Filters and Methode Estimate

Yin Chheanyun

July 2024

Contents

1 Kalman Filter(KF) 3
1.1 What is Kalman Filter? 3
1.2 Kalman Filter Algorithm 3
1.3 Python Implementation of Kalman Filter Algorithm 4

2 Extended Kalman Filter(EKF) 5
2.1 What is Externded Kalman Filter? 5
2.2 Extended Kalman Filter Algorithm 5
2.3 Python Implementation of Extended Kalman Filter(EKF) Algorithm 7

3 Unscented Kalman Filter (UKF) 8
3.1 What is Unscented Kalman Filter? 8
3.2 Unscented kalman filter Algorithm 8
3.3 Python Implementation of Unscented Kalman Filter(UKF) Algorithm 10

4 Particle Filter 11
4.1 What is Particle Filter? 11
4.2 Application of Particle 11
4.3 Particle Algorithm 11
4.4 Python Implementation of Particle Filter Algorithm 16

5 Complementary Filter 17
5.1 What is Complementary Filter? 17
5.2 Complementary Filter Algorithm 17
5.3 Python Implementation of Complementary Filter Algorithm 18

6 Least Squares Estimation (LSE) 19
6.1 What is Least Squares Estimation? 19
6.2 LSE Algorithm 19
6.3 Python Implementation of Least Squares Estimation(LSE) Algorithm 22

7 Least Mean Squares (LMS) 23
7.1 What is Least Mean Squares? 23
7.2 Least Mean Squares Algorithm 23
7.3 Python Implementation of Least Mean Squares(LMS) 24

8 Maximum Likelihood Estimation (MLE) 25
8.1 What is Maximum Likelihood Estimation? 25
8.2 Review Probability 25
8.3 MLE Algorithm 27
8.4 Python Implementation of Maximum Likelihood Estimation(MLE) 29

List of Figures

1 Kalman Filter Prediction 3
2 Flow to use Kalman Filter 3
3 Flow to use Extended Kalman Filter 5
4 Process of UKF Algorithm 8
5 Step-by-step of the Particle Filter in the context of position estimation for mobile robots in indoor environments. 11
6 Visualizations of the process and measurement models used in the running 11
7 Complementary Filter Block diagram 17
8 Least square estimate 19
9 LSE Prediction error 20
10 Typical Adaptive Filter 23
11 Graph of Maximum Likelihood estimate 25

List of Tables

1 Kalman Filter(KF)

1.1 What is Kalman Filter?

Kalman Filter, also know as linear quadratic estimation(LQE) is the algorithm that uses series of measurements that are observed over time and that contains statistical noise and other inaccuracies that are found in the given system.

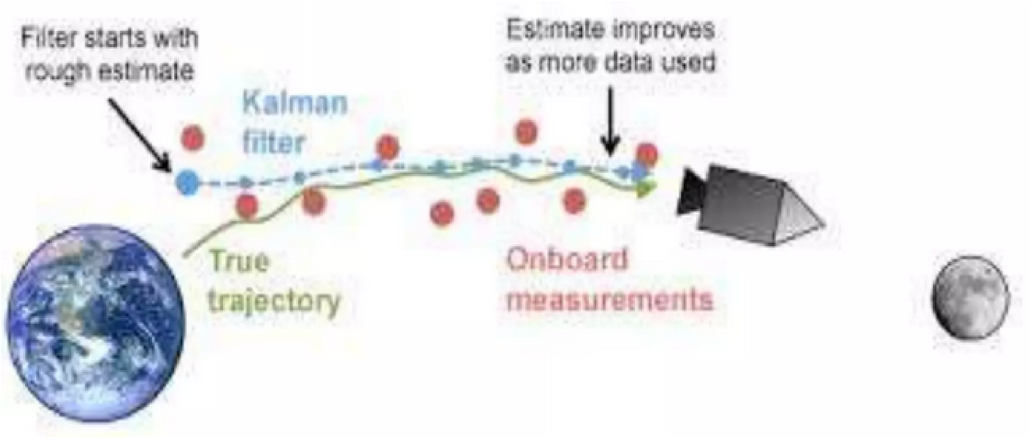
Filter starts with rough estimate
Kalman filter
Estimate improves as more data used
True trajectory
Onboard measurements

Figure 1: Kalman Filter Prediction

1.2 Kalman Filter Algorithm

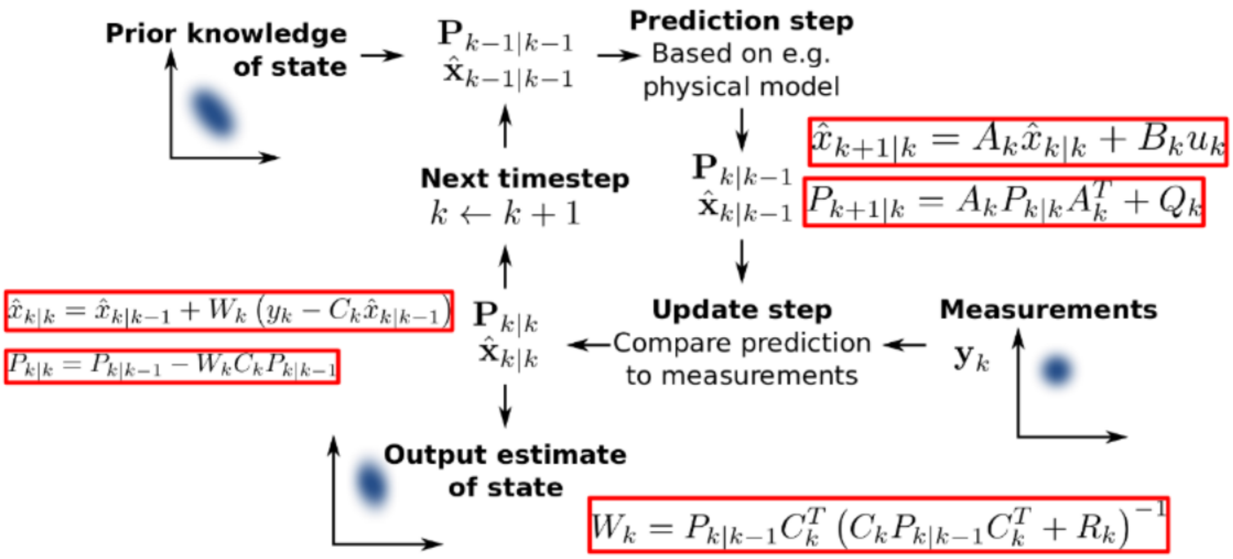
Prior knowledge of state
$\mathbf{P}_{k-1|k-1}$
$\hat{\mathbf{x}}_{k-1|k-1}$
Prediction step
Based on e.g. physical model
$\hat{x}_{k+1|k} = A_k\hat{x}_{k|k} + B_k u_k$
$P_{k+1|k} = A_k P_{k|k} A_k^T + Q_k$
$\mathbf{P}_{k|k-1}$
$\hat{\mathbf{x}}_{k|k-1}$
Next timestep
$k \leftarrow k+1$
$\hat{x}_{k|k} = \hat{x}_{k|k-1} + W_k\left(y_k - C_k\hat{x}_{k|k-1}\right)$
$P_{k|k} = P_{k|k-1} - W_k C_k P_{k|k-1}$
$\mathbf{P}_{k|k}$
$\hat{\mathbf{x}}_{k|k}$
Update step
Compare prediction to measurements
Measurements
$\mathbf{y}_k$
Output estimate of state
$W_k = P_{k|k-1}C_k^T\left(C_k P_{k|k-1} C_k^T + R_k\right)^{-1}$

Figure 2: Flow to use Kalman Filter

Given : $\hat{x}_{0|-1}, P_{0|-1}$

- **Process Model:**

$$\mathbf{x_k} = \mathbf{A x_{k-1}} + \mathbf{B u_k} + \mathbf{w_k}$$

  - A : State transition matrix.
  - B : Control matrix
  - u : Control input.
  - w : Process noise

- **Measurement Model:**

$$\mathbf{y_k = Cx_k + v_k}$$

  - C : Observation matrix.
  - v : Measurement noise.

- **Compute Kalman gain:**

$$\mathbf{K = P_{k|k-1}C_k^T(C_kP_{k|k-1}C_k^T + R_k)^{-1}}$$

- **Update (modify) predicted value by observation**

$$\mathbf{\hat{x}_{k|k} = \hat{x}_{k|k-1} + K(y_k - C_k\hat{x}_{k|k-1})}$$

- **Compute prediction at the next time step**

$$\mathbf{P_{k|k} = P_{k|k-1} - KC_kP_{k|k-1}}$$
$$\mathbf{P_{k+1|k} = A_kP_{k|K}A_k^T + Q_k}$$

1.3 Python Implementation of Kalman Filter Algorithm

2 Extended Kalman Filter(EKF)

2.1 What is Externded Kalman Filter?

The Extended Kalman Filter is an algorithm that leverages our knowledge of the physics of motion of the system (i.e. the state space model) to make small adjustments to (i.e. to filter) the actual sensor measurements (i.e. what the robot's sensors actually observed) to reduce the amount of noise, and as a result, generate a better estimate of the state of a system.

- The **state estimate** for the previous timestep t-1.
- The **time interval** dt from one timestep to the next.
- The **linear and angular velocity** of the car at the previous time step t-1 (i.e. previous control inputs. . . i.e. commands that were sent to the robot to make the wheels rotate accordingly).
- An **estimate of random noise** (typically small values. . . when we send velocity commands to the car, it doesn't move exactly at those velocities so we add in some random noise).

