Assignment 12

Joint assimilation of navigation data coming from different sources

Group 6:

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


```
clc
clear
close all
```

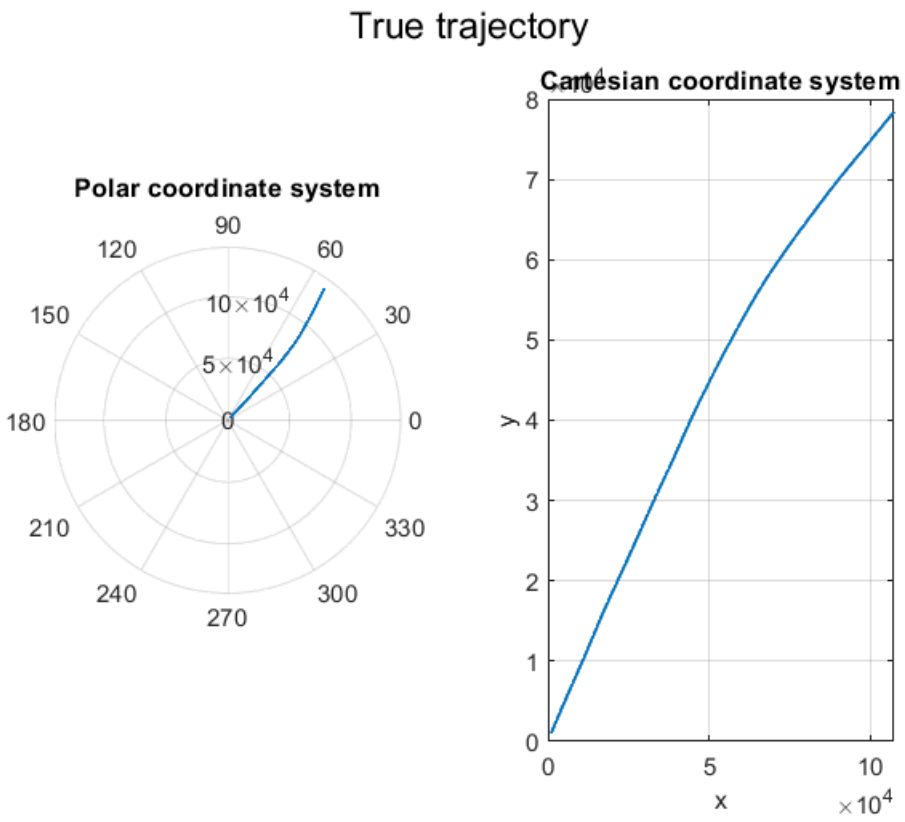
Task 1: Generate a true trajectory X

```
n = 500; % Size of trajectory
T = 2;  % Interval between measurements

%Initial components of velocity
V = zeros(2,n);
V(1,1)= 1e2; % V_x
V(2,1)= 1e2; % 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
```



Task 3: Generate measurements of D and provided by first observer that arrive every 4 seconds.

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

eta_odd = randn(2,n);
eta_odd(1,:) = eta_odd(1,:)*sqrt(var_D);    % eta_D_odd
eta_odd(2,:) = eta_odd(2,:)*sqrt(var_beta); % eta_beta_odd

D_odd = zeros(1,n);
beta_odd = zeros(1,n);
for i = 1:n
    if mod(i,2) == 0
        D_odd(i) = NaN;
```

```
        beta_odd(i) = NaN;
    else
        D_odd(i) = D(i) + eta_odd(1,i);
        beta_odd(i) = beta(i) + eta_odd(2,i);
    end
end
```

Task 4: Generate more accurate measurements of azimuth provided by second observer that arrive between measurement times of the first observer.

