



# Kalman Filter Simplified

## Table of Contents

Introduction .....	1
Background & Theory .....	2
Single valued Kalman Filter .....	2
Calculating the Kalman Gain, Current Estimate, Error Estimate.....	3
Single Valued Example .....	3
Adding Noise .....	5
Adding Noise Example .....	5
Multi-Variable Kalman Filter .....	8
State Space Representation .....	9
Matrix Operations.....	9
Statistical Background.....	10
Multi-Dimensional Kalman Filter Flow Chart and Discussion .....	14
Using Kalman Filters in Software .....	18
Python 2-Dimensional: Position and Velocity Predictor .....	18
Kalman Filter Code .....	23
GUI Code .....	26

## Introduction

A Kalman Filter used in modern systems that have numerous sensors that estimate hidden states based on series of measurements. Like for example, a GPS receiver which provides location and velocity estimation, where there is a rate of change of time of the satellite signal's arrival for location and velocity. One of the biggest challenges if tracking and control systems is providing a **quick, accurate and precise estimation of a system with the presence of uncertainty**. This uncertainty can depend on the application and environment of the system but for GPS receivers it typically depends on thermal noise, atmospheric effects, satellite positioning, receiver clock precision, and more.

More Examples of Applications of Kalman Filter:

- Object Tracking – Use for estimating the location of a vehicle
  - Uncertainty – Discrete noise, variance in acceleration and

- Modeling of Interest Rates – Used to predict future possible interest rates of a loan or federal reserve.
- Estimating Temperature of a system or room.

When dealing with uncertainty with any estimation calculation the Kalman filter is one of the most important and common estimation algorithms. It can also predict the future system state based on past estimations. The Kalman filter is also very useful method when dealing with large amounts of data needs to be considered because of fast computation algorithm. In this lab, the basic principles of the method will be explained and to show how we can use it to estimate a model's parameters with two Kalman filters. The first one is a simple filter, which estimate linear data and the second one, will work with non-linear models. Later, two additional Kalman filters will be prototyped in Python with its test before using it in C++ for different applications. Finally, the Disadvantages and advantages of the Kalman filter will be mentioned in terms of implementation that is usually not mentioned in literature.

## Background & Theory

### Single valued Kalman Filter

All Kalman Filter will use State Space Representation when modeling. But first let's start to understand a simple case: single measured value Kalman filter. A flowchart for a single measured Kalman filter is shown below:

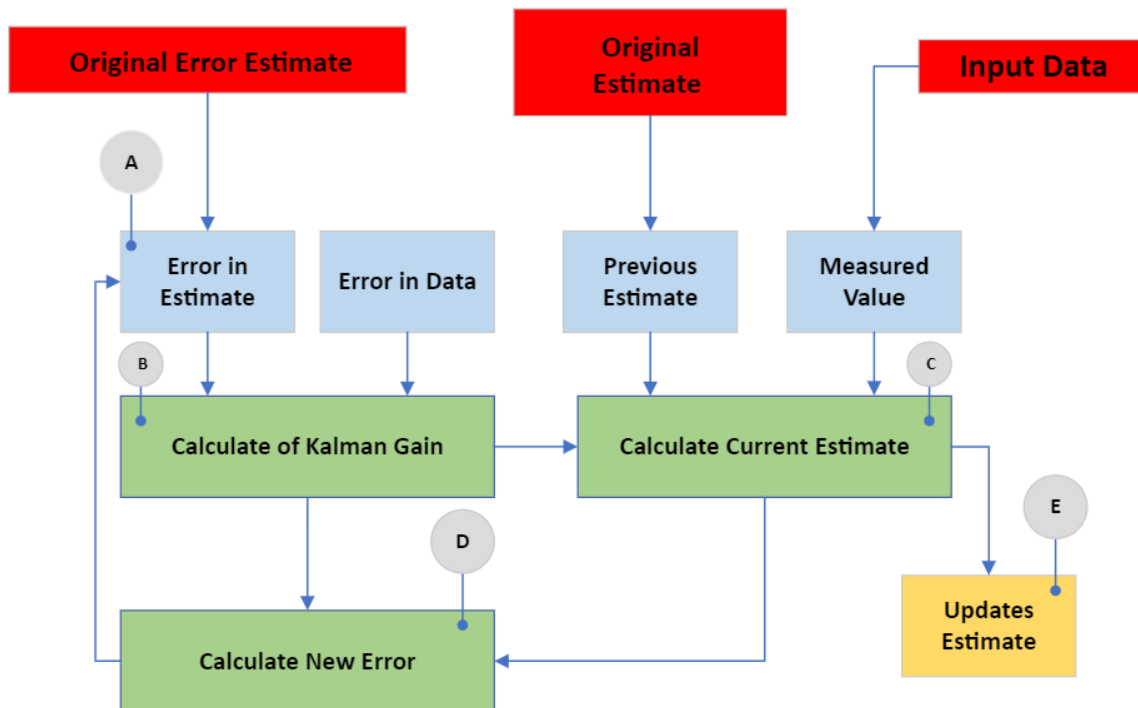


Figure 1. Flowchart for Single Variable Kalman Filter

As shown on the flowchart there are three main actions that must happen, (1) Calculate the Kalman Gain which happens in each iteration of the process, (2) the current estimate must be calculated based on the Previous Estimate, Measured Value, and Kalman Gain. (3) The New error will be calculated and fed back to the System to update the Error in the estimate. As noted by the balloon footnote. **Balloon A**, the error in the estimate must be known for the system. **Balloon B**, the Kalman gain is calculated based on the error in the estimate and error in the data. The Kalman gain will give us a relative importance in the error in the estimate versus the error in the data. It tells us which error to trust more when calculating the current Estimate. **Balloon C**, the current estimate is calculated based on the measured and previous value. A weight is added to either are based on the coefficient of the Kalman gain giving us are updated estimate for **Balloon E**. The new error is calculated based on the Kalman Gain and current estimate.

### Calculating the Kalman Gain, Current Estimate, Error Estimate

The Kalman gain calculation is shown below:

$$Kalman\ Gain = \frac{E_{Est}}{E_{Est} + E_{Mea}}$$

Where  $E_{Est}$  is the Error in Estimate and  $E_{Mea}$  is the Error in Measurement. The Kalman Gain can range from 0 to 1. If the Kalman Gain is close to 0 this indicates that the Error in the Measurement are much larger than the error in the estimate and Measurements are inaccurate. If the Kalman Gain is close to 1 this means that there is no errors in measurements. Later, when calculating the current estimate, we will see how this indicates the stability of the estimate.

The current estimate is calculated by this simple equation:

$$EST_t = EST_{t-1} + KalmanGain[MEA - EST_{t-1}]$$

If the Kalman Gain is large then the difference between the measured value and estimated will affect the new estimate value more, indicating that the system has high uncertainty making it unstable. The inverse is also true, a small gain factor indicates that the estimated value will have a greater affect in our system and or system will be stable.

Lastly the estimated error denoted as  $E_{EST_t}$  is calculated by using this equation:

$$E_{EST_t} = [1 - KalmanGain](E_{EST_{t-1}})$$

If the Kalman Gain is small, then the error in the measurement is large and will change the error in estimate quickly and thus move slower to the true value. However, a larger Kalman gain will move to the true value much quicker with less data points. Regardless of speed this system will always have some type of uncertainty associated with it.

### Single Valued Example

An example is shown below of Kalman filter applied to a temperature control system. The true temperature value of the room was found to be 72 Celsius by using the most accurate thermometer possible. We are measuring the temperature of the room via a thermistor and MCU via an ADC channel.

