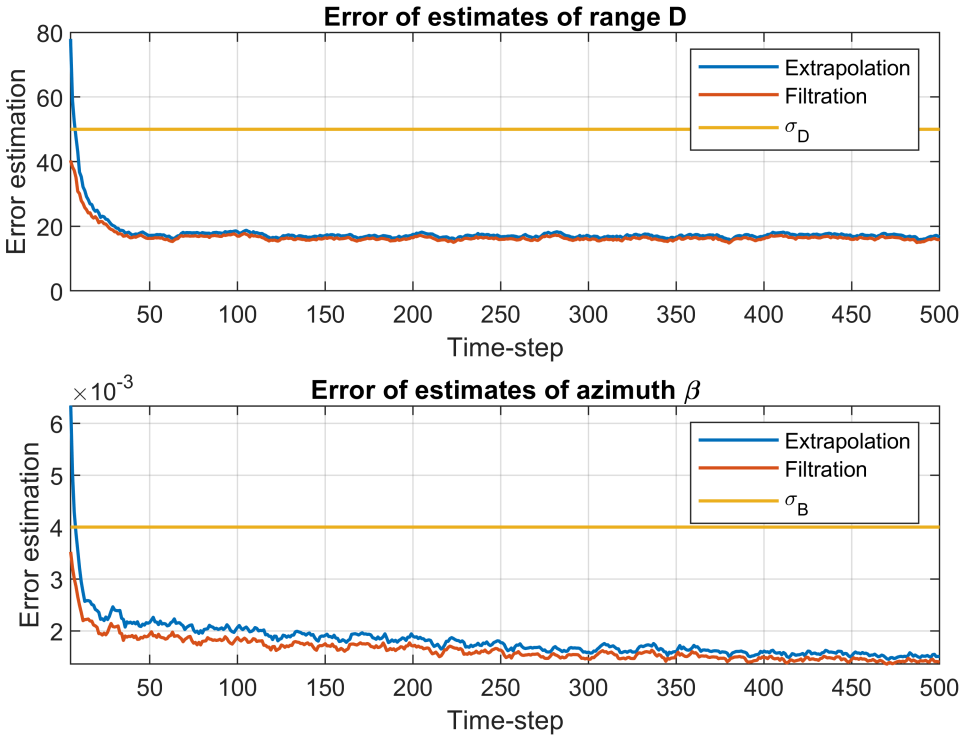
Task 10: Run Kalman filter algorithm over = 500 runs.

```
M = 500;
V = zeros(2,n);
V(1,1)= 10; % V_x
V(2,1)= 10; % V_y

% Initial coordinates
x = zeros(2,n);
x(1,1)= 1e3; % x
x(2,1)= 1e3; % y

Final_Error_estr = zeros(2,n);
error_estr = zeros(2,n);
Final_Error_filt = zeros(2,n);
error_filt = zeros(2,n);
for run = 1:M
    a = sqrt(var_a)*randn(2,n);

    for i = 2:n
        x(1,i) = x(1,i-1) + V(1,i-1)*T + a(1,i-1)*T^2/2;
        x(2,i) = x(2,i-1) + V(2,i-1)*T + a(2,i-1)*T^2/2;
        V(1,i) = V(1,i-1) + a(1,i-1)*T;
        V(2,i) = V(2,i-1) + a(2,i-1)*T;
    end

    D = sqrt(x(1,:).^2 + x(2,:).^2);
```

```
beta = atan(x(1,:)./x(2,:));

eta = randn(2,n);
eta(1,:) = eta(1,:)*sqrt(var_D);    % eta_D
eta(2,:) = eta(2,:)*sqrt(var_beta); % eta_beta

D_meas = D + eta(1,:);
beta_meas = beta + eta(2,:);

z = [D_meas ; beta_meas]; % Measurements

X = zeros(4,n);
X_pred = X;
polar_e = zeros(2,n);
X(:,1) = [D_meas(1)*sin(beta_meas(1)) 0 D_meas(1)*cos(beta_meas(1)) 0]'; % Initial filtered
P = 1e10*eye(4);                        % Initial filtration error covariance matrix
polar_f(1,1) = sqrt(X(1,1)^2+X(3,1)^2); % Initial filtered estimate of D
polar_f(2,1) = atan(X(1,1)/X(3,1));     % Initial filtered estimate of beta

for i = 2:n
    % Prediction
    X_pred(:,i) = phi*X(:,i-1);
    P_pred = phi*P*phi' + Q;
    polar_e(1,i) = sqrt(X_pred(1,i)^2+X_pred(3,i)^2);   % extrapolated estimate of D
    polar_e(2,i) = atan(X_pred(1,i)/X_pred(3,i));       % extrapolated estimate of beta

    % dh/dX
    dh = zeros(2,4);
    dh(1,1) = X_pred(1,i)/sqrt(X_pred(1,i)^2 + X_pred(3,i)^2);
    dh(1,3) = X_pred(3,i)/sqrt(X_pred(1,i)^2 + X_pred(3,i)^2);
    dh(2,1) = X_pred(3,i)/(X_pred(1,i)^2 + X_pred(3,i)^2);
    dh(2,3) = -X_pred(1,i)/(X_pred(1,i)^2 + X_pred(3,i)^2);

    % Filtration
    K = P_pred*dh'*inv(dh*P_pred*dh'+R);
    X(:,i) = X_pred(:,i) + K*(z(:,i)-polar_e(:,i));
    P = (eye(4)-K*dh)*P_pred;

    % Change of coordinates
    polar_f(1,i) = sqrt(X(1,i)^2+X(3,i)^2); %D
    polar_f(2,i) = atan(X(1,i)/X(3,i));     %beta
end

for i = 3:n
    %Errors of extrapolation estimates of range D
    error_estr(1,i) = (D(i)-polar_e(1,i))^2;
    Final_Error_estr(1,i)= Final_Error_estr(1,i)+error_estr(1,i);
    %Errors of filtration estimates of range D
    error_filt(1,i) = (D(i)-polar_f(1,i))^2;
    Final_Error_filt(1,i)= Final_Error_filt(1,i)+error_filt(1,i);

    %Errors of extrapolation estimates of azimuth beta
    error_estr(2,i) = (beta(i)-polar_e(2,i))^2;
    Final_Error_estr(2,i)= Final_Error_estr(2,i)+error_estr(2,i);
```

```
        %Errors of filtration estimates of azimuth beta
        error_filt(2,i) = (beta(i)-polar_f(2,i))^2;
        Final_Error_filt(2,i)= Final_Error_filt(2,i)+error_filt(2,i);
    end
end
Final_Error_estr = sqrt(Final_Error_estr/(M-1));
Final_Error_filt = sqrt(Final_Error_filt/(M-1));

figure
subplot(2,1,1)
sgtitle('Fig 4: Estimation and Measurement errors')
plot(Final_Error_estr(1,:), 'linewidth',1.2)
hold on
plot(Final_Error_filt(1,:),'linewidth',1.2)
plot(sqrt(var_D)*ones(1,n),'linewidth',1.2)
grid on
title('Error of estimates of range D')
legend('Extrapolation','Filtration','\sigma_D')
ylabel('Error estimation')
xlabel('Time-step')
xlim([5 inf])

subplot(2,1,2)
plot(Final_Error_estr(2,:), 'linewidth',1.2)
hold on
plot(Final_Error_filt(2,:),'linewidth',1.2)
plot(sqrt(var_beta)*ones(1,n),'linewidth',1.2)
title('Error of estimates of azimuth \beta')
legend('Extrapolation','Filtration','\sigma_B')
ylabel('Error estimation')
xlabel('Time-step')
grid on
xlim([5 inf])
```

