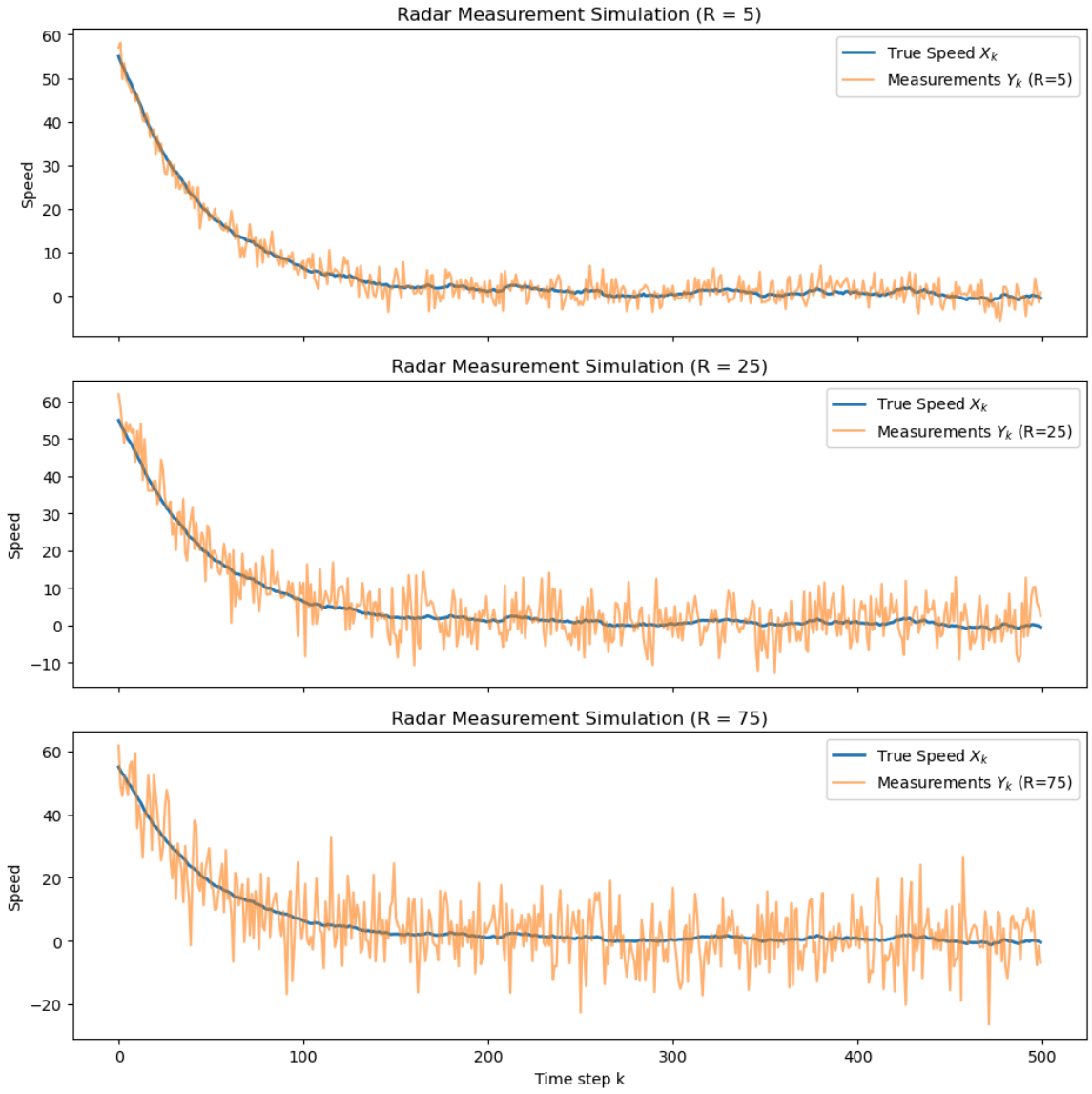
Numerical Implementation

a) Generate simulated plots over k = 0,…,499. Use R = 5, 25, 75.

