# 最小二乘估计

## 思路

1. 首先需要明确，卡尔曼滤波的Xk估计值的更新公式实际上是由加权最小二乘的目标函数J得来的，加权最小二乘的目标函数J是因，卡尔曼滤波的Xk估计值是果，将J对Xk求导，令求导之后的函数为0，即可得到当Xk为何值时，目标函数的值最小。求出来的Xk的结果即为卡尔曼滤波的Xk估计值更新公式。其中，求导过程参考矩阵求导公式。[https://blog.csdn.net/daaikuaichuan/article/details/80620518]，如果是复合函数，从外层函数到内层函数依次求导，如果遇到的函数是关于x的向量，则按向量对向量的求导规则来，如果遇到的函数是关于x的标量，则按向量对标量的求导规则来。以及用到了Woodbury matrix identity ==》 
2. 分为当前时刻先验估计和前一时刻最优估计两种方法。两种方法都必须先找出线性回归形式，然后求出线性回归的最优估计值。
3. 当前时刻先验估计方法，首先写出加权最小二乘的目标函数，然后根据目标函数写出其线性回归形式，由于新的线性回归形式Y=Hx+e中的e发生了变化，误差e的协方差矩阵也发生了变化，在写出其最优估计值形式的时候，其R矩阵也变为了新的误差e的协方差矩阵。该误差协方差矩阵刻画的是先验估计误差协方差矩阵P\_pre(先验x和真实x之间误差方差)和R(观测方程噪声误差方差）之间的关系。
4. 前一时刻最优估计方法，首先根据最初的系统状态方程和观测方程变换其形式，凑出X\_{k-1|k-1}的形式，新的线性回归方程同样需要与3点一样，找出其新的误差e的协方差矩阵，写出新的R矩阵。论文中对线性回归形式做了归一化，归一化后的线性回归形式中误差的协方差矩阵变成了单位阵，Y-Hx矩阵中的每一行都变为了1。相当于最开始的加权最小二乘目标函数中的权重都变为了1。如果 要把最小二乘目标函数更换为其他的目标函数，归一化后会更方便。

## 公式1 当前时刻的先验估计



**公式1.1 最小二乘的目标函数**

其中X\_{k}是k时刻的X真实值，X\_hat{k|k-1}是k时刻的先验估计值，P\_{k|k-1}是k时刻的先验误差协方差矩阵，Rk是观测方程误差方差矩阵。



**公式1.2 先验估计误差**



**公式1.3 观测方程误差**



**公式1.4 新的线性回归（忽略中间的横线)**

**说明：这是将公式1.2和公式1.3写成矩阵形式之后的格式，这样的格式就是线性回归，从而解得线性回归的最小二乘解。将公式1.4看成公式1.4.1**

****

**公式1.4.1 线性回归矩阵形式**

**其中，,,**

根据《最优状态估计》P60的推导，可以得到公式1.5.



其中，

**公式1.5 加权最小二乘估计值**

X\_hat即为k时刻的最优估计值，也就是说我们根据k时刻的先验估计值得到了k时刻的最优估计值，在下一次的循环迭代过程中，我们根据公式1.6得到了k+1时刻的先验估计值，从而继续迭代。



**公式1.6 先验预测**

## 公式2 前一时刻的最优估计



**公式2.1 系统状态方程和观测方程**



**公式2.2 线性回归方程Hx=Y+e**

**说明：在《robust kalman filter based on a generalized maximum-likelihood>>论文和《robust kalmanfiltering》两篇论文中，解释道该线性回归方程的来历，首先根据公式2.1写出大致的公式2.2的形式，其次，要凑出前一时刻的最有估计这个式子，因为这个公式就是根据前一时刻的最有估计来做的。**



再同时将方程左右两边同时乘以Sk^(-1)归一化。归一化的目的见前文。



**公式2.3 归一化后的线性回归方程**

对于公式2.3，，，，

其中，β最终可以写为公式2.5.

原先的，对于Sk的求解，由于相乘是一个对角矩阵，所以，归一化后的误差协方差矩阵变为单位阵，，相当于将公式1.1变为了公式2.4。



**公式2.4 变换后的目标函数**



公式2.5 最小二乘估计值

## 代码1 当前时刻先验估计

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%function main3\_3\_2 最小二乘 用当前时刻的先验估计

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N=100; %仿真时间，时间序列总数

vector=zeros(1000,100); % 位移的误差

vector1=zeros(1000,100); % 速度的误差

zero1=[0 ;0];

zero2=[0 0];

for kk=1:1000

% 定义一个行向量存储每次实验每个时刻的方差

% 噪声

Q=[0,0;0,0]; % 过程噪声方差为0，即下落过程忽略空气阻力

R=1; % 观测噪声方差

W=sqrt(Q)\*randn(2,N);% 既然Q为0，则W=0；在此写出，方便对照理解

V=sqrt(R)\*randn(1,N);% 测量噪声V(k)