Fig 4: Estimation and Measurement errors



Conclusions:

- The true trajectory is generated for an object motion disturbed by normally disturbed random acceleration in cartesian coordinate system. The true values of range (D) and azimuth (beta) were calculated for polar coordinate system. The true trajectory is plotted and shown in Fig 1.
- The mesaurements for the range (D) and azimuth (beta) are generated with predefined respective variances of measurement noises.
- Since there are two different coordinate systems (i.e. the cartesian coordinate for motion model and polar coordinate system for measurements), an Extended Kalman filter algorithm is used for this case. The initial conditions are fitted into the system for the given conditions. The transition matrix, state noise covariance matrix and measurement noise covarianice matrix are created.
- Due to the conditions of different coordinate systems (non-linear system), the measurements are linearized by differentiating the extrapolated state vector at filtration step of Extended Kalman filter algorithm. The result of implementation of this algorithm is shown in Fig 2 and Fig 3 for cartesian and polar coordinate systems respectively. The predicted and filtered estimates coincides with the true trajectory and hence the Extended Kalman filter works.
- This Extended Kalman filter algorithm is ran for 500 runs for calculating the estimation errors for range (D) and azimuth (beta) for extrapolated and filtered estimates. Fig 4 shows the comparison of errors of estimates for Extrapolated and Filtered states with their respetive standard deviation. The filter reduces the errors by a factor of ~3.3 in range (D) and by ~2.6 in azimuth (beta) and reduces them almost to zero

for azimuth. Hence, for a non-linear system a Exteneded Kalman filter algorithm can be used to reduce the errors by noises in the system.