2.2 Extended Kalman Filter Algorithm

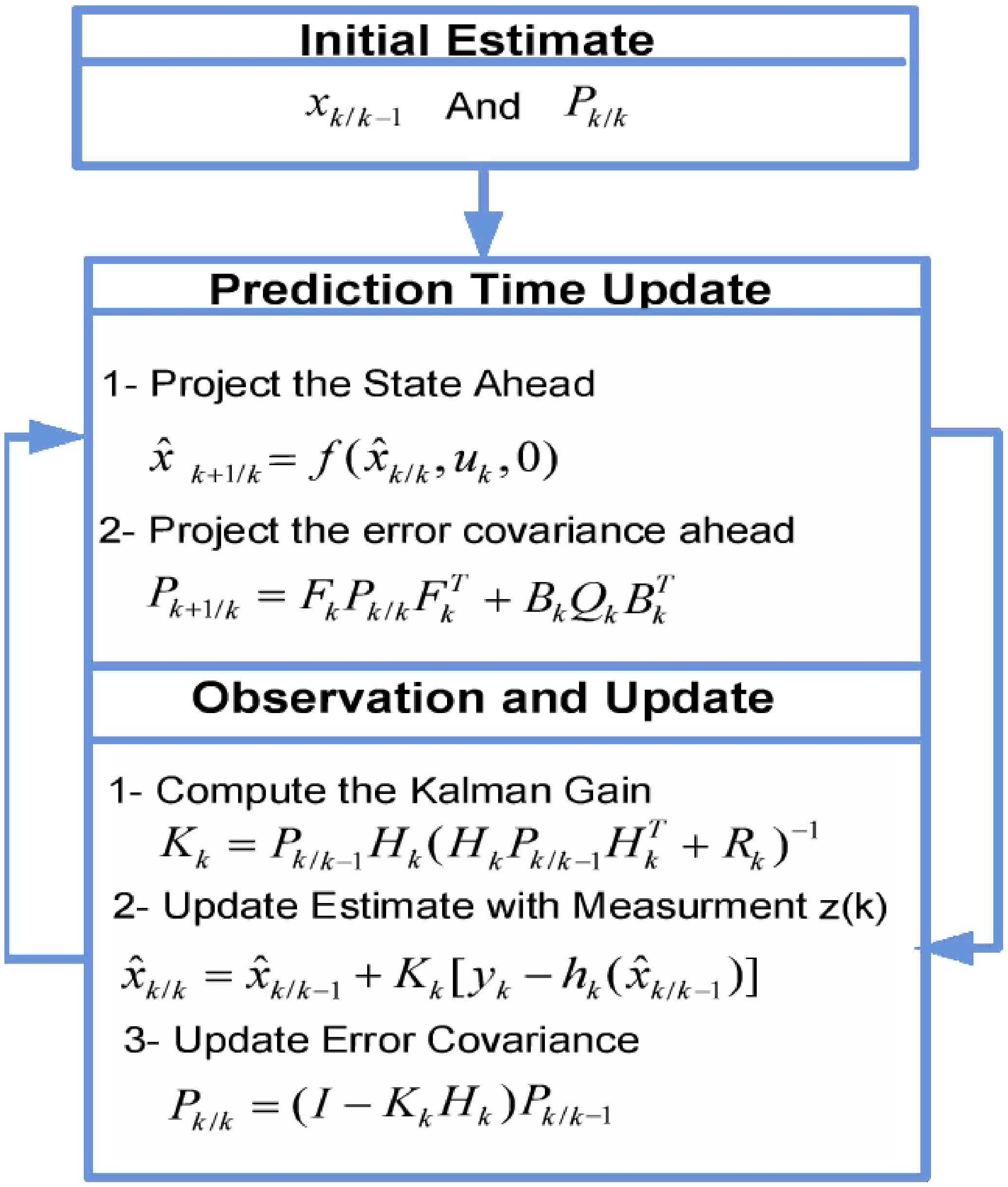
Initial Estimate
$x_{k/k-1}$ And $P_{k/k}$
Prediction Time Update
1- Project the State Ahead
$\hat{x}_{k+1/k} = f(\hat{x}_{k/k}, u_k, 0)$
2- Project the error covariance ahead
$P_{k+1/k} = F_k P_{k/k} F_k^T + B_k Q_k B_k^T$
Observation and Update
1- Compute the Kalman Gain
$K_k = P_{k/k-1} H_k (H_k P_{k/k-1} H_k^T + R_k)^{-1}$
2- Update Estimate with Measurment z(k)
$\hat{x}_{k/k} = \hat{x}_{k/k-1} + K_k [y_k - h_k(\hat{x}_{k/k-1})]$
3- Update Error Covariance
$P_{k/k} = (I - K_k H_k) P_{k/k-1}$

Figure 3: Flow to use Extended Kalman Filter

- **Initialization**:
  - For the first iteration of EKF, we start at time k. In other words, for the first run of EKF, we assume the current time is k.
  - We initialize the state vector and control vector for the previous time step k-1.
- **Predicted State Estimate**:

$$\mathbf{\hat{X}_{k|k-1} = f(\hat{X}_{k-1|k-1}, u_k) = A_{k-1}X_{k-1|k-1} + B_{k-1}U_{k-1} + V_{k-1}}$$

  - We use the state space model, the state estimate for timestep k-1, and the control input vector at the previous time step (e.g. at time k-1) to predict what the state would be for time k (which is the current timestep).
  - That equation above is the same thing as our equation below. Remember that we used t in my earlier tutorials.
- **Predicted Covariance of the State Estimate**

$$\mathbf{P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k}$$

  - $P_{k-1|k-1}$ is a square matrix. It has the same number of rows (and columns) as the number of states in the state vector x.
  - P (or, commonly, sigma) is a 3×3 matrix. The P matrix has variances on the diagonal and covariances on the off-diagonal.
  - we initialize $P_{k-1|k-1}$ to some guessed values (eg 0.1 along the diagonal part of the matrix and 0s elsewhere).

$$\boldsymbol{P_{k-1|k-1}} = \begin{bmatrix} \mathbf{0.1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0.1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0.1} \end{bmatrix}$$

  - F_k = are equivalent to A_{k-1} respectively, from my state space model tutorial.
  - Q_k = is the state model noise covariance matrix. It is also a 3×3 matrix in our running robot car example because there are three states.

$$\boldsymbol{Q_k} = \begin{bmatrix} \boldsymbol{cov(x,x)} & \boldsymbol{cov(x,y)} & \boldsymbol{cov(x,\gamma)} \\ \boldsymbol{cov(y,x)} & \boldsymbol{cov(y,y)} & \boldsymbol{cov(y,\gamma)} \\ \boldsymbol{cov(\gamma,x)} & \boldsymbol{cov(\gamma,y)} & \boldsymbol{cov(\gamma,\gamma)} \end{bmatrix}$$

  - The covariance between two variables that are the same is actually the variance. For example, Cov(x, x) = Var(x).
  - We can start by letting Q be the identity matrix and tweak the values through trial and error.

$$\boldsymbol{Q_k} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

  - when Q is large, it means we trust our actual sensor observations more than we trust our predicted sensor measurements from the observation model. . . more on this later in this tutorial.
- **Innovation or Measurement Residual**

$$\mathbf{\tilde{y_k} = z_k - h(\hat{X}_{k|k-1})}$$

  - we calculate the difference between actual sensor observations and predicted sensor observations.
  - z_k is the observation vector. It is a vector of the actual readings from our sensors at time k.Matrix in z k in mobile robot:

$$\boldsymbol{z_k} = \begin{bmatrix} \boldsymbol{x_k} \\ \boldsymbol{y_k} \\ \boldsymbol{\gamma_k} \end{bmatrix}$$

  - $h(\hat{X}_{k|k-1})$ is our observation model. It represents the predicted sensor measurements at time k given the predicted state estimate at time k from Step 2. That hat symbol above x means "predicted" or "estimated".

$$\boldsymbol{h(\hat{X}_{k|k-1}) = H_k \hat{X}_{k|k-1} + W_k}$$

- Measurement matrix H k (which is used to convert the predicted state estimate at time k into predicted sensor measurements at time k)

$$\boldsymbol{H_k} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

- **Innovation (or residual) Covariance**

$$\mathbf{S_k = H_k P_{k|k-1} H_k^T + R_k}$$

  - use the predicted covariance of the state estimate $P_{k|k-1}$
  - The measurement matrix H_k and its transpose.
  - R_k (sensor measurement noise covariance matrix. . . which is a covariance matrix that has the same number of rows as sensor measurements and same number of columns as sensor measurements)
  - To start out by making R_k the identity through trial and error

$$R_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- **Near-optimal Kalman Gain**

$$\mathbf{K_k = P_{k|k-1} H_k^T S_k^{-1}}$$

  - If sensor measurement noise is large, then K approaches 0, and sensor measurements will be mostly ignored.
  - If prediction noise (using the dynamical model/physics of the system) is large, then K approaches 1, and sensor measurements will dominate the estimate of the state [x,y,yaw angle].

- **Updated State Estimate**

$$\mathbf{\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k \tilde{y}_k}$$

  - we calculate an updated (corrected) state estimate

- **Updated Covariance of the State Estimate**

$$\mathbf{P_{k|k} = (I - K_k H_k) P_{k|k-1}}$$

2.3 Python Implementation of Extended Kalman Filter(EKF) Algorithm

3 Unscented Kalman Filter (UKF)

3.1 What is Unscented Kalman Filter?

The unscented Kalman filter algorithm and Unscented Kalman Filter block use the unscented transformation to capture the propagation of the statistical properties of state estimates through nonlinear functions. The algorithm first generates a set of state values called sigma points.

3.2 Unscented kalman filter Algorithm

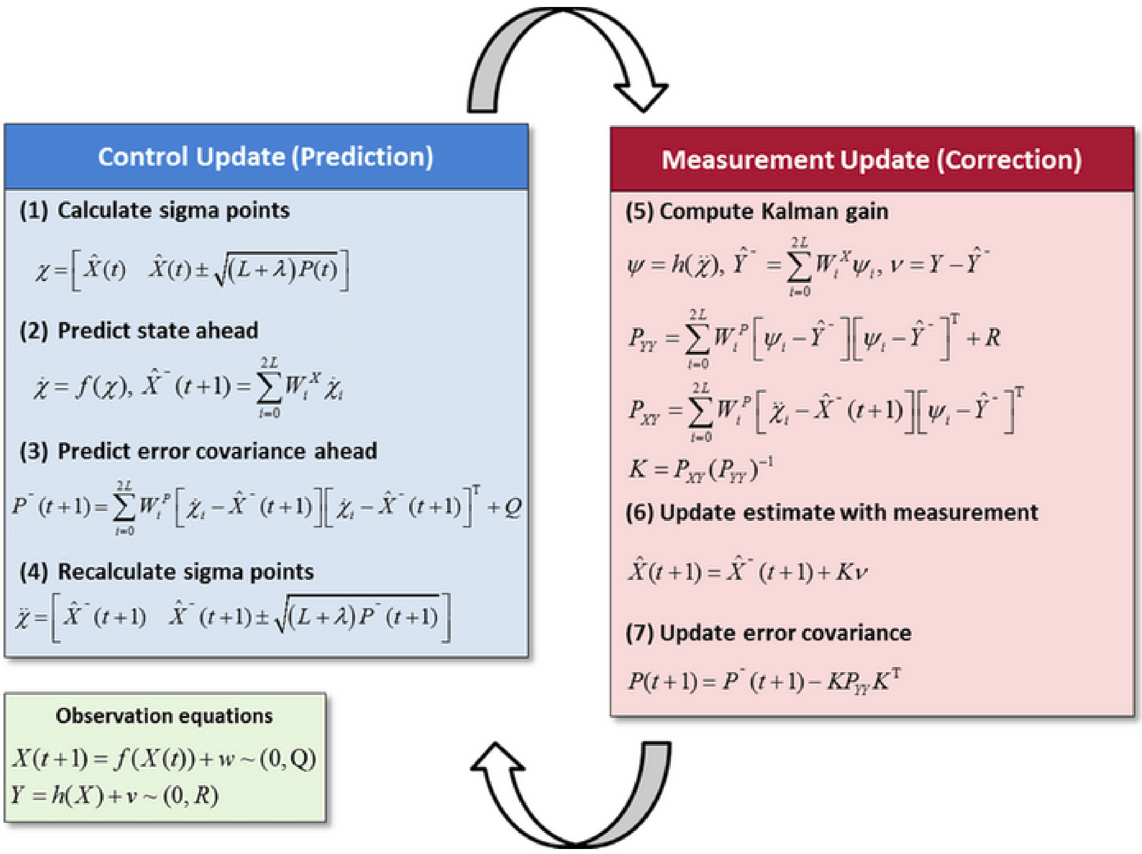
Control Update (Prediction)
(1) Calculate sigma points
$\chi = \left[\hat{X}(t) \quad \hat{X}(t) \pm \sqrt{(L+\lambda)P(t)} \right]$
(2) Predict state ahead
$\breve{\chi} = f(\chi),\ \hat{X}^{-}(t+1) = \sum_{i=0}^{2L} W_i^X \breve{\chi}_i$
(3) Predict error covariance ahead
$P^{-}(t+1) = \sum_{i=0}^{2L} W_i^P \left[\breve{\chi}_i - \hat{X}^{-}(t+1) \right] \left[\breve{\chi}_i - \hat{X}^{-}(t+1) \right]^T + Q$
(4) Recalculate sigma points
$\breve{\chi} = \left[\hat{X}^{-}(t+1) \quad \hat{X}^{-}(t+1) \pm \sqrt{(L+\lambda)P^{-}(t+1)} \right]$
Observation equations
$X(t+1) = f(X(t)) + w \sim (0, Q)$
$Y = h(X) + v \sim (0, R)$
Measurement Update (Correction)
(5) Compute Kalman gain
$\psi = h(\breve{\chi}),\ \hat{Y}^{-} = \sum_{i=0}^{2L} W_i^X \psi_i,\ v = Y - \hat{Y}^{-}$
$P_{YY} = \sum_{i=0}^{2L} W_i^P \left[\psi_i - \hat{Y}^{-} \right] \left[\psi_i - \hat{Y}^{-} \right]^T + R$
$P_{XY} = \sum_{i=0}^{2L} W_i^P \left[\breve{\chi}_i - \hat{X}^{-}(t+1) \right] \left[\psi_i - \hat{Y}^{-} \right]^T$
$K = P_{XY} (P_{YY})^{-1}$
(6) Update estimate with measurement
$\hat{X}(t+1) = \hat{X}^{-}(t+1) + Kv$
(7) Update error covariance
$P(t+1) = P^{-}(t+1) - K P_{YY} K^T$