```
var_beta_even = 0.001^2;
eta_even = randn(1,n)*sqrt(var_beta_even);

beta_even = zeros(1,n);
for i = 4:n
    if mod(i,2) == 0
        beta_even(i) = beta(i) + eta_even(i);
    else
        beta_even(i) = NaN;
    end
end
```

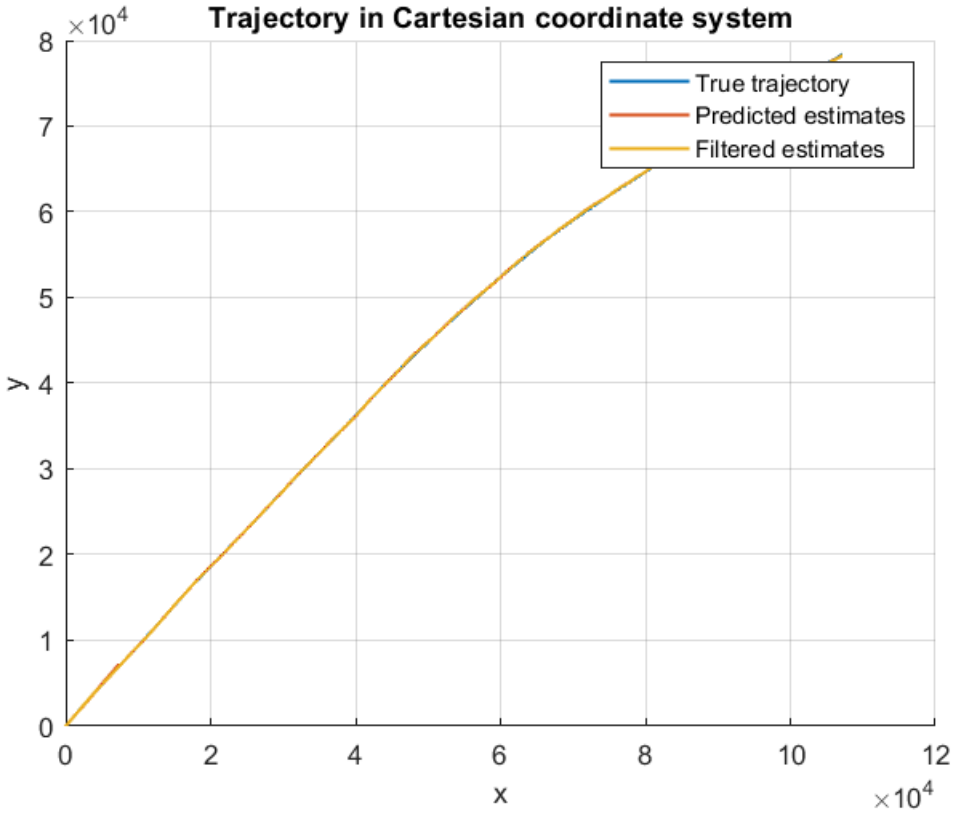
Task 5: Initial conditions for Extended Kalman filter algorithm

```
x1_m = D_odd(1)*sin(beta_odd(1));
x3_m = D_odd(3)*sin(beta_odd(3));
y1_m = D_odd(1)*cos(beta_odd(1));
y3_m = D_odd(3)*cos(beta_odd(3));

X = zeros(4,n);
X(:,3) = [x3_m (x3_m-x1_m)/(2*T) y3_m (y3_m-y1_m)/(2*T)]'; % Initial filtered estimate of state
P = 1e10*eye(4); % Initial filtration error covariance matrix

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

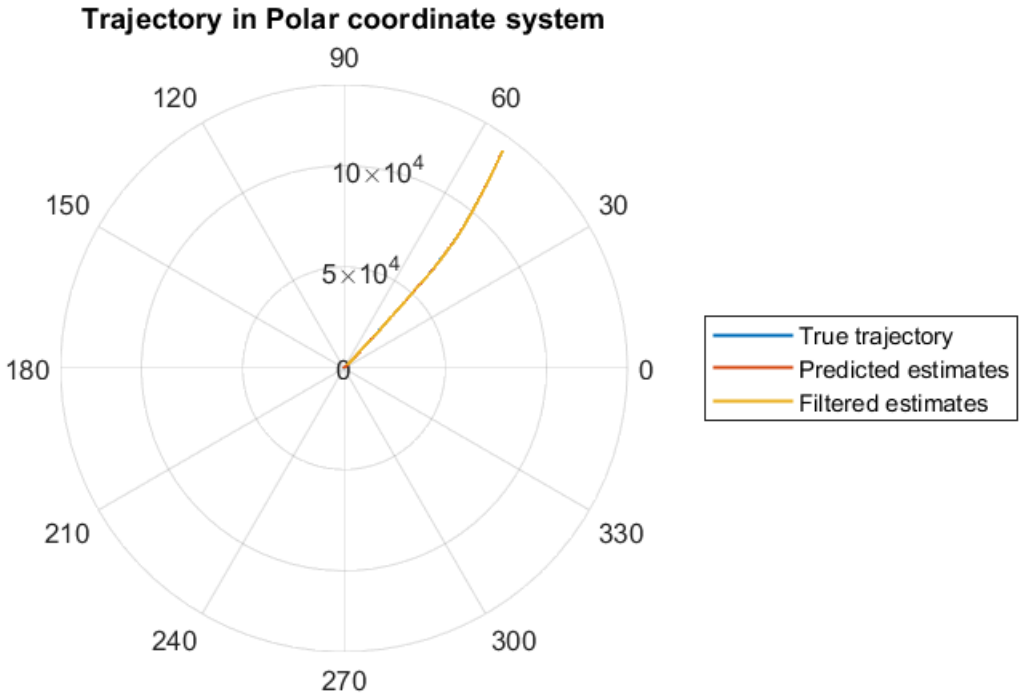
Task 6: Develop Kalman filter algorithm to estimate state vector (extrapolation and filtration).

