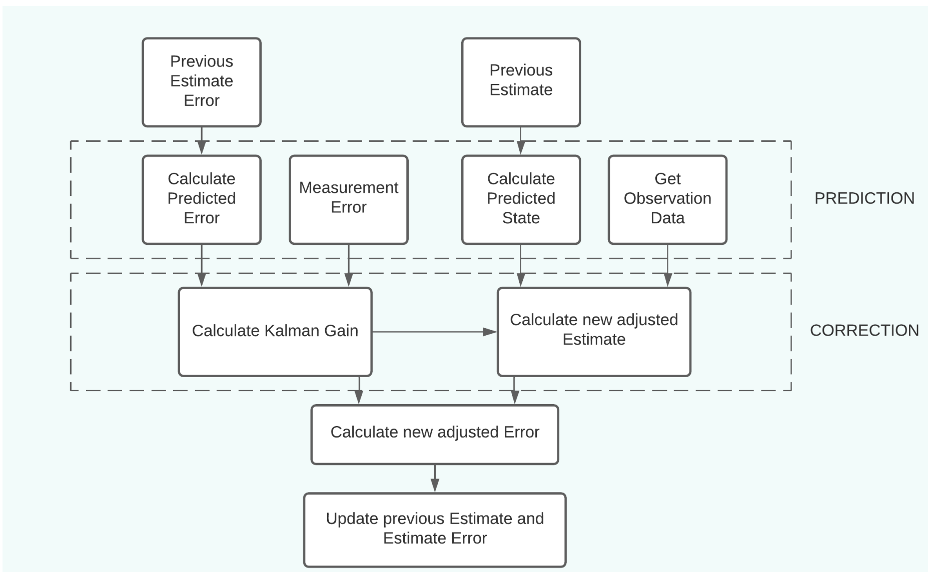
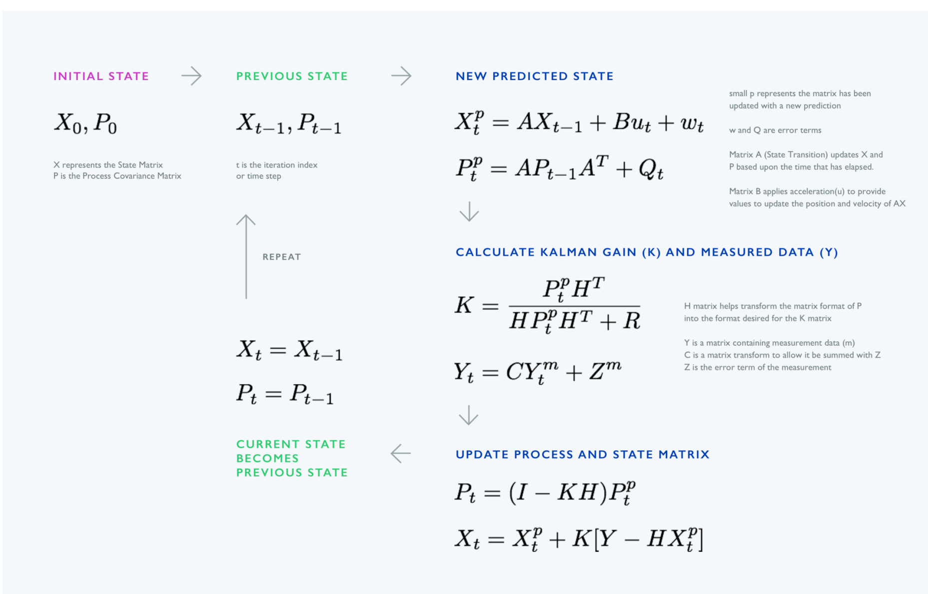
Kalman Filter - 1D motion example - Assignment 1

Dimitriy Georgiev

March 16, 2021

1 Flowchart of the Kalman Filter





2 Implementation in Python

0. Include necassary libraries

[1]:
```
import numpy as np
import pandas as pd
import openpyxl
import matplotlib.pyplot as plt
```

