the simpliest example of Kalman filter implementation

Bilgin Esme

July 31, 2015

1 Rudolf Emil Kalman

Rudolf Kalman was born in Budapest, Hungary, and obtained his bachelor's degree in 1953 and master's degree in 1954 from MIT in electrical engineering. his doctorate in 1957 was from Columbia University

Kalman is an electrical engineer by training, and is famous for his co-invention of the Kalman filter, a mathematical technique widely used in control systems and avionics to extract a signal from a series of incomplete and noisy measurements

Kalman's ideas on filtering were initially met with skepticism, so much so that he was forced to first publish his results in a mechanical (rather than electrical) engineering journal

he had more success in presenting his ideas, however, while visiting Stanley F. Schmidt at the NASA Ames Research Center in 1960. this led to the use of Kalman filters during the Apollo program

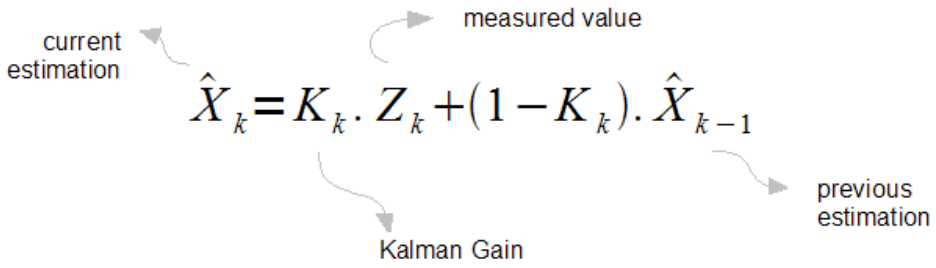
2 intro

it's nearly impossible to grasp the meaning of **Kalman Filter** by starting from definitions and complicated equations

first of all, it's not a filter at all, it's an ESTIMATOR. **it's a very, very important thing**, it's not an overemphasize - believe me!

it's a recursive method, which means, for each instance, you use the previous output as an input

for most cases, the state matrices drop out and we obtain the below equation, which is much easier to start with



the k's on the subscript are states. here we can treat it as discrete time intervals, such as k=1 means 1ms, k=2 means 2ms

the only unknown component in this equation is the **Kalman Gain** K, because we have the measurement values, and we already have the previous estimated signal

you should calculate this **Kalman Gain** for each consequent state. this is not easy of course

on the other hand, let's assume **Kalman gain** to be 0.5, what do we get? it's a simple averaging!

in other words, we should find smarter **Kalman gain** coefficients at each state

Kalman filter finds the most optimum averaging factor for each consequent state

also somehow remembers a little bit about the past states

isn't this amazing? here's a simple step-by-step guide for a quick start to Kalman filtering

3 example

let's try to estimate a scalar random constant, such as a "voltage reading" from a source

so let's assume that it has a constant value of aV (volts), but of of course we have some noisy readings

and we assume that the standard deviation of the measurement noise is 0.1V

and finally, let's assume that we have the following measurement values:

ms	1	2	3	4	5	6	7	8	9	10
V	0.39	0.50	0.48	0.29	0.25	0.32	0.34	0.48	0.41	0.45

4 model

let's build our model:

$$x_t = A * x_{t-1} + B * u_t + w_t$$

$$z_t = H * x_t + v_t$$

where

x - signal vector
A - system dynamic matrix
B - system control matrix
u - control vector
w - signal white noise vector

z - measurement vector
H - measurement behaviour matrix
v - noise of the measurement vector

1) but, fortunatly, we have a 1-dimensional signal problem, so every entity in our model is a numerical value, not a matrix

2) as the signal is a constant value, the constant A is just 1, because we already know that the next value will be same as the previous one. we are lucky that we have a constant value in this example

3) we have no such thing as control signal u_t, and it's out of the game

4) the white noise of oure voltage source is zero

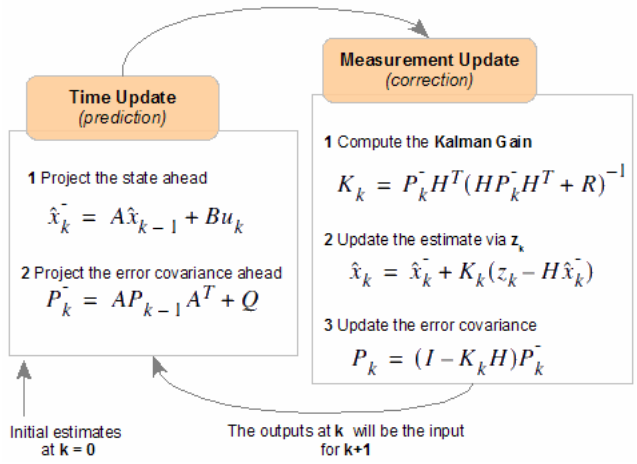
5) the value $H = 1$, because we know that the measurement is composed of the state value and some noise (you'll rarely encounter real life cases that H is different from 1)