Python

```python
import numpy as np
import matplotlib.pyplot as plt
# Parameters
np.random.seed(42)
N = 500         # time steps
a = 0.98        # state transition
Q = 0.04        # process noise variance
b = 1           # measurement gain
initial_mean = 50
initial_var = 100
# Generate true state X
X = np.zeros(N)
X[0] = np.random.normal(initial_mean, np.sqrt(initial_var))
for k in range(1, N):
    X[k] = a * X[k-1] + np.random.normal(0, np.sqrt(Q))

# Measurement noise scenarios
R_values = [5, 25, 75]
Y_all = {}

for R in R_values:
    V = np.random.normal(0, np.sqrt(R), size=N)
    Y = b * X + V
    Y_all[R] = Y

# Plot results
fig, axs = plt.subplots(3, 1, figsize=(10, 10), sharex=True)
for i, R in enumerate(R_values):
    axs[i].plot(X, label="True Speed $X_k$", linewidth=2)
    axs[i].plot(Y_all[R], label=f"Measurements $Y_k$ (R={R})", alpha=0.6)
    axs[i].set_ylabel("Speed")
    axs[i].set_title(f"Radar Measurement Simulation (R = {R})")
    axs[i].legend(loc="upper right")

axs[-1].set_xlabel("Time step k")
plt.tight_layout()
plt.show()
```



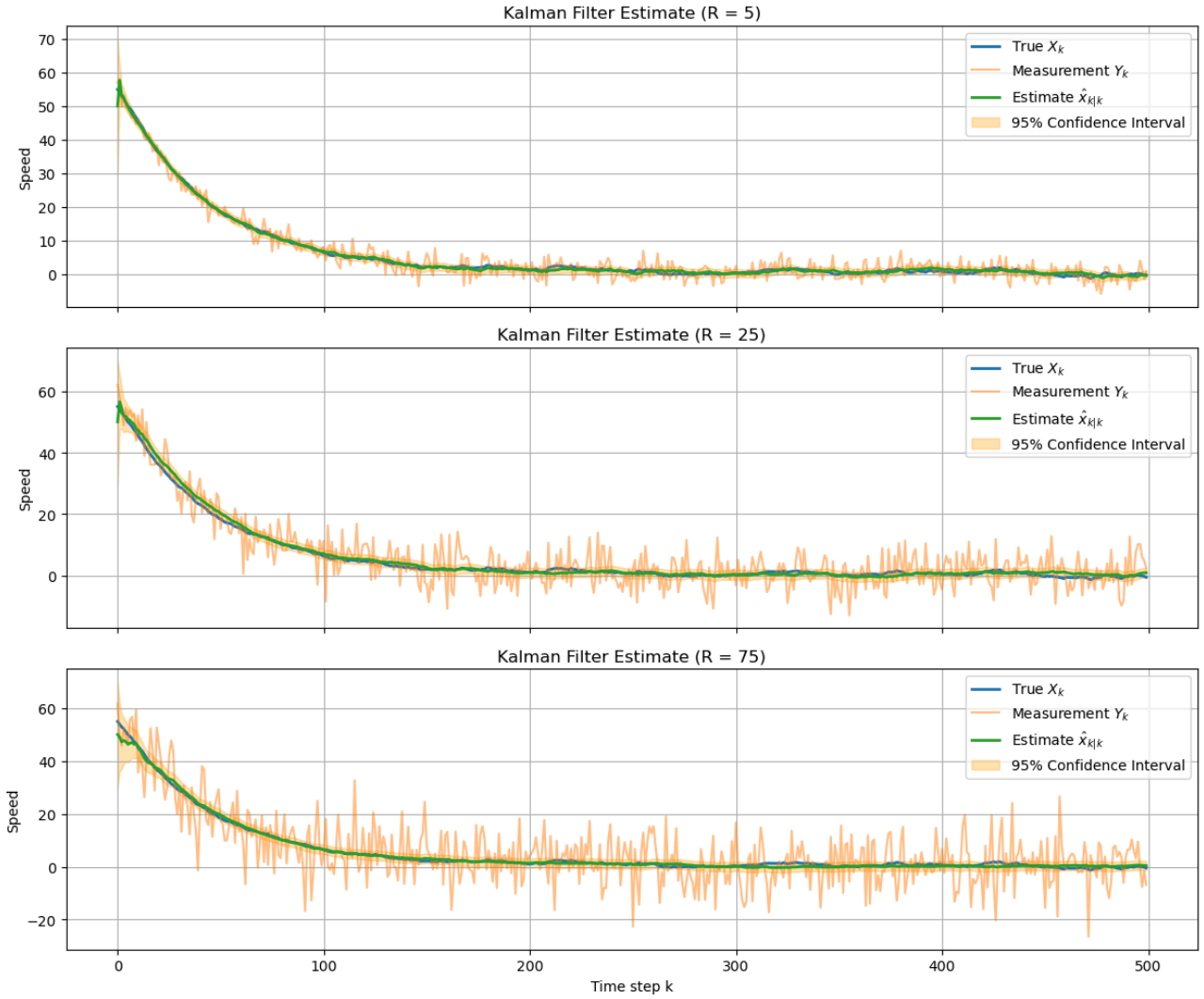
In this simulation, we modeled the true speed of a vehicle over 500 time steps using a simple linear system with process noise. We then generated radar measurements by adding varying levels of Gaussian noise to the true speed, using three different values of measurement noise variance R: 5, 25, and 75. The resulting plots show how measurement accuracy degrades as noise increases for low R; the measurements closely follow the true speed, while for high R, the readings are much more scattered and unreliable. This simulation highlights the challenge of inferring the true state from noisy observations and sets the foundation for using the Kalman filter in later parts to improve state estimation.

