Exercise_round_9

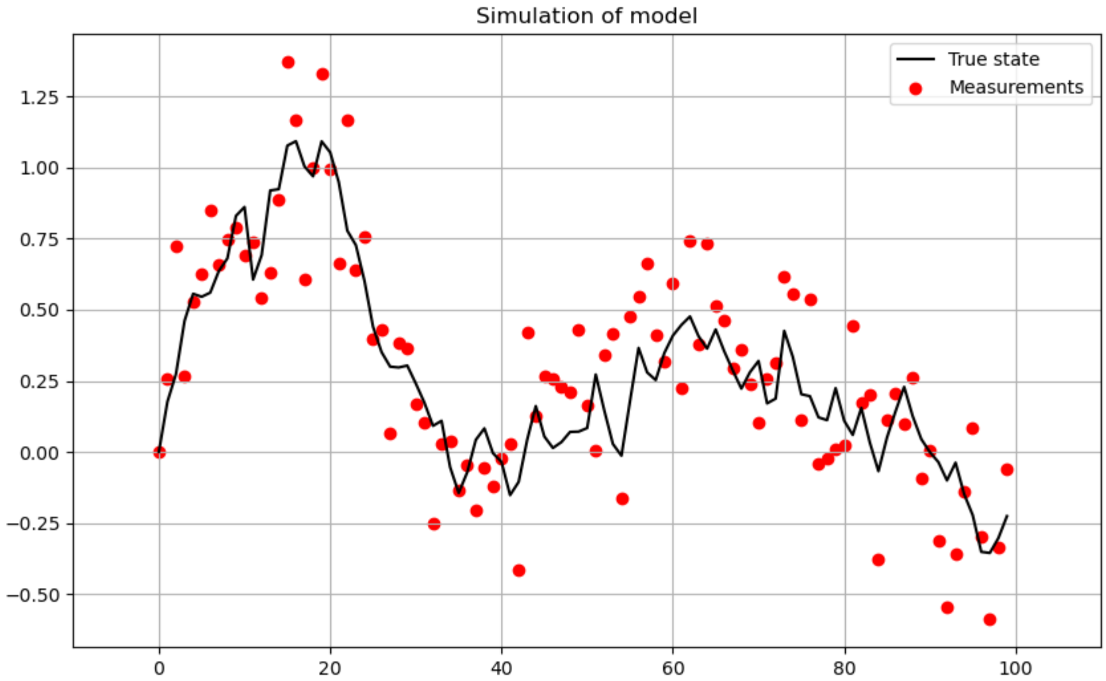
March 20, 2025

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

0.1 Exercise 2. (Smoother for Gaussian Random Walk)

[2]:
```
# Gaussian random walk model
np.random.seed(0)

steps = 100

Q = 0.1
R = 0.2

x0 = 0

xs = np.zeros(steps)
ys = np.zeros(steps)

xs[0] = x0
ys[0] = xs[0]

# Model simulation
for k in range(1, steps):
    xs[k] = xs[k-1] + np.random.normal(0, Q)
    ys[k] = xs[k] + np.random.normal(0, R)


# Plot simulation
plt.figure(figsize=(10, 6))
plt.grid(True)

plt.plot(xs, label="True state", color="black")
plt.scatter(range(steps), ys, label="Measurements", color="red")
plt.title("Simulation of model")
```

```
plt.xlim(steps-(steps+10), steps+10)
plt.legend(loc='upper right')
plt.show()
```



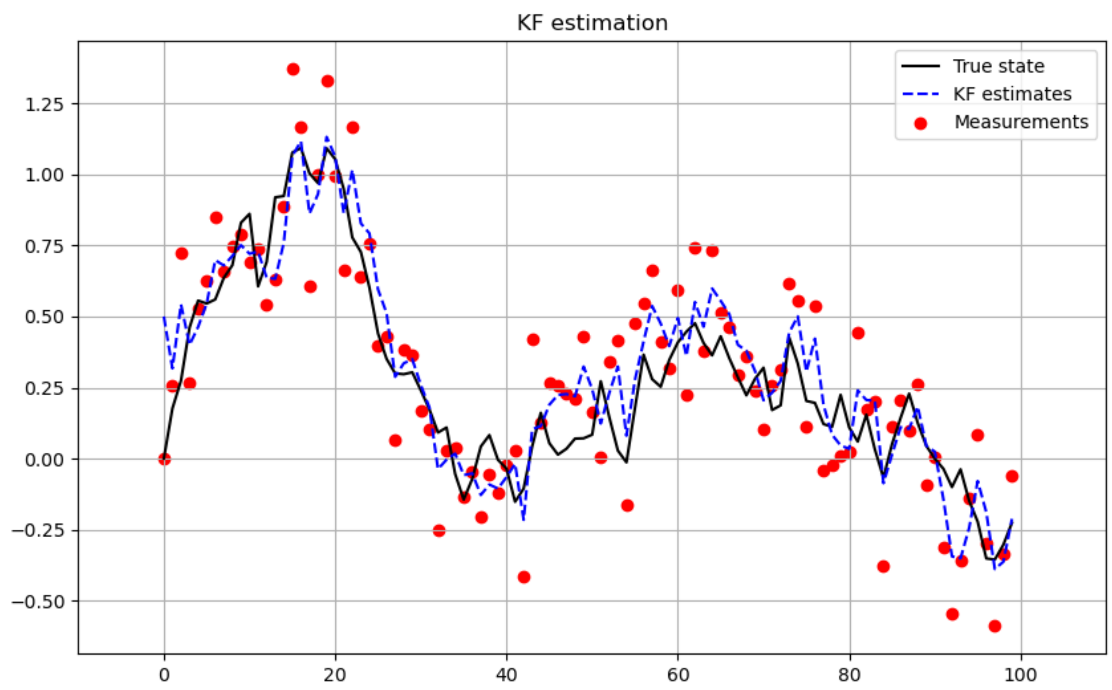
[3]:

