Kalman 滤波和 MATLAB 实现

卡尔曼滤波器（Kalman Filter）是一个最优化自回归数据处理算法（optimal recursive data processing algorithm）。对于解决很大部分的问题，他是最优，效率最高甚至是最有用的。他的广泛应用已经超过 30 年，包括机器人导航，控制，传感器数据融合甚至在军事方面的雷达系统以及导弹追踪等等。近年来更被应用于计算机图像处理，例如头脸识别，图像分割，图像边缘检测等等

简单来说，卡尔曼滤波器是一个“optimal recursive data processing algorithm（最优化自回归数据处理算法）”。对于解决很大部分的问题，他是最优，效率最高甚至是最有用的。他的广泛应用已经超过 30 年，包括机器人导航，控制，传感器数据融合甚至在军事方面的雷达系统以及导弹追踪等等。近年来更被应用于计算机图像处理，例如头脸识别，图像分割，图像边缘检测等等。

为了可以更加容易的理解卡尔曼滤波器，这里应用形象的描述方法讲解，不像参考书那样罗列一大堆的数学公式和数学符号。但是，他的 5 条公式是其核心内容。结合现代的计算机，其实卡尔曼的程序相当的简单，只要你理解了他的那 5 条公式。

假设我们要研究的对象是一个房间的温度。根据你的经验判断，这个房间的温度是恒定的，也就是下一分钟的温度等于现在这一分钟的温度。假设你对你的经验不是 100%的相信，可能会有上下偏差几度。我们把这些偏差看成是高斯白噪声，也就是这些偏差跟前后时间是没有关系的而且符合高斯分配。另外，我们在房间里放一个温度计，但是这个温度计也是不准确的，测量值会比实际值有偏差。我们也把这些偏差看成是高斯白噪声。

好了，现在对于某一分钟我们有两个有关该房间的温度值：你根据经验的预测值（系统的预测值）和温度计的值（测量值）。下面我们要用这两个值结合他们各自的噪声来估算出房间的实际温度值。

假如我们要估算 k 时刻的实际温度值。首先你要根据 k-1 时刻的温度值，来预测 k 时刻的温度。因为你相信温度是恒定的，所以你会得到 k 时刻的温度预测值是跟 k-1 时刻一样的，假定是 23 度，同时该值的高斯噪声的偏差是 5 度（5 是这样得到的：如果 k-1 时刻估算出的最优温度值的偏差是 3，你对自己预测的不确定度是 4 度，他们平方相加再开方，就是 5）。然后，你从温度计那里得到了 k 时刻的温度值，假设是 25 度，同时该值的偏差是 4 度。

由于我们用于估算 k 时刻的实际温度有两个温度值，分别是 23 度和 25 度。究竟实际温度是多少呢？相信自己还是相信温度计呢？究竟相信谁多一点，我们可以用他们的协方差（covariance）来判断。因为 Kg^2=5^2/(5^2+4^2)，所以 Kg=0.78，我们可以估算出 k 时刻的实际温度值是：23+0.78*(25-23)=24.56 度。可以看出，因为温度计的 covariance 比较小（比较相信温度计），所以估算出的最优温度值偏向温度计的值。

现在我们已经得到 k 时刻的最优温度值了，下一步就是要进入 k+1 时刻，进行新的最优估算。到现在为止，好像还没看到什么自回归的东西出现。对了，在进入 k+1 时刻之前，我们还要算出 k 时刻那个最优值（24.56 度）的偏差。算法如下：((1-Kg)*5^2)^0.5=2.35。这里的 5 就是上面的 k 时刻你预测的那个 23 度温度值的偏差，得出的 2.35 就是进入 k+1 时刻以后 k 时刻估算出的最优温度值的偏差（对应于上面的 3）。

就是这样，卡尔曼滤波器就不断的把 covariance 递归，从而估算出最优的温度值。他运行的很快，而且它只保留了上一时刻的 covariance。上面的 Kg，就是卡尔曼增益（Kalman Gain）。他可以随不同的时刻而改变他自己的值。

