homework 9

January 3, 2025

1 Homework 9 (DL Monday, November 22 at 12:00 PM)

ELEC-E8740 - Basics of sensor fusion - Autumn 2024

[1]:
```python
import numpy as np
import scipy.linalg as linalg
import matplotlib.pyplot as plt
```

Consider a 1D Gaussian random walk model

$$x_k = x_{k-1} + q_{k-1}, y_k = x_k + r_k,$$

where $x_0 \sim \mathcal{N}(0, 1), q_{k-1} \sim \mathcal{N}(0, 1)$, and $r_k \sim \mathcal{N}(0, 1)$.

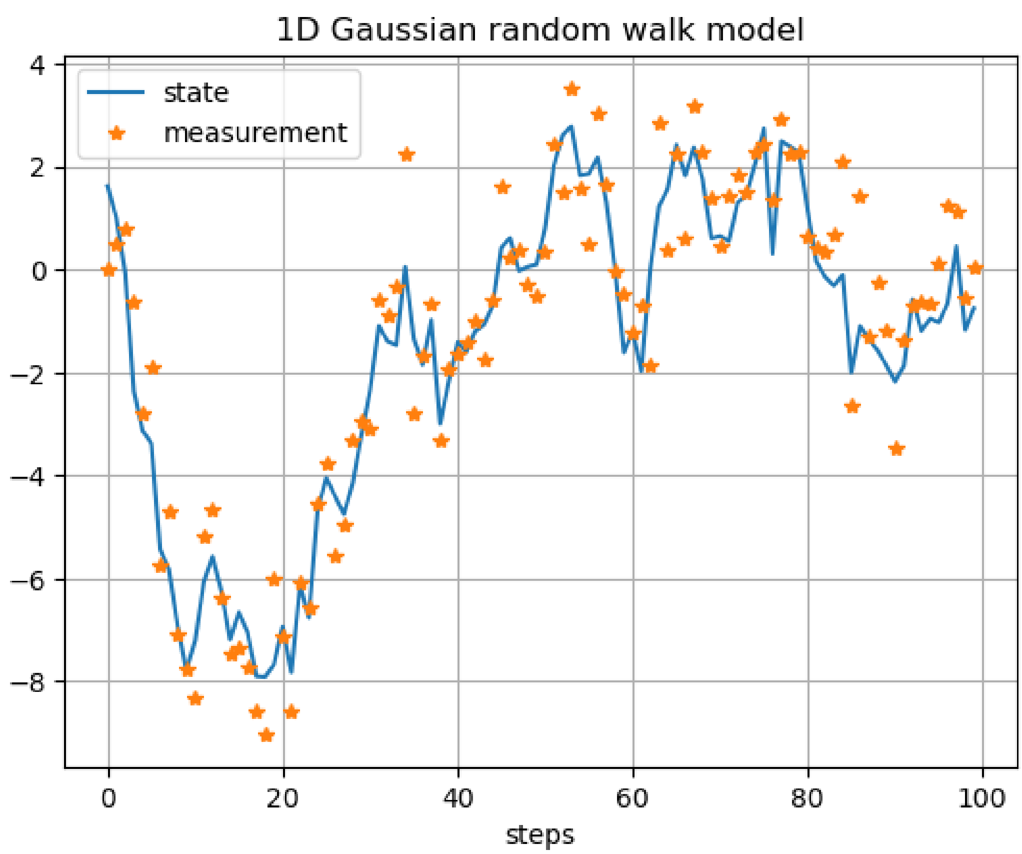
1.0.1 Part a (1 point): Simulate state and measurements from the model for 100 time steps. Plot the data.

[2]:
```python
def model_simulation(seed_number, steps):
    """
    model simulation for 1D Gaussian random walk model
    -------
    Input:
        seed_number: it is used to generate the same sequence of random numbers
        steps: number of steps
    Output:
        xs: state trajectory
        ys: measurement tajectory

    """
    np.random.seed(seed_number)       # do not change this line
    xs = np.zeros((steps, 1))         # do not change this line
    ys = np.zeros((steps, 1))         # do not change this line
    # To draw random samples from a normal (Gaussian) distribution, you could␣
 ↪use np.random.normal function
    # YOUR CODE HERE
    # Initial state x_0
    xs[0] = np.random.normal(0, 1)

    for k in range(1, steps):
```

```python
        q_k = np.random.normal(0, 1)
        xs[k] = xs[k-1] + q_k

        r_k = np.random.normal(0, 1)
        ys[k] = xs[k] + r_k

    return xs, ys # do not change this line
```

Feel free to uncomment and run the given code below.

[3]:
```python
plt.plot(model_simulation(1, 100)[0], label='state')
plt.plot(model_simulation(1, 100)[1], '*', label='measurement')
plt.title('1D Gaussian random walk model')
plt.xlabel('steps')
plt.legend()
plt.grid();
```



[ ]:
```python

```

1.0.2 part b (2 points) Implement a Kalman filter for the model and plot the results.