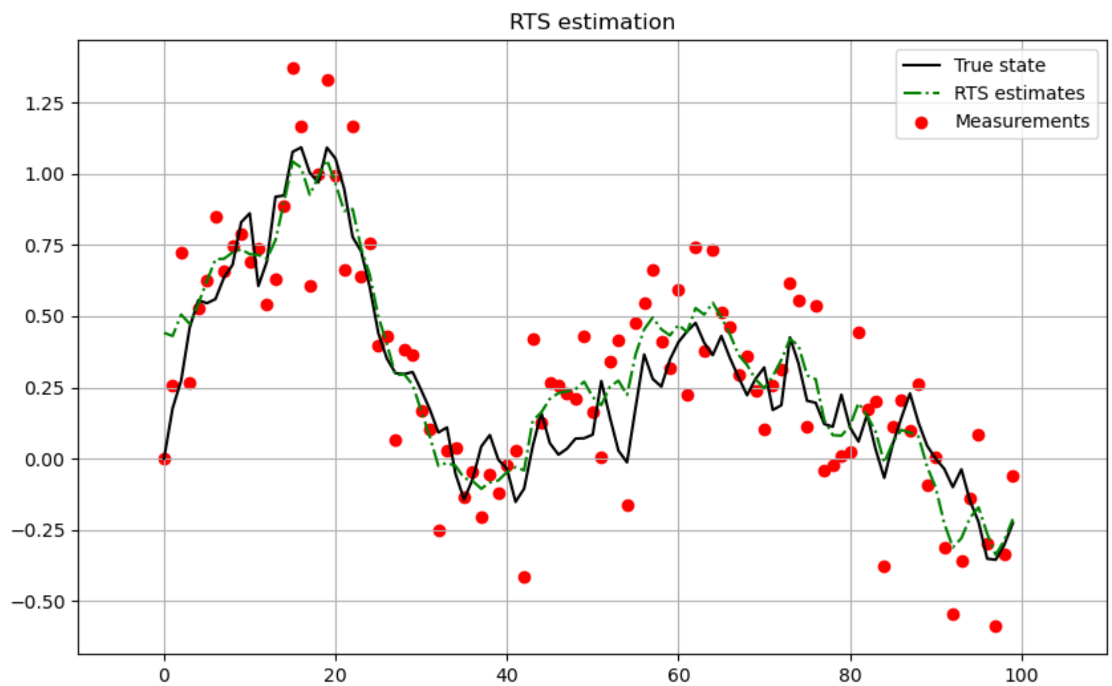
```
# Kalman Filter
def KF_walk(y, m0, P0, Q, R, steps):

    m_kf = np.zeros(steps)
    P_kf = np.zeros(steps)

    m_kf[0] = m0
    P_kf[0] = P0

    for k in range(1, steps):
        # Prediction
        m_pred = m_kf[k-1]
        P_pred = P_kf[k-1] + Q

        # Update
        S = P_pred + R
        K = P_pred / S
        m_kf[k] = m_pred + K * (y[k] - m_pred)
        P_kf[k] = P_pred - K * S * K

    return m_kf, P_kf
```

[4]:
```
# RTS smoother
def RTS_walk(m_kf, P_kf, Q, steps):

    m_rts = np.zeros(steps)
    P_rts = np.zeros(steps)

    m = m_kf[-1]
    P = P_kf[-1]

    m_rts[-1] = m
    P_rts[-1] = P

    for k in range(steps-2, -1, -1):
        # Prediction
        m_pred = m_kf[k]
        P_pred = P_kf[k] + Q

        # Smoothing gain
        Gk = P_kf[k] / P_pred

        # Backwards update
        m = m_pred + Gk * (m - m_pred)
        P = P_pred + Gk * (P - P_pred) * Gk

        m_rts[k] = m
        P_rts[k] = P

    return m_rts, P_rts
```