Figure 4: Process of UKF Algorithm

- **Unscented Transform Steps:**

$$\boldsymbol{x_{n+1} = f_n(x_n) + u_n}$$
$$\mathbf{y_n = h_n(x_n) + v_n}$$

  - Unscented transform is based on following ideas
    - First, it is easy to perform nonlinear transform on a point(Difficult to perform transform on a PDF)
    - Second, We can find a set of points in state space that adequately capture the first and second order moments (mean and covariance) of a distribution.
  - Suppose we know the mean $\bar{\mathbf{x}}$ and covariance $\mathbf{P}$ of the state $\mathbf{x}$
  - We can find a set of points with a sample mean and covariance equal to $\bar{x}$ and covariance P
    - These points are very similar to particles use in particle filters
    - The points are called **sigma points**
  - We then apply our nonlinear function $\mathbf{y} = \mathbf{h}(\mathbf{x})$ to each of these points
  - The sample mean and covariance is a good estimate of the true mean and covariance of $\mathbf{y}$

- **Sigma Point Properties**
Sigma points are only random in the sense that they depend on the current estimate of the state and state error covariance

$$\boldsymbol{x_n^i = \hat{x}_n + \tilde{x}^i fori = 1, 2, \ldots, 2n_x}$$

$$\tilde{x}_n^i = [row_i(\sqrt{n_x P_n})]^T fori = 1, \ldots, n_x$$

$$\tilde{x}_n^i = -[row_i(\sqrt{n_x P_n})]^T fori = n_x + 1, \ldots, 2n_x$$

Where $row_i(A)$ denotes the i^{th} row vector of the matrix A and $\sqrt{n_x P_n}$ is a matrix square root of $(n_x P_n)$ such that

$$\sqrt{\mathbf{n_x P_n}}^{\mathbf{T}} \sqrt{\mathbf{n_x P_n}} = \mathbf{n_x P_n}$$

The sigma points are chosen carfully sunch that

$$\hat{x}_n = \frac{1}{2n_x} \sum_{i=1}^{2n_x} x_n^i$$

$$P_n = \frac{1}{2n_x} \sum_{i=1}^{2n_x} (x_n^i - \hat{x}_n)(x_n^i - \hat{x}_n)^T$$

Note: That only the mean and covariance (first and second order moments) are preserved

- **Estimating Mean of a Nonlinear Function**

$$\mathbf{y} = \mathbf{h}(\mathbf{x})$$

We use sigma points to estimate the mean and covariance as follows

$$\mathbf{y^i} = \mathbf{h}(\mathbf{x^i})$$

$$\hat{y}_u = \frac{1}{2n_x} \sum_{i=1}^{2n_x} y^i$$

$$R_{\tilde{y},u} = \frac{1}{2n_x} \sum_{i=1}^{2n_x} (y^i - \hat{y}_u)(y^i - \hat{y}_u)^T$$

- **Time Update (Prediction Step)**

$$\hat{\mathbf{x}}_{\mathbf{0|0}} = \mathbf{E}[\mathbf{x_0}]$$

$$P_{\tilde{x},0|0} = ||x_0 - \hat{x}_{0|0}||^2$$

For $n = 1, \ldots$

$$x_{n-1}^i = \hat{x}_{n-1|n-1} + \tilde{x}^i fori = 1, \ldots, 2n_x$$

$$\tilde{x}_n^i = [row_i(\sqrt{n_x P_{\tilde{x},n-1|n-1}})]^T fori = 1, \ldots, n_x$$

$$\tilde{x}_n^i = -[row_i(\sqrt{n_x P_{\tilde{x},n-1|n-1}})]^T fori = n_x + 1, \ldots, 2n_x$$

$$\hat{x}_n^i = f_n(\hat{x}_{n-1}^i)$$

$$\hat{x}_{n|n-1} = \frac{1}{2n_x} \sum_{i=1}^{2n_x} \hat{x}_n^i$$

$$P_{\tilde{x},n|n-1} = \frac{1}{2n_x} \sum_{i=1}^{2n_x} (\hat{x}_n^i - \hat{x}_{n|n-1})(\hat{x}_n^i - \hat{x}_{n|n-1})^T$$

- **Sigma Point Update**

$$x_n^i = \hat{x}_n + \tilde{x}^i fori = 1, 2, \ldots, 2n_x$$

$$\tilde{x}_n^i = [row_i(\sqrt{n_x P_{\tilde{x},n|n-1}})]^T fori = 1, \ldots, n_x$$

$$\tilde{x}_n^i = -[row_i(\sqrt{n_x P_{\tilde{x},n|n-1}})]^T fori = n_x + 1, \ldots, 2n_x$$

- **Measurement Update (Filter Step)**

$$\boldsymbol{y_n^i = h_n(x_n^i)}$$

$$\boldsymbol{\hat{y}_{n|n-1} = \frac{1}{2n}\sum_{i=1}^{2n_x} y_n^i}$$

$$\boldsymbol{P_{\tilde{y}} = \frac{1}{2n}\sum_{i=1}^{2n_x}(y_n^i - \hat{y}_{n|n-1})(y_n^i - \hat{y}_{n|n-1})^T + R_{n-1}}$$

$$\boldsymbol{P_{\tilde{x}\tilde{y}} = \frac{1}{2n}\sum_{i=1}^{2n_x}(x_n^i - \hat{x}_{n|n-1})(y_n^i - \hat{y}_{n|n-1})^T}$$

$$\boldsymbol{K_n = P_{\tilde{x}\tilde{y}}P_{\tilde{y}^{-1}}}$$

$$\boldsymbol{\hat{x}_{n|n} = \hat{x}_{n|n-1} + K_n(y_n - n|\hat{n} - 1)}$$

$$\boldsymbol{P_{\tilde{x},n|n} = P_{\tilde{x},n|n-1} - K_n P_{\tilde{y}} K_n^T}$$

3.3 Python Implementation of Unscented Kalman Filter(UKF) Algorithm

4 Particle Filter

4.1 What is Particle Filter?

A particle filter is a computational algorithm used for estimating the state of a dynamic system from noisy measurements. It's commonly used in fields such as robotics, computer vision, and signal processing.

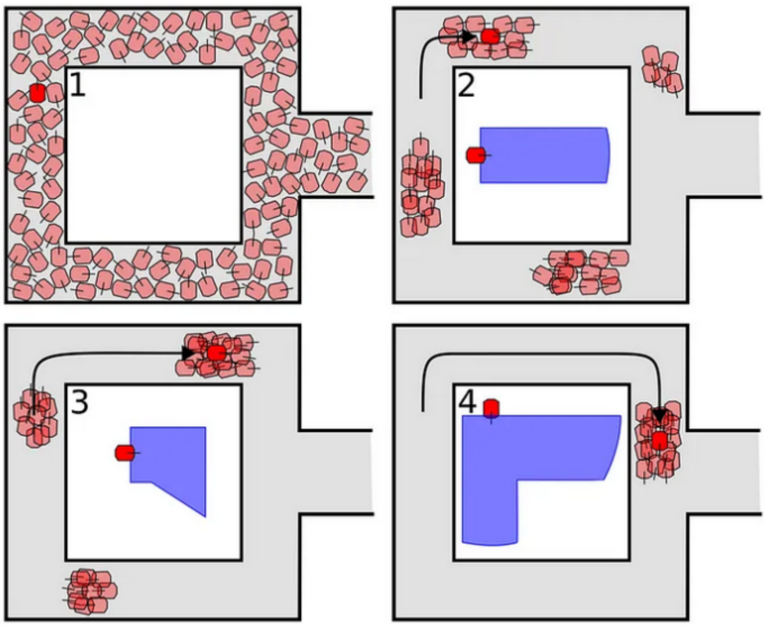

Figure 5: Step-by-step of the Particle Filter in the context of position estimation for mobile robots in indoor environments.

4.2 Application of Particle

- Estimating the position and orientation of a vehicle with respect to a map using measurements of landmarks whose positions are known from that map
- Estimating asset returns in econometrics
- Building a map of a robot's environment while navigating this environment
- Fault detection by state estimation, e.g., in a chemical process

4.3 Particle Algorithm

1. **Model Problem:**

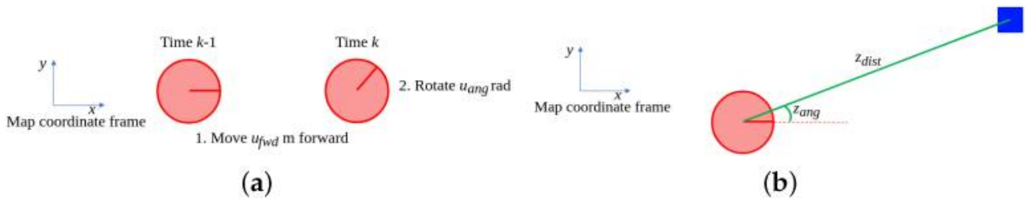

Figure 6: Visualizations of the process and measurement models used in the running

- **Process Model:**

$$\mathbf{x_k} = \mathbf{f_k}(\mathbf{x_{k-1}}, \mathbf{u_k}, \mathbf{v_{k-1}}) = \mathbf{A}\mathbf{x_k} + \mathbf{B}\mathbf{u_k} + \mathbf{v_{k-1}} \qquad (eq.1)$$

Example:

$$\begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}_k = \begin{bmatrix} x_r \\ y_r \\ \theta_r \end{bmatrix}_{k-1} + \begin{bmatrix} cos(\theta_{r,k-1} & 0 \\ sin(\theta_{r,k-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_x \\ \omega \end{bmatrix}$$

- **Measurement Model:**

$$\mathbf{z_k} = \mathbf{h_k}(\mathbf{x_k}, \mathbf{n_k}) \qquad (eq.2)$$

Example:

$$\begin{bmatrix} z_{dist} \\ z_{ang} \end{bmatrix}_k = \begin{bmatrix} \sqrt{(x_{r,k} - x_{lm})^2 + (y_{r,k} - y_{lm})^2} \\ arctan\left(\frac{y_{r,k} - y_{lm}}{x_{r,k} - x_{lm}}\right) \end{bmatrix}$$

Where:

- x_{lm}, y_{lm} = landmark with position.
- Notice that for reasons of simplicity, the robot's heading is not considered in the simulated measurement, i.e., both z_{dist} and z_{ang} are independent of the robot's heading θ_r.

