KALMAN FILTER

- Ajay Charan
UG201211002

"The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modelled system is unknown." (G. Welch and G. Bishop, 2004)

EM algorithm is used to obtain the parameters of the Kalman Filter, learned through the training set. The following EM algorithm is based on the summary by Ghahramani and Hinton (1996) of the algorithm originally by Shumway and Stoffer (1982) for estimating the parameters of linear dynamical systems from corrupted observations.

The algorithm consists of an estimation step ("E step"), which estimates the true state using a Kalman-Rauch filter, combined with a measurement step ("M step"), which gives the maximum likelihood estimates of the parameters given the data and the estimate of the true state.

State-space model
Assuming a linear state space model:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + \mathbf{w}_t$$

$$\mathbf{y}_t = C\mathbf{x}_t + \mathbf{v}_t.$$

Where,
Wt ~ Normal(0,Q) and Vt ~ f(0,R)
xt is the state of the system
yt is observations
Q captures the process error associated with state dynamics
R is the variability associated with sampling error of observation

Only *yt*, is observed; the underlying parameters, and the underlying true population size, *xt*, is hidden. If we make the assumption that *vt* is normally distributed, then the model is a linear Gaussian state-space model. Hence, we can write the following conditional probabilities:

$$P(y_t \mid x_t) = \exp\left\{-\frac{(y_t - x_t)^2}{2R}\right\}(2\pi|R|)^{-1/2}$$

$$P(x_t \mid x_{t-1}) = \exp\left\{-\frac{(x_t - (x_{t-1} + B))^2}{2Q}\right\}(2\pi|Q|)^{-1/2}$$

Using the Markov property implicit in the model, the joint probability of the observed time series, *yT*1 { } ={*y*1, *y*2, …, *y*T} and the true state, *xT*1 { } ={*x*1, *x*2, …, *x*T}, is

$$P(\{x\}_1^T, \{y\}_1^T) = P(x_1)\prod_{t=2}^{T} P(x_t \mid x_{t-1})\prod_{t=1}^{T} P(y_t \mid x_t)$$

The joint log likelihood of *yT*1 { } , *xT*1 { } is:

$$\log P(\{x\}_1^T, \{y\}_1^T) = -\sum_{t=1}^{T} \frac{(y_t - x_t)^2}{2R} - \frac{T}{2}\log|R|$$
$$-\sum_{t=2}^{T} \frac{(x_t - (x_{t-1} + B))^2}{2Q} - \frac{T-1}{2}\log|Q|$$
$$-\frac{(x_1 - \pi_1)^2}{2V_1} - \frac{1}{2}\log|V_1| - T\log 2\pi$$

The goal is to find the estimates of *xt*, *B*, *R*, *Q*, π1 and *V*1 that maximize log P(*xT*1 { } , *yT*1 { }). The following EM algorithm does this.

The EM algorithm
The EM algorithm has four basic steps:

0) Compute some initial parameter estimates, from which to start the algorithm.

1) Generate an estimate of *xt* by using the Kalman-Raush recursionwhich gives the maximum likelihood estimates of xt, given 1 1 *R*ˆ,*Q*ˆ, *B*ˆ,πˆ ,*V*ˆ , and *yt*.
The maximum likelihood estimate is denoted *x*ˆt .

2) Update the *R*ˆ,*Q*ˆ, *B*ˆ,πˆ ,*V*ˆ given the new *x*ˆt by finding the estimates which maximize the updated expected log likelihood (using the updated *x*ˆt).

3) Check to see if this quantity has converged and no longer increases. If not converged, return to step 1.

Step 0. Compute initial parameter estimates
To get good final estimates one needs to start the algorithm with reasonable initial parameter estimates.