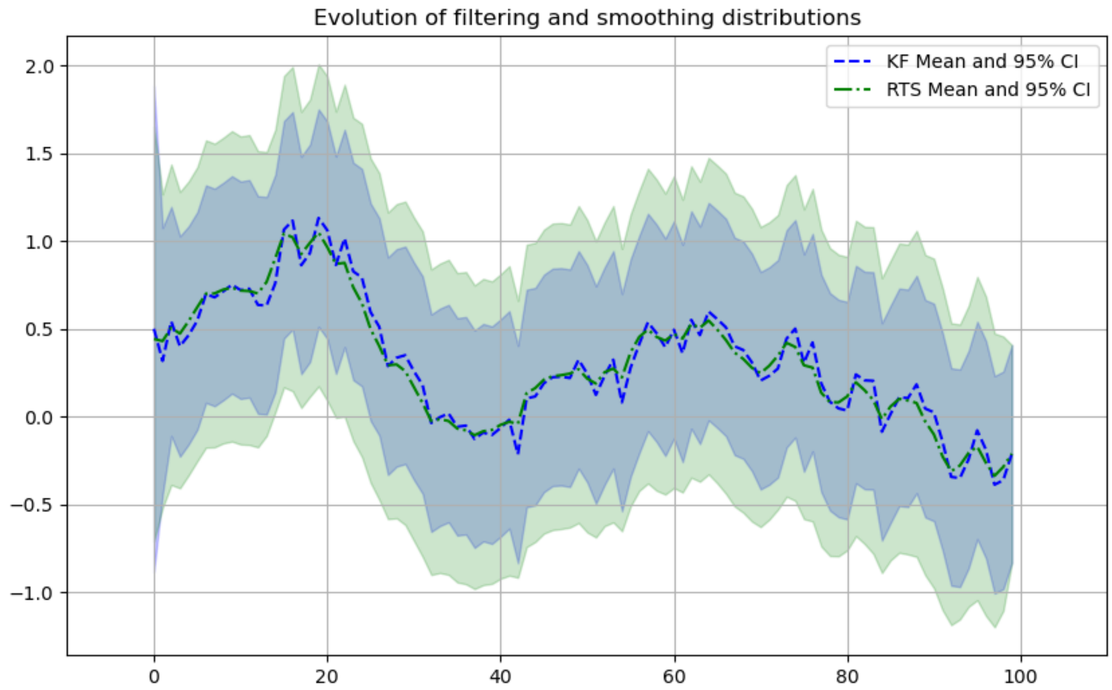
[5]:
```
# Simulate KF and RTS smoother

m0 = 0.5
P0 = 0.5

m_kf, P_kf = KF_walk(ys, m0, P0, Q, R, steps)
m_rts, P_rts = RTS_walk(m_kf, P_kf, Q, steps)

# Plot results

# KF
plt.figure(figsize=(10, 6))
plt.grid(True)

plt.plot(xs, label="True state", color="black")
plt.plot(m_kf, label="KF estimates", color="blue", linestyle="--")
plt.scatter(range(steps), ys, label="Measurements", color="red")
plt.title("KF estimation")
```

```python
plt.xlim(steps-(steps+10), steps+10)
plt.legend(loc='upper right')
plt.show()

# RTS
plt.figure(figsize=(10, 6))
plt.grid(True)

plt.plot(xs, label="True state", color="black")
plt.plot(m_rts, label="RTS estimates", color="green", linestyle="-.")
plt.scatter(range(steps), ys, label="Measurements", color="red")
plt.title("RTS estimation")
plt.xlim(steps-(steps+10), steps+10)
plt.legend(loc='upper right')
plt.show()
```





[6]:
```
# Plot evolution of filtering and smoothing distributions
kf_sd = np.sqrt(P_kf)
rts_sd = np.sqrt(P_rts)

plt.figure(figsize=(10, 6))
plt.grid(True)

# KF mean and 95% CI
plt.plot(range(steps), m_kf, label="KF Mean and 95% CI", color="blue",␣
↪linestyle="--")
plt.fill_between(range(steps), m_kf - 1.96*kf_sd, m_kf + 1.96*kf_sd,␣
↪color="blue", alpha=0.2)

# RTS mean and 95% CI
plt.plot(range(steps), m_rts, label="RTS Mean and 95% CI", color="green",␣
↪linestyle="-.")
plt.fill_between(range(steps), m_rts - 1.96*rts_sd, m_kf + 1.96*rts_sd,␣
↪color="green", alpha=0.2)

plt.title("Evolution of filtering and smoothing distributions")
plt.xlim(steps-(steps+10), steps+10)
plt.legend(loc='upper right')
plt.show()
```