1. Define initial conditions:

[2]:
```
a = 1  # Acceleration (Control Signal) [m/s^2]
sd_noise = 0.2 # Acceleration noise, due to road surface, wind, etc. [m/s^2],␣
↪one standard deviation (sd)
x_0 = 0  # Position [m]
v_0 = 0  # Velocity [m/s]
t = 0.1  # Time step [s], equal. 10Hz

x_err_process = 0  # Position uncertainty for the first prediction [m]
v_err_process = 0  # Velocity uncertainty for the first prediction [m/s]

x_err_measure = 10  # Position uncertainty in observation data [m]

# Define the transformation matrices describing the state equations
A = np.array([[1, t], [0, 1]])
B = np.array([[0.5 * (t ** 2)], [t]])
C = np.array([1, 0]).transpose() # Observations are made only on the position
control_variable_matrix = np.array([a])

# Define Sw as the covariance between position and velocity error due to␣
↪acceleration noise
process_noise_matrix = np.array([[((0.5 * (t ** 2)) ** 2) * (sd_noise ** 2), ((0.
↪5 * (t ** 2)) * sd_noise) * (t * sd_noise)],
                                 [((0.5 * (t ** 2)) * sd_noise) * (t *␣
↪sd_noise), (t ** 2) * (sd_noise ** 2)]])
# or if pre-calculated -> estimation_noise_matrix = np.array([[10**-6, 2 * 10␣
↪**-5], [2 * 10 **-5, 4 * 10 **-4]])
```

2. Import observation data from Excel file

[3]:
```
data_frame = pd.read_excel("KF_data.xlsx", engine="openpyxl")
collected_data = data_frame["data"].to_numpy()
```

3. Define initial state of the system:

[4]:
```
prev_process_covariance = np.array([[x_err_process ** 2, 0], [0, v_err_process␣
↪** 2]]) # Covariance terms are set to 0 ('x' does not affect 'v')
prev_state = np.array([x_0, v_0]).transpose()
```

4. Begin the recursive process of the filter:

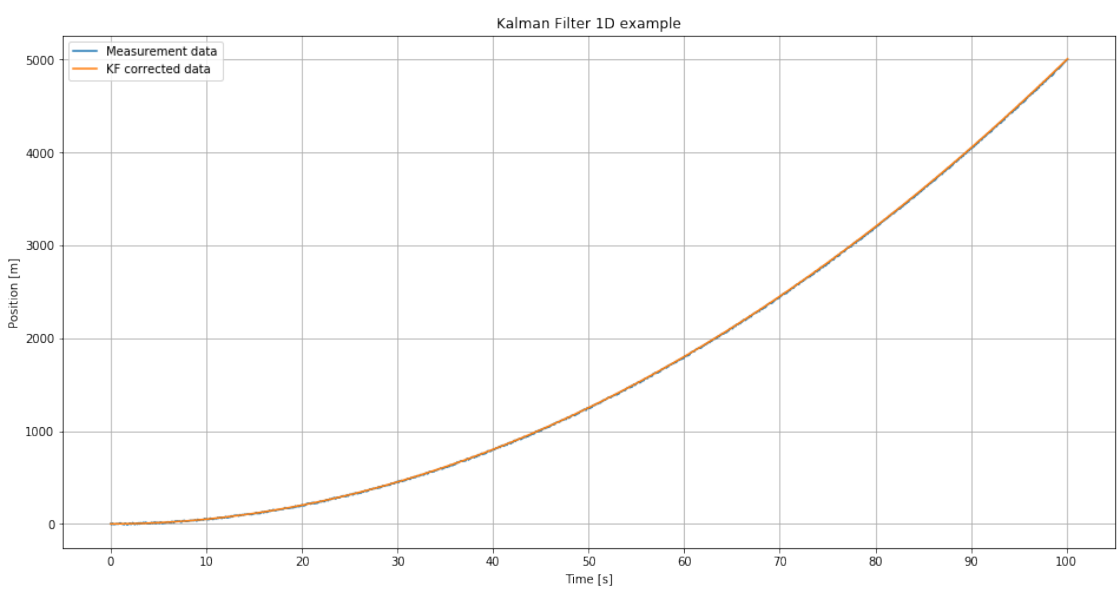
[5]:
```
# Define variables to store filtered data
time = np.linspace(0, 100.1, num=1001)
kalman_corrected_data = []

for x_measurement in collected_data:
        # Predict the state
        state_estimate = A.dot(prev_state) + B.dot(control_variable_matrix) + 0

        # Predict the error in the estimate
        process_covariance_estimate = A @ prev_process_covariance @ A.
 ↪transpose() + 0 # process_noise_matrix
        process_covariance_estimate = np.diag(np.
 ↪diag(process_covariance_estimate)) # Set covariance terms to 0 ('x' does not␣
 ↪affect 'v')

        # Take a state measurement
        state_measurement = C * x_measurement + 0 # Assume measurement errors␣
 ↪due to unknown factors are 0

        # Calculate the error in the measurement, use standard deviation of GPS␣
 ↪error
        measurement_covariance = np.array([x_err_measure ** 2])

        # Calculate the weigthing factor (kalman gain)
        kalman_gain = process_covariance_estimate.take(0).item() /␣
 ↪(process_covariance_estimate.take(0).item() + measurement_covariance.item())
        kalman_gain = np.array([kalman_gain, 0]).transpose()

        # Calculate the (new) adjusted state, based on the estimate and the␣
 ↪measurement
        H = np.identity(2)
        adjusted_state = state_estimate + kalman_gain * np.
 ↪array([state_measurement.take(0) - (H @ state_estimate).take(0)])

        # Calculate the (new) adjusted error in the estimate, based on the␣
 ↪weigthing factor
        I = np.identity(2)
        adjusted_process_covariance = (I - kalman_gain.transpose() @ H) @␣
 ↪process_covariance_estimate
        adjusted_process_covariance = np.diag(np.
 ↪diag(adjusted_process_covariance)) # Set covariance terms to 0 ('x' does not␣
 ↪affect 'v')

        # Update the previous state and process covariance
        prev_state = adjusted_state
```

```
        prev_process_covariance = adjusted_process_covariance

        kalman_corrected_data.append(adjusted_state.take(0).item())

dictionary = {'Observed': collected_data, 'KF adjusted': kalman_corrected_data}
data_frame = pd.DataFrame(data=dictionary)
print(data_frame.tail())

    #print("Observed: ", x_measurement, "       ", "KF adjusted: ",␣
 ↪adjusted_state.take(0)
```

```
      Observed  KF adjusted
996     4964.0     4970.045
997     4974.0     4980.020
998     4984.0     4990.005
999     4998.0     5000.000
1000    5003.0     5010.005
```