(b) [30 pts] Use the Kalman filter to track the vehicle speed Y_k generated in (a). Plot $\hat{x}_{k|k}$ as well as the confidence tube around it using $\hat{x}_{k|k} \pm 2\sqrt{P_{k|k}}$. Compare the estimates with the true values. Discuss your results and observations.

Python

```python
import numpy as np
import matplotlib.pyplot as plt

# Parameters
np.random.seed(42)
N = 500           # number of time steps
a = 0.98          # state transition coefficient
Q = 0.04          # process noise variance
b = 1             # measurement gain
initial_mean = 50
initial_var = 100

# Generate true state X
X = np.zeros(N)
X[0] = np.random.normal(initial_mean, np.sqrt(initial_var))
for k in range(1, N):
    X[k] = a * X[k-1] + np.random.normal(0, np.sqrt(Q))

# Measurement noise scenarios
R_values = [5, 25, 75]
Y_all = {}

# Generate noisy observations for each R
for R in R_values:
    V = np.random.normal(0, np.sqrt(R), size=N)
    Y = b * X + V
    Y_all[R] = Y

# Kalman filter function
def run_kalman_filter(Y, a, b, Q, R, mu_0, P_0):
    N = len(Y)
    x_est = np.zeros(N)         # Posterior mean estimate \hat{x}_{k|k}
    P_est = np.zeros(N)         # Posterior variance P_{k|k}
    x_pred = np.zeros(N)        # Predicted state \hat{x}_{k|k-1}

    # Initial conditions
    x_est[0] = mu_0
    P_est[0] = P_0
```

```
    for k in range(1, N):
        # Predict
        x_pred[k] = a * x_est[k-1]
        P_pred = a**2 * P_est[k-1] + Q

        # Kalman gain
        K = (P_pred * b) / (b**2 * P_pred + R)

        # Update
        x_est[k] = x_pred[k] + K * (Y[k] - b * x_pred[k])
        P_est[k] = P_pred - K * b * P_pred

    return x_est, P_est

# Initial Kalman filter parameters
mu_0 = 50
P_0 = 100

# Plot estimates for each R on a single figure with subplots
fig, axs = plt.subplots(3, 1, figsize=(12, 10), sharex=True)

for i, R in enumerate(R_values):
    Y = Y_all[R]
    x_est, P_est = run_kalman_filter(Y, a, b, Q, R, mu_0, P_0)

    axs[i].plot(X, label="True $X_k$", linewidth=2)
    axs[i].plot(Y, label="Measurement $Y_k$", alpha=0.5)
    axs[i].plot(x_est, label="Estimate $\hat{x}_{k|k}$", linewidth=2)
    axs[i].fill_between(
        range(len(X)),
        x_est - 2 * np.sqrt(P_est),
        x_est + 2 * np.sqrt(P_est),
        color="orange",
        alpha=0.3,
        label="95% Confidence Interval"
    )
    axs[i].set_title(f"Kalman Filter Estimate (R = {R})")
    axs[i].set_ylabel("Speed")
    axs[i].legend(loc="upper right")
    axs[i].grid(True)

axs[-1].set_xlabel("Time step k")
plt.tight_layout()
plt.show()
```