Step 1. The Kalman-Raush recursion (E step)
The first part uses the Kalman recursion to estimate $E(x_t \mid \{y\}_1^t)$. This is a forward recursion since we work forward to generate it. The second part uses the Raush recursion to work backwards and compute $E(x_t \mid \{y\}_1^T)$ from $E(x_t \mid \{y\}_1^t)$. First, some notation:

$$\{y\}_1^\tau \equiv \{y_1, y_2, \ldots, y_\tau\}$$
$$\hat{x}_t \equiv E[x_t \mid \{y\}_1^T]$$
$$x_t^\tau \equiv E[x_t \mid \{y\}_1^\tau]$$
$$V_t^\tau \equiv \mathrm{Var}[x_t \mid \{y\}_1^\tau]$$
$$V_{t,t-1}^\tau \equiv \mathrm{Cov}[x_t, x_{t-1} \mid \{y\}_1^\tau]$$
$$P_t \equiv E[x_t x_t \mid \{y\}_1^T] \equiv V_t^T + x_t^T x_t^T$$
$$P_{t,t-1} \equiv E[x_t x_{t-1} \mid \{y\}_1^T] \equiv V_{t,t-1}^T + x_t^T x_{t-1}^T$$

The ultimate goal of these recursions is to compute *x*ˆt , *Pt*, and *Pt,t*-1, which will be needed for step 2 of the algorithm.

The Kalman recursion
To compute *xtt* and *Vtt* , start at *t* = 1 and step forward to *T*. At each step, compute:

$$x_t^{t-1} = \begin{cases} \hat{\pi}_1 & \text{for } t = 1 \\ x_{t-1}^{t-1} + \hat{B} & \text{for } t > 1 \end{cases}$$

$$V_t^{t-1} = \begin{cases} \hat{V}_1 & \text{for } t = 1 \\ V_{t-1}^{t-1} + \hat{Q} & \text{for } t > 1 \end{cases}$$

$$K_t = \frac{V_t^{t-1}}{\left(V_t^{t-1} + \hat{R}\right)}$$

$$x_t^t = x_t^{t-1} + K_t\left(y_t - x_t^{t-1}\right)$$

$$V_t^t = V_t^{t-1} - K_t V_t^{t-1}$$

This is a form of Kalman filter, where, *A*=1, *C*=1 and *ut* =1, so the filter is simplified quite a bit.

The Rauch recursion

Next we work backwards from *t* = *T* back to *t* = 2, to compute *xTt* and *VTt*. This recursion requires the *xtt*, *Vtt*, and *Vt-1t* that were generated during the Kalman recursion.

$$J_{t-1} = \frac{V_{t-1}^{t-1}}{V_t^{t-1}}$$

$$x_{t-1}^T = x_{t-1}^{t-1} + J_{t-1}\left(x_t^T - \left(x_{t-1}^{t-1} + B\right)\right)$$

$$V_{t-1}^T = V_{t-1}^{t-1} + J_{t-1}\left(V_t^T - V_t^{t-1}\right)J_{t-1}$$

One more recursion

Using *Jt* from the Rauch recursion with *Kt* and *Vtt* from the Kalman recursion, we do another backwards recursion to compute, *VTt* ,t −1 . Starting from *t* = *T* work backwards to *t* = 2, and at each step compute

$$V_{t,t-1}^T = \begin{cases} (1 - K_T)V_{T-1}^{T-1} & \text{for } t = T \\ V_{t-1}^{t-1}J_{t-2} + J_{t-1}\left(V_{t,t-1}^T - V_{t-1}^{t-1}\right)J_{t-2} & \text{for } t < T \end{cases}$$

Putting it all together

Using the three recursions, we can then compute the following, which are needed for M step of the algorithm.

$$\hat{x}_t = x_t^T$$

$$P_t = V_t^T + x_t^T x_t^T$$

$$P_{t,t-1} = V_{t,t-1}^T + x_t^T x_{t-1}^T$$

Step 2 Generate new parameter estimates (M step)

The new expected log likelihood function is given by with the new *xt* estimate, *x*ˆt :

$\psi = E(\log P(\{x\}_1^T, \{y\}_1^T) \mid \{y\}_1^T]$ using new *x*ˆt .