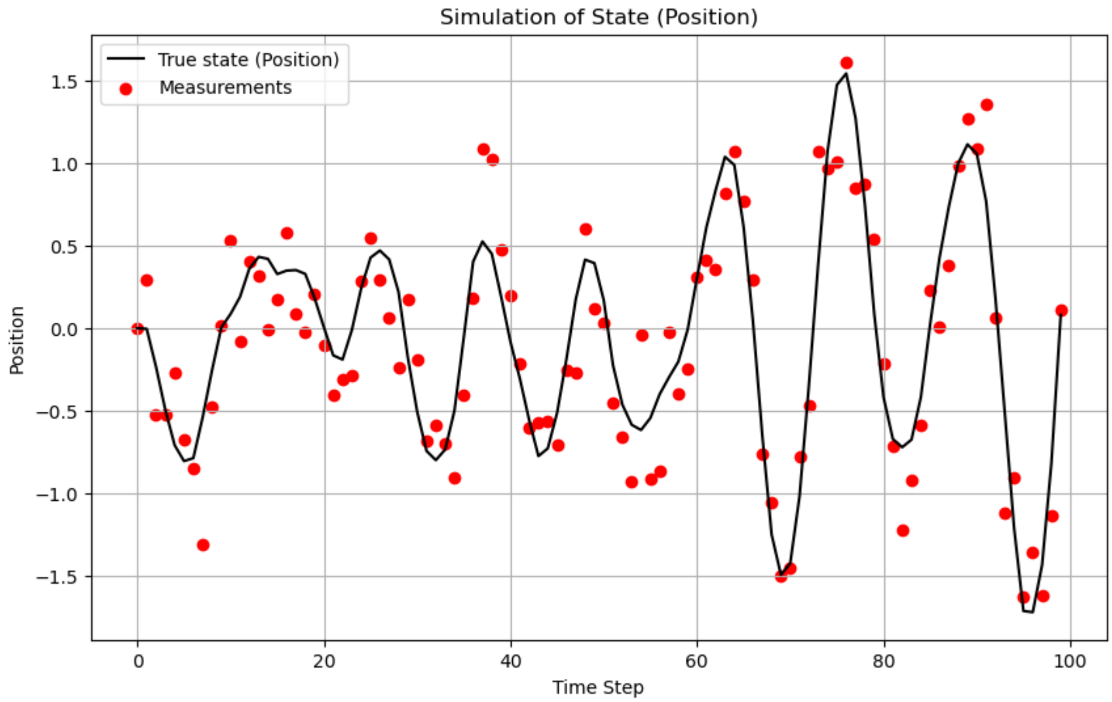
[7]:
```
# Compare RMSE of KF and RTS solutions

# KF
rmse_kf = np.sqrt(np.mean((m_kf - xs) ** 2))

# RTS
rmse_rts = np.sqrt(np.mean((m_rts - xs) ** 2))

print(f"RMSE (KF): {rmse_kf:.4f}")
print(f"RMSE (RTS): {rmse_rts:.4f}")
```

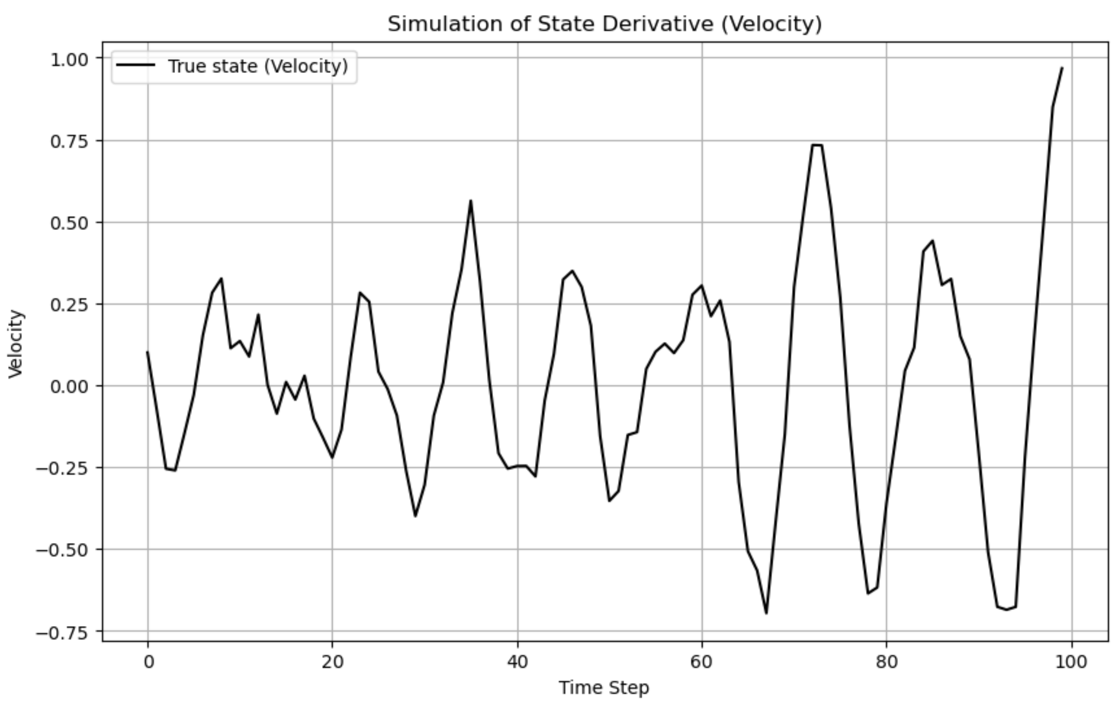
```
RMSE (KF): 0.1402
RMSE (RTS): 0.1202
```