2. **Review Probability**

Let $p(\mathbf{z}_1|\mathbf{z}_2)$ be an arbitrary conditional probability density function,let $\mathbf{z}_1$, and $\mathbf{z}_2$ be two vectors.

$$\mathbf{p}(\mathbf{z_1}|\mathbf{z_2}) = \frac{\mathbf{p}(\mathbf{z_1}, \mathbf{z_2})}{\mathbf{p}(\mathbf{z_2})} \qquad (eq.1)$$

or

$$\mathbf{p}(\mathbf{z_1}, \mathbf{z_2}) = \mathbf{p}(\mathbf{z_1}|\mathbf{z_2})\mathbf{p}(\mathbf{z_2}) \qquad (eq.2)$$

On the other hand, we also have

$$\mathbf{p}(\mathbf{z_1}, \mathbf{z_2}) = \mathbf{p}(\mathbf{z_2}|\mathbf{z_1})\mathbf{p}(\mathbf{z_1}) \qquad (eq.3)$$

By combining (2) and (3), we have

$$\mathbf{p}(\mathbf{z_1}|\mathbf{z_2})\mathbf{p}(\mathbf{z_2}) = \mathbf{p}(\mathbf{z_2}|\mathbf{z_1})\mathbf{p}(\mathbf{z_1})$$

From the last equation, we obtain

$$\mathbf{p}(\mathbf{z_1}|\mathbf{z_2}) = \frac{\mathbf{p}(\mathbf{z_2}|\mathbf{z_1})\mathbf{p}(\mathbf{z_1})}{\mathbf{p}(\mathbf{z_2})}$$

In the context of the Bayesian estimation, we want to infer statistics or an estimate of $\mathbf{z}_1$ from the observation $\mathbf{z}_2$. The density $p(\mathbf{z}_1|\mathbf{z}_2)$ is called the posterior and we want to compute this density. The density $p(\mathbf{z}_2|\mathbf{z}_1)$ is called the likelihood. The density $p(\mathbf{z}_1)$ is called the prior density, and the density $p(\mathbf{z}_2)$ is called the marginal density. Usually, the marginal density is not important and it serves as a normalizing factor. Consequently, the Bayes' rule can be written as

$$\boldsymbol{p(z_1|z_2) = \gamma p(z_2|z_1)p(z_1)}$$

Where: γ is the normalizing constant.
if we would have three vectors $\mathbf{z}_1$, $\mathbf{z}_2$, and $\mathbf{z}_3$, the Bayes' rule takes this form

$$\mathbf{p}(\mathbf{z_1}|\mathbf{z_2}, \mathbf{z_3}) = \frac{\mathbf{p}(\mathbf{z_2}|\mathbf{z_1}, \mathbf{z_3})\mathbf{p}(\mathbf{z_1}|\mathbf{z_3})}{\mathbf{p}(\mathbf{z_2}|\mathbf{z_3})}$$

or in the normalized form

$$\boldsymbol{p(z_1|z_2, z_3) = \gamma p(z_2|z_1, z_3)p(z_1|z_3)}$$

3. **Mathematical Problem**

- The state estimate is represented by a pdf that quantifies both the estimated state and the uncertainty associated with the estimated value. The state of interest is referred to as the posterior state: the estimate of the state at time k given all measurements and inputs up to time k and is denoted by the conditional pdf:

$$\mathbf{p}(\mathbf{x_k}|\mathbf{u_{1:k}}, \mathbf{z_{1:k}}) \qquad (eq.3)$$

Where:

- $u_{1:k} = \{u_i; i = 1, 2, \ldots, k\}$ denotes the sequence of known control inputs.
- $z_{1:k} = \{z_i; i = 1, 2, \ldots, k\}$ denotes the measurement sequence.

- The state sequence is modelled by a Markov chain. This means the past is assumed to be adequately summarized by only the state at the previous time step. Mathematically this implies:

$$\mathbf{p}(\mathbf{x_k}|\mathbf{x_{1:k-1}}, \mathbf{z_{1:k-1}}) = \mathbf{p}(\mathbf{x_k}|\mathbf{x_{k-1}}), \mathbf{p}(\mathbf{z_k}|\mathbf{x_{1:k}}) = \mathbf{p}(\mathbf{z_k}|\mathbf{x_k}) \qquad (eq.4)$$

- Bayes' theorem can be written as:

$$\mathbf{p(A|B) = \frac{p(B|A)P(A)}{p(B)}}$$

$$\mathbf{posterior = \frac{likelihood * prior}{marginal\_likelihood}} \qquad (eq.5)$$

- The posterior distribution at the previous time step, $p(x_{k-1}|z_{1:k-1})$, is combined with the process model that describes how the state evolves over time in the prediction step. The result is referred to as the prior state:

$$\mathbf{p(x_k|z_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z_{1:k-1})dx_{x-1}} \qquad (eq.6)$$

- the update step, the measurement z_k at time k is used to compute the posterior using Bayes' theorem:

$$\mathbf{p(x_k|z_{1:k}) = \frac{p(z_k|x_k)p(x_k|z_{1:k-1})}{p(z_k|z_{1:k-1})}} \qquad (eq.7)$$

- The likelihood $p(z_k|x_k)$ represents the conditional probability of a measurement given the predicted state, $p(x_k|z_{1:k-1})$ is the prior computed using (eq.6) and the normalizing constant represents the probability of the measurement. It can be computed using:

$$\boldsymbol{p(z_k|z_{1:k-1}) = \int p(z_k|x_k)p(x_k|z_{1:k-1})dx_k} \qquad (eq.8)$$

4. **Basic Idea of Particle Filter**

- **The particle filter is derived by first considering the expectation involving the smoothing density**

$$\boldsymbol{E(\beta(x_{0:k})|z_{0,k},u_{0,k-1}) = \int_{x_{0:k}\in\tilde{S}} \beta(x_{0:k})p(x_{0:k}|z_{0:k},u_{0:k-1})dx_{0:k}} \qquad (eq.1)$$

where $\boldsymbol{\beta}(\mathbf{x}_{0:k})$ is a nonlinear function of the state sequence, $E[\cdot]$ is the expectation operator, and $\tilde{S}$ is the state support (domain) of the smoothed posterior density.

- **The remedy is to use the importance sampling method.**

$$\boldsymbol{E(\beta(x_{0:k})|z_{0,k},u_{0,k-1} \approx \sum_{i=1}^{N} \tilde{w}_k^{(i)}\beta(x0:k^(i))} \qquad (eq.2)$$

where $\mathbf{x}_{0:k}^{(i)}$ is a sample of the state sequence $\mathbf{x}_{0:k}$ sampled from the smoothed proposal (importance) density $q(\mathbf{x}_{0:k}|\mathbf{y}_{0:k},\mathbf{u}_{0:k-1})$, and the weight is defined by

$$\mathbf{\tilde{w}_k^{(i)} = \frac{p(x_{0:k}^{(i)}|y_{0:k},u_{0:k-1})}{q(x_{0:k}^{(i)}|y_{0:k},u_{0:k-1})}} \qquad (eq.3)$$

the general form of the weight is

$$\mathbf{\tilde{w}_k = \frac{p(x_{0:k}|y_{0:k},u_{0:k-1})}{q(x_{0:k}|y_{0:k},u_{0:k-1})}} \qquad (eq.4)$$

- **the derivation of the recursive relation for $\mathbf{\tilde{w}_k}$. We can formally write the smoothing density as follows**

$$\mathbf{p(x_{0:k}|y_{0:k},u_{0:k-1}) = p(x_{0:k}|y_k,y_{0:k-1},u_{0:k-1})} \qquad (eq.6)$$

Then, by using Bayes' rule, we have

$$\begin{aligned} p(\mathbf{x}_{0:k}|\mathbf{y}_{0:k},\mathbf{u}_{0:k-1}) &= p(\mathbf{x}_{0:k}|\mathbf{y}_k,\mathbf{y}_{0:k-1},\mathbf{u}_{0:k-1}) \\ &= \gamma \cdot p(\mathbf{y}_k|\mathbf{x}_{0:k},\mathbf{y}_{0:k-1},\mathbf{u}_{0:k-1})p(\mathbf{x}_{0:k}|\mathbf{y}_{0:k-1},\mathbf{u}_{0:k-1}) \end{aligned}$$

The state-space model is a Markov state-space model, we have

$$\mathbf{p(y_k|x_{0:k},y_{0:k-1},u_{0:k-1}) = p(y_k|x_k)} \quad (eq.8)$$

The density $p(\mathbf{y}_k|\mathbf{x}_k)$ is the measurement probability density function explained in the first tutorial part.
By substituting (eq.8) in (eq.7), we obtain

$$\mathbf{p(x_{0:k}|y_{0:k},u_{0:k-1})} = \gamma \cdot \mathbf{p(y_k|x_k)p(x_{0:k}|y_{0:k-1},u_{0:k-1})} \quad (eq.9)$$

Next, we can write

$$\mathbf{p(x_{0:k}|y_{0:k},u_{0:k-1})} = \gamma \cdot \mathbf{p(y_k|x_k)p(x_k,x_{0:k-1}|y_{0:k-1},u_{0:k-1})} \quad (eq.10)$$

write the density $p(\mathbf{x}_k, \mathbf{x}_{0:k-1}|\mathbf{y}_{0:k-1}, \mathbf{u}_{0:k-1})$ as follows

$$\mathbf{p(x_k,x_{0:k-1}|y_{0:k-1},u_{0:k-1}) = p(x_k|x_{0:k-1},y_{0:k-1},u_{0:k-1})p(x_{0:k-1}|y_{0:k-1},u_{0:k-1})} \quad (eq.11)$$

the state-space model is Markov, the probability density $p(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{y}_{0:k-1}, \mathbf{u}_{0:k-1})$ is

$$\mathbf{p(x_k|x_{0:k-1},y_{0:k-1},u_{0:k-1}) = p(x_k|x_{k-1},u_{k-1})} \quad (eq.12)$$

The probability density function of the state at the time instant k depends only on the state and input at the time step k-1. The density $p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_{k-1})$ in (27) is the state transition density of the state-space model introduced in the first tutorial part.
On the other hand, the density $p(\mathbf{x}_{0:k-1}|\mathbf{y}_{0:k-1}, \mathbf{u}_{0:k-1})$ should not depend on the input $\mathbf{u}_{k-1}$. This is because from the state equation of (1) it follows that the state at the time instant k-1 depends on $\mathbf{u}_{k-2}$ and not on $\mathbf{u}_{k-1}$. That is, we have

$$\mathbf{p(x_{0:k-1}|y_{0:k-1},u_{0:k-1}) = p(x_{0:k-1}|y_{0:k-1},u_{0:k-2})} \quad (eq.13)$$

By substituting (eq.12) and (eq.13) in (eq.11), we obtain

$$\mathbf{p(x_k,x_{0:k-1}|y_{0:k-1},u_{0:k-1}) = p(x_k|x_{k-1},u_{k-1})p(x_{0:k-1}|y_{0:k-1},u_{0:k-2})} \quad (eq.14)$$

By substituting (eq.14) in (eq.10), we obtain

$$\mathbf{p(x_{0:k}|y_{0:k},u_{0:k-1})} = \gamma \cdot \mathbf{p(y_k|x_k)p(x_k|x_{k-1},u_{k-1})p(x_{0:k-1}|y_{0:k-1},u_{0:k-2})} \quad (eq.15)$$

- Next, we need to derive a recursive equation for **the smoothed proposal density** (also known as **smoothed importance density**). Again, we start from the general form of the proposal density that takes into account the complete state sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$:

$$\mathbf{q(x_{0:k}|y_{0:k},u_{0:k-1})} \quad (eq.16)$$

This density can be written as

$$\mathbf{q(x_{0:k}|y_{0:k},u_{0:k-1}) = q(x_k,x_{0:k-1}|y_{0:k},u_{0:k-1})} \quad (eq.17)$$

we have

$$\begin{aligned} q(\mathbf{x}_{0:k}|\mathbf{y}_{0:k}, \mathbf{u}_{0:k-1}) &= q(\mathbf{x}_k, \mathbf{x}_{0:k-1}|\mathbf{y}_{0:k}, \mathbf{u}_{0:k-1}) \\ &= q(\mathbf{x}_k|\mathbf{x}_{0:k-1}, \mathbf{y}_{0:k}, \mathbf{u}_{0:k-1})q(\mathbf{x}_{0:k-1}|\mathbf{y}_{0:k}, \mathbf{u}_{0:k-1}) \quad (eq.18) \end{aligned}$$