Due to ADC converter Error in the Measurement will be found because of quantization error that occurs when converted an analog signal to a finite bit size. This Error in the measurement is found to be 4 and is a constant error. Initial Estimates, Measurements, and Error in Estimate are given. Sample Calculations of the Kalman Gain, Error in Estimate and Estimate are shown below:

$$Kalman\ Gain_{t=0} = \frac{E_{Est_{t=0}}}{E_{Est_{t=0}} + E_{Mea_{t=0}}} = \frac{2}{2 + 4} = \frac{1}{3}$$

$$EST_{t=0} = EST_{t-1} + KalmanGain_t[MEA_t - EST_{t-1}] = 68 + \frac{1}{3}[75 - 68] = 70.33$$

$$E_{EST_{t=0}} = [1 - KalmanGain](E_{EST_{t-1}}) = \left[1 - \frac{1}{3}\right](2) = \frac{4}{3}$$

A plot of the measured vs the Estimate Temperature values of the room is shown by Figure 3. Visually the Estimate value is following a somewhat linear curve towards the true value of 72. The estimate value will fluctuate between the true value for each measurement with less uncertainty when compared to the measured values.

True Temperature Celsius		Initial Estimate	Initial Error in Estimate	Initial MEA	Error in MEA
72		68	2	75	4
Time	Measurement	Error in Measurement	Kalman Gain	Error in Estimate	Estimate
-1	0	0	0.00	2.00	68.00
0	75	4	0.33	1.33	70.33
1	73	4	0.25	1.00	71.00
2	71	4	0.20	0.80	71.00
3	75	4	0.17	0.67	71.67
4	73	4	0.14	0.57	71.86

Figure 2. Calculation for Kalman Filter Example

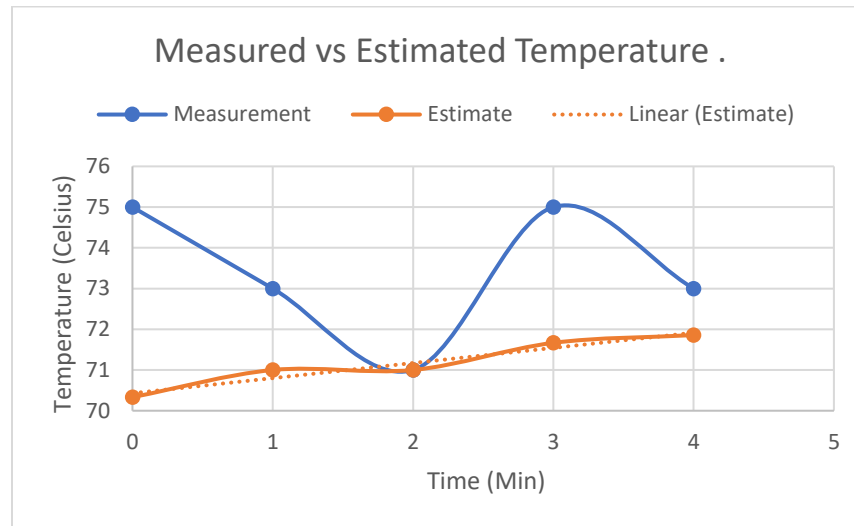


Figure 3. Plot of Measured Temperature vs Estimated Kalman Filter Temp

The difference between the measurements (blue samples) and the true value, 72 are measurement errors. Since these are random these can be described as variance: ( $\sigma^2$ ). This error can be found from the ADC converters themselves or the code that derived from the code itself. Regardless of the specific reason this is an example of measurement uncertainty (Measurement Error).

Likewise, the difference between the estimates (Orange Value) and the true value is the estimate error. This error becomes smaller and smaller as we make additional measurements, and it converges towards zero, while the estimated values converge towards the true value.

### Adding Noise

To finish the one-dimensional Kalman Filter, process noise variable must be added to the Estimated Error equation also called the Covariance Update Equation. In the real world, uncertainties in a system model no matter how closed a system is. For example, when wanting to estimate the resistance value of a resistor, we assume that the resistance doesn't change between the measurements. However, this isn't always the case as its resistance changes due to the fluctuation of the temperature around it.

The uncertainty of a model is called the process noise which produces estimation errors. How does this affect our example if temperature fluctuations.

$$E_{EST_t} = C + W_n$$

Where  $W_n$  is the random process noise with variance  $q$  and  $C$  is the constant Estimated error.

### Adding Noise Example

Let's say we would like to estimate the temperature of tank with water instead of a room via our MCU. We will assume that the Tank is in steady state and that the temperature of the water is constant. However, some fluctuations in the true water temperature are possible due to the external environment. Assume that the true temperature of the water is 35 degrees Celsius with a process noise

variance of 0.1. The error in the measurement (Standard deviation) is 3.7 degree Celsius. Our MCU samples every second. Data was collected for a period and the true liquid temperature values were collected from a Calibrated State of the Art thermistor and our MCU. The figure below compares the true liquid temperatures, and the measurements from our MCU.

$$\text{Initial } E_{MEA} = \sigma^2 = 3.7^2 = 13.69$$

$$W_n = 0.1$$

Time (Seconds)	TRUE Value	Measurement
0	34.9	31
1	35.1	39
2	34.85	32
3	35.01	37
4	34.99	34
5	34.5	36
6	35.3	35
7	35.2	31
8	35.6	39
9	34.6	38
10	34.1	33

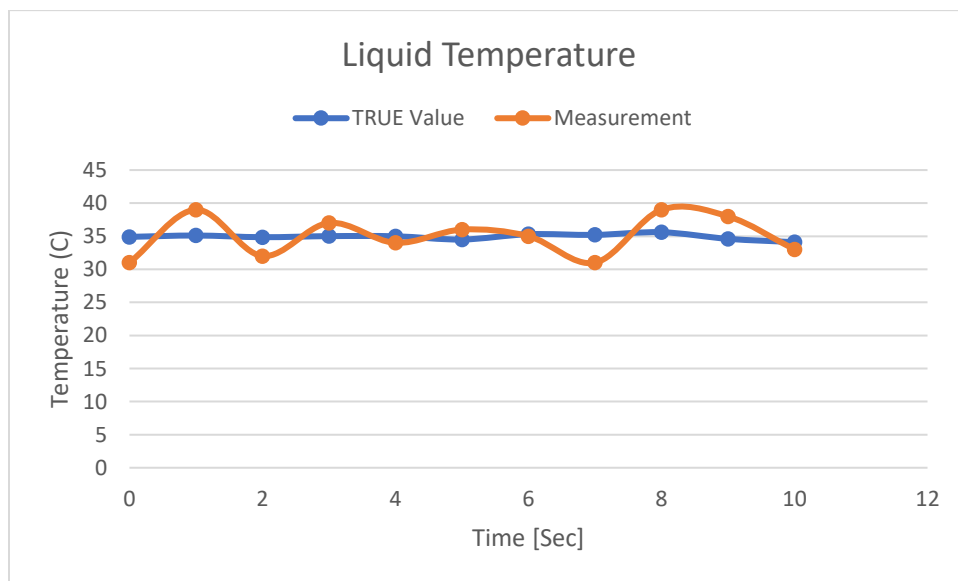


Figure 4. Scatter Plot comparison of true water temp and the Measurements from MCU



Let's say we do not know the true temperature of the liquid tank and based of experience our guess is that the water temperature is around 90 Celsius (Really HOT).

$$Initial\ EST_t = 90$$

Our guess is just a guess and there imprecise, so our system will have a high initial estimated error, the standard deviation will be 100. Since the Estimated Uncertainty is the error's variance we can find as:

$$Initial\ Non - Noisy\ E_{EST} = \sigma^2 = 100^2 = 10,000$$

$$Initial\ Noisy\ E_{EST_t} = E_{est} + W_n = 10,000 + 0.1 = 10,000.1$$

The variance is very high therefore this Kalman Filter will converge faster if we initialize it with a more meaningful value.

Now that we know all the variables needed for the Kalman Filter flowchart lets calculate the first Iteration of the system. First the Kalman Fain will be calculated as below:

$$Kalman\ Gain_{t=1} = \frac{E_{Est}}{E_{Est} + E_{Mea}} = \frac{10,000.1}{10,000.1 + 13.69} = 0.998632$$

Notice that for the iteration we are using the initial Error in the estimate with Noisy value and the initial error in the measurement. The Kalman Gain is almost 1, indicating that our estimate error is much bigger than the measurement error. Thus, more weight will be put on the measurements then our estimates.