0.2 Exercise 3. (Smoother for Stochastic Resonator)

[8]:
```
# Stochastic resonator model
np.random.seed(0)

steps = 100

x0 = np.array([0, 0.1])

w = 0.5
q = 0.01
```

```
R = 0.3
Q = 0.5 * q * np.array([[(w - np.cos(w) * np.sin(w)) / w ** 3, np.sin(w) ** 2 /␣
 ↪w ** 2],
                            [np.sin(w) ** 2 / w ** 2, (w + np.cos(w) * np.
 ↪sin(w)) / w]])

A = np.array([[np.cos(w), np.sin(w) / w],
              [(-w)*np.sin(w), np.cos(w)]])

H = np.array([[1, 0]])

xs = np.zeros((steps, 2))
ys = np.zeros((steps, 1))

xs[0] = x0

# Model simulation
for k in range(1, steps):
    xs[k] = A @ xs[k-1] + np.random.multivariate_normal([0, 0], Q)
    ys[k] = H @ xs[k] + np.random.normal(0, R)

# Plot true state and measurements
plt.figure(figsize=(10, 6))
plt.grid(True)
plt.plot(xs[:, 0], label="True state (Position)", color="black")
plt.scatter(range(steps), ys[:, 0], label="Measurements", color="red")
plt.title("Simulation of State (Position)")
plt.xlabel("Time Step")
plt.ylabel("Position")
plt.legend(loc='upper left')
plt.show()

# Plot second state (velocity)
plt.figure(figsize=(10, 6))
plt.grid(True)
plt.plot(xs[:, 1], label="True state (Velocity)", color="black")
plt.title("Simulation of State Derivative (Velocity)")
plt.xlabel("Time Step")
plt.ylabel("Velocity")
plt.legend(loc='upper left')
plt.show()
```


Simulation of State (Position)
True state (Position)
Measurements
Position
Time Step
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
20
40
60
80
100


Simulation of State Derivative (Velocity)
True state (Velocity)
Velocity
Time Step
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
20
40
60
80
100