Note: the input of the following "Kalman_Filter" function is only the measurements. Please do not change that and define any nesessary parameters inside the function.

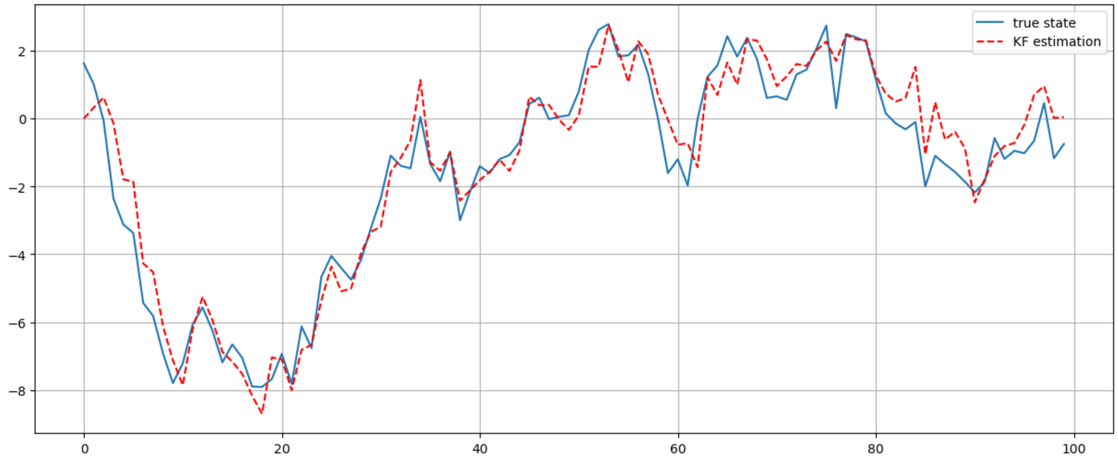
The output should be Kalman filter means and covariances of the whole trajectory.

[4]:
```python
def Kalman_Filter(Y):
    """
    Kalman filter state estimation for 1D Gaussian random walk model

    -------
    Input:
        Y: measurements
    Output:
        mean_kf: Kalman filter mean estimation
        cov_kf: Kalman filter covariance estimation

    """
    steps = Y.shape[0]
    mean_kf = np.zeros((steps, 1))      # do not change this line
    cov_kf = np.zeros((steps, 1, 1))    # do not change this line
    # YOUR CODE HERE
    mean_kf[0] = 0
    cov_kf[0] = 1


    A = 1
    Q = 1
    H = 1
    R = 1


    for k in range(1, steps):
        # Prediction Step
        mean_pred = A * mean_kf[k-1]
        cov_pred = A * cov_kf[k-1] * A + Q

        # Update Step
        y_k = Y[k]
        K = cov_pred * H / (H * cov_pred * H + R)
        mean_kf[k] = mean_pred + K * (y_k - H * mean_pred)
        cov_kf[k] = (1 - K * H) * cov_pred

    return mean_kf, cov_kf  # do not change this line
```

[ ]:
```python

```

Feel free to uncomment and run the given code below.

[5]:
```python
observations =  model_simulation(1, 100)[1]
x_kalman, cov_kalman = Kalman_Filter(observations)
plt.figure(figsize=(15,6))
plt.plot(model_simulation(1, 100)[0], label='true state')
plt.plot(x_kalman[:,0], 'r--', label='KF estimation')
plt.legend()
plt.grid();
```



1.0.3 Part b (1 point): Compare its state estimates (= mean) in RMSE sense to using pure measurements as estimates ($x_k \approx y_k$) for the state.

[9]:
```python
def rmse_comparison(true_state, kf_mean_estimation, pure_measurements):
    """
    RMSE comparison for 1D Gaussian random walk model using kalman filter␣
↪estimates and raw measurements estimates

    -------
    Input:
        true_state: true states come from the model
        kf_mean: kalman filter mean estimation
        pure_measurements: measurements
    Output:
        rmse_kf: rmse of Kalman filter estimates and true states
        rmse_raw: rmse of raw measurements estimates and true states

    """
    rmse_kf = np.sqrt(np.mean((true_state - kf_mean_estimation) ** 2))
    rmse_raw = np.sqrt(np.mean((true_state - pure_measurements) ** 2))


    return rmse_kf, rmse_raw
```

[ ]:

[11]:
```
seed_number = 1
steps = 100
true_state, measurements = model_simulation(seed_number, steps)

kf_mean, cov_kf = Kalman_Filter(measurements)

rmse_kf, rmse_raw = rmse_comparison(true_state, kf_mean, measurements)

plt.figure(figsize=(10, 6))
plt.plot(true_state, label='True state', color='blue')
plt.plot(measurements, '*', label='Measurements', color='orange')
plt.plot(kf_mean, label='Kalman Filter Estimate', color='green')
plt.title('Kalman Filter State Estimation for 1D Gaussian Random Walk Model')
plt.xlabel('Steps')
plt.legend()
plt.grid()
plt.show()
```