Since the proposal density is not strictly related to our dynamical system, the Markov property does not hold by default. However, we can always make an assumption that similar to the Markov property. Consequently, we assume

$$\mathbf{q(x_k|x_{0:k-1},y_{0:k},u_{0:k-1}) = q(x_k|x_{k-1},u_{k-1})} \quad (eq.19)$$

Also, the conditional density $q(\mathbf{x}_{0:k-1}|\mathbf{y}_{0:k}, \mathbf{u}_{0:k-1})$ does not depend on the future output $\mathbf{y}_k$. Also, this density does not depend on the input $\mathbf{u}_{k-1}$. This is because from the state equation of the model (1) it follows that the last state in the sequence $\mathbf{x}_{0:k-1}$, that is the state $\mathbf{x}_{k-1}$, does not depend on $\mathbf{u}_{k-1}$. Instead, the state $\mathbf{x}_{k-1}$ depends on $\mathbf{u}_{k-2}$. Consequently, we have that the conditional density has the following form

$$\mathbf{q(x_{0:k-1}|y_{0:k},u_{0:k-1}) = q(x_{0:k-1}|y_{0:k-1},u_{0:k-2})} \quad (\mathbf{eq.20})$$

By substituting (eq.19) and (eq.20) in (eq.18), we obtain the final equation for the batch importance density:

$$\mathbf{q(x_{0:k}|y_{0:k},u_{0:k-1}) = q(x_k|x_{k-1},u_{k-1})q(x_{0:k-1}|y_{0:k-1},u_{0:k-2})} \quad (eq.21)$$

- **Now, we can substitute the recursive relations given by (eq.15) and (eq.21) into the equation for the importance weight (eq.4) of the smoothed posterior density**

$$\begin{aligned}\tilde{w}_k &= \frac{p(\mathbf{x}_{0:k}|\mathbf{y}_{0:k}, \mathbf{u}_{0:k-1})}{q(\mathbf{x}_{0:k}|\mathbf{y}_{0:k}, \mathbf{u}_{0:k-1})} \\ &= \frac{\gamma \cdot p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_{k-1})p(\mathbf{x}_{0:k-1}|\mathbf{y}_{0:k-1}, \mathbf{u}_{0:k-2})}{q(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_{k-1})q(\mathbf{x}_{0:k-1}|\mathbf{y}_{0:k-1}, \mathbf{u}_{0:k-2})} \\ &= \frac{\gamma \cdot p(\mathbf{y}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_{k-1})}{q(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_{k-1})} \cdot \frac{p(\mathbf{x}_{0:k-1}|\mathbf{y}_{0:k-1}, \mathbf{u}_{0:k-2})}{q(\mathbf{x}_{0:k-1}|\mathbf{y}_{0:k-1}, \mathbf{u}_{0:k-2})} \qquad (eq.21)\end{aligned}$$

The last term in the last equation is actually the value of the weight of the smoothed posterior density at the time step k-1:

$$\tilde{\mathbf{w}}_{\mathbf{k-1}} = \frac{\mathbf{p}(\mathbf{x}_{\mathbf{0:k-1}}|\mathbf{y}_{\mathbf{0:k-1}}, \mathbf{u}_{\mathbf{0:k-2}})}{\mathbf{q}(\mathbf{x}_{\mathbf{0:k-1}}|\mathbf{y}_{\mathbf{0:k-1}}, \mathbf{u}_{\mathbf{0:k-2}})} \qquad (eq.22)$$

Consequently, we can write (eq.20) as follows

$$\tilde{\mathbf{w}}_{\mathbf{k}} = \frac{\gamma \cdot \mathbf{p}(\mathbf{y}_{\mathbf{k}}|\mathbf{x}_{\mathbf{k}})\mathbf{p}(\mathbf{x}_{\mathbf{k}}|\mathbf{x}_{\mathbf{k-1}}, \mathbf{u}_{\mathbf{k-1}})}{\mathbf{q}(\mathbf{x}_{\mathbf{k}}|\mathbf{x}_{\mathbf{k-1}}, \mathbf{u}_{\mathbf{k-1}})} \cdot \tilde{\mathbf{w}}_{\mathbf{k-1}}$$

- **One of the simplest possible particle filters, called the bootstrap particle filter or the Sequential Importance Resampling (SIR) filter selects this function as the state transition density**

$$\mathbf{q}(\mathbf{x}_{\mathbf{k}}|\mathbf{x}_{\mathbf{k-1}}, \mathbf{u}_{\mathbf{k-1}}) = \mathbf{p}(\mathbf{x}_{\mathbf{k}}|\mathbf{x}_{\mathbf{k-1}}, \mathbf{u}_{\mathbf{k-1}})$$

we obtain

$$\mathbf{w}_{\mathbf{k}} \propto \mathbf{p}(\mathbf{y}_{\mathbf{k}}|\mathbf{x}_{\mathbf{k}}) \cdot \mathbf{w}_{\mathbf{k-1}}$$

his implies that for the sample $\mathbf{x}_k^{(i)}$, the weight $w_k^{(i)}$ is computed as follows

$$\mathbf{w}_{\mathbf{k}}^{(\mathbf{i})} \propto \mathbf{p}(\mathbf{y}_{\mathbf{k}}|\mathbf{x}_{\mathbf{k}}^{(\mathbf{i})}) \cdot \mathbf{w}_{\mathbf{k-1}}^{(\mathbf{i})}$$

5. **Summerize Particle Filter Algorithm**

- **For the initial set of particles**

$$\{(\mathbf{x}_{\mathbf{0}}^{(\mathbf{i})}, \mathbf{w}_{\mathbf{0}}^{(\mathbf{i})})|\mathbf{i} = \mathbf{1}, \mathbf{2}, \mathbf{3}, \ldots, \mathbf{N}\}$$

- **The optimal Bayesian solution is approximated by a sum of weighted samples:**

$$\boldsymbol{p(x_{0:k}|z_{1:k}) = \sum_{i=1}^{N_s} W_k^i \delta(x_{0:k} - x_{0:k}^i)} \qquad (eq.14)$$

Where:

  - $\{W_k^i, x_k^i\}_{i=1}^{N_s}$ is a set containing N_s sample and weight.
  - $x_{0:k}^i$: is sample represents a possible realization of the state sequence
  - $\sum_{i=1}^{N_s} W_k^i = 1$: because $W_0^i = \frac{1}{N}$
  - $\Delta(.)$: denotes the Dirac delta function. The Dirac delta function $\Delta(a)$ is zero everywhere except for a, its integral is equal to one.

- **The correct way of computing the weights is:**

$$\boldsymbol{W_k^i = \frac{p(x_{0:n}^i|y_{0:n})}{q(x_{0:n}^i|y_{0:n})}}$$

  - The samples are chosen by importance sampling
  - You git to pick the importance density $q(x_{0:n}|y_{0:n})$
  - This is where the samples (particles) x^i come from!
  - You generate them from random numbers
  - We'll talk about how to pick good importance densities later
  - For now, assume that we've chosen one (Gaussian, say)
  - How the do we recursively and efficiently calculate the weights?

- the found previously that

$$\mathbf{W_n} = \frac{\mathbf{p(y_n|x_n)p(x_n|x_{n-1})}}{\mathbf{q(x_n|x_{n-1}, y_n}}\mathbf{w_{n-1}}$$

- **Weight Update Discussion**
  With the particle index i, this becomes

$$w_n^i \propto \frac{p(y_n|x_n^i)p(x_n^i|x_{n-1}^i)}{q(x_n^i|x_{n-1}^i, y_n}w_{n-1}^i$$

  - Thus, we end up with this beautiful recursion for the weights
  - Consider the terms
    * We get to pick $q(x_n|x_{0:n-1}, y_{0:n})$
    * $p(y_n|x_n)$ can be obtained from the measurement model $y = h_n(x_n, v_n)$
    * $p(x_n|x_{n-1}$ can be obtained from the process model $x = f_n(x_n, w_n)$
    * w_{n-1}^i is known from the previous time step for each particle i
- **How do we obtain the Marginal Posterior?**

$$\hat{p}(x_{0:n}|y_{0:n}) \triangleq \sum_{i=1}^{n_p} w_n^i \delta(x_{0:n} - x_{0:n}^i)$$

  - We now have a beautiful mean of estimating the joint posterior
  - But what we really need for estimation is the marginal posterior, $p(x_n|y_{0:n})$
- **Obtaning the Marginal from the joint**

$$\hat{p}(x_{0:n}|y_{0:n}) \triangleq \sum)i = 1^{n_p} w_n^i \delta(x_{0:n} - x_{0:n}^i)$$

  Note that

$$\delta(x_{0:n}) = \prod_{k=0}^{n} \delta(x_k) = \delta(x_0)\delta(x_1)\ldots\delta(x_n)$$

  therefore

$$\hat{p}(x_n|y_{0:n}) = \int \sum_{i=1}^{n_p} w_n^i \delta(x_n - x_n^i)$$

- **Sequential Importance Sampling (SIS) Algorithm**

$$\Big[\{x_n^i, w_n^i\}_{i=1}^{n_p}\Big] = SIS\Big[\{x_{n-1}^i, w_{n-1}^i\}_{i=1}^{n_p}, y_n\Big]$$

  - Calculate the unnormalized weights

$$\tilde{w_n^i} = \frac{p(y_n|x_n^i)p(x_n^i|x_{n-1}^i)}{q(x_n^i|x_{n-1}^i, y_n}w_{n-1}^i$$

  - Calculate the normalized weights

$$w_n^i = \frac{\tilde{w_n^i}}{\sum_{i=1}^{n_p} \tilde{w_n^i}}$$

4.4 Python Implementation of Particle Filter Algorithm

5 Complementary Filter

5.1 What is Complementary Filter?

A complementary filter is a data fusion algorithm used primarily in signal processing to combine data from multiple sensors, often to estimate the orientation or position of an object. It is particularly common in applications involving inertial measurement units (IMUs) that include accelerometers and gyroscopes.

5.2 Complementary Filter Algorithm

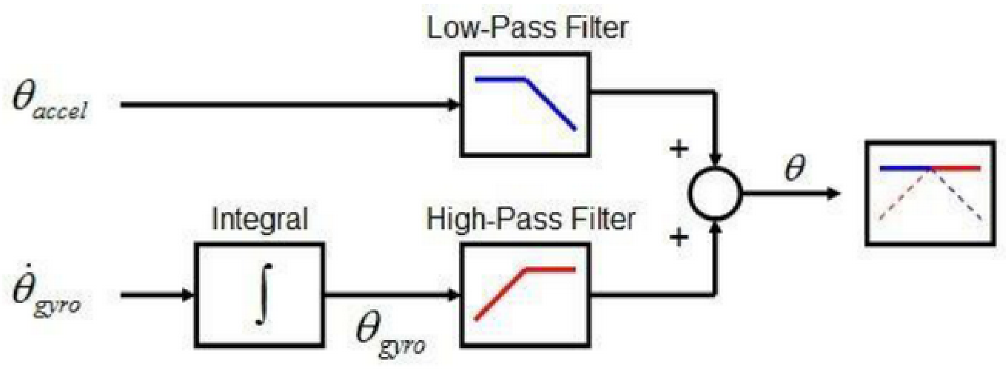

Figure 7: Complementary Filter Block diagram

- **Equation for low-pass filter:**

$$\boldsymbol{Y[n] = (1-\alpha)X[n] + \alpha Y[n]} \quad \text{(use this for angles obtained from accelerometers)}$$

  - **x**[**n**] : is the pitch/roll/yaw that you get from the accelerometer.
  - **y**[**n**] : is the filtered final pitch/roll/yaw which you must feed into the next phase of your program.