Dr Kalman 的卡尔曼滤波器。涉及一些基本的概念知识，包括概率（Probability），随机变量（Random Variable），高斯或正态分配（Gaussian Distribution）等。

首先，要引入一个离散控制过程的系统。该系统可用一个线性随机微分方程来描述：

X(k)=A X(k-1)+B U(k)+W(k)

再加上系统的测量值：

Z(k)=H X(k)+V(k)

上两式子中，X(k)是 k 时刻的系统状态，U(k)是 k 时刻对系统的控制量。A 和 B 是系统参数，对于多模型系统，他们为矩阵。Z(k)是 k 时刻的测量值，H 是测量系统的参数，对于多测量系统，H 为矩阵。W(k)和 V(k)分别表示过程和测量的噪声。他们被假设成高斯白噪声，他们的协方差（covariance ）分别是 Q，R（这里我们假设他们不随系统状态变化而变化）。

对于满足上面的条件(线性随机微分系统，过程和测量都是高斯白噪声)，卡尔曼滤波器是最优的信息处理器。下面我们来用他们结合他们的 covariances 来估算系统的最优化输出（类似上一节那个温度的例子）。

首先我们要利用系统的过程模型，来预测下一状态的系统。假设现在的系统状态是 k，根据系统的模型，可以基于系统的上一状态而预测出现在状态：

X(k|k-1)=A X(k-1|k-1)+B U(k) ……... (1)

式(1)中，X(k|k-1)是利用上一状态预测的结果，X(k-1|k-1)是上一状态最优的结果，U(k)为现在状态的控制量，如果没有控制量，它可以为 0。

到现在为止，我们的系统结果已经更新了，可是，对应于 X(k|k-1)的 covariance 还没更新。我们用 P 表示 covariance：

P(k|k-1)=A P(k-1|k-1) A’+Q …… (2)

式(2)中，P(k|k-1)是 X(k|k-1)对应的 covariance，P(k-1|k-1)是 X(k-1|k-1)对应的 covariance，A’表示 A 的转置矩阵，Q 是系统过程的 covariance。式子 1，2 就是卡尔曼滤波器 5 个公式当中的前两个，也就是对系统的预测。

现在我们有了现在状态的预测结果，然后我们再收集现在状态的测量值。结合预测值和

测量值，我们可以得到现在状态(k)的最优化估算值 X(k|k)：

X(k|k)= X(k|k-1)+Kg(k) (Z(k)-H X(k|k-1)) …… (3)

其中 Kg 为卡尔曼增益(Kalman Gain)：

Kg(k)= P(k|k-1) H’ / (H P(k|k-1) H’ + R) …… (4)

到现在为止，我们已经得到了 k 状态下最优的估算值 X(k|k)。但是为了要另卡尔曼滤波器不断的运行下去直到系统过程结束，我们还要更新 k 状态下 X(k|k)的 covariance：

P(k|k)=（I-Kg(k) H）P(k|k-1) …… (5)

其中 I 为 1 的矩阵，对于单模型单测量，I=1。当系统进入 k+1 状态时，P(k|k)就是式子(2)的 P(k-1|k-1)。这样，算法就可以自回归的运算下去。

卡尔曼滤波器的原理基本描述了，式子 1，2，3，4 和 5 就是他的 5 个基本公式。根据这 5 个公式，可以很容易的实现计算机的程序。

这里举一个简单的例子来说明卡尔曼滤波器的工作过程。把房间看成一个系统，然后对这个系统建模。房间的温度是跟前一时刻的温度相同的，所以 A=1。没有控制量，所以 U(k)=0。因此得出：

X(k|k-1)=X(k-1|k-1) …….. (6)

式子（2）可以改成：

P(k|k-1)=P(k-1|k-1) +Q …… (7)

因为测量的值是温度计的，跟温度直接对应，所以 H=1。式子 3，4，5 可以改成以下：

X(k|k)= X(k|k-1)+Kg(k) (Z(k)-X(k|k-1)) …… (8)