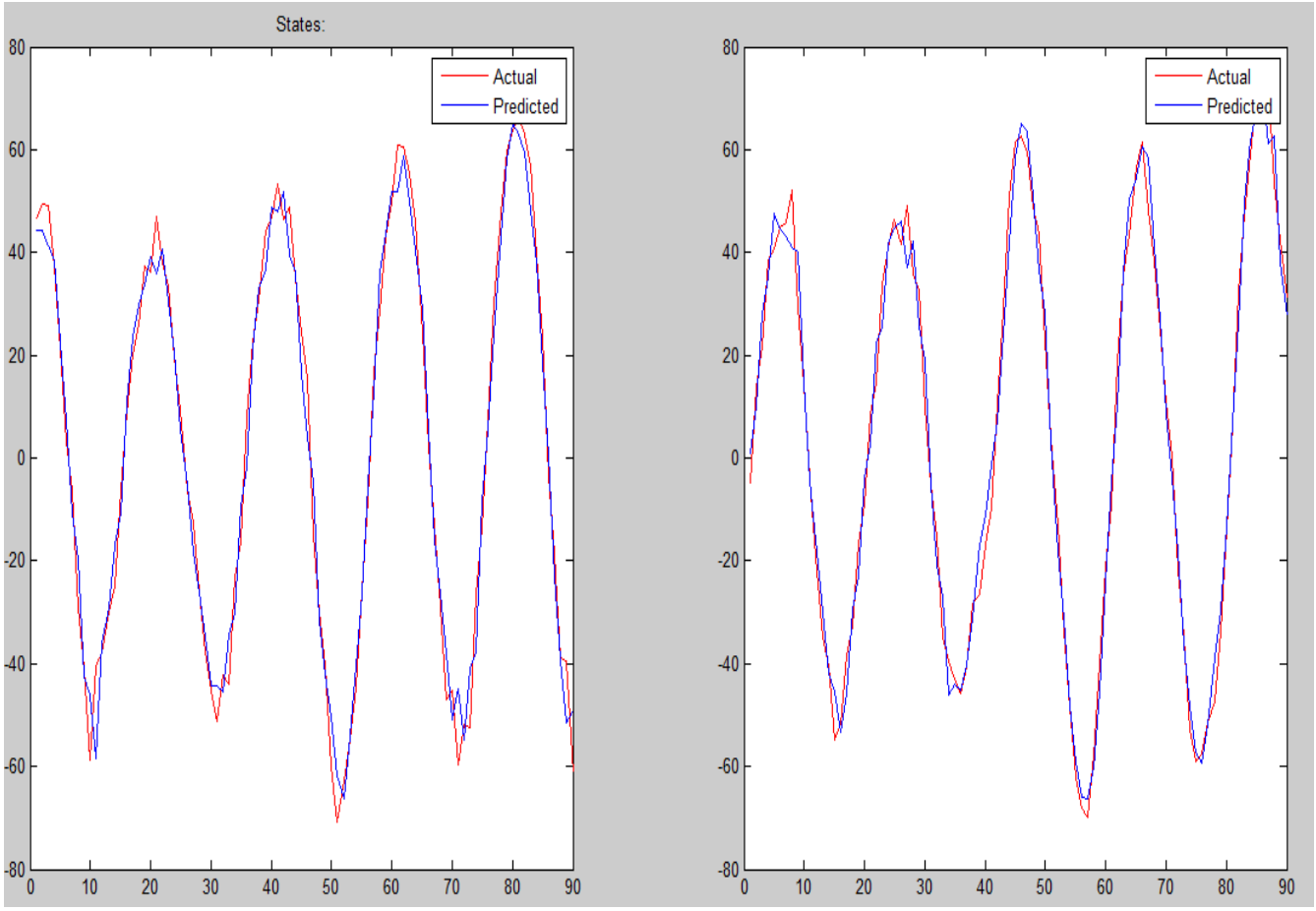
To compute the new parameter estimates, we find the new 1*R*ˆ,*Q*ˆ, *B*ˆ,πˆ ,*V*ˆ that maximize the new ψ. To do this, we take the partial derivative of ψ with respect to each parameter, set the derivative to zero and solve for the maximizing parameter.

Step 3 Check for convergence

We compare the new ψ to the previously estimated ψ and check if the difference is less than some threshold.