- **Equation for high-pass filter:**

$$\boldsymbol{Y[n] = (1-\alpha)y[n-1] + (1-\alpha)(X[n] - X[n-1])} \quad \text{(use this for angles obtained from)}$$

  - **X**[**n**] : is the pitch/roll/yaw that you get from the gyroscope
  - **Y**[**n**] : is the filtered final pitch/roll/yaw which you must feed into the next phase of your program.

- **How to choose alpha?**

$$\boldsymbol{\alpha = \frac{\tau}{\tau + dt}}$$

  - tau is the desired time constant (how fast you want the readings to respond)
  - dt = 1/fs where fs is your sampling frequency.
  - alpha is related to time-constant. it defines the boundary where the accelerometer readings stop and the gyroscope readings take over and vice-versa. It controls how much you want the output to depend on the current value or a new value that arrives. Both the alpha's have to be the same. alpha is usually ¿ 0.5 using the definitions above.

- **Equation of Complementary Filter**:

$$\boldsymbol{\theta = \alpha\theta_{gyr} + (1-\alpha)\theta_{acc}}$$

  - $\boldsymbol{\theta_{acc}}$:angles computed from the accelerometer measurements

$$\boldsymbol{\theta_{acc} = \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \end{bmatrix} = \begin{bmatrix} arctan2(a_y, a_z) \\ arctan2(-a_x, \sqrt{a_y^2 + a_x^2}) \\ 0 \end{bmatrix}}$$

Where: $(\theta_x, \theta_y, \theta_z) : roll, pich, yaw$
If a magnetometer sample is available, the yaw angle, θ_z can be computed. First compensate the measured magnetic field using the tilt:

$$\boldsymbol{\begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} m_x cos(\theta_x) + m_y sin(\theta_x) sin(\theta_y) + m_z sin(\theta_x) cos(\theta_y) \\ m_y cos(\theta_y) - m_z sin(\theta_y) \\ -m_x sin(\theta_x) + m_y sin(\theta_x) cos(\theta_y) + m_z * cos(\theta_x) cos(\theta_y) \end{bmatrix}}$$

$$=> \boldsymbol{\theta_z = arctan2(-b_y, b_x)}$$

- $\boldsymbol{\theta_{gyr}}$: is the attitude estimated from the gyroscope.

$$\boldsymbol{\theta_{gyr} = \begin{bmatrix} \theta_{x_t} \\ \theta_{y_t} \\ \theta_{z_t} \end{bmatrix} = \begin{bmatrix} \theta_{x_t} + \omega_x * dt \\ \theta_{y_t} + \omega_y * dt \\ \theta_{z_t} + \omega_z * dt \end{bmatrix}}$$

  Where: ω is angular velocity(orientation), dt is the time interval between the current and previous measurements, a.k.a. the sampling period or time step.

- $\boldsymbol{\alpha}$ is filter gain must be a floating value within the range[0.0. 1.0]

5.3 Python Implementation of Complementary Filter Algorithm

6 Least Squares Estimation (LSE)

6.1 What is Least Squares Estimation?

Least square approach is one of the most popular estimation techniques that has been widely applied to diverse fields such as machine learning, system identification, and adaptive control etc. The least square estimation (LSE) is mainly used to approximate a parameter of a model by measuring the system inputs and outputs, so it also called “linear regression” in statistics.

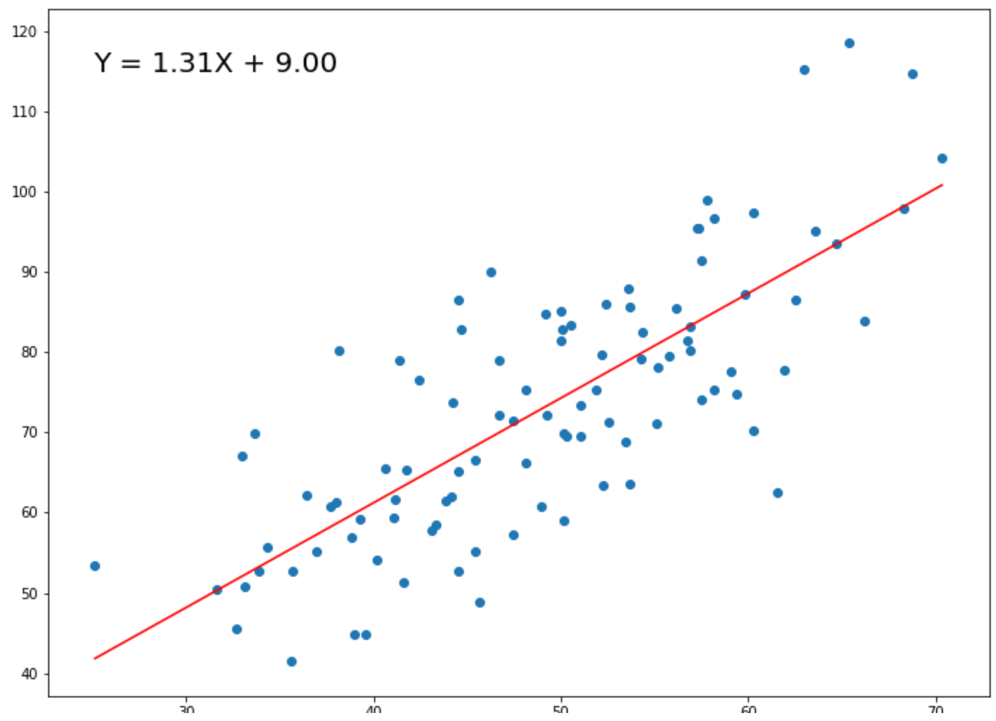

Figure 8: Least square estimate

6.2 LSE Algorithm

In general, Consider a linear homogeneous equation

$$\mathbf{y} = \mathbf{b_1 u_1} + \mathbf{b_2 u_2} + \ldots + \mathbf{b_m u_m} \qquad (eq.1)$$

Where:

$$\boldsymbol{\theta} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \qquad \text{Parameter Vector}$$

$$\boldsymbol{\psi} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix} \qquad \text{Regressor}$$

$$\boldsymbol{y} = \boldsymbol{\theta^T \psi} = \boldsymbol{\psi^T \theta} \qquad \text{Linear Regression} \qquad \text{(eq.2)}$$

- **For Finding parameters from Data, y(t) and ψ(t)**
  - Prediction Error formalism

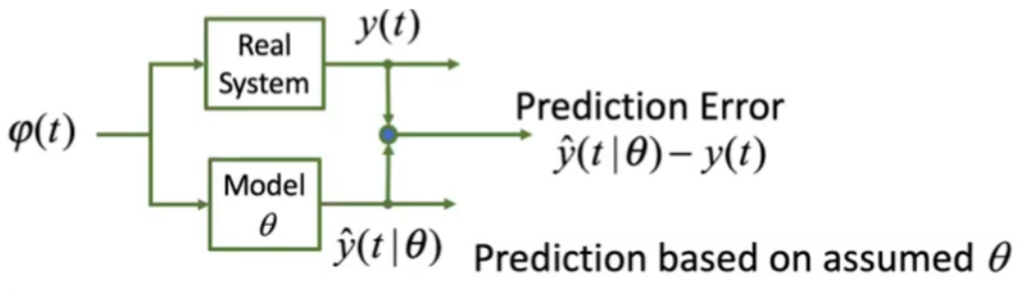

Figure 9: LSE Prediction error

- Mean Squared Error:

$$\boldsymbol{V_N(\theta) = \frac{1}{N}\sum_{t=1}^{N}(\hat{y}(t|\theta) - y(t))^2} \quad (eq.3)$$

- Least squares estimate(LSE) provided the parameter vector that minimizes the above Mean squard Error:

$$\boldsymbol{\hat{\theta}^{LS} = argmin_{\theta}V_N(\theta) \quad (eq.4)}$$

- **Prediction error formalism**
  Broadly used in estimation, system identification, and machine learning:
  Solution: The necessary conditions for V_N to take a minimum:

$$\boldsymbol{\frac{dV_N(\theta)}{d\theta} = 0}$$

Let $x_t = \hat{y}(t|\theta) - y(t)$

$$\boldsymbol{\frac{\partial V_N(\theta)}{\partial\theta_i} = \frac{1}{N}\sum_{t=1}^{N}\frac{dx_t^2}{dx_t}\frac{\partial x_t}{\partial\theta_i} = \frac{2}{N}\sum_{t=1}^{N}x_t\frac{\partial\hat{y}(t|\theta)}{\partial\theta_i} = \frac{2}{N}\sum_{t=1}^{N}x_t\psi_i(t) = \frac{2}{N}\sum_{t=1}^{N}(\hat{y}(t|\theta) - y(t))\psi_i(t)}$$

$$\boldsymbol{\textstyle\sum_{t=1}^{N}(\hat{y}(t|\theta) - y(t))\psi_i(t) = 0 \quad (eq.5)}$$

Where: from (eq.2) $y = \theta^T\psi = \psi^T(t)\theta$

$$\boldsymbol{\textstyle\sum_{t=1}^{N}(\psi^T(t)\theta - y(t))\psi_i(t))\psi(t) = 0 \quad => \sum_{t=1}^{N}[\psi^T(t)\theta]\psi(t) = \sum_{t=1}^{N}y(t)\psi(t)}$$

$$\boldsymbol{\sum_{t=1}^{N}[\psi^T(t)\psi(t)]\theta = \sum_{t=1}^{N}y(t)\psi(t)}$$

The least square estimate we have obtained is an **offline** batch processing algorithm after collecting all data.

$$\boldsymbol{\hat{\theta}^{LS} = \Big[\sum_{t=1}^{N}\psi^T(t)\psi(t)\Big]^{-1}\sum_{t=1}^{N}y(t)\psi(t)} \quad (eq.6)$$

- **The Recursive least squares algorithm**
  Rather than waiting until all the data are obtained you want to obtain the optional estimate based on the daa you have obtained so far. If the optimal estimate $\hat{\theta}(t)$ depends only on $\theta(t \hat{-} 1)$ and new data $\psi(t), y(t)$, we can forget old data find are cursive fomular:

$$\boldsymbol{\hat{\theta}^{LS}(t) = \hat{\theta}^{LS}(t-1) + [correction/updatebasedon\psi(t), y(t)]} \quad (eq.7)$$

Let us derive the recursive formula to update the estimate with new observation $\psi(t)$ and $y(t)$

- **Step1:** We assume that enough data are initially available so that the matrix $\sum_{i=1}^{t}\psi(i)\psi^T(i)$

  Define $\boldsymbol{P_t = \Big[\sum_{i=1}^{t}\psi^T(i)\psi(i)\Big]^{-1}}$ and $\boldsymbol{B_t = \sum_{i=1}^{t}y(i)\psi(i)}$
  So that : $\boldsymbol{\hat{\theta}^{LS} = P_tB_t}$

  Split each of these into the new observation and the one from the previous step

$$\boldsymbol{B_t = \sum_{i=1}^{t-1} y(i)\psi(i) + y(t)\psi(t) = B_{t-1} + y(t)\psi(t)} \qquad \text{(eq.8)}$$
$$\boldsymbol{P_t^{-1} = \sum_{i=1}^{t} \psi^T(i)\psi(i) = \sum_{i=1}^{t-1} \psi^T(i)\psi(i) + \psi^T(t)\psi(t)} \qquad (eq.9)$$

– **Step2: The matrix inversion lemma on (eq.9)**
  Can we compute P_t recursively without taking matrix inversion? Yes,We can :
  * Pre-multiply P_t and Post-multiply P_{t-1}:
    We have $\boldsymbol{P_t^{-1} = P_{t-1}^{-1} + \psi(t)\psi(t)^T}$

$$\boldsymbol{P_t P_t^{-1} P_{t-1} = P_t P_{t-1}^{-1} P_{t-1} + P_t \psi(t)\psi(t)^T P_{t-1}}$$