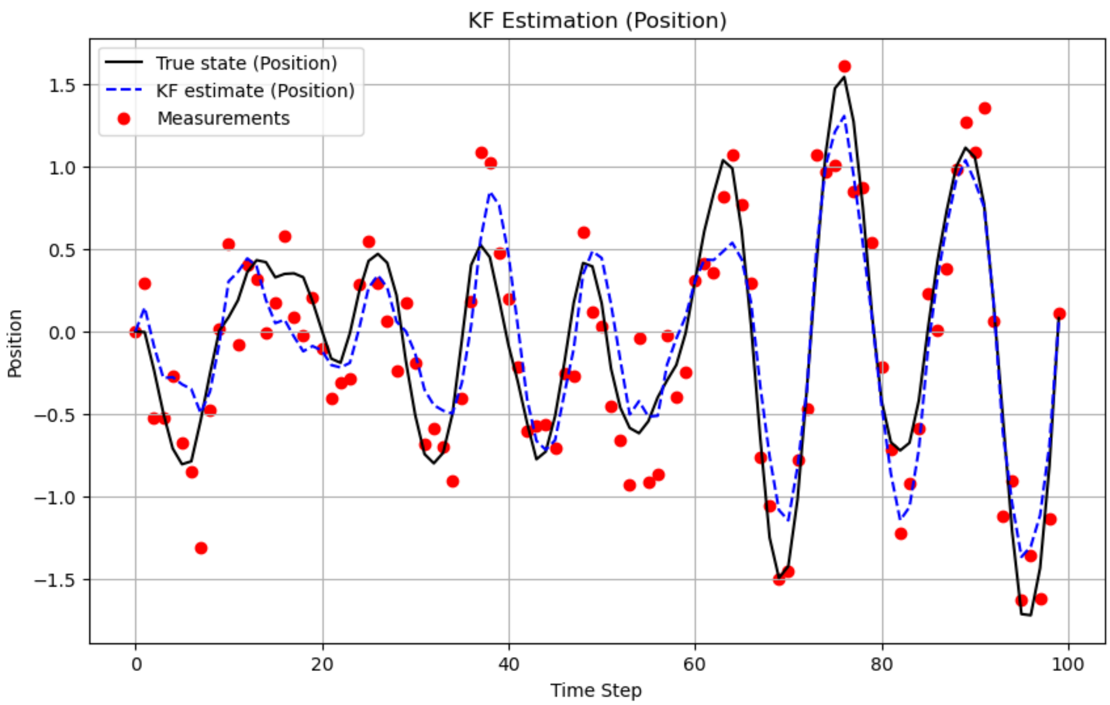
[9]:
```python
# KF for resonator
def KF_res(y, m0, P0, A, H, Q, R, steps):

    m_kf = np.zeros((steps, m0.shape[0]))
    P_kf = np.zeros((steps, P0.shape[0], P0.shape[1]))

    m_kf[0] = m0
    P_kf[0] = P0

    for k in range(1, steps):
        # Prediction
        m_pred = A @ m_kf[k-1]
        P_pred = A @ P_kf[k-1] @ A.T + Q

        # Update
        S = H @ P_pred @ H.T + R
        K = scipy.linalg.solve(S, H @ P_pred, assume_a='pos').T
        m_kf[k] = m_pred + K @ (y[k] - H @ m_pred)
        P_kf[k] = P_pred - K @ S @ K.T

    return m_kf, P_kf
```

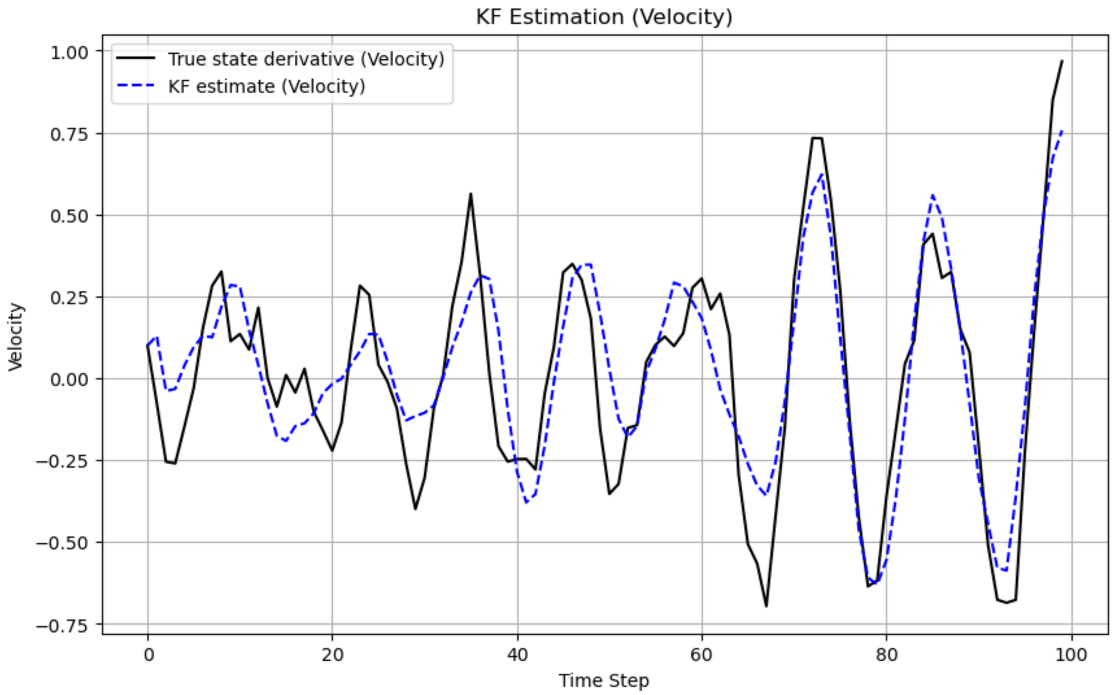
[10]:
```python
# RTS for resonator
def RTS_res(m_kf, P_kf, A, Q, steps):

    m_rts = np.zeros_like(m_kf)
    P_rts = np.zeros_like(P_kf)

    m = m_kf[-1]
    P = P_kf[-1]

    m_rts[-1] = m
    P_rts[-1] = P

    for k in range(steps-2, -1, -1):
        # Prediction
        m_pred = A @ m_kf[k]
        P_pred = A @ P_kf[k] @ A.T + Q

        # Smoothing gain
        Gk = P_kf[k] @ A.T @ scipy.linalg.solve(P_pred, np.eye(2),␣
 ↪assume_a='pos')

        # Backwards update
        m = m_kf[k] + Gk @ (m - m_pred)
        P = P_kf[k] + Gk @ (P - P_pred) @ Gk.T
```