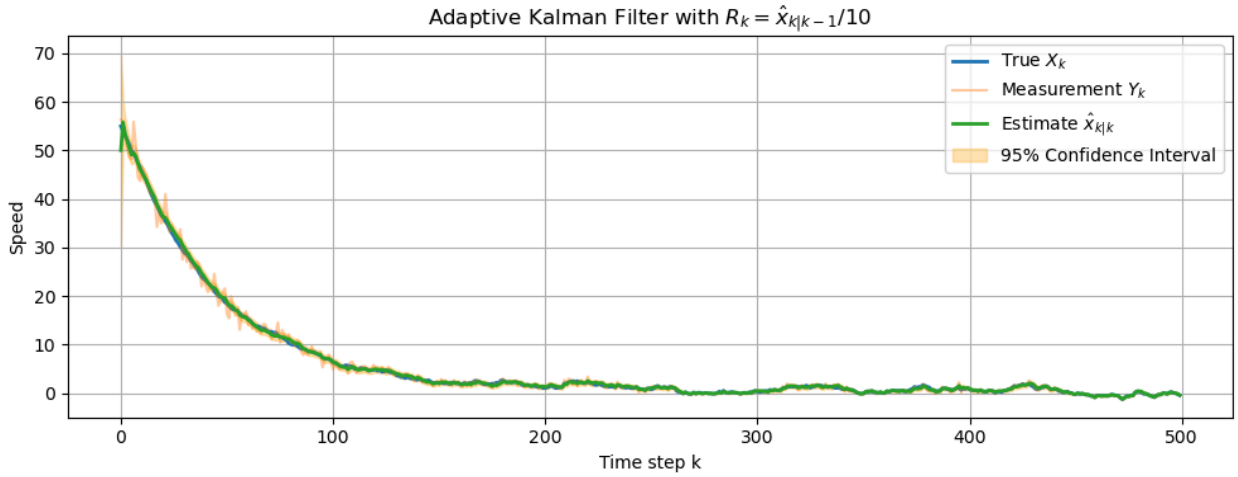
In this part, we applied the standard Kalman filter to estimate the true vehicle speed from noisy radar measurements generated in part (a). Using known system parameters and constant measurement noise values R=5,25,75, the filter was run over all time steps to produce both the estimated speed and its associated uncertainty at each step. The plots show how well the filter tracks the true speed under different noise conditions: when R is low, the estimate closely follows the true signal with narrow confidence intervals, while higher R leads to more cautious updates and wider uncertainty bands. Overall, the Kalman filter performs well, effectively combining the model's predictions with noisy measurements to produce smooth and reliable estimates of the vehicle's speed.

c) [20 pts] Suppose now that the measurement noise variance R is 1/10 of the actual vehicle speed. Can you still use the Kalman filter? Come up with a practical idea and implement it. (Of course you have to generate Y again.) Show the results. Does the scheme perform well? Why or why not? Discuss your results and observations.

Python

```
def run_kalman_v2(Y, a, b, Q, mu_0, P_0):
    N = len(Y)
    x_est = np.zeros(N)         # \hat{x}_{k|k}
    P_est = np.zeros(N)         # P_{k|k}
    x_pred = np.zeros(N)        # \hat{x}_{k|k-1}
    R_vals = np.zeros(N)        # Store R_k values for analysis

    # Initialize
    x_est[0] = mu_0
    P_est[0] = P_0
    R_vals[0] = 1 / 10 * mu_0

    for k in range(1, N):
        # Predict
        x_pred[k] = a * x_est[k-1]
        P_pred = a**2 * P_est[k-1] + Q

        # Estimate R_k from prediction
        R_k = max(0.01, 1/10 * x_pred[k])   # Avoid division by zero or
negative R
        R_vals[k] = R_k

        # Kalman gain
        K = (P_pred * b) / (b**2 * P_pred + R_k)

        # Update
        x_est[k] = x_pred[k] + K * (Y[k] - b * x_pred[k])
        P_est[k] = P_pred - K * b * P_pred

    return x_est, P_est, R_vals

# Simulate new noisy measurement using true X
R_true = np.maximum(1e-3, 1/10 * X)
V = np.random.normal(0, np.sqrt(R_true))
Y_dynamic = b * X + V

# Run adaptive Kalman filter
x_est_dyn, P_est_dyn, R_used = run_kalman_v2(Y_dynamic, a, b, Q, mu_0, P_0)
```

```python
# Plot results
plt.figure(figsize=(10, 4))
plt.plot(X, label="True $X_k$", linewidth=2)
plt.plot(Y_dynamic, label="Measurement $Y_k$", alpha=0.4)
plt.plot(x_est_dyn, label="Estimate $\hat{x}_{k|k}$", linewidth=2)
plt.fill_between(
    range(len(X)),
    x_est_dyn - 2 * np.sqrt(P_est_dyn),
    x_est_dyn + 2 * np.sqrt(P_est_dyn),
    color="orange",
    alpha=0.3,
    label=r"95% Confidence Interval"
)
plt.title("Adaptive Kalman Filter with $R_k = \hat{x}_{k|k-1}/10$")
plt.xlabel("Time step k")
plt.ylabel("Speed")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



This part modifies the standard Kalman filter to handle a case where the measurement noise variance is not constant, but instead varies with the true vehicle speed. Since the true state X_k is unknown during filtering, the variance R_k is approximated at each step using the predicted state $\hat{x}_{k|k-1}$. This is the key change: rather than keeping R fixed, the filter dynamically recalculates the Kalman gain using an estimated R_k based on the prediction. The measurements themselves are also regenerated using the true R_k values to simulate a more realistic environment with state-dependent noise. This adaptive version of the filter adjusts its trust in each observation depending on the predicted speed, giving less weight to noisier, high-speed readings. The resulting estimates track the true state closely, and the confidence intervals respond appropriately by widening when uncertainty increases. Overall, this practical modification shows that the Kalman filter can remain effective even when the measurement noise depends on the unknown state.