Estimating the current state is as the following:

$$EST_{t=1} = EST_{t-1} + KalmanGain_t[MEA_t - EST_{t-1}] = 90 + 0.998632[31 - 90] = 31.080$$

Finally, we will update or Error in the Estimate by using the following equation.

$$Noisy\ E_{EST_{t=1}} = [1 - KalmanGain](E_{EST_{t-1}}) + W_n = [1 - 0.998632](10,000.1) + 0.1 = 13.780$$

The table below shows the Error in the Measurement, Error in the Estimate, Kalman Gain, and Measured Values for 10 samples. A Graph was done to compare the true values, measured values, and estimates. Based on the graph we can see that the estimated value converges towards the true value.

Noisy Coefficient		Initial Estimate		Initial Error in Estimate	Initial MEA	Error in MEA
0.1		90		10,000.10	31	13.69
Time	Measurement	True Value	Error in Measurement	Kalman Gain	Error in Estimate	Estimate
-1					10000.10	90.00
0	31	34.9	13.69	0.9986	13.77	31.08
1	39	35.1	13.69	0.5015	6.97	35.05
2	32	34.85	13.69	0.3372	4.72	34.02

3	37	35.01	13.69	0.2562	3.61	34.79
4	34	34.99	13.69	0.2086	2.96	34.62
5	36	34.5	13.69	0.1776	2.53	34.87
6	35	35.3	13.69	0.1560	2.24	34.89
7	31	35.2	13.69	0.1404	2.02	34.34
8	39	35.6	13.69	0.1287	1.86	34.94
9	38	34.6	13.69	0.1197	1.74	35.31
10	33	34.1	13.69	0.1127	1.64	35.05



Figure 5. Kalman Filter result of True Value, Estimate, and Measured value

The Estimated Uncertainty also decays but looks like it will deviate around 1.64 which is the variance of the estimated values. The estimate error standard deviation is found to be 1.28 Celsius. Thus, we can say that the liquid temperature estimate is around  $Estimate \pm \sqrt{E_{Estimate}} = 35.05 \pm 1.28 \text{ Celsius}$

### Multi-Variable Kalman Filter

Many Times, systems will have more the just one process sometimes two, three, or even more dimensions. For instance, a Kalman filter that is attempting to derive a plan position and velocity in x,y,z direction will have 6 processes. It is common practice to describe multidimensional process via state space representation. But firstly, some review on State Space Representation, Matrix Operations, and Key Statistical Background is needed.



## State Space Representation

The basic principle of the Kalman filter is the use of a data set that tracks a sample over time of observable variables to reconstitute the value of the non-observable variables. This is often done in state-space format. In control engineering, a state-space representation is a mathematical model of a system as a set of inputs, outputs, and state variables. The state variables are variables that change based on the value given at that moment. The output variables values depend on the values of the state variables. When a system is a linear Time-Invariant system (LTI) the state space model can be used, by the first and second equations.

$$\dot{X} = AX + BU$$

$$Y = CX + DU$$

Where:

- $X, \dot{X}$  are the state vectors and the differential state vector respectively.
- $Y$  are input vectors and output vectors respectively.
- $A$  is a system matrix,  $D$  is the Feed-Forward matrix.
- $B$  and  $C$  are input and output matrices.

## Matrix Operations

### Addition and Multiplication

Adding Matrices is simply and only requiring one rule, the two matrices must have an equal value of rows and columns. Example is shown below:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} j & k \\ l & m \end{bmatrix} = \begin{bmatrix} a+j & b+k \\ c+l & d+m \end{bmatrix}$$

To multiply a matrix by another matrix we need to do the dot product of rows and columns. The number of columns of the 1<sup>st</sup> matrix must be equal to the number of rows of the 2<sup>nd</sup> matrix. Let's say we have a matrix of 1 row with 3 columns:  $A = 1 \times 3$  and a second matrix named  $B$  with  $3 \times 2$ . Multiplication is possible since the Columns of Matrix  $A$  is equal to the Rows of Matrix  $B$

$$\begin{bmatrix} a & b & c \end{bmatrix} + \begin{bmatrix} d & e \\ f & g \\ h & i \end{bmatrix} = \begin{bmatrix} a(d) + b(f) + c(h) & a(e) + b(g) + c(i) \end{bmatrix}$$

### Transpose and Inverses

The Transpose of a matrix is simply a flipped version of the original matrix. We can transpose a matrix by switching its rows with its columns. This is usually denoted as  $A^T$  where  $A$  is a matrix. For example:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$$

The inverse of a Matrix is when a matrix can be multiplied by another matrix to get its Identity Matrix. The identity matrix is equivalent to the number "1" on a number line. Identity matrix are often Denoted as  $I$  and Inverse Matrices as  $A^{-1}$ . Not all matrices have an identity matrix but overall if we have a square matrix which means that the number of rows is equal to the number columns a inverse exist if and only if the determinant is equal not equal to zero.

$$AA^{-1} = I$$

$$I = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix}$$

Example:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \text{ and } A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

### Symmetric Matrices

A matrix is symmetric if the Transpose Matrix is equal to the Original Matrix. Because equal matrices have equal dimensions, only square matrices can be symmetric. These matrices are symmetric with respect to the main diagonal. So, for every  $i$ th row and  $j$ th column. The index is equal.

$$A \text{ is symmetric if and only if } A = A^T$$

$$\text{For every Index in } A \text{ named as 'a'. } a_{ij} = a_{ji}$$

Example:

$$A = \begin{bmatrix} 1 & 7 & 3 \\ 7 & 4 & 5 \\ 3 & 5 & 0 \end{bmatrix}$$

### Statistical Background

#### Random Variable and Expectation

Hopefully you know what a random variable already is but if a quick refresher is needed that is alright. A random variable is a function that assigns values to each sample space's outcomes onto the real value line from 0 to 1. The random variable is described by the probability density function which is characterization of **moments**. **The first raw moment of a random variable is the expectation which is the mean of the sequence of measurements. The second central moment is the variance of the sequence of measurements.**

The expectation of a random variable is simply the mean of the random variable.

$$E(X) = \mu_X$$

Where  $\mu_X$  is the mean of the random variable.

Here are some basic rules of Expectations where  $X$  &  $Y$  is a random variable and  $a, b$  is a constant:

- $E(X) = \mu_X = \sum xp(x)$
- $E(a) = a$
- $E(aX) = aE(X)$
- $E(a \pm bX) = a \pm bE(X)$
- $E(X \pm Y) = E(X) \pm bE(Y)$
- $E(XY) = E(X)E(Y)$  (If X and Y are independent)

Variance measures the variability of certain sections in the data based on its mean ie, the average of each point from the mean. Standard deviation measures the square root of the variance. Variance can be calculated as:

$$Var = \frac{\sum (X_i - \bar{X})^2}{n}$$

Where N is the number of observations and  $\bar{X}$  is the mean value of all observations. Common rules for variance are **(Where a is constant and X is a random variable)** :

- $Var(a) = 0$
- $Var(a \pm X) = Var(X)$
- $Var(X) = E(X^2) - \mu_X^2$

#### *Covariance of Two Random Variables*

The definition of covariance is as the following, assuming you have two random variable X and Y their covariance can be found as:

$$Cov(X, Y) = E[(X - \mu_X)(Y - \mu_Y)]$$

If the  $X = Y$  then the covariance simplifies to the variance.

$$Cov(X, Y) = E[(X - \mu_X)(X - \mu_X)] = E(X^2) - \mu_X^2 = Var[X] \text{ if } X = Y$$

Lastly if two random variables are independent then there uncorrelated. However, if two random variables are found to be uncorrelated then this does not imply independence.