so, in our case these two equations transform into:

$$x_t = x_{t-1}$$

$$z_t = x_t + v_t$$

we have two distinct set of equations:
"Time Update Set" – **prediction set** and "Measurement Update Set" – **correction set**



both equation sets are applied at each k^{th} state

Q - process noise covariation (in our case: 0.0)

R - measurement noise covariation (in our case: 0.1)

well, let's write the **TimeUpdate** and the **MeasurementUpdate** equations:

PREDICTION

$$\hat{x}_t^- = \hat{x}_{t-1}$$

$$p_t^- = p_{t-1}$$

where p^- - predictor

CORRECRTION

$$k_t = p_t^- / (p_t^- + r)$$

$$\hat{x}_t = \hat{x}_t^- + k_t * (z_t - \hat{x}_t^-)$$

$$p_t = (1 - k_t) * p_t^-$$

where r - regression error

INITIAL VALUES

OK, we should start from somewhere. we should find (or assume) some initial state. let's assume:

$$x_0 = 0$$

$$p_0 = 1$$

why didn't we choose predictor $p_0 = 0$ for example?

it's simple. if we chose that way, this would mean that there's no noise in the environment, and this assumption would lead all the consequent to be zero (remaining as the initial state). so we choose p_0 something other than zero

5 calculations

now, let's calculate the $\hat{x}_t$ values for each iteration

t	1	2	3	4	5	6	7	8	9	10
x_{t-1}	0	0.355	0.424	0.442	0.405	0.375	0.365	0.362	0.377	0.380
p_t^-	1	0.091	0.048	0.032	0.024	0.020	0.016	0.014	0.012	0.011
p_t	0.091	0.048	0.032	0.024	0.020	0.016	0.014	0.012	0.011	0.010
z_t	0.390	0.500	0.480	0.290	0.250	0.320	0.340	0.480	0.410	0.450
$\hat{x}_t$	0.355	0.424	0.442	0.405	0.375	0.365	0.362	0.377	0.380	0.387

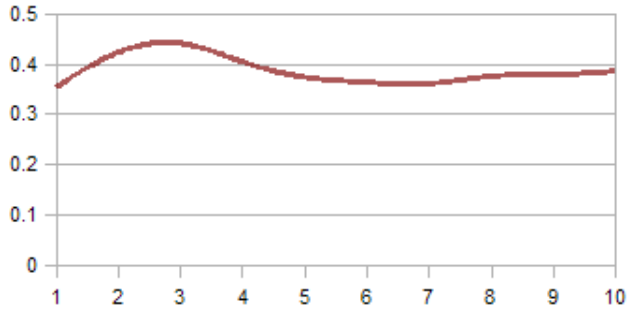
if you try to write calculation as an algorithm, you'll discover that **Kalman Filter** is very easy to implement

6 example of calculating in R

> z = rnorm(130, .4, .1)
> require(dlm)
Loading required package: dlm
> x <- dlmFilter (z, dlmModPoly(1, dV=0.1, dW=0.0))
> plot (x$f,type="l")

7 plotting

the chart here shows that the **Kalman Filter** algorithm converges to the true voltage value



here, only the first 10 iterations are displayed and we clearly see the signs of convergence

in 50 or so iterations, it'll converge even better

8 conclusion

to enable the convergence in fewer steps, you should
– model the system more elegantly
– estimate the noise more precisely

9 silly questions

can I deploy Kalman Filter to all DSP problems?

I've seen lots of papers that use Kalman Filter for a variety of problems, such as noise filtering, sub-space signal analysis, feature extraction and so on. the bottom line is, you can use Kalman Filter with a quite approximation and clever modeling

can I use it for Image Processing?

of course

where do we find these Time Update and Measurement Update equations? it seems that they suddenly appeared from nowhere?

you can derive it from the linear stochastic difference equations, by taking the partial derivative and setting them to zero (for minimizing the estimation error). of course they're hard and time consuming

10

OK. we're done