5. Plot the observed data & KF corrected data:

[6]:
```
plt.rcParams['figure.figsize'] = [16, 8]
plt.plot(time, collected_data, label="Measurement data")
plt.plot(time, kalman_corrected_data, label="KF corrected data")
plt.xlabel("Time [s]")
plt.xticks(np.arange(0, 100.1, step=10))
plt.ylabel("Position [m]")
plt.title("Kalman Filter 1D example")
plt.legend()
plt.grid()
plt.show()
```



NOTE: In order to match the filtered data with the provided numbers in the excel sheet, the process covariance matrix had to be set to 0 and accelerometer noise ignored, however by doing this we are essentially relying only on the estimate values from the model and disregarding the measurement readings.

6. Calculate with added noise:

[7]:
```
prev_process_covariance = np.array([[x_err_process ** 2, 0], [0, v_err_process␣
↪** 2]]) #
prev_state = np.array([x_0, v_0]).transpose()

kalman_corrected_data = []

for x_measurement in collected_data:
        # Predict the state
        state_estimate = A.dot(prev_state) + B.dot(control_variable_matrix) + 0

        # Predict the error in the estimate
        process_covariance_estimate = A @ prev_process_covariance @ A.
↪transpose() + process_noise_matrix
        process_covariance_estimate = np.diag(np.
↪diag(process_covariance_estimate)) #

        # Take a state measurement
        state_measurement = C * x_measurement + 0

        # Calculate the error in the measurement, use standard deviation of GPS␣
↪error
        measurement_covariance = np.array([x_err_measure **2])

        # Calculate the weigthing factor (kalman gain)
        kalman_gain = process_covariance_estimate.take(0).item() /␣
↪(process_covariance_estimate.take(0).item() + measurement_covariance.item())
        kalman_gain = np.array([kalman_gain, 0]).transpose()

        # Calculate the (new) adjusted state, based on the estimate and the␣
↪measurement
        H = np.identity(2)
        adjusted_state = state_estimate + kalman_gain * np.
↪array([state_measurement.take(0) - (H @ state_estimate).take(0)])

        # Calculate the (new) adjusted error in the estimate, based on the␣
↪weigthing factor
        I = np.identity(2)
```

```
        adjusted_process_covariance = (I - kalman_gain.transpose() @ H) @ 
 ↪process_covariance_estimate
        adjusted_process_covariance = np.diag(np.
 ↪diag(adjusted_process_covariance)) # Set covariance terms to 0 ('x' does not 
 ↪affect 'v')

        # Update the previous state and process covariance
        prev_state = adjusted_state
        prev_process_covariance = adjusted_process_covariance

        kalman_corrected_data.append(adjusted_state.take(0).item())

dictionary = {'Observed': collected_data, 'KF adjusted': kalman_corrected_data}
data_frame = pd.DataFrame(data=dictionary)
print(data_frame.tail())
```

```
      Observed  KF adjusted
996     4964.0  4961.765439
997     4974.0  4971.754021
998     4984.0  4981.752620
999     4998.0  4991.785246
1000    5003.0  5001.797531
```

We get slightly different values when we add the accelerometer noise to the process covariance matrix