But what does this mean? Based on the equation we can induce that covariance is a measure of the relationship between two random variables. Positive covariance indicates that the two variables tend to move in the same direction. While Negative covariance reveals that two variables tend to move inverse directions. This is very different from Correlation since covariance since covariance can only gauge the direction not strength of the relationship. A simpler equation of Covariance can be expressed:

$$Cov(X, Y) = E(XY) - E(X)E(Y)$$

In finance, the concept is primarily used in portfolio theory most used in diversification method. Using Covariance between assets in a portfolio, choosing assets that don't have high positive covariance with each other is a key known way to eliminate any unsystematic risk.

### Covariance and Variance Extended

Expectations in multiply dimensions is affect defined as such:

Let  $X = [X_1, X_2, \dots, X_n]^T$  to be a random vector. The expectation is :

$$\vec{\mu} = E[X] = \begin{bmatrix} E[X_1] \\ \vdots \\ E[X_n] \end{bmatrix}$$

The resulting vector is the mean vector. And are the individual elements for each random variable. This is the same as computing their marginal PDF's which is challenging to do by hand but easily done in computers. Marginal PDF's is the pdf result when we integrate for one random variable. Mathematically this is expressed as:

$$f_x(x) = \int_{\Omega_y}^{\infty} f_{x,y}(x, y) dy \quad \& \quad f_y(y) = \int_{\Omega_x}^{\infty} f_{x,y}(x, y) dx$$

This is two the extent in which we will talk about variance matrices. But more information can be found online or in an undergraduate book of probability theory.

The covariance matrix of a random vector called  $X$  can be found as such.

$$Cov(X) = \begin{bmatrix} Var[X_1] & \cdots & Cov(X_1, X_n) \\ \vdots & \ddots & \vdots \\ Cov(X_n, X_1) & \cdots & Var[X_n] \end{bmatrix}$$

A more compact way of writing this is shown below:

$$Cov(X) = E[(X - \vec{\mu})(X - \vec{\mu})^T]$$

Where  $\vec{\mu} = E[X]$  is the mean vector.

An example is shown below that deal with length, width, and Height:

The data matrix  $X$  is shown below:

$$\begin{matrix} & L & W & H \\ \begin{bmatrix} 4.0 & 2.0 & 0.60 \\ 4.2 & 2.1 & 0.59 \\ 3.9 & 2.0 & 0.56 \\ 4.3 & 2.1 & 0.62 \\ 4.1 & 2.3 & 0.63 \end{bmatrix} \end{matrix}$$

The mean vector for Matrix A is:  $\vec{\mu} = E[\mathbf{X}] = [4.1 \quad 2.1 \quad 0.6]$ . Variance for all three random variables will need to be found next. Arithmetic is shown below:

$$Var_L = \frac{\sum (l_i - \bar{l})^2}{n} = \frac{1}{5} [(4.0 - 4.1)^2 + \dots + (4.1 - 4.1)^2] = 0.02$$

$$Var_W = \frac{\sum (W - \bar{w})^2}{n} = \frac{1}{5} [(2.0 - 2.1)^2 + \dots + (2.3 - 2.1)^2] = 0.012$$

$$Var_H = \frac{\sum (H - \bar{H})^2}{n} = \frac{1}{5} [(0.60 - 0.60)^2 + \dots + (0.63 - 0.60)^2] = 0.0006$$

Lastly Covariance will have to found for the difference between two random variables. Calculations are shown below.

$$Cov(W, L) = E(WL) - E(W)E(L) = \frac{\sum (W_i - \bar{W})^2 \sum (L_i - \bar{L})^2}{n} = 0.006$$

$$Cov(W, H) = E(WH) - E(W)E(H) = \frac{\sum (W_i - \bar{W})^2 \sum (H_i - \bar{H})^2}{n} = 0.002$$

$$Cov(L, H) = E(LH) - E(L)E(H) = \frac{\sum (L_i - \bar{L})^2 \sum (H_i - \bar{H})^2}{n} = 0.0026$$

$$Cov(\mathbf{X}) = \begin{bmatrix} Var[L] & Cov(L, W) & Cov(L, H) \\ Cov(W, L) & Var[W] & Cov(W, H) \\ Cov(H, L) & Cov(H, W) & Var[H] \end{bmatrix} = \begin{bmatrix} 0.02 & 0.006 & 0.0026 \\ 0.006 & 0.012 & 0.002 \\ 0.0026 & 0.002 & 0.0006 \end{bmatrix}$$

## Multi-Dimensional Kalman Filter Flow Chart and Discussion

A Flowchart of the Kalman Filter Multi-Dimensional Model is shown below:

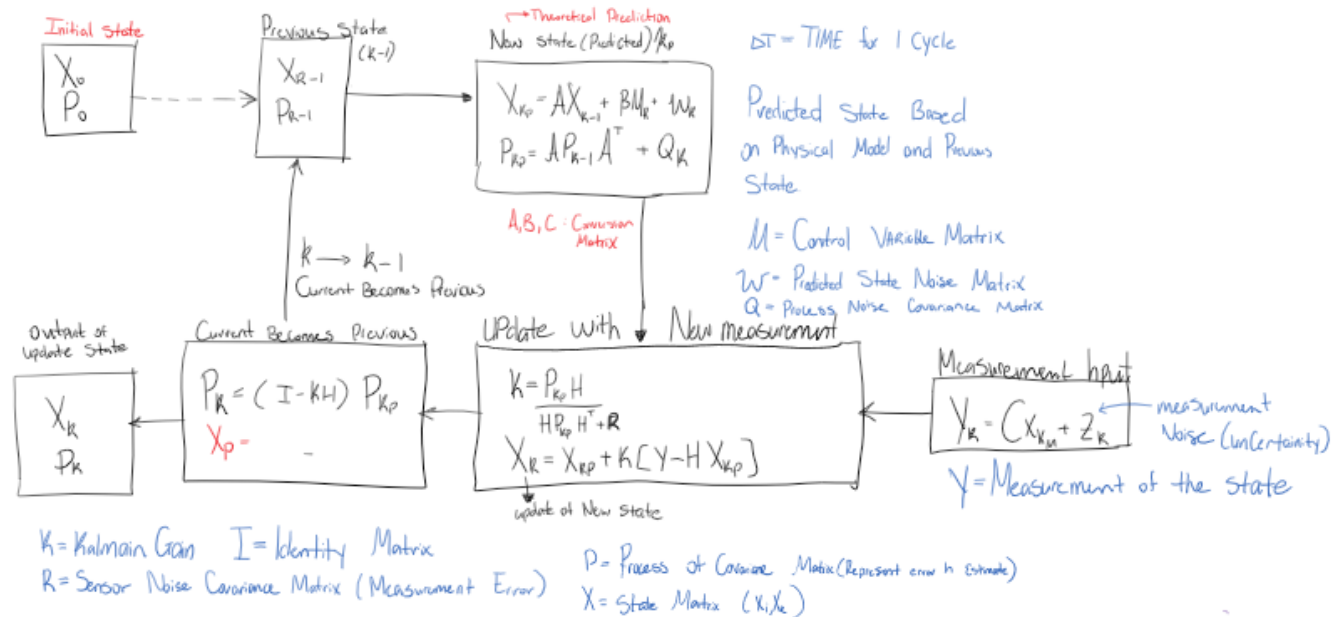


Figure 6. Multi-Dimensional Kalman Filter Flowchart

As always, we will have an initial state that has a state matrix  $X_0$  and a process covariance matrix  $P_0$ . The state matrix typical can contain position, velocity, or acceleration one can think of the covariance matrix as the set of errors in the estimate but for multiple dimensions. Next the Previous state is found which will be used to predict the new state. This is a theoretical estimate that is based on the previous state, control variable matrix, and predicted noise in our system. Finally based on the predicted new state and Measurements inputs we will find the Kalman Gain and updated of the new state is created. Error in the process is conducted and fed back into the system before doing an output of the updated state. **If this doesn't make sense don't worry, we will break it down further and do an example for further explanation.**