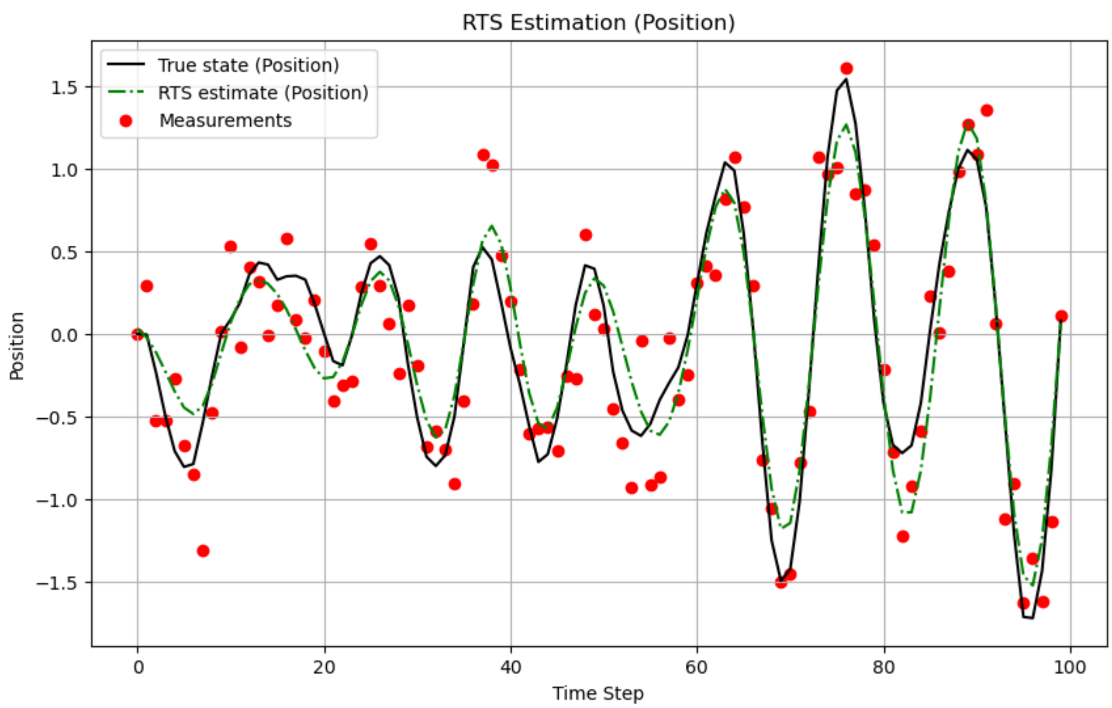
```
        m_rts[k] = m
        P_rts[k] = P

    return m_rts, P_rts
```