  * Further post-multiply $\psi(t)$

$$\boldsymbol{P_{t-1}\psi(t) = P_t\psi(t) + P_t\psi(t)\psi^T(t)P_{t-1}\psi(t) = P_t\psi(t)(1 + \psi^T(t)P_{t-1}\psi(t))}$$

  * Note that $1 + \psi^T(t)P_{t-1}\psi(t)$ is a scalar quantity .

$$\boldsymbol{P_t = \frac{P_{t-1}}{1 + \psi^T(t)P_{t-1}\psi(t)}}$$
$$\boldsymbol{P_t\psi^T(t)P_{t-1}\psi(t) = P_{t-1} + P_t}$$
$$\boldsymbol{P_t = P_{t-1} - P_t\psi(t)\psi^T(t)P_{t-1}}$$

    And

$$\boldsymbol{P_t\psi(t) = \frac{P_{t-1}\psi(t)}{1 + \psi^T(t)P_{t-1}\psi(t)}}$$

  * So the And Equation:

$$\boldsymbol{P_t = P_{t-1} - \frac{P_{t-1}\psi(t)\psi^T(t)P_{t-1}}{1 + \psi^T(t)P_{t-1}\psi(t)}} \qquad \boldsymbol{(eq.10)}$$

– **Step3: Reduce $\hat{\theta}^{LS} = P_t B_t$ to a recursive formula**
  * We can show that the recursis given by the following form:

$$\boldsymbol{\hat{\theta}^{LS}(t) = \hat{\theta}^{LS}(t-1) + K_t[y(t) - \hat{y}(t)\hat{\theta}^{LS}(t-1)]} \qquad (eq.11)$$

  K_t : is optimal gain for correcting the estimate
  * By definition:

$$\boldsymbol{\hat{\theta}^{LS}(t) - \hat{\theta}^{LS}(t-1) = P_t B_t - P_{t-1}B_{t-1}} \qquad (eq.12)$$

  * Input (eq.10) and (eq.8) to (eq.12):

$$\boldsymbol{\hat{\theta}^{LS}(t) - \hat{\theta}^{LS}(t-1) = \Big(P_{t-1} - \frac{P_{t-1}\psi(t)\psi^T(t)P_{t-1}}{1 + \psi^T(t)P_{t-1}\psi(t)}\Big)(B_{t-1} + y(t)\psi(t)) - P_{t-1}B_{t-1}}$$

  So

$$\boldsymbol{\hat{\theta}^{LS}(t) - \hat{\theta}^{LS}(t-1) = P_{t-1}y\psi - \frac{P_{t-1}\psi\psi^T P_{t-1}}{1 + \psi^T P_{t-1}\psi}(B_{t-1} + y\psi)}$$

  Scalar y can be moved:

$$\boldsymbol{\hat{\theta}^{LS}(t) - \hat{\theta}^{LS}(t-1) = \frac{P_{t-1}y\psi + P_{t-1}y\psi\psi^T P_{t-1}\psi - P_{t-1}\psi\psi^T P_{t-1}B_{t-1} - P_{t-1}\psi\psi^T P_{t-1}y\psi}{1 + \psi^T P_{t-1}\psi}}$$

$$\boldsymbol{\hat{\theta}^{LS}(t) - \hat{\theta}^{LS}(t-1) = \frac{P_{t-1}y\psi - P_{t-1}\psi\psi^T P_{t-1}B_{t-1}}{1 + \psi^T P_{t-1}\psi}}$$

  Factoring out:

$$\boldsymbol{\hat{\theta}^{LS}(t) - \hat{\theta}^{LS}(t-1) = \frac{P_{t-1}\psi}{1 + \psi^T P_{t-1}\psi}(y - \psi^T P_{t-1}B_{t-1})}$$

  So We get:

$$\boldsymbol{\hat{\theta}^{LS}(t) = \hat{\theta}^{LS}(t-1) + \frac{P_{t-1}\psi}{1 + \psi^T P_{t-1}\psi}(y - \psi^T P_{t-1}B_{t-1})} \qquad (eq.13)$$

    * Compare (eq.11) with (eq.13): So We get:

$$\boldsymbol{K_t = \frac{P_{t-1}\psi}{1+\psi^T P_{t-1}\psi}} \qquad (eq.14)$$

  - **Step4: Recursive least squares with exponential forgetting factor**

$$\boldsymbol{P_t = \frac{1}{\alpha}\Big[P_{t-1} - \frac{P_{t-1}\psi(t)\psi^T(t)P_{t-1}}{1+\psi^T(t)P_{t-1}\psi(t)}\Big]}$$

    Where $\boldsymbol{\alpha}$ is Exponential forgetting factor ($0 < \alpha < 1$)

- **Summarize Algorithm**
  Given $\hat{\theta}(0)$ and P_0
  Where: θ is parameter to estimate ; ψ is input ;y is output of model

  - Model :

$$\boldsymbol{y(t) = \psi^T(t)\theta(t)}$$

  - Estimate Model:

$$\boldsymbol{\theta^{LS}(t) = \theta^{LS}(t-1) + K(t)e(t)}$$

  - Calculate Error:

$$\boldsymbol{e(t) = y(t) - \hat{y}(t)\hat{\theta}(t-1)}$$

  - Optimal Gain:

$$\boldsymbol{K_t = \frac{P_{t-1}\psi(t)}{1+\psi(t)^T P_{t-1}\psi(t)}}$$

  - Update Covariance P:

$$\boldsymbol{P_t = \frac{1}{\alpha}\Big[P_{t-1} - \frac{P_{t-1}\psi(t)\psi^T(t)P_{t-1}}{1+\psi^T(t)P_{t-1}\psi(t)}\Big]}$$

6.3 Python Implementation of Least Squares Estimation(LSE) Algorithm

7 Least Mean Squares (LMS)

7.1 What is Least Mean Squares?

The Least Mean Squares (LMS) method is an adaptive algorithm widely used for finding the coefficients of a filter that will minimize the mean square error between the desired signal and the actual signal. It is mainly utilized in training algorithms such as gradient descent, where the network finalizes a target function by iteratively adjusting its weights w.r.t. the error between predicted and actual outputs.

7.2 Least Mean Squares Algorithm

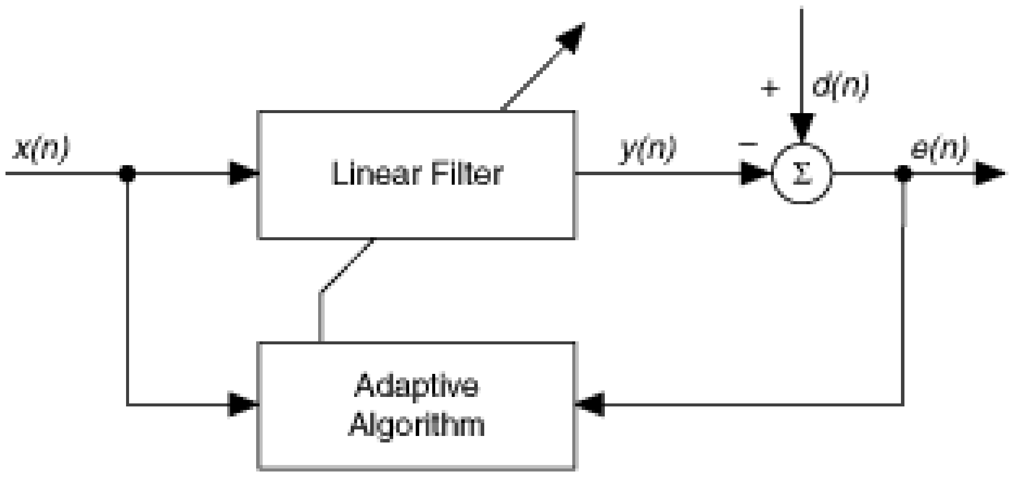

Figure 10: Typical Adaptive Filter

- **Initialization**
$$\mathbf{w(0) = zeros(p)}$$
Where p is filter order
- **Filter Output**
$$\boldsymbol{y(k) = x^T(k)w(k)}$$
Where:
  - k is discrete time index
  - x is input vector (for a filter of size n) : $\mathbf{x(k) = [x_1(k), \dots, x_n(k)]}$
  - n is number of the current input sample.
  - w is weight vector of filter adaptive parameters
  - y is output vector filtered signal
- **Estimation of error**
$$\mathbf{e(k) = d(k) - y(k)}$$
Where: d is desired output value
- **The squared error**
Squaring the error using vector notation, we get:
$$\boldsymbol{e^2 = (d - x^T w)^2 = (d^T d - 2d^T x^T w + w^T x^T x w)}$$
$$\boldsymbol{\sum_{k=1}^{N} e^2(k) = \sum_{k=1}^{N} d^2(k) + w^T R w - 2P^T w}$$
Define the Quadratic error (QE):
$$\boldsymbol{QE = \sum_{k=1}^{N} e(k)^2 - \sum_{k=1}^{N} d(k)^2 = w^T R w - 2P^T w}$$
Where:

  - $R = x^T x$ is the cross-correlation of the input vectors.
  - $P^T = d^T x$ is the cross-correlation of the desired output with the input

- **The gradient descent**
  The gradient of the mean-square-error can be obtained by differentiating the error with respect to the weights, or:
  $$\boldsymbol{\bigtriangledown = \frac{\partial \xi}{\partial w} = Rw - P = x^T x w - d^T x = x(x^T w - d) = -xe}$$
  Where:
  $\xi = \sum_{k=1}^{N} e^2(k)$

- **Update weights adaptation**
  $$\boldsymbol{w(k+1) = w(k) + \Delta w(k)}$$
  $$\boldsymbol{w(k+1) = w(k) - \alpha \bigtriangledown (k)}$$
  $$\boldsymbol{w(k+1) = w(k) + \mu \cdot e(k) \cdot x(k)}$$
  Where:

  - μ is step size $(0 < \mu < \frac{2}{\lambda_{max}})$
  $$\boldsymbol{\mu = \frac{2}{\lambda_{max} + \lambda_{min}}}$$
  - λ_{max} is the greatest eigenvalue of the autocorrelation matrix R .
  - λ_{min} is the smallest eigenvalue of R

7.3 Python Implementation of Least Mean Squares(LMS)

8 Maximum Likelihood Estimation (MLE)

8.1 What is Maximum Likelihood Estimation?

Maximum Likelihood Estimation (MLE) is a statistical method used to estimate the parameters of a probability distribution that best describe a given dataset. The fundamental idea behind MLE is to find the values of the parameters that maximize the likelihood of the observed data, assuming that the data are generated by the specified distribution.