### New State Predicted Model:

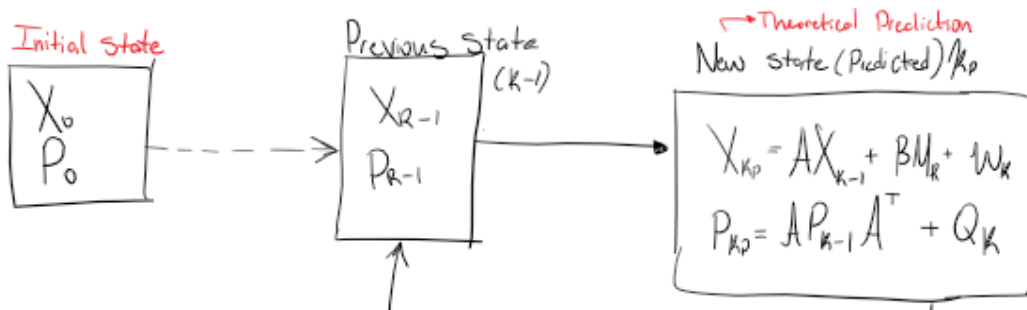


Figure 7. New State Flow for Part1 of the Kalman Filter

Using the State extrapolation equation denoted as  $X_{kp}$  the general form of the state extrapolation equation in matrix notation is:

$$X_{kp} = AX_{k-1} + Bu_k + w_k$$

Where:

- $X_{kp}$  is the predicted system state vector at new time interval.
- $X_{k-1}$  is the estimated system state vector of the Previous state
- $u_k$  is the control input variable which is a measurable input of the system.
- $w_k$  models noise or disturbances in the system
- A and B are known as the Transition matrix.

The State Transition Matrix A and System state vector X is typically modeled as such for a 2-dimensional problem:

$$AX = \begin{bmatrix} 1 & 0 & \Delta T & 0 \\ 0 & 1 & 0 & \Delta T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} x + \Delta T \dot{x} \\ y + \Delta T \dot{y} \\ \dot{x} \\ \dot{y} \end{bmatrix}$$

The Control system section is broken down into the Control Input variable  $u_k$  multiplied by the control transition matrix B. An example is shown below for a 2-dimensional example for an object moving down in the X and Y direction of an object.

$$Bu_k = \begin{bmatrix} \frac{1}{2}\Delta T^2 & 0 \\ 0 & \frac{1}{2}\Delta T^2 \\ \Delta T & 0 \\ 0 & \Delta T \end{bmatrix} \begin{bmatrix} a_x \\ a_y \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\Delta T^2 a_x \\ \frac{1}{2}\Delta T^2 a_y \\ \Delta T a_x \\ \Delta T a_y \end{bmatrix}$$

This is where acceleration in the x and y direction are being modeled. And if we ignore Noise or disturbances in the system, we can see that the predicted state of the matrix for this example matches to the kinematic equation for a project motion.

$$X_{kp} = AX_{k-1} + Bu_k = \begin{bmatrix} x + \Delta T \dot{x} \\ y + \Delta T \dot{y} \\ \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta T^2 a_x \\ \frac{1}{2}\Delta T^2 a_y \\ \Delta T a_x \\ \Delta T a_y \end{bmatrix} = \begin{bmatrix} x + \Delta T \dot{x} + \frac{1}{2}\Delta T^2 a_x \\ y + \Delta T \dot{y} + \frac{1}{2}\Delta T^2 a_y \\ \dot{x} + \Delta T a_x \\ \dot{y} + \Delta T a_y \end{bmatrix}$$

The Process covariance matrix of the current state  $P_{kp}$  (Error in the estimate) is modeled as such:

$$P_{kp} = AP_{k-1}A^T + Q_k$$

Where:

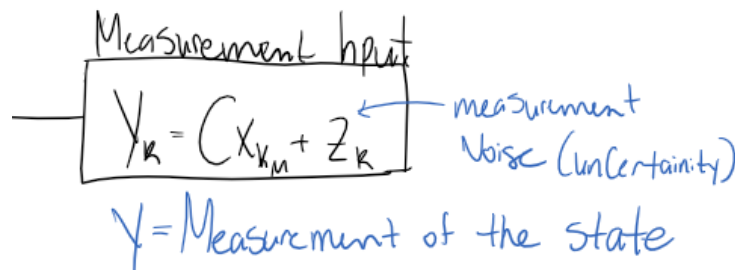
- A is the State transition matrix
- Q is the Process noise matrix (**Keeps the State Covariance Matrix from becoming too small or going to 0**)
- $P_{k-1}$  is the uncertainty of a predication of the previous state.

In the multidimensional Kalman Filter, the process of noise is a covariance matrix denoted by Q. Recall that is modeled as:

$$Cov(X) = \begin{bmatrix} Var[X_1] & \cdots & Cov(X_1, X_n) \\ \vdots & \ddots & \vdots \\ Cov(X_n, X_1) & \cdots & Var[X_n] \end{bmatrix}$$

Typically, A, and predication of the previous state will be different based on the Application of the Kalman filter and whether the system is being modeled in a discrete time or continuous time. More information can be found online but an example would work better with discussing on this topic.

*Measurement Input:*



$$y_k = Cx_k + z_k$$

Measurement input

measurement Noise (uncertainty)

$y = \text{Measurement of the state}$

Figure 8. Measurement Input State

The measurement equation represents a true system state in addition to the random measurements noise found caused by the measuring device. This measurement noise variance can be constant for each measurement, or it can be different for each measurement



Updated of New Measurement and Kalman Gain:

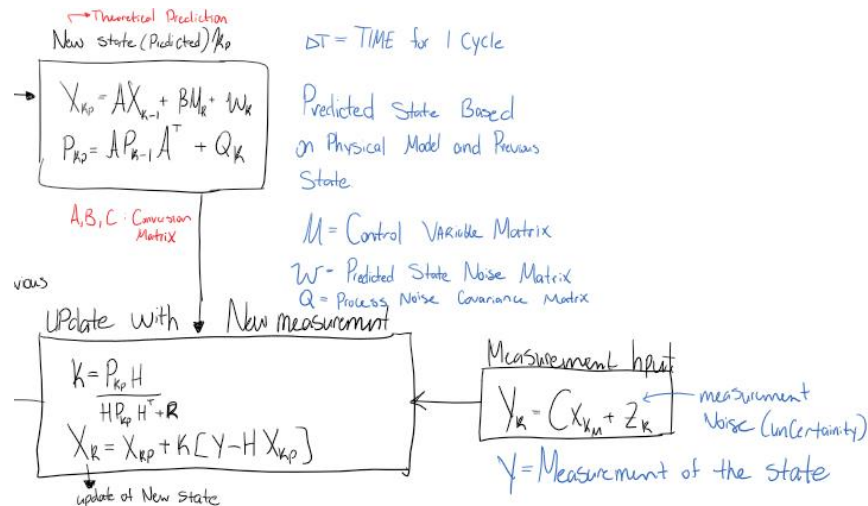


Figure 9. Updating State and Covariance Flowchart of Kalman Filter

The Kalman Gain equation and updated new state matrix is provided below:

$$K = \frac{(P_{kp}H^T)}{HP_{kp}H^T + R}$$

$$X_k = X_{kp} + K[Y - HX_{kp}]$$

Where:

- K is the Kalman gain
- $P_{kp}$  is the previous predicted covariance vector
- H is the observation matrix
- Y is the measurement state
- R is the Noise Covariance matrix

Not much can be said about the update state matrix other than that conducts the same thing as a one-dimensional Kalman filter but rather in more dimensions. Again, we the Kalman gain calculates the amount of weight that the new state will have when compared to the predicted state to the measurement values. The main difference with multi-dimensional Kalman filter to a one-dimensional filter is the possible variation between two values can exist and how to account for this.