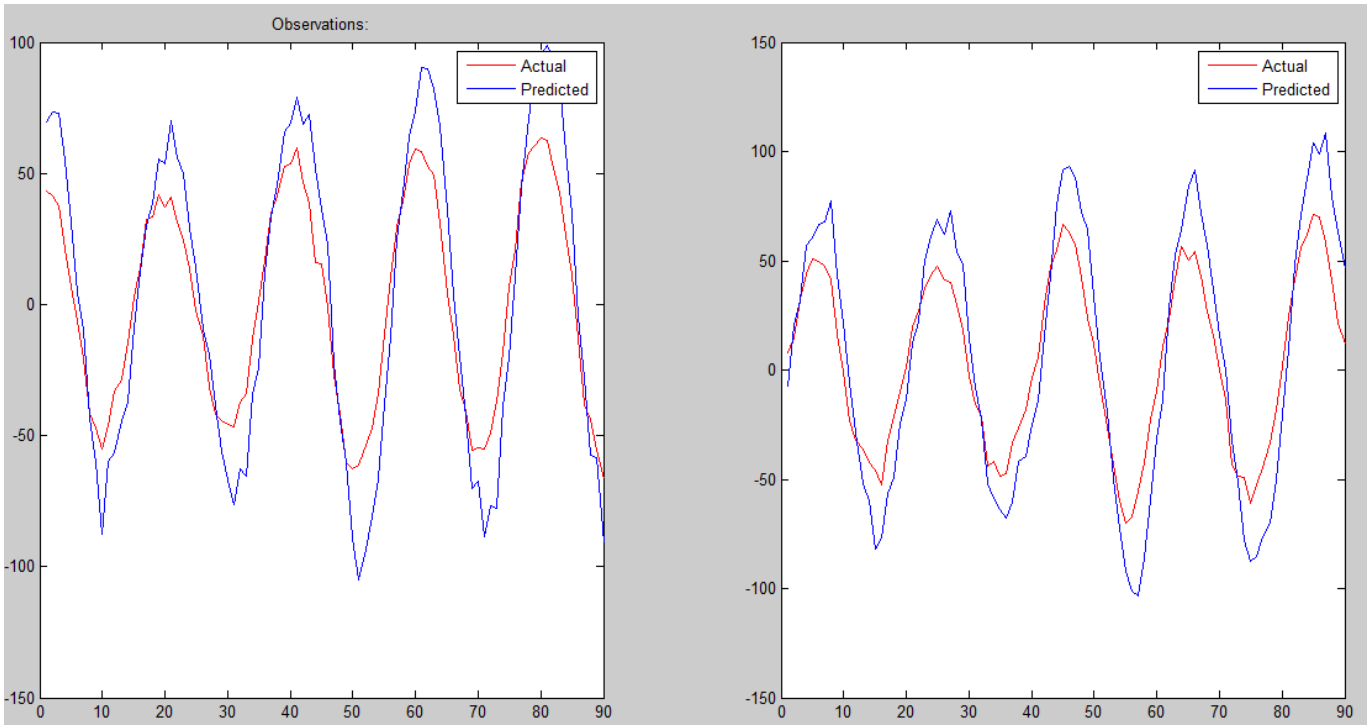
Results:

We take the provided dataset and perform a split of 80:20 where we take 361 training examples and remaining data for testing purpose.
On training the Kalman filter for 50 iterations and threshold of 0.01, and then testing, following results were obtained:



Here we see that the predicted states closely follow the actual ones on the testset.

For the case of observations, we have the following plot:



Again, we see that the predicted observations follow the actual ones but the relative error in this case is higher.

