```matlab
%% Extended Kalman filter for navigation and tracking

% Written by Irina Yareshko and Luca Breggion, Skoltech 2022

close all
clear
clc

set(0,'defaulttextInterpreter','latex');
set(groot,'defaultAxesTickLabelInterpreter','latex');
set(groot,'defaultLegendInterpreter','latex');

%%

T=1; N=500;
InititalState = [1000;10;1000;10]; %[x0; Vx0; y0; Vy0]
sigma_a = 0.3;      %Variance of acceleration noise
sigma_D = 50;       %variance of range noise of measurements
sigma_b = 0.004;    %variance of azimuth noise of measurements

%Creating of arrays for running Kalman filter M times
M=500;                            %number of runs
Err_range_filtered  = zeros(M,N); %Filtraion error of range
Err_range_forecast  = zeros(M,N); %Predicted error of range
Err_azimut_filtered = zeros(M,N); %Filtraion error of azimut
Err_azimut_forecast = zeros(M,N); %Predicted error of azimut
Condition_nums      = zeros(M,N); %array of average condition number
K_matr_values       = zeros(M,N);

Mean_Z_x     = zeros(1,N);
Mean_azimuth = zeros(1,N);

Mean_range = zeros(1,N);
for i=1:M
    %Generation of deterministic trajectory and its measurements
    [TruePolar,TrueCart,Z_c,Z_p] = ...
        Generation_true_determ(N,T,sigma_D,sigma_b,InititalState, sigma_a);

    %Applying or Kalman filter for measurements
    [Z_filtered, FiltrErr_CovMatr, PredErr_CovMatr, range_fe, azimuth_fe, CondMatr, K_matr] = ...
        Kalman(Z_c,Z_p,T,sigma_D,sigma_b, sigma_a);
    Err_range_filtered(i,:) = (TruePolar(1,:) - range_fe(1,:)).^2;
    Err_range_forecast(i,:) = (TruePolar(1,:) - range_fe(2,:)).^2;
    Err_azimut_filtered(i,:)= (TruePolar(2,:) - azimuth_fe(1,:)).^2;
    Err_azimut_forecast(i,:)= (TruePolar(2,:) - azimuth_fe(2,:)).^2;

    Condition_nums(i,:) = CondMatr;
    K_matr_values(i,:) = K_matr(1,1,:);

    Mean_Z_x = Mean_Z_x + Z_filtered(1,2:N+1);
    Mean_azimuth = Mean_azimuth + azimuth_fe(1,:);
end
```

```
Mean_Z_x = Mean_Z_x/M;
Mean_azimuth = Mean_azimuth/M;

FinalErr_range_filtered =sqrt(1/(M-1)*sum(Err_range_filtered));
FinalErr_range_forecast =sqrt(1/(M-1)*sum(Err_range_forecast));
FinalErr_azimut_filtered = sqrt(1/(M-1)*sum(Err_azimut_filtered));
FinalErr_azimut_forecast = sqrt(1/(M-1)*sum(Err_azimut_forecast));
Final_CN=sum(Condition_nums)/M;
Final_K_matr=sum(K_matr_values)/M;

figure(1)
polarplot(TruePolar(2,:),TruePolar(1,:), 'b', 'LineWidth', 1.2);
legend('True motion', 'FontSize', 20);
grid on; grid minor;

figure(2)
polarplot(Z_p(2,:),Z_p(1,:),'r.', 'LineWidth', 1.2);
legend('Measurements', 'FontSize', 20);
grid on; grid minor;

figure(3)
polarplot(azimuth_fe(1,:), range_fe(1,:), 'c', 'LineWidth', 1.2);
legend('Filtered Estimate', 'FontSize', 20);
grid on; grid minor;

figure(4)
polarplot(azimuth_fe(2,:), range_fe(2,:), 'b', 'LineWidth', 1.2);
legend('Extrapolated Estimate', 'FontSize', 20);
grid on; grid minor;

figure(5)
plot(3:N,FinalErr_range_filtered(3:N),...
     3:N,FinalErr_range_forecast(3:N),...
     3:N,sigma_D*ones(1,N-2), 'black', 'LineWidth', 1.2);
legend('True filtration error','True extrapolation error','$\sigma_D$',
'FontSize', 20);
title('a) Errors of range', 'FontSize', 20);
xlabel('Step', 'FontSize', 20)
ylabel('Errors', 'FontSize', 20)
grid on; grid minor;

figure(6)
plot(3:N,FinalErr_azimut_filtered(3:N), ...
     3:N,FinalErr_azimut_forecast(3:N), ...
     3:N,sigma_b*ones(1,N-2), 'black', 'LineWidth', 1.2);
legend('True filtration error','True extrapolation error','$\sigma_\beta$',
'FontSize', 20);
title('b) Errors of azimuth', 'FontSize', 20);
xlabel('Step', 'FontSize', 20)
ylabel('Errors', 'FontSize', 20)
grid on; grid minor;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                          %
%                                FUNCTION                                  %
%                                                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [True_polar,True_Cartesian,Z_c,Z_p] = ...