*Calculating the new Process Covariance in the Measurement:*

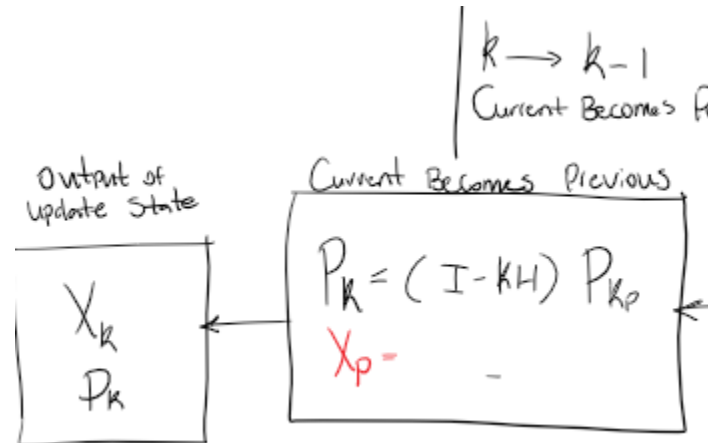


Figure 10. Ending stage of the Kalman Filter

In many textbooks, the simplified form of the covariance update equation is modeled as above.

$$P_k = (I - KH)P_{kp}$$

Where:

- K is the Kalman Gain
- H is the observation transition matrix
- I is the identity matrix
- P\_KP is the Predicted Processed Covariance vector

Notice that both new updated state and updated Predicted Covariance get fed back into the system for the next iteration. This is the beauty of Kalman filtering in which it is an iterative process with a closed loop system.

## Using Kalman Filters in Software

### Python 2-Dimensional: Position and Velocity Predictor

In this example, a Kalman Filter has been designed to find the position and velocity of an object moving in one dimension. We will be measuring this device every sampling point Sampling and plotting on a real-time graph. As we know in any real system noise will exist with our measurement and the estimation itself.

Five variables will affect our Kalman filter prediction, these are the initial estimated position, estimated velocity, sampling period, derivative of time, error(variance) in the measurement, and error(variance) in the acceleration.

$$X_{kp} = AX_{k-1}$$

$$P_{kp} = AP_{k-1}A^T + Q_k$$

Where:

- $X_{kp}$  is the predicted system state vector at new time interval.
- $X_{k-1}$  is the estimated system state vector of the Previous state
- A and B are known as the Transition matrix.
- P is Predicted Processed covariance state
- $Q_k$  is Processed covariance noise.

Notice that the for the predicted state there is no control matrices or noise in the modeling state. In terms of state space representation our Predicted system state and predicted processed covariance matrices are:

$$X_{kp} = AX_{k-1}$$

$$X_{kp} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_k \\ V_k \end{bmatrix}$$

$$P_{kp} = AP_{k-1}A^T + Q_k$$

$$P_{kp} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_k \\ P_k \end{bmatrix} \begin{bmatrix} 1 & 0 \\ dt & 1 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}dt^2 \\ dt \end{bmatrix}$$

In terms of code this is accomplished by using the python library numpy. The function **np.eye( argsws)** generates an identity matrix and the **np.dot()** function multiplies the rows and columns of the matrix together. The function **predict** for our Kalman filter object conducts the calculations. Code is shown below:

```
def predict(self, dt: float) -> None:
    # x = A x_kp
    # P = A P_kp A^T + Q_K

    """
    'A' will be our state transition matrix and it simply
    a conversion matrix. We set the matrix oh to have data
    that matches the sampling time. Lastly the dot product is
    done between A and our State Matrix to get the Predicted
    New State. Here is where we can Predicted noise if wanted or
    needed and our control variable with its control transition matrix
    """
    A = np.eye(NUMVARS)
    A[iX, iV] = dt
```

```

new_x = A.dot(self._x)

"""
'Q_K' will be our Process Noise Covariance Matrix which is a size
of 2 x 1 matrix for this example.
Q_K = | 0 |
      | 0 |
We think initialize the Process noise with 1/2 of the rate of change in
time squared
and the delT for velocity. Lastly we calculated the predicted covariance
matrix
by the simple equation below.
new_P = A.dot(self._P).dot(A.T) + Q_K.dot(Q_K.T) *
self._accel_variance
"""
Q_K = np.zeros((2, 1))
Q_K[iX] = 0.5 * dt**2
Q_K[iV] = dt
new_P = A.dot(self._P).dot(A.T) + Q_K.dot(Q_K.T) * self._accel_variance

"""
We update our state or state vector and processed covariance matrix with
the new predicted state
"""
self._P = new_P
self._x = new_x

```

Next, the partial is measured based on the theoretical equation of the position plus the random variable multiplied but its standard deviation of the measured variance.

$$X_{tk} = X_{tk} + dt * V_{Tk}$$

$$Meas_{X_k} = X_{tk} + N(\mu, \sigma) \sqrt{E_{Meas}}$$

Where:

- $X_{tk}$  is the theoretical position of the object
- $V_{tk}$  is the theoretical velocity of the object and  $dt$  is the rate of change of time
- $N(\mu, \sigma)$  is a random variable that is gaussian

In terms of cod this is done like this:

```

real_x = real_x + ddt * real_v
measured_x = real_x + np.random.randn() * np.sqrt(mea_var)

```

Lastly the Kalman filter gets updated once every sampling period as shown by the code below:

```

if step != 0 and step % MEAS_EVERY_STEPS == 0:
    kf.update(meas_value=measured_x,

```

```
meas_variance=mea_var)

measured_xs.append(measured_x)
real_xs.append(real_x)
real_vs.append(real_v)
```

In the updated state the Kalman gain will get calculated which is based on the generic equation of the Kalman gain for multi-dimensional matrix. As shown below, based on the Kalman gain value the next predicted position and velocity will give more weight to the either the measured value or the theoretical estimated value. As time goes by the Kalman gain value will decrease and more will be put on the estimated values rather than the measured ones. Likewise, the processed covariance matrix is also calculated and fed back to the system.

$$K = \frac{(P_{kp}H^T)}{HP_{kp}H^T + R}$$

$$X_k = X_{kp} + K[Y - HX_{kp}]$$

$$P_k = (I - KH)P_{kp}$$

Where:

- K is the Kalman gain
- $P_{kp}$  is the previous predicted covariance vector
- $X_{kp}$  is the previous predicted state vector
- H is the observation matrix
- Y is the measurement state
- R is the Noise Covariance matrix

```
def update(self, meas_value: float, meas_variance: float):
    # y = H - Z_k
    # K = P_kp H / (H P Ht + R)
    # x = x + K(y - HX_kp)
    # P = (I - K H) * P_kp

    """
    In this function the measurement Noise matrix is being
    created with the number of rows of the array being
    the index of the filter.
    """

    Z_k = np.zeros((1, NUMVARS))
    Z_k[0, iX] = 1

    """
    Measured values are adding into the measurement matrix
    and finally the final Measurement vector was created
    """

    X_km = np.array([meas_value])
    y = X_km - Z_k.dot(self._x)
    self._Measured = y
```

```

"""
Next the Kalman Gain will be calculated in this project
which is typically modeled as:

$$K = P_{kp} H / (H P_{kp} H^T + R)$$


First the Sensor Noise Covariance Matrix or
Measurement Error which is denoted as R.

The Bottom half of the Kalman Gain is then calculated
based on the error measurement and process of covariance
matrix. '_P' and the Observation matrix is also shown.
Finally the Kalman Gain is called as shown below.
"""

R = np.array([meas_variance])
S = Z_k.dot(self._P).dot(Z_k.T) + R
K = self._P.dot(Z_k.T).dot(np.linalg.inv(S))

"""
Next the new-state vector and process covariance matrix
are Calculated
"""

new_x = self._x + K.dot(y)

new_P = (np.eye(2) - K.dot(Z_k)).dot(self._P)

self._P = new_P
self._x = new_x

```

Overall, a Python GUI was designed with 6 sliders and two graph plotting position and velocity over time. Figure is shown below. Users can then change specific parameters of the system to view how that affects the prediction of the Kalman filter. A legend is provided, and the two python files needed to create this script.