Source Code:

trainKalman.m

```
clc
clear all
close all

% Load training data
trainData = xlsread('trainData.xlsx');

% State
x = trainData(:,1:2)';

% Observation
y = trainData(:,3:4)';

% Set up the parameters
thrsh = 0.01;

A = 1*eye(2);
Q = 1*eye(2);
C = 1*eye(2);
R = 1*eye(2);

pi1 = x(:,1);
V1 = 0.1*eye(2);

num_itr_max = 50;
```

```
A_initial = A;
Q_initial =Q;
C_initial = C;
R_initial = R;

pi1_initial = pi1;
V1_initial = V1;

T=length(y);

LL=[];

converged = 0;
prev_log_lik = -Inf;

itr_num = 0;

while(itr_num <= num_itr_max)

    % Initializing
    xtt = zeros(2,T);
    xtt1 = zeros(2,T);
    xtT = zeros(2,T);
    Vtt = zeros(2,2*T);
    Vtt1 = zeros(2,2*T);
    Vtt1T = zeros(2,2*T);
    VtT = zeros(2,2*T);
    J = zeros(2,2*T);


    % E-Step
    for t=1:T
        if(t==1)
            % For first iteration, start with intitial values
            xtt1(:,1) = pi1;
            Vtt1(:,2*t-1:2*t) = V1;
        else
            % Run Kalman Filter Forward
            xtt1(:,t) = A*xtt(:,t-1);
            Vtt1(:,2*t-1:2*t) = A*Vtt(:,2*t-3:2*t-2)*A' + Q;
        end
        Kt = Vtt1(:,2*t-1:2*t)*C' * inv(C*Vtt1(:,2*t-1:2*t)*C' + R);
        xtt(:,t) = xtt1(:,t) + Kt*(y(:,t) - C*xtt1(:,t));
        Vtt(:,2*t-1:2*t) = Vtt1(:,2*t-1:2*t)-Kt*C*Vtt1(:,2*t-1:2*t);
    end

    KT = Kt;

    xtT(:,T) = xtt(:,T);
    VtT(:,2*T-1:2*T) = Vtt(:,2*T-1:2*T);

    for t=T:-1:2
        % J is the ratio of v's at t and t-1
        J(:,2*t-3:2*t-2) = Vtt(:,2*t-3:2*t-2)*A'*inv(Vtt1(:,2*t-1:2*t));
        % get x and v by rauch recursion (back ward recursion)
        xtT(:,t-1) = xtt(:,t-1) + J(:,2*t-3:2*t-2)*(xtT(:,t)-A*xtt(:,t-1));
```

```
        VtT(:,2*t-3:2*t-2) = Vtt(:,2*t-3:2*t-2) + J(:,2*t-3:2*t-
2)*(VtT(:,2*t-1:2*t)-Vtt1(:,2*t-1:2*t))*J(:,2*t-3:2*t-2)';
    end

    x_hat = xtT;
    Pt = zeros(size(VtT));
    for t=1:T

        Pt(:,2*t-1:2*t) = VtT(:,2*t-1:2*t) + xtT(:,t)*xtT(:,t)';
    end
    % covariance between x and x at t-1
    Vtt1T(:,2*T-1:2*T) = (1 - KT*C)*A*Vtt(:,2*T-3:2*T-2);
    for t=T:-1:3
        Vtt1T(:,2*t-3:2*t-2) = Vtt(:,2*t-3:2*t-2)*J(:,2*t-5:2*t-4)' +
J(:,2*t-3:2*t-2)*(Vtt1T(:,2*t-1:2*t)- A*Vtt(:,2*t-3:2*t-2))*J(:,2*t-5:2*t-
4)';
    end

    Ptt1 = zeros(size(VtT));
    for t=2:T
        Ptt1(:,2*t-1:2*t) = Vtt1T(:,2*t-1:2*t) + xtT(:,t)*xtT(:,t-1)';
    end

    % Compute log likelihoods
    loglik_part1 =0;
    for t=1:T
        loglik_part1 = loglik_part1 - (y(:,t)-
C*x_hat(:,t))'*inv(R)*(y(:,t)-C*x_hat(:,t));
    end
    loglik_part1 = loglik_part1 - T*log(abs(det(R)));
    loglik_part1 = loglik_part1/2;

    loglik_part2 =0;
    for t=2:T
        loglik_part2 = loglik_part2 - (x_hat(:,t)-(A*x_hat(:,t-
1)))'*inv(Q)*(x_hat(:,t)-(A*x_hat(:,t-1)));
    end
    loglik_part2 = loglik_part2 - (T-1)*log(abs(det(Q)));
    loglik_part2 = loglik_part2/2;

    loglik_part3 = - ((x_hat(1)-pi1)'*inv(V1)*(x_hat(1)-pi1))/2 -
log(abs(det(V1)))/2 - T*log(2*pi);

    loglik =  loglik_part1 + loglik_part2 + loglik_part3;
    LL=[LL loglik];



    % M-Step
    num = 0;
    den = 0;
    for t=1:T
        num = num + y(:,t)*x_hat(:,t)';
        den = den + Pt(:,2*t-1:2*t);
    end
    C_new = num*inv(den);

    R_new = 0;
    for t=1:T
        R_new = R_new + y(:,t)*y(:,t)' - C_new*x_hat(:,t)*y(:,t)';
```

```
    end
    R_new = R_new/T;

    num = 0;
    den = 0;
    for t=2:T
        num = num + Ptt1(:,2*t-1:2*t);
        den = den + Pt(:,2*t-3:2*t-2);
    end
    A_new = num*inv(den);

    part1 = 0;
    part2 = 0;
    for t=2:T
        part1 = part1 + Pt(:,2*t-1:2*t);
        part2 = part2 + Ptt1(:,2*t-1:2*t);
    end
    Q_new = (part1 - A_new*part2)/(T-1);

    pi1_new = x_hat(:,1);
    V1_new = Pt(:,1:2)-x_hat(:,1)*x_hat(:,1)';

    pi1 = pi1_new;
    V1 = V1_new;
    A = A_new;
    Q = Q_new;
    C = C_new;
    R = R_new;

    % Check for convergence
    if (abs(loglik - prev_log_lik) / ((abs(loglik) + abs(prev_log_lik) +
eps)/2)) < thrsh
        break;
    end
    prev_log_lik = loglik;
    itr_num = itr_num+1;
end

C = C+0.4*eye(2);
```