Kg(k)= P(k|k-1) / (P(k|k-1) + R) …… (9)

P(k|k)=（1-Kg(k)）P(k|k-1) …… (10)

现在模拟一组测量值作为输入。假设房间的真实温度为 25 度，模拟 200 个测量值，这些测量值的平均值为 25 度，但是加入了标准偏差为几度的高斯白噪声。

为了令卡尔曼滤波器开始工作，需要告诉卡尔曼两个零时刻的初始值，是 X(0|0)和 P(0|0)。他们的值不用太在意，随便给一个就可以了，因为随着卡尔曼的工作，X 会逐渐的收敛。但是对于 P，一般不要取 0，因为这样可能会令卡尔曼完全相信你给定的 X(0|0)是系统最优的，从而使算法不能收敛。选取 X(0|0)=1 度，P(0|0)=10。

该系统的真实温度为 25 度，图中用黑线表示。图中红线是卡尔曼滤波器输出的最优化结果（该结果在算法中设置了 Q=1e-6，R=1e-1）。

附 matlab 下面的 kalman 滤波程序：

```
clear

N=200;

w(1)=0;

w=randn(1,N)

x(1)=0;

a=1;

for k=2:N;

x(k)=a*x(k-1)+w(k-1);

end

V=randn(1,N);

q1=std(V);

Rvv=q1.^2;

q2=std(x);

Rxx=q2.^2;

q3=std(w);

Rww=q3.^2;

c=0.2;

Y=c*x+V;

p(1)=0;
```

tiankong

```
s(1)=0;
for t=2:N;
p1(t)=a.^2*p(t-1)+Rww;
b(t)=c*p1(t)/(c.^2*p1(t)+Rvv);
s(t)=a*s(t-1)+b(t)*(Y(t)-a*c*s(t-1));
p(t)=p1(t)-c*b(t)*p1(t);
end
t=1:N;
plot(t,s,'r',t,Y,'g',t,x,'b');
function [x, V, VV, loglik] = kalman_filter(y, A, C, Q, R, init_x, init_V, varargin)
% Kalman filter.
% [x, V, VV, loglik] = kalman_filter(y, A, C, Q, R, init_x, init_V, ...)
%
% INPUTS:
% y(:,t) - the observation at time t
% A - the system matrix
% C - the observation matrix
% Q - the system covariance
% R - the observation covariance
% init_x - the initial state (column) vector
% init_V - the initial state covariance
%
```

```
% OPTIONAL INPUTS (string/value pairs [default in brackets])
% 'model' - model(t)=m means use params from model m at time t [ones(1,T) ]
% In this case, all the above matrices take an additional final dimension,
% i.e., A(:,:,m), C(:,:,m), Q(:,:,m), R(:,:,m).
% However, init_x and init_V are independent of model(1).
% 'u' - u(:,t) the control signal at time t [ [] ]
% 'B' - B(:,:,m) the input regression matrix for model m
%
% OUTPUTS (where X is the hidden state being estimated)
% x(:,t) = E[X(:,t) | y(:,1:t)]
% V(:,:,t) = Cov[X(:,t) | y(:,1:t)]
% VV(:,:,t) = Cov[X(:,t), X(:,t-1) | y(:,1:t)] t >= 2
% loglik = sum{t=1}^T log P(y(:,t))
%
% If an input signal is specified, we also condition on it:
% e.g., x(:,t) = E[X(:,t) | y(:,1:t), u(:, 1:t)]
% If a model sequence is specified, we also condition on it:
% e.g., x(:,t) = E[X(:,t) | y(:,1:t), u(:, 1:t), m(1:t)]
[os T] = size(y);
ss = size(A,1); % size of state space
% set default params
model = ones(1,T);
```

tiankong

```
u = [];

B = [];

ndx = [];

args = varargin;

nargs = length(args);

for i=1:2:nargs

switch args

case 'model', model = args{i+1};

case 'u', u = args{i+1};

case 'B', B = args{i+1};

case 'ndx', ndx = args{i+1};

otherwise, error(['unrecognized argument ' args])

end

end

x = zeros(ss, T);

V = zeros(ss, ss, T);

VV = zeros(ss, ss, T);

loglik = 0;

for t=1:T

m = model(t);

if t==1

%prevx = init_x(:,m);
```