% 系统矩阵

A=[1,1;0,1]; %状态转移矩阵

B=[0.5;1]; %控制量

U=-1;

H=[1,0]; %观测矩阵

% 初始化

X=zeros(2,N); % 物体真实状态

X(:,1)=[95;1]; % 初始位移和速度

P0=[10,0;0,1]; % 初始误差

Z=zeros(1,N);

Z(1)=H\*X(:,1); % 初始观测值

Xls=zeros(2,N);% Kalman估计状态初始化

Xls(:,1)=X(:,1)+sqrt(P0)\*randn(2,1); % 在初始化的时候加入误差方差估计值 从而保持吻合

err\_P=zeros(N,2);

err\_P(1,1)=P0(1,1);

err\_P(1,2)=P0(2,2);

I=eye(2); % 二维系统

% 第一列

vector(kk,1)=Xls(1,1)-X(1,1);

vector1(kk,1)=Xls(2,1)-X(2,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for k=2:N

%物体下落，受状态方程的驱动

X(:,k)=A\*X(:,k-1)+B\*U+W(k);

% 位移传感器对目标进行观测

Z(k)=H\*X(:,k)+V(k);

% Kalman滤波

X\_pre=A\*Xls(:,k-1)+B\*U; %状态预测

P\_pre=A\*P0\*A'+Q; %协方差预测

Hf=[eye(2);H]; % 线性回归Y=Hx+e 中的H矩阵

Y=[X\_pre;Z(k)]; % 线性回归Y=Hx+e 中的Y矩阵

Rt=[P\_pre zeros(2,1);zeros(1,2) R];% 新的Rt, Y=Hx+e 中e的误差协方差矩阵

Kg=P\_pre\*H'\*(H\*P\_pre\*H'+R)^(-1); %计算Kalman增益

Xls(:,k)=(Hf'\*Rt^(-1)\*Hf)^(-1)\*Hf'\*Rt^(-1)\*Y; % 最小二乘估计

P0=(I-Kg\*H)\*P\_pre\*(I-Kg\*H)'+Kg\*R\*Kg';%方差更新

vector(kk,k)=Xls(1,k)-X(1,k);

vector1(kk,k)=Xls(2,k)-X(2,k);

% 误差均方值

err\_P(k,1)=P0(1,1);

err\_P(k,2)=P0(2,2);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

% 计算vector矩阵中 每一列的方差 存储在行向量中

Difference=var(vector,0,1);

Difference1=var(vector1,0,1);

errPx=transpose(err\_P);

errPx1=errPx(1,:);

errPx2=errPx(2,:);

% 将err\_P(k,1)和Difference()画在同一张图中，比较卡尔曼滤波估计出来的Pk与实际的方差相差多少.

figure

plot(Difference,'-bo'); % 实际的位移误差方差

hold on;

plot(errPx1,'-g+');

legend('通过最小二乘得到的实际位移误差方差','kalman估计方差');

xlabel('采样时间/s');

ylabel('方差');

figure

plot(Difference1,'-bo'); %实际的速度误差方差

hold on;