    Generation_true_determ(size_,T,sigma_D,sigma_b, InititalState, sigma_a)
% Generation of true trajectory
    X   = zeros(1,size_); X(1)   = InititalState(1);
    Y   = zeros(1,size_); Y(1)   = InititalState(3);
    V_x = zeros(1,size_); V_x(1) = InititalState(2);
    V_y = zeros(1,size_); V_y(1) = InititalState(4);
    a_x = zeros(1,size_ - 1);
    a_y = zeros(1,size_ - 1);
    for i=2:size_
        a_x(i-1) = randn * sigma_a;
        a_y(i-1) = randn * sigma_a;
        V_x(i)=V_x(i-1) + a_x(i-1)*T;
        V_y(i)=V_y(i-1) + a_y(i-1)*T;
        X(i)=X(i-1)+V_x(i-1)*T + 0.5*a_x(i-1)*T^2;
        Y(i)=Y(i-1)+V_y(i-1)*T + 0.5*a_y(i-1)*T^2;
    end
    True_Cartesian = [X; V_x; Y; V_y];
    %Generation of true values of range D and aimuth b
    D = sqrt(X.^2+Y.^2);
    b = atan(X./Y);
    True_polar = [D;b];
    %Generation of measurements
    D_m=zeros(1,size_); %array of measurements of range
    b_m=zeros(1,size_); %array of measurements of azimuth
    x_m=zeros(1,size_); %array of pseudo-measurements of x
    y_m=zeros(1,size_); %array of pseudo-measurements of y
    Z_c=zeros(2,size_); %array of pseudo-measurements in Cartesian coordinates
    Z_p=zeros(2,size_); %array of measurements in polar coordinates
    for i=1:size_
        D_m(i)=D(i)+randn*sigma_D;
        b_m(i)=b(i)+randn*sigma_b;

        x_m(i)=D_m(i)*sin(b_m(i));
        y_m(i)=D_m(i)*cos(b_m(i));

        Z_p(:,i)=[D_m(i);b_m(i)];
        Z_c(:,i)=[x_m(i);y_m(i)];
    end
end

function [Z_filtered, P, P_pred, range_fe, azimuth_fe, CondMatr, K] ...
    = Kalman(Z_cart, Z_polar, T, sigma_D,sigma_b, sigma_a)
%   Description:
    size_ = length(Z_cart);
    Z_filtered = zeros(4, size_ + 1);            %Filtered data
    Z_forecast = zeros(4, size_);                %Forecast data [x; Vx; y; Vy]
```

```
    P = zeros(4, 4, size_ + 1);                     %Filtration error covariance matrix
    P_pred  = zeros(4, 4, size_ + 1);               %Prediction error covariance matrix
    K = zeros(4, 2, size_);
    Z_filtered(:, 1) = [Z_polar(1,1)*sin(Z_polar(2,1)); 0;...
                        Z_polar(1,1)*cos(Z_polar(2,1)); 0]; %Initian state vector
    P(:, :, 1) = [10^10 0 0 0; 0 10^10 0 0; 0 0 10^10 0; 0 0 0 10^10];

    TransMatr = [1 T 0 0; 0 1 0 0; 0 0 1 T; 0 0 0 1]; %Φ
    InputMatr = [0.5*T^2 0; T 0; 0 0.5*T^2; 0 T]; %G
    CovMatr_StateNoise   = (InputMatr*InputMatr')*sigma_a^2; %Covariance matrix of state noise
    CovMatr_MeasureNoise = [sigma_D^2 0; 0 sigma_b^2];

    range_fe = zeros(2,size_);   %array of filtered and extrapolated range
    azimuth_fe = zeros(2,size_); %array of filtered and extrapolated azimuth
    CondMatr = zeros(1,size_);
    dhdx = zeros(2, 4);
    for i = 2:size_+1
        %Prediction part
        Z_forecast(:,i-1) =  TransMatr*Z_filtered(:,i-1);
        P_pred(:,:,i-1) = ...
            TransMatr*P(:,:,i-1)*TransMatr' + CovMatr_StateNoise;
        %Filtrarion part
        h = [sqrt(Z_forecast(1,i-1)^2 + Z_forecast(3,i-1)^2 );...
             atan(Z_forecast(1,i-1)/Z_forecast(3,i-1)) ];
        dhdx(1,1) =  Z_forecast(1,i-1)/sqrt( (Z_forecast(1,i-1))^2 + (Z_forecast(3,i-1))^2 );
        dhdx(1,3) =  Z_forecast(3,i-1)/sqrt( (Z_forecast(1,i-1))^2 + (Z_forecast(3,i-1))^2 );
        dhdx(2,1) =  Z_forecast(3,i-1)/(Z_forecast(1,i-1)^2 + Z_forecast(3,i-1)^2 );
        dhdx(2,3) = -Z_forecast(1,i-1)/(Z_forecast(1,i-1)^2 + Z_forecast(3,i-1)^2 );

        K(:,:,i-1) = P_pred(:,:,i-1)*(dhdx') * ...
            (dhdx*P_pred(:,:,i-1)*dhdx' + CovMatr_MeasureNoise)^(-1);
        P(:, :, i) = (eye(4)-K(:,:,i-1)*dhdx)*P_pred(:,:,i-1);
        Z_filtered(:,i) = Z_forecast(:,i - 1) + ...
            K(:,:,i-1)*(Z_polar(:,i-1) - h);

        range_fe(1,i-1)=sqrt( Z_filtered(1,i)^2+Z_filtered(3,i)^2 );
        range_fe(2,i-1)=sqrt( Z_forecast(1,i-1)^2+Z_forecast(3,i-1)^2);
        azimuth_fe(1,i-1)=atan(Z_filtered(1,i)/Z_filtered(3,i));
        azimuth_fe(2,i-1)=atan(Z_forecast(1,i-1)/Z_forecast(3,i-1));

        if (sigma_D^2) > (Z_polar(1,i - 1)^2*sigma_b^2)
            CondMatr(i - 1) = (sigma_D^2)/(Z_polar(1,i - 1)^2*sigma_b^2);
        else
            CondMatr(i - 1) = (Z_polar(1,i - 1)^2*sigma_b^2)/(sigma_D^2);
        end
```

```
    end
end
```