Kalman Filters let us determine a hidden state based on noisy observations. Hidden state can be location of a robot, value of a stock, and the like. KF makes the assumption that hidden variable changes in x_t and observations y_t change through linear functions, along with Gaussian noise, which means x_t and y_t is distributed as Gaussians as well. Equations look like:

$x_{t+1} = Ax_t + v$

$y_t = Hx_t + w$

where v and w are distributed as Gaussians with covariances Q and R. After lengthy derivations, one gets a tractable formula which lets us compute and estimate x_t as we get new y_t's. So generatively (say for simulations, if we wanted to generate test data from our formulas, with given A, H, Q, R), we could go from $x_t -> x_{t+1} -> y_{t+1}$ but KF formulas gives us the reverse of this. So we can discover the hidden state, and update our estimate of this hidden world with each given y_t.

Since we update estimate of x_t as we get new y_t, this algorithm can be used online. Robotics applications for example could represent a robot's location as x_t where each new measurement from robot's sensors could be used to update the hidden belief, help determine the location of the robot.

Line Fitting

If we had all data that is needed for a line, we could simply use least squares to fit a straight line.

KF let us to fit a line continuously, as we get new data. Line fitting success of KF, once it has received every observation data point (that is, after having processed all previous data as well) is equal to that of least squares with all given data in one shot.

How do we use Kalman Filters for line fitting? We need to little magic with our representation. We ask ourselves: What is the state of this system? Since we would like to determine the "slope" of a line, we can include slope as a parameter inside x_t. We also include the y-axis values of the line as part of our observation. Note: To distinquish between y's and x's of KF, we will call the values off the axis as xx_t and yy_t. Then the x_t "vector" looks like

$$x_t = \begin{bmatrix} yy_t \\ a \end{bmatrix}$$

where letter 'a' represents slope. Since a is a constant, KF will be forced to estimate a hidden state that never changes (usually KF examples use a changing x_t, this has to do with the domain of the application, rather than a mathematical necessity).

We also need a little magic with our A matrix. This is the transformation matrix that takes x_t and gives us an x_{t+1}. A_t would be equal to

$$A_t = \begin{bmatrix} 1 & \Delta xx \\ 0 & 1 \end{bmatrix}$$

If you do the multiplication with x_t, you will see that x_{t+1} will indeed contain the next yy_{t+1} because (upper cell) is $yy_t * 1 + a * \Delta_{xx}$. This is the next point on the line.

Our H matrix looks like [1 0, 1 0] which once multiplied with x_t will give us the observation values containing yy_t (twice, see the note why).

Note: In order for some matrix operations to work for all dimensions, we had to make y_t dimension equal to x_t (which is 2) and as a result H is taken as above and this results yy_t value to be repeated in both cells of y_t. We could not find a way to map 1x2 x_t into 1x1 y_t which only carries yy observations and not break our matrix code. If you can figure this out, let me know. Repeating yy value twice in obs[] vector is a minor inconvenience however, not a big issue.