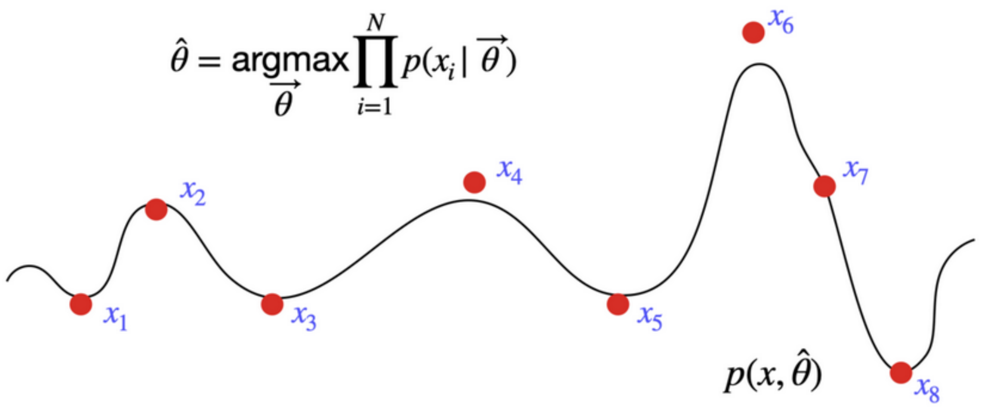
$\hat{\theta} = \underset{\vec{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p(x_i \mid \vec{\theta})$
x_1
x_2
x_3
x_4
x_5
x_6
x_7
x_8
$p(x, \hat{\theta})$

Figure 11: Graph of Maximum Likelihood estimate

8.2 Review Probability

- **Normal Distribution or Gaussian distribution**
  the normal distribution or Gaussian distribution or bell curve is one of the most important continuous probability distributions. The normal distribution is defined as the probability density function f(x) for the continuous random variable, say x, in the system. A normal distribution is a very important statistical data distribution pattern occurring in many natural phenomena, such as height, blood pressure, lengths of objects produced by machines, etc. Here, we are going to discuss the normal distribution formula and examples in detail.
  - probability density function(pdf):
  $$\boldsymbol{f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma}}}$$
  - x $\in [-\infty, \infty]$
  - $X \backsim N(\mu, \sigma^2)$
- **Bernoulli Distribution**
  Bernoulli Distribution is a special kind of distribution that is used to model real-life examples and can be used in many different types of applications. A random experiment that can only have an outcome of either 1 or 0 is known as a Bernoulli trial. Such an experiment is used in a Bernoulli distribution.
  - probability mass function(PMF):
  $$\boldsymbol{f(x, p) = p^x(1-p)^{1-x}, \quad x \in [0, 1]}$$
  - $X \backsim Ber(p)$
  - p is probability
  - Mean: $\mu = p$
  - Variance: $Var = p(1-p)$
- **Poisson Distribution**
  A Poisson distribution is a discrete probability distribution, meaning that it gives the probability of a discrete (i.e., countable) outcome. For Poisson distributions, the discrete outcome is the number of times an event occurs, represented by k.

- probability mass function(PMF):

$$\mathbf{P(x = k) = \frac{e^{-\lambda}\lambda^{k}}{k!}}$$

  Where:
- $X \backsim Po(\lambda)$
- X is a random variable following a Poisson distribution
- k is the number of times an event occurs$[0, \infty]$
- λ is the average number of times an event occurs

- **Exponential Distribution**
  the exponential distribution is a continuous probability distribution that often concerns the amount of time until some specific event happens. It is a process in which events happen continuously and independently at a constant average rate. The exponential distribution has the key property of being memoryless. The exponential random variable can be either more small values or fewer larger variables. For example, the amount of money spent by the customer on one trip to the supermarket follows an exponential distribution.

  - probability density function(PDF):

$$\boldsymbol{f_X(x|\lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{for} \quad x > 0 \\ 0 & \text{for} \quad x \leq 0 \end{cases}}$$

  - λ is called the distribution rate.
  - Mean: $E(x) = \mu = \frac{1}{\lambda}$
  - Variance: $Var(x) = \frac{1}{\lambda}$

- **Gamma Distribution**
  The gamma distribution term is mostly used as a distribution which is defined as two parameters – shape parameter and inverse scale parameter, having continuous probability distributions. It is related to the normal distribution, exponential distribution, chi-squared distribution and Erlang distribution.

  - cumulative distribution function(CDF):

$$\boldsymbol{f(X : \alpha, \beta) = \begin{cases} \frac{1}{\beta^{\alpha}\gamma(\alpha)X} X^{\alpha-1} e^{\frac{-X}{\beta}} & \text{for} \quad X > 0 \\ 0 & \text{for} \quad X \leq 0 \end{cases}}$$

  - Gamma function:

$$\boldsymbol{\gamma(\alpha) = \int_{0}^{\infty} X^{\alpha-1} e^{-X} dx}$$

  Where:
  - $\alpha > 0, \beta > 0$

- **Uniform Distribution**
  A continuous probability distribution is a Uniform distribution and is related to the events which are equally likely to occur. It is defined by two parameters, x and y, where x = minimum value and y = maximum value. It is generally denoted by u(x, y).

  - probability density function(PDF):

$$\boldsymbol{f(x) = \begin{cases} \frac{1}{b-a}, & a \leq x \leq b \\ 0, & otherwise \end{cases}}$$

  - $X \backsim U(a, b)$ Where:
  - $E(x) = \frac{a+b}{2}$
  - $V(x) = \frac{(b-a)^2}{12}$

8.3 MLE Algorithm

1. Maximum Likelihood Estimate:
$$\hat{\theta}^{MLE} = argmax_{\theta}[log(L_n(X_1, X_2, \ldots, X_n, p))]$$
2. likelihood function:
$$L_n(X_1, X_2, \ldots, X_n, \theta) = \prod_{i=1}^{n} p_{\theta}(x_i)$$
3. log-likelihood fucntion:
$$log(L_n(X_1, X_2, \ldots, X_n, \theta)) = log\Big(\prod_{i=1}^{n} p_{\theta}(x_i)\Big)$$
4. Calculate of the first derivative:
$$\frac{\partial}{\partial\theta}[log(L_n(X_1, X_2, \ldots, X_n, p))] = 0$$

- **For Bernoulli Distribution:**
  - Model:
$$(\{0, 1\}, X \backsim Ber(p))$$
  - Parameter $\theta = p$
  - Probability Mass Function:
$$p_{\theta}(X) = p_p(X) = p^x(1-p)^{1-x}$$
  - likelihood function:
$$L_n(X_1, X_2, \ldots, X_n, p) = \prod_{i=1}^{n} p^{x_i}(1-p)^{1-x_i}$$
$$\Rightarrow L_n(X_1, X_2, \ldots, X_n, p) = p^{\sum_{i=1}^{n} x_i}(1-p)^{n-\sum_{i=1}^{n} x_i}$$
  - Log-likelihood Function:
$$log(L_n(X_1, X_2, \ldots, X_n, p)) = log(p^{\sum_{i=1}^{n} x_i}(1-p)^{n-\sum_{i=1}^{n} x_i})$$
$$\Rightarrow log(L_n(X_1, X_2, \ldots, X_n, p)) = \sum_{i=1}^{n} x_i log(p)\Big(n - \sum_{i=1}^{n} x_i\Big)log(1-p)$$
  - Maximum Likelihood Estimator:
$$\hat{\theta}^{MLE} = argmax_{\theta}[log(L_n(X_1, X_2, \ldots, X_n, p))]$$
$$\Rightarrow \hat{p}^{MLE} = argmax_{p\in\{0,1\}}\Big[\sum_{i=1}^{n} x_i log(p)\Big(n - \sum_{i=1}^{n} x_i\Big)log(1-p)\Big]$$
  - Calculation of the First derivative:
$$\frac{\partial}{\partial\theta}[log(L_n(X_1, X_2, \ldots, X_n, p))] = \frac{\partial}{\partial p}\Big[\sum_{i=1}^{n} x_i log(p)\Big(n - \sum_{i=1}^{n} x_i\Big)log(1-p)\Big] = \frac{\sum x_i}{p} - \frac{(n - \sum x_i)}{1-p}$$
  - Calculation of Critical Points in (0, 1)
$$\frac{\partial}{\partial\theta}[log(L_n(X_1, X_2, \ldots, X_n, p))] = 0$$
$$\Rightarrow \frac{\sum x_i}{p} - \frac{(n - \sum x_i}{1-p} = 0 \Rightarrow \frac{\sum x_i}{p} = \frac{(n - \sum x_i}{1-p}$$
$$\Rightarrow (1-p)\sum x_i = p(n - \sum x_i) \Rightarrow pn = \sum_{i=1}^{n} x_i$$
$$\Rightarrow p = \frac{1}{n}\sum_{i=1}^{n} x_i \quad (eq.1)$$

- Calculation of the Second derivative:

$$\frac{\partial^2}{\partial\theta^2}[log(L_n(X_1,X_2,\dots,X_n,p))] = \frac{\partial}{\partial p}\Big[\frac{\sum x_i}{p} - \frac{(n-\sum x_i}{1-p}\Big] = -\frac{\sum x_i}{p^2} - \frac{(n-\sum x_i}{(1-p)^2}$$

- Substituting (eq.1) in the above expression, we obtain,

$$-\frac{n^2\sum x_i}{(\sum x_i)^2} - \frac{n^2(n-\sum x_i)}{(n-\sum x_i)^2} = -\Big(\frac{n^2}{\sum x_i} + \frac{n^2}{n-\sum x_i}\Big) < 0$$

- Therefore maximizer of the log-likelihood is

$$\hat{p}^{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- **For Poisson Distribution:**
  - Model:

$$(N \cup \{0\}, X \backsim Po(\lambda)$$

  - Probability Mass Function:

$$p_\theta(X) = p_\lambda(X) = \frac{\lambda^x e^{-\lambda}}{x!}$$

  - Likelihood Function:

$$L_n(X_1,X_2,\dots,X_n,\lambda) = \prod_{i=1}^{n}\frac{\lambda^{x_i}e^{-\lambda}}{x_i!}$$

$$\Rightarrow L_n(X_1,X_2,\dots,X_n,\lambda) = \frac{\lambda^{\sum x_i}e^{-n\lambda}}{\prod x_i!}$$

  - Log-likelihood Function:

$$log(L_n(X_1,X_2,\dots,X_n,\lambda)) = log\Big(\frac{\lambda^{\sum x_i}e^{-n\lambda}}{\prod x_i!}\Big)$$

$$\Rightarrow log(L_n(X_1,X_2,\dots,X_n,\lambda)) = \sum_{i=1}^{n} x_i log\lambda - n\lambda - \sum_{i=1}^{n} log x_i!$$

  - Maximum Likelihood Estimator:

$$\hat{\theta}^{MLE} = argmax_\theta[log(L_n(X_1,X_2,\dots,X_n,\lambda))]$$

$$\hat{\lambda}^{MLE} = argmax_{\lambda\in(0,\infty)}\Big[\sum_{i=1}^{n} x_i log\lambda - n\lambda - \sum_{i=1}^{n} log x_i!\Big]$$

  - Calculation of the First derivative:

$$\frac{\partial}{\partial\lambda}[log(L_n(X_1,X_2,\dots,X_n,\lambda))] = \frac{\partial}{\partial\lambda}\Big[\sum_{i=1}^{n} x_i log\lambda - n\lambda - \sum_{i=1}^{n} log x_i!\Big] = \frac{\sum x_i}{\lambda} - n$$

  - Critical Points in (0, ∞)

$$\frac{\partial}{\partial\lambda}[log(L_n(X_1,X_2,\dots,X_n,\lambda))] = 0$$

$$\Rightarrow \frac{\sum x_i}{\lambda} - n = 0 \Rightarrow \frac{\sum x_i}{\lambda} = n$$

$$\Rightarrow \lambda = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad (eq.1)$$

  - Second derivative:

$$\frac{\partial^2}{\partial\lambda^2}[log(L_n(X_1,X_2,\dots,X_n,\lambda))] = \frac{\partial}{\partial\lambda}\Big[\frac{\sum x_i}{\lambda} - n\Big] = -\frac{\sum x_i}{\lambda^2}$$

– Substituting (eq.1) in the above expression, we obtain,

$$-n^2 \frac{\sum x_i}{(\sum x_i)^2} = -\frac{n^2}{(\sum x_i)^2} < 0$$

– Therefore is the maximizer of the log-likelihood.Thus

$$\hat{\lambda}^{MLE} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- **For Exponential Distribution:**
  The maximizer of the log-likelihood is :

$$\hat{\lambda}^{MLE} = \frac{n}{\sum x_i}$$

- **For Normal or Gaussian Distribution**
  The maximizer of the log-likelihood is :

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

8.4 Python Implementation of Maximum Likelihood Estimation(MLE)