tiankong

```
%prevV = init_V(:,:,m);

prevx = init_x;

prevV = init_V;

initial = 1;

else

prevx = x(:,t-1);

prevV = V(:,:,t-1);

initial = 0;

end

if isempty(u)

[x(:,t), V(:,:,t), LL, VV(:,:,t)] = ...

kalman_update(A(:,:,m), C(:,:,m), Q(:,:,m), R(:,:,m), y(:,t), prevx, prevV, 'initial', initial);

else

if isempty(ndx)

[x(:,t), V(:,:,t), LL, VV(:,:,t)] = ...

kalman_update(A(:,:,m), C(:,:,m), Q(:,:,m), R(:,:,m), y(:,t), prevx, prevV, ...

'initial', initial, 'u', u(:,t), 'B', B(:,:,m));

else

i = ndx;

% copy over all elements; only some will get updated

x(:,t) = prevx;

prevP = inv(prevV);
```

```
prevPsmall = prevP(i,i);

prevVsmall = inv(prevPsmall);

[x(i,t), smallV, LL, VV(i,i,t)] = ...

kalman_update(A(i,i,m), C(:,i,m), Q(i,i,m), R(:,:,m), y(:,t), prevx(i), prevVsmall, ...

'initial', initial, 'u', u(:,t), 'B', B(i,:,m));

smallP = inv(smallV);

prevP(i,i) = smallP;

V(:,:,t) = inv(prevP);

end

end

loglik = loglik + LL;

end
```

最佳线性滤波理论起源于40年代美国科学家Wiener和前苏联科学家Колмогоров等人的研究工作，后人统称为维纳滤波理论。从理论上说，维纳滤波的最大缺点是必须用到无限过去的数据，不适用于实时处理。为了克服这一缺点，60年代Kalman把状态空间模型引入滤波理论，并导出了一套递推估计算法，后人称之为卡尔曼滤波理论。卡尔曼滤波是以最小均方误差为估计的最佳准则，来寻求一套递推估计的算法，其基本思想是：采用信号与噪声的状态空间模型，利用前一时刻地估计值和现时刻的观测值来更新对状态变量的估计，求出现时刻的估计值。它适合于实时处理和计算机运算。

现设线性时变系统的离散状态方程和观测方程为：

$X(k) = F(k,k-1)\cdot X(k-1)+T(k,k-1)\cdot U(k-1)$

$Y(k) = H(k)\cdot X(k)+N(k)$

其中

X(k)和 Y(k)分别是 k 时刻的状态矢量和观测矢量

F(k,k-1)为状态转移矩阵

tiankong

U(k)为 k 时刻动态噪声

T(k,k-1)为系统控制矩阵

H(k)为 k 时刻观测矩阵

N(k)为 k 时刻观测噪声

则卡尔曼滤波的算法流程为：

预估计 X(k)^= F(k,k-1)•X(k-1)

计算预估计协方差矩阵
C(k)^=F(k,k-1)×C(k)×F(k,k-1)'+T(k,k-1)×Q(k)×T(k,k-1)'
Q(k) = U(k)×U(k)'

计算卡尔曼增益矩阵
K(k) = C(k)^×H(k)'×[H(k)×C(k)^×H(k)'+R(k)]^(-1)
R(k) = N(k)×N(k)'

更新估计
X(k)~=X(k)^+K(k)×[Y(k)-H(k)×X(k)^]

计算更新后估计协防差矩阵
C(k)~ = [I-K(k)×H(k)]×C(k)^×[I-K(k)×H(k)]'+K(k)×R(k)×K(k)'

X(k+1) = X(k)~
C(k+1) = C(k)~
重复以上步骤

Matlab 实现代码
****
****

%%%% Constant Velocity Model Kalman Filter Simulation %%%%

%==========
======

clear all; close all; clc;