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

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

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

polar_e = zeros(2,n);
X_pred = zeros(4,n);
```

```
for i = 4: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

    if mod(i,2) == 0
        R = var_beta_even;
        z = beta_even;

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

        % Filtration
        K = P_pred*dH'/(dH*P_pred*dH'+R);
        X(:,i) = X_pred(:,i) + K*(z(:,i)-polar_e(2,i));
        P = (eye(4)-K*dH)*P_pred;
    else
        R = [var_D 0; 0 var_beta];
        z = [D_odd ; beta_odd];

        % 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;
    end

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



Task 7: Run Kalman filter algorithm over M = 500 runs.

```
M = 500;

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

    eta_odd = randn(2,n);
    eta_odd(1,:) = eta_odd(1,:)*sqrt(var_D);    % eta_D_odd
    eta_odd(2,:) = eta_odd(2,:)*sqrt(var_beta); % eta_beta_odd
    eta_even = randn(1,n)*sqrt(var_beta_even);  % eta_beta_even

    beta_even = zeros(1,n);
    D_odd = zeros(1,n);
    beta_odd = zeros(1,n);
    for i = 1:n
        if mod(i,2) == 0
            D_odd(i) = NaN;
```

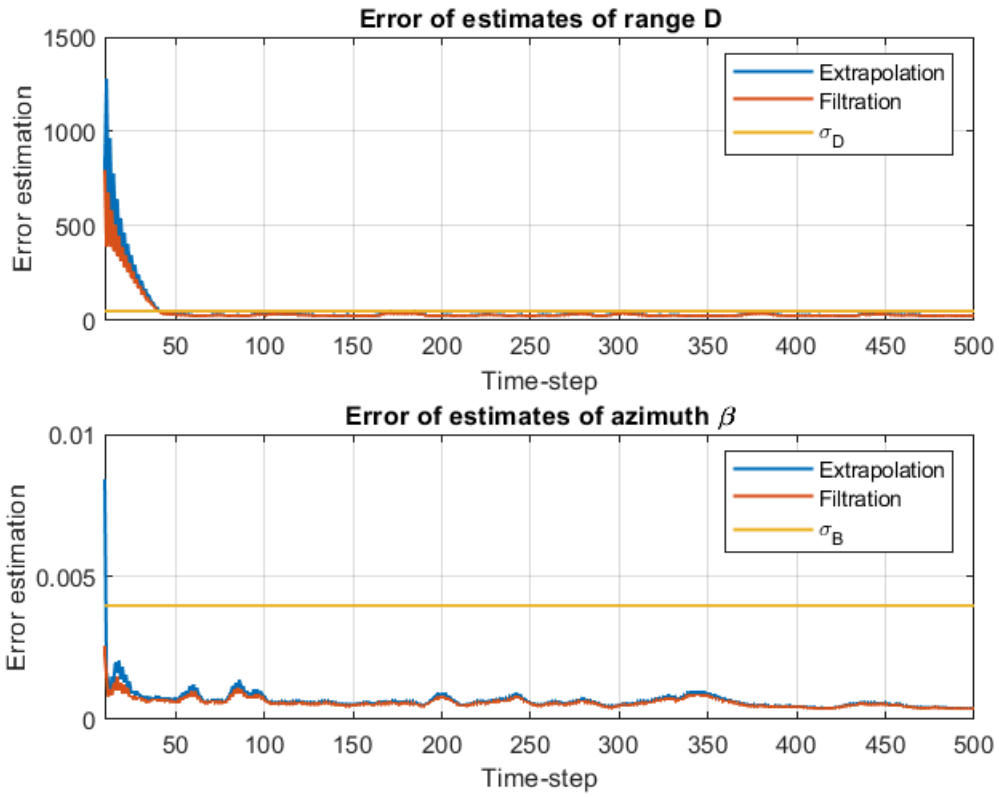
```
            beta_odd(i) = NaN;
            beta_even(i) = beta(i) + eta_even(i);
        else
            D_odd(i) = D(i) + eta_odd(1,i);
            beta_odd(i) = beta(i) + eta_odd(2,i);
            beta_even(i) = NaN;
        end
    end

    x1_m = D_odd(1)*sin(beta_odd(1));
    x3_m = D_odd(3)*sin(beta_odd(3));
    y1_m = D_odd(1)*cos(beta_odd(1));
    y3_m = D_odd(3)*cos(beta_odd(3));

    X = zeros(4,n);
    X(:,3) = [x3_m (x3_m-x1_m)/(2*T) y3_m (y3_m-y1_m)/(2*T)]'; % Initial filtered estimate of s
    P = 1e10*eye(4); % Initial filtration error covariance matrix

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

    polar_e = zeros(2,n);
    X_pred = zeros(4,n);
    for i = 4: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

        if mod(i,2) == 0
            R = var_beta_even;
            z = beta_even;

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

            % Filtration
            K = P_pred*dH'/(dH*P_pred*dH'+R);
            X(:,i) = X_pred(:,i) + K*(z(:,i)-polar_e(2,i));
            P = (eye(4)-K*dH)*P_pred;
        else
            R = [var_D 0; 0 var_beta];
            z = [D_odd ; beta_odd];

            % 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;
        end

        % 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 = 4: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));
```