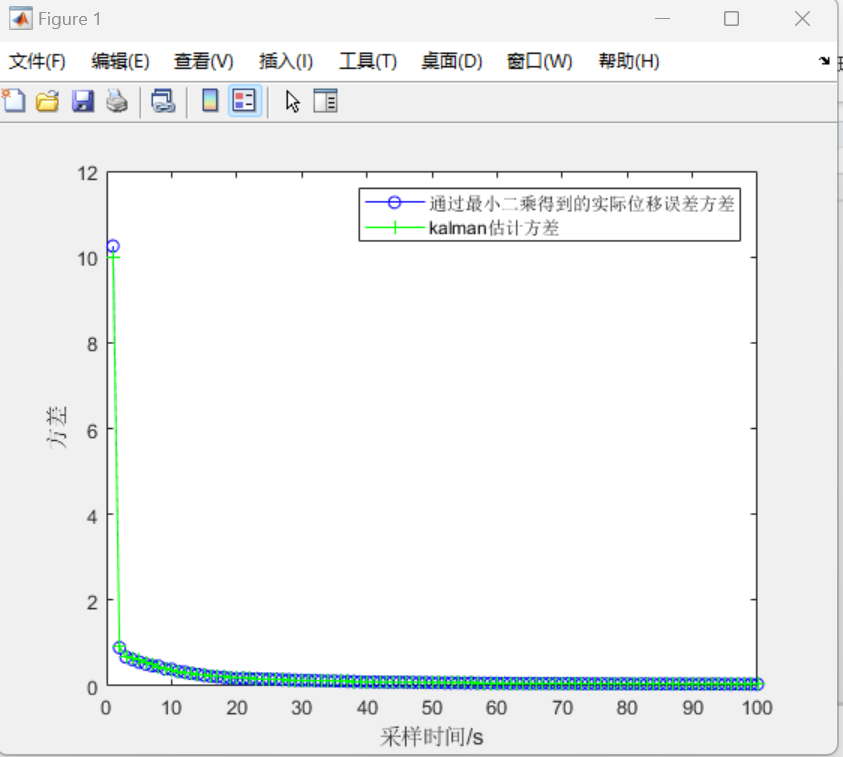
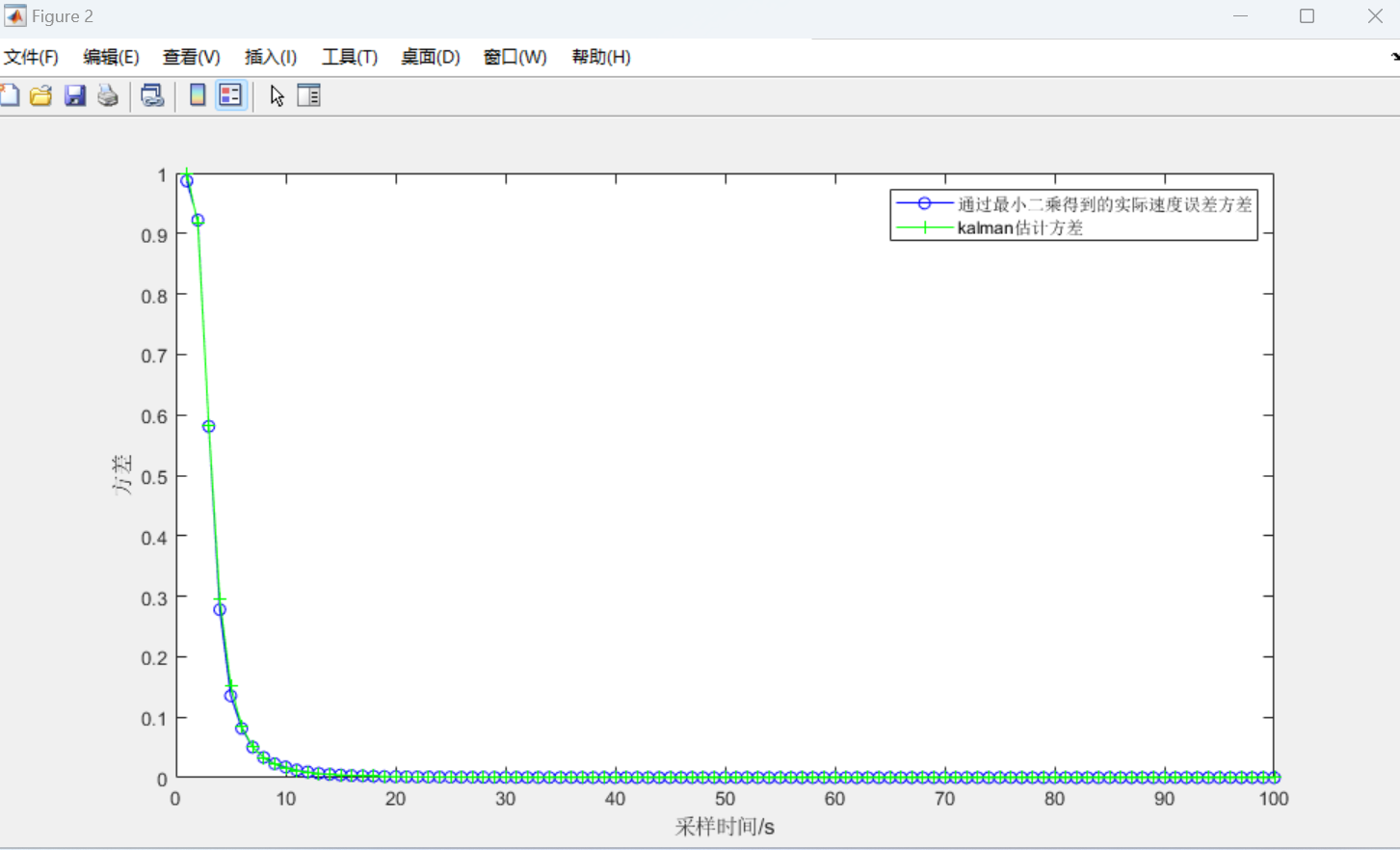
plot(errPx2,'-g+');

legend('通过最小二乘得到的实际速度误差方差','kalman估计方差');

xlabel('采样时间/s');

ylabel('方差');