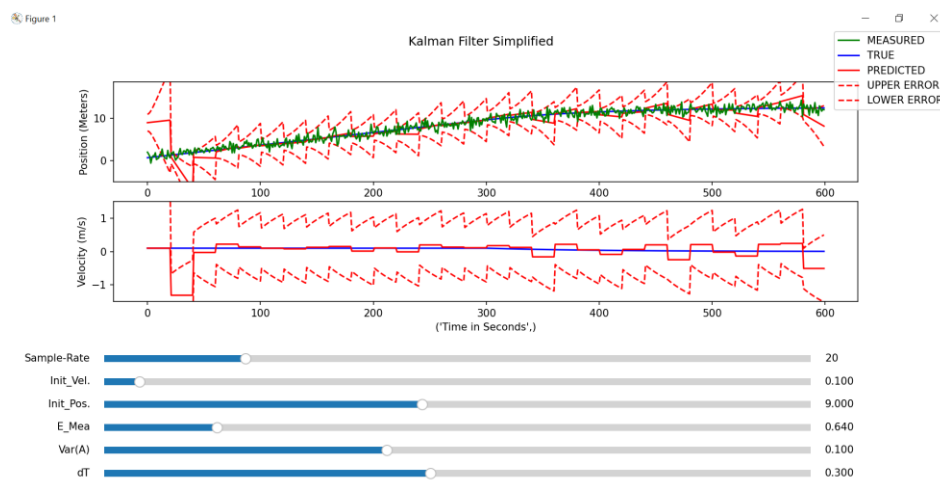


Figure 11. Start of the Kalman Filter App



## Kalman Filter Code

```
import numpy as np

# offsets of each variable in the state vector
iX = 0
iV = 1
NUMVARS = iV + 1

"""
Creating a class named KF for Kalman Filter that initializes itself
three arguments initial position, velocity, and the acceleration variance.
"""
class KF:
    def __init__(self, initial_x: float,
                  initial_v: float,
                  accel_variance: float) -> None:

        """
        mean of state GRV
        This outputs a new array given the shape of the new array.
        So for example if NUMVARS is 5 then X will be
        X = [0,0,0,0,0]
        This will serve as out state matrix
        """
        self._x = np.zeros(NUMVARS)

        """
        two class variables that are arrays have been created that have
        the Initial Values. Lastly an accelerate variance has also been
        added from our initial acceleration variance.
        """

        self._x[iX] = initial_x
        self._x[iV] = initial_v
        self._accel_variance = accel_variance

        self._Measured = 0;
        """
        This P will serve as the Initial Process of Covariance Matrix
        """
        self._P = np.eye(NUMVARS)

        """
        This function will run the predicted State Matrix value and
        Predicted Process of Covariance matrix. The general equations are:
        X_kp = AX_kp-1 + BU_k + W_k
        P_kp = AP_kp-1A^T + Q_K

        Notice that in our Predicted State Matrix we do not have a control
        variable
        or a predicted noise of the state for simplicity.
        """
```

```

"""
def predict(self, dt: float) -> None:
    # x = A x_kp
    # P = A P_kp A^T + Q_K

    """
    'A' will be our state transition matrix and it simply
    a conversion matrix. We set the matrix oh to have data
    that matches the sampling time. Lastly the dot product is
    done between A and our State Matrix to get the Predicted
    New State. Here is where we can Predicted noise if wanted or
    needed and also our control variable with its control transition
matrix
    """
    A = np.eye(NUMVARS)
    A[iX, iV] = dt
    new_x = A.dot(self._x)

    """
    'Q_K' will be our Process Noise Covariance Matrix which is a size
    of 2 x 1 matrix for this example.
    Q_K = | 0 |
          | 0 |

    We think initialize the Process noise with 1/2 of the rate of change
in time squared
    and the delT for velocity. Lastly we calculated the predicted
covariance matrix
    by the simple equation below.
        new_P = A.dot(self._P).dot(A.T) + Q_K.dot(Q_K.T) *
self._accel_variance
    """
    Q_K = np.zeros((2, 1))
    Q_K[iX] = 0.5 * dt**2
    Q_K[iV] = dt
    new_P = A.dot(self._P).dot(A.T) + Q_K.dot(Q_K.T) *
self._accel_variance

    """
    We update our state or state vector and processed covariance matrix
with the new predicted state
    """
    self._P = new_P
    self._x = new_x

    """
    This function takes care of calculating the Kalman Gain and
    updating the new state based on the predicted value and the measurement
    input.
    Y = CX_km + Z_k
    x = x + K(y - HX_kp)

```



```

K = P_kp H/ (H P Ht + R)
"""

def update(self, meas_value: float, meas_variance: float):
    # y = H - Z_k
    # K = P_kp H/ (H P Ht + R)
    # x = x + K(y - HX_kp)
    # P = (I - K H) * P_kp

    """
    In this function the measurement Noise matrix is being
    created with the number of rows of the array being
    the index of the filter.
    """
    Z_k = np.zeros((1, NUMVARS))
    Z_k[0, ix] = 1

    """
    Measured values are adding into the measurement matrix
    and finally the final Measurement vector was created
    """
    X_km = np.array([meas_value])
    y = X_km - Z_k.dot(self._x)
    self._Measured = y
    """
    Next the Kalman Gain will be calculated in this project
    which is typically modeled as:
    K = P_kp H/ (H P_kp H^T + R )

    First the Sensor Noise Covariance Matrix or
    Measurement Error which is denoted as R.

    The Bottom half of the Kalman Gain is then calculated
    based on the error measurement and process of covariance
    matrix. '_P' and the Observation matrix is also shown.
    Finally the Kalman Gain is called as shown below.
    """
    R = np.array([meas_variance])
    S = Z_k.dot(self._P).dot(Z_k.T) + R
    K = self._P.dot(Z_k.T).dot(np.linalg.inv(S))

    """
    Next the new-state vector and process covariance matrix
    are Calculated
    """
    new_x = self._x + K.dot(y)

    new_P = (np.eye(2) - K.dot(Z_k)).dot(self._P)

    self._P = new_P
    self._x = new_x

```

```

"""
Function for Covariance
"""
@property
def cov(self) -> np.array:
    return self._P

"""
Function for Mean
"""
@property
def mean(self) -> np.array:
    return self._x

"""
Function for Position
"""
@property
def pos(self) -> float:
    return self._x[iX]

"""
Function for Velocity
"""
@property
def vel(self) -> float:
    return self._x[iV]

"""
Function for Velocity
"""
@property
def Measured(self) -> float:
    return self._Measured[iX]

```

## GUI Code

```

import functools
import numpy as np
import pylab
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
from kf import KF

"""
This function is responsible of plotting the X and Y variable on the plot.
"""
def draw_function(p, v, mus, covs, measured_p):
    Pxlabel = "Time in Seconds",
    Pylabel = "Position (Meters)"
    Vylabel = "Velocity (m/s)"
    figure, (ax1, ax2) = plt.subplots(2)