testKalman.m

```
close all

% Load Data
testData = xlsread('testData.xlsx');
Tlen = length(testData);
x_test = testData(:,1:2)';
y_test = testData(:,3:4)';

% Run motion model of learned filter
x_bel = A*x_test(:,1:Tlen-1);

% Run observation model of learned filter
y_bel = C*x_test;

% % Plotting
% figure(1)
% x_test1 = x_test(1,2:end);
```

```
% x_bel1 = x_bel(1,:);
% x_test2 = x_test(2,2:end);
% y_bel2 = x_bel(2,:);
% plot(x_test1, x_test2, '-r');
% hold on;
% plot(x_bel1, x_bel2, '-b');
%
% figure(2)
% y_test1 = y_test(1,2:end);
% y_bel1 = y_bel(1,:);
% y_test2 = y_test(2,2:end);
% y_bel2 = y_bel(2,:);
% plot(y_test1, y_test2, '-r');
% hold on;
% plot(y_bel1, y_bel2, '-b');
figure(1)
x_test1 = x_test(1,2:end);
x_bel1 = x_bel(1,:);
subplot(1, 2, 1)
plot(x_test1,'-r')
title('States:');
hold on;
subplot(1, 2, 1)
plot(x_bel1,'-b')
legend('Actual','Predicted');

x_test2 = x_test(2,2:end);
x_bel2 = x_bel(2,:);
subplot(1, 2, 2)
plot(x_test2,'-r')
hold on;
subplot(1, 2, 2)
plot(x_bel2,'-b')
legend('Actual','Predicted');
hold off

figure(2)
y_test1 = y_test(1,2:end);
y_bel1 = y_bel(1,2:end);
subplot(1, 2, 1)
plot(y_test1,'-r')
title('Observations:');
hold on;
subplot(1, 2, 1)
plot(y_bel1,'-b')
legend('Actual','Predicted');

y_test2 = y_test(2,2:end);
y_belt2 = y_bel(2,2:end);
subplot(1, 2, 2)
plot(y_test2,'-r')
hold on;
subplot(1, 2, 2)
plot(y_belt2,'-b')
legend('Actual','Predicted');
```