%% Initial condition
ts = 1; % Sampling time
t = [0:ts:100];

```
T = length(t);

%% Initial state
x = [0 40 0 20]';
x_hat = [0 0 0 0]';

%% Process noise covariance
q = 5
Q = q*eye(2);

%% Measurement noise covariance
r = 5
R = r*eye(2);

%% Process and measurement noise
w = sqrt(Q)*randn(2,T);      % Process noise
v = sqrt(R)*randn(2,T);      % Measurement noise

%% Estimate error covariance initialization
p = 5;
P(:,:,1) = p*eye(4);

%================================================================
======

%% Continuous-time state space model
%{
x_dot(t) = Ax(t)+Bu(t)
z(t) = Cx(t)+Dn(t)
%}
A = [0 1 0 0;
        0 0 0 0;
        0 0 0 1;
        0 0 0 0];
B = [0 0;
        1 0;
        0 0;
        0 1];
C = [1 0 0 0;
        0 0 1 0];
D = [1 0;
        0 1];

%% Discrete-time state space model
```

tiankong

```
%{
x(k+1) = Fx(k)+Gw(k)
z(k) = Hx(k)+Iv(k)

Continuous to discrete form by zoh
%}
sysc = ss(A,B,C,D);
sysd = c2d(sysc, ts, 'zoh');
[F G H I] = ssdata(sysd);

%% Practice state of target
for i = 1:T-1
    x(:,i+1) = F*x(:,i);
end

x = x+G*w;      % State variable with noise
z = H*x+I*v; % Measurement value with noise

%==========================================================================
======

%%% Kalman Filter
for i = 1:T-1

%% Prediction phase
    x_hat(:,i+1) = F*x_hat(:,i);
    % State estimate predict
    P(:,:,i+1) = F*P(:,:,i)*F'+G*Q*G';
    % Tracking error covariance predict
    P_predicted(:,:,i+1) = P(:,:,i+1);

%% Kalman gain
    K = P(:,:,i+1)*H'*inv(H*P(:,:,i+1)*H'+R);

%% Updata step
    x_hat(:,i+1) = x_hat(:,i+1)+K*(z(:,i+1)-H*x_hat(:,i+1));
    % State estimate update
    P(:,:,i+1) = P(:,:,i+1)-K*H*P(:,:,i+1);
    % Tracking error covariance update
    P_updated(:,:,i+1) = P(:,:,i+1);

end

%==========================================================================
```

tiankong

======

```
%% Estimate error
    x_error = x-x_hat;

%% Graph 1 practical and tracking position
figure(1)
plot(x(1,:),x(3,:),'r');
hold on;
plot(x_hat(1,:),x_hat(3,:),'g.');
title('2D Target Position')
legend('Practical Position','Tracking Position')
xlabel('X axis [m]')
ylabel('Y axis [m]')
hold off;

%% Graph 2
figure(2)
plot(t,x(1,:)),grid on;
hold on;
plot(t,x_hat(1,:),'r'),grid on;
title('Practical and Tracking Position on X axis')
legend('Practical Position','Tracking Position')
xlabel('Time [sec]')
ylabel('Position [m]')
hold off;

%% Graph 3
figure(3)
plot(t,x_error(1,:)),grid on;
title('Position Error on X axis')
xlabel('Time [sec]')
ylabel('Position RMSE [m]')
hold off;

%% Graph 4
figure(4)
plot(t,x(2,:)),grid on;
hold on;
plot(t,x_hat(2,:),'r'),grid on;
title('Practical and Tracking Velocity on X axis')
legend('Practical Velocity','Tracking Velocity')
xlabel('Time [sec]')
ylabel('Velocity [m/sec]')
```

```
hold off;

%% Graph 5
figure(5)
plot(t,x_error(2,:)),grid on;
title('Velocity Error on X axis')
xlabel('Time [sec]')
ylabel('Velocity RMSE [m/sec]')
hold off;

%=========================================================================
======
```