```

```

plt.subplots_adjust(left=0.11, bottom=0.40)

ax1.grid
lines1 = ax1.plot(p, "-b")
lines1a = ax1.plot([mu[0] for mu in mus], 'r')

line1Cov1 = ax1.plot([mu[0] - 2 * np.sqrt(cov[0, 0]) for mu, cov in
zip(mus, covs)], 'r--')

line1Cov2 = ax1.plot([mu[0] + 2 * np.sqrt(cov[0, 0]) for mu, cov in
zip(mus, covs)], 'r--')

measuredline = ax1.plot(measured_p, 'g', label="MEASURED")
# ax1.set_xlabel(Pxlabel)
ax1.set_ylabel(Pylabel)
ax1.set_ylim(-5, max(p) * 1.5)

ax2.grid
lines2 = ax2.plot(v, "-b", label='TRUE')
lines2a = ax2.plot([mu[1] for mu in mus], 'r', label='PREDICTED')

line2Cov1 = ax2.plot([mu[1] - 2 * np.sqrt(cov[1, 1]) for mu, cov in
zip(mus, covs)], 'r--', label='UPPER ERROR')

line2Cov2 = ax2.plot([mu[1] + 2 * np.sqrt(cov[1, 1]) for mu, cov in
zip(mus, covs)], 'r--', label='LOWER ERROR')

ax2.set_xlabel(Pxlabel)
ax2.set_ylabel(Vylabel)
ax2.set_ylim(max(v) * -15, max(v) * 15)
plt.gcf().legend()

return figure, ax1, lines1, lines1a, line1Cov1, line1Cov2, ax2, lines2,
lines2a, line2Cov1, line2Cov2, measuredline

"""
This function is set to control what the follows are set to be.
We will have 5 Sliders on our GUI which can help the student to see the
efforts
of variance error and initial measurements.
"""
def draw_interactive_controls(E_mea, Var, dt, init_X, init_V,
samplingperiod):
    # Placing Sliders onto the Plot GUI
    axK = plt.axes([0.1, 0.01, 0.75, 0.03])
    axB = plt.axes([0.1, 0.06, 0.75, 0.03])
    axN = plt.axes([0.1, 0.11, 0.75, 0.03])
    axX = plt.axes([0.1, 0.16, 0.75, 0.03])
    axV = plt.axes([0.1, 0.21, 0.75, 0.03])
    axS = plt.axes([0.1, 0.26, 0.75, 0.03])

```

```

Nslider = Slider(axN, "E_Mea", 0.0, 4, valinit=E_mea, valfmt='%1.3f')
Bslider = Slider(axB, "Var(A)", 0, 0.25, valinit=Var, valfmt='%1.3f')
Kslider = Slider(axK, "dT", 0.00, 0.65, valinit=dt, valfmt='%1.3f')

Samplingslider = Slider(axS, "Sample-Rate", 0.00, 100,
valinit=samplingperiod, valstep=1)

Xslider = Slider(axX, "Init_Pos.", 0, 20, valinit=init_X, valfmt='%1.3f')

Vslider = Slider(axV, "Init_Vel.", 0.00, 2, valinit=init_V,
valfmt='%1.3f')

return Nslider, Bslider, Kslider, Xslider, Vslider, Samplingslider

"""
This function updates the plot of the figure
"""
def update_plot(val, line1=None, line2=None, line3=None, line4=None,
measuredLine=None, ax1=None, line5=None, line6=None, line7=None, line8=None,
ax2=None, Nslider=None, Bslider=None, Kslider=None, Xslider=None,
Vslider=None, Mslider=None):

    # getting the slider values from the system
    measured_variance = Nslider.val
    accel_variance = Bslider.val
    DT = Kslider.val
    initial_x = Xslider.val
    initial_v = Vslider.val
    MeasuredStep = Mslider.val

    # print("Measured Variance:", measured_variance)
    # print("Initial X:", initial_x)
    # run the Kalman filter and get the updated position and velocity

    position, velocity, mus, covs, measured_P, step =
RunKalmanFilter(init_x=initial_x, init_v=initial_v, accel_var=accel_variance,
mea_var=measured_variance, ddt=DT, MEAS_EVERY_STEPS=MeasuredStep)

    # Now that we have the updated position and velocity
    # go ahead and redraw the plots on the graph
    # print(position)

    line1[0].set_ydata(position) # Plotting of True value
    line2[0].set_ydata([mu[0] for mu in mus]) # Plotting Predicted Values

    line3[0].set_ydata([mu[0] - 2 * np.sqrt(cov[0, 0]) for mu, cov in
zip(mus, covs)]) # Upper Uncertainty Bound

    line4[0].set_ydata([mu[0] + 2 * np.sqrt(cov[0, 0]) for mu, cov in
zip(mus, covs)]) # Lower Uncertainty Bound

```

```

measuredLine[0].set_ydata(measured_P) # Measured position
ax1.set_ylim(-5, max(position) * 1.5)

line5[0].set_ydata(velocity) # Plotting of True value
line6[0].set_ydata([mu[1] for mu in mus]) # Plotting Predicted Values

line7[0].set_ydata([mu[1] - 2 * np.sqrt(cov[1, 1]) for mu, cov in
zip(mus, covs)]) # Upper Uncertainty Bound

line8[0].set_ydata([mu[1] + 2 * np.sqrt(cov[1, 1]) for mu, cov in
zip(mus, covs)]) # Lower Uncertainty Bound

ax2.set_ylim(max(velocity) * -15, max(velocity) * 15)
pylab.draw()

"""
This function runs the Kalman Filter and outputs the basic it takes the
initial position, initial velocity, sample time,
and variance in the measured and acceleration.
Returns: Estimated Position, Velocity, Measured Value, and Covariance
"""
def RunKalmanFilter(init_x=None, init_v=None, accel_var=None, mea_var=None,
ddt=None, MEAS_EVERY_STEPS=None):
    # initial values for the number of steps and Measurements for each steps
    NUM_STEPS = 600
    real_x = 0.8 ** 2
    real_v = 0.1

    mus = []
    covs = []
    real_xs = []
    measured_xs = []
    real_vs = []

    kf = KF(initial_x=init_x, initial_v=init_v, accel_variance=accel_var)

    """
    Running the system
    """
    for step in range(NUM_STEPS):
        if step > 300:
            real_v *= 0.99

        covs.append(kf.cov)
        mus.append(kf.mean)

        real_x = real_x + ddt * real_v
        measured_x = real_x + np.random.randn() * np.sqrt(mea_var)
        kf.predict(dt=DT)

```

```

        if step != 0 and step % MEAS_EVERY_STEPS == 0:
            kf.update(meas_value=measured_x,
                    meas_variance=mea_var)

        measured_xs.append(measured_x)
        real_xs.append(real_x)
        real_vs.append(real_v)

    return real_xs, real_vs, mus, covs, measured_xs, MEAS_EVERY_STEPS

if __name__ == "__main__":

    # Values that can be controlled
    measured_variance = 0.8 ** 2
    initial_x = 9.0
    initial_v = 0.1
    accel_variance = 0.1
    DT = 0.3
    MEAS_EVERY_STEPS = 20

    # Running the Kalman Filter
    real_xs, real_vs, mus, covs, measured_xs, MEAS_EVERY_STEPS =
RunKalmanFilter(init_x=initial_x, init_v=initial_v, accel_var=accel_variance,
mea_var=measured_variance, ddt=DT, MEAS_EVERY_STEPS=MEAS_EVERY_STEPS)

    # setup initial graph and control settings
    fig, ax1, Pos, PosPredicted, PosCov1, PosCov2, \
    ax2, VelMea, VelPredicted, VelCov1, VelCov2, PosMeasured =
draw_function(real_xs, real_vs, mus, covs, measured_xs)

    # generated the needed sliders
    E_Measlider, E_Acelslider, DTslider, initial_xslider, initial_vslider,
MSlider = draw_interactive_controls( measured_variance, accel_variance, DT,
initial_x, initial_v, MEAS_EVERY_STEPS)

    # specify updating function for interactive controls
    updatefxn = functools.partial(update_plot, line1=Pos, line2=
PosPredicted, line3=PosCov1, line4=PosCov2, line5=VelMea, line6=VelPredicted,
line7=VelCov1, line8=VelCov2, measuredLine=PosMeasured, ax1=ax1, ax2=ax2,
Nslider=E_Measlider, Bslider=E_Acelslider, Kslider=DTslider,
Xslider=initial_xslider, Vslider=initial_vslider, Mslider=MSlider)

    # update fxn function when the slider value gets changed
    E_Measlider.on_changed(updatefxn)
    E_Acelslider.on_changed(updatefxn)

    DTslider.on_changed(updatefxn)

```



```
initial_xslider.on_changed(updatefxn)
initial_vslider.on_changed(updatefxn)

MSlider.on_changed(updatefxn)

# Show the gui screen
fig.suptitle("Kalman Filter Simplified")
pylab.show()
```