## 结果1 当前时刻先验估计



## 代码2前一时刻的最优估计

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%function main3\_3\_2 最小二乘 用当前时刻的先验估计

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N=100; %仿真时间，时间序列总数

vector=zeros(1000,100); % 位移的误差

vector1=zeros(1000,100); % 速度的误差

zero1=[0 ;0];

zero2=[0 0];

for kk=1:1000

% 定义一个行向量存储每次实验每个时刻的方差

% 噪声

Q=[0,0;0,0]; % 过程噪声方差为0，即下落过程忽略空气阻力

R=1; % 观测噪声方差

W=sqrt(Q)\*randn(2,N);% 既然Q为0，则W=0；在此写出，方便对照理解

V=sqrt(R)\*randn(1,N);% 测量噪声V(k)

% 系统矩阵

A=[1,1;0,1]; %状态转移矩阵

B=[0.5;1]; %控制量

U=-1;

H=[1,0]; %观测矩阵

% 初始化

X=zeros(2,N); % 物体真实状态

X(:,1)=[95;1]; % 初始位移和速度

P0=[10,0;0,1]; % 初始误差

Z=zeros(1,N);

Z(1)=H\*X(:,1); % 初始观测值

Xls=zeros(2,N);% Kalman估计状态初始化

Xls(:,1)=X(:,1)+sqrt(P0)\*randn(2,1); % 在初始化的时候加入误差方差估计值 从而保持吻合

err\_P=zeros(N,2);

err\_P(1,1)=P0(1,1);

err\_P(1,2)=P0(2,2);

I=eye(2); % 二维系统

% 第一列

vector(kk,1)=Xls(1,1)-X(1,1);

vector1(kk,1)=Xls(2,1)-X(2,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for k=2:N

%物体下落，受状态方程的驱动

X(:,k)=A\*X(:,k-1)+B\*U+W(k);

% 位移传感器对目标进行观测

Z(k)=H\*X(:,k)+V(k);

% Kalman滤波

X\_pre=A\*Xls(:,k-1)+B\*U; %状态预测

P\_pre=A\*P0\*A'+Q; %协方差预测

Rt=[P\_pre zeros(2,1);zeros(1,2) R];

Sk=Rt^(0.5);

Xmat=[eye(2);H];

Xlinear=Sk^(-1)\*Xmat;

Ymat=[A\*Xls(:,k-1);Z(k)];

Ylinear=Sk^(-1)\*Ymat;

Kg=P\_pre\*H'\*(H\*P\_pre\*H'+R)^(-1); %计算Kalman增益

%Xkf(:,k)=X\_pre+Kg\*(Z(k)-H\*X\_pre); % 状态更

Xls(:,k)=(Xlinear'\*Xlinear)^(-1)\*Xlinear'\*Ylinear; % 新的线性回归的估计值

P0=(I-Kg\*H)\*P\_pre\*(I-Kg\*H)'+Kg\*R\*Kg';%方差更新

%vector(kk,k)=Xkf(1,k)-X(1,k); % 第kk次实验的第k个时刻 位移的误差

%vector1(kk,k)=Xkf(2,k)-X(2,k); % 第kk次实验的第k个时刻 速度的误差

vector(kk,k)=Xls(1,k)-X(1,k);

vector1(kk,k)=Xls(2,k)-X(2,k);

% 误差均方值

err\_P(k,1)=P0(1,1);

err\_P(k,2)=P0(2,2);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

% 计算vector矩阵中 每一列的方差 存储在行向量中

Difference=var(vector,0,1);

Difference1=var(vector1,0,1);

errPx=transpose(err\_P);

errPx1=errPx(1,:);

errPx2=errPx(2,:);

% 将err\_P(k,1)和Difference()画在同一张图中，比较卡尔曼滤波估计出来的Pk与实际的方差相差多少.

figure

plot(Difference,'-bo'); % 实际的位移误差方差

hold on;

plot(errPx1,'-g+');

legend('通过最小二乘得到的实际位移误差方差','kalman估计方差');

xlabel('采样时间/s');

ylabel('方差');

figure

plot(Difference1,'-bo'); %实际的速度误差方差

hold on;

plot(errPx2,'-g+');

legend('通过最小二乘得到的实际速度误差方差','kalman估计方差');

xlabel('采样时间/s');

ylabel('方差');

## 结果2