Task 8: Compare estimation results with measurement errors of D and beta.

```
figure
sgtitle("Fig.4 Errors comparison")
subplot(2,1,1)
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([10 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([10 inf])
```



Conclusions:

1) The true trajectory is generated for an object motion with normally distributed random acceleration in cartesian coordinate system. The true values of range (D) and azimuth (beta) are generated in polar coordinate system. The true trajectory is plotted and shown in Fig 1.

2) The measurements for the range (D) and azimuth (beta) are generated. They are provided by two observers. Signal from the first observer arrives every 4 seconds with respective variances of measurement noises. The more accurate measurements of the azimuth (beta) are generated provided by the second observer which arrive between the measurement times of the first observer with different variance of measurement noise. The measurements for first observer are gained at odd time steps and for second observer at even time steps.

3) For the filtration of this joint assimilation of navigation data from different sources, an Extended Kalman filter algorithm is developed. The initial conditions are fitted into the system for the given conditions. The transition matrix and state noise covariance matrix are created. The measurement noise covariance matrix is created differently for first and second observers as the second observer has only azimuth (beta) measurements. Due to the conditions of different coordinate system (non-linear system), the measurements are linearized in Extended Kalman filtering process at filtration step by differentiating the extrapolated state vector which is also different for both observers. The plot for this algorithm is plotted in Fig 2 and Fig 3 for cartesian and polar coordinate systems respectively. The predicted and filtered estimates coincide with the true trajectory and hence the Extended Kalman filter works.

4) This Extended Kalman filter algorithm is ran 500 times for calculating the estimation errors for range (D) and azimuth (beta) at extrapolated and filtered estimates. Fig 4 shows the comparison of errors of estimates for Extrapolated and Filtered states with their respective standard deviation. There are not much change in the errors of estimates for range (D), the error decreases by ~1.5 factor to the standard deviation. On the other hand we can observe that there is large reduction in the errors of filtration for azimuth (beta). This is because the accurate measurements provided by the 2nd observer increased the accuracy of the filter thus providing better results.

5) Oscillations with opposite phases are observed on the plot with Extrapolation and Filtration graphs. Because at time step i (even) filtration uses better measurement of beta (beta even) while extrapolation is made with less accurate estimation (at the previous time step), and at i+1 extrapolation uses more accurate filtration and filtration uses beta_odd