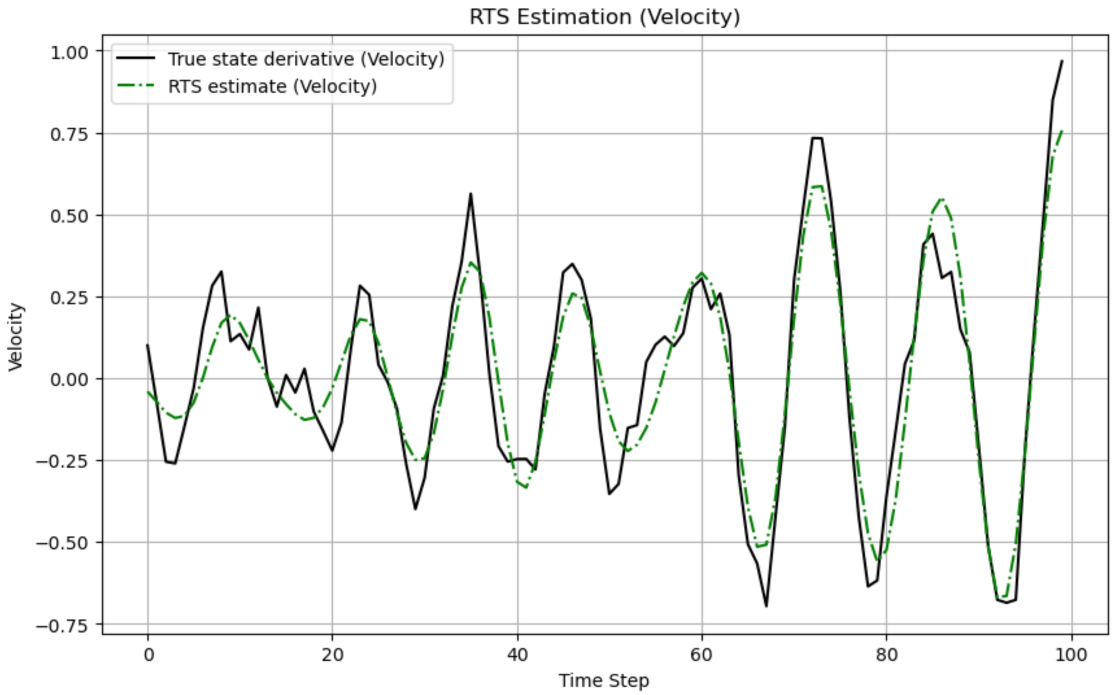
[11]:
```
# Run KF and RTS on the model

m0 = np.array([0, 0.1])
P0 = np.array([[0.01, 0.0],
              [0.0, 0.1]])

m_kf, P_kf = KF_res(ys, m0, P0, A, H, Q, R, steps)

m_rts, P_rts = RTS_res(m_kf, P_kf, A, Q, steps)

# Plot KF results
plt.figure(figsize=(10, 6))
plt.grid(True)
plt.plot(xs[:, 0], label="True state (Position)", color="black")
plt.plot(m_kf[:, 0], label="KF estimate (Position)", color="blue",␣
 ↪linestyle="--")
plt.scatter(range(steps), ys[:, 0], label="Measurements", color="red")
plt.title("KF Estimation (Position)")
plt.xlabel("Time Step")
plt.ylabel("Position")
plt.legend(loc='upper left')
plt.show()

plt.figure(figsize=(10, 6))
plt.grid(True)
plt.plot(xs[:, 1], label="True state derivative (Velocity)", color="black")
plt.plot(m_kf[:, 1], label="KF estimate (Velocity)", color="blue",␣
 ↪linestyle="--")
plt.title("KF Estimation (Velocity)")
plt.xlabel("Time Step")
plt.ylabel("Velocity")
plt.legend(loc='upper left')
plt.show()

# Plot RTS results
plt.figure(figsize=(10, 6))
plt.grid(True)
plt.plot(xs[:, 0], label="True state (Position)", color="black")
plt.plot(m_rts[:, 0], label="RTS estimate (Position)", color="green",␣
 ↪linestyle="-.")
plt.scatter(range(steps), ys[:, 0], label="Measurements", color="red")
plt.title("RTS Estimation (Position)")
```

```
plt.xlabel("Time Step")
plt.ylabel("Position")
plt.legend(loc='upper left')
plt.show()

plt.figure(figsize=(10, 6))
plt.grid(True)
plt.plot(xs[:, 1], label="True state derivative (Velocity)", color="black")
plt.plot(m_rts[:, 1], label="RTS estimate (Velocity)", color="green",␣
 ↪linestyle="-.")
plt.title("RTS Estimation (Velocity)")
plt.xlabel("Time Step")
plt.ylabel("Velocity")
plt.legend(loc='upper left')
plt.show()
```




KF Estimation (Velocity)
True state derivative (Velocity)
KF estimate (Velocity)
Velocity
Time Step
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
20
40
60
80
100


RTS Estimation (Position)
True state (Position)
RTS estimate (Position)
Measurements
Position
Time Step
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
20
40
60
80
100



[12]:
```
# Compare RMSE of KF and RTS solutions

# KF
rmse_kf_p = np.sqrt(np.mean((m_kf[:, 0] - xs[:, 0]) ** 2))
rmse_kf_v = np.sqrt(np.mean((m_kf[:, 1] - xs[:, 1]) ** 2))

# RTS
rmse_rts_p = np.sqrt(np.mean((m_rts[:, 0] - xs[:, 0]) ** 2))
rmse_rts_v = np.sqrt(np.mean((m_rts[:, 1] - xs[:, 1]) ** 2))

print(f"RMSE of position (KF vs RTS): {rmse_kf_p:.4f} vs {rmse_rts_p:.4f}")
print(f"RMSE of velocity (KF vs RTS): {rmse_kf_v:.4f} vs {rmse_rts_v:.4f}")
```

```
RMSE of position (KF vs RTS): 0.2542 vs 0.2006
RMSE of velocity (KF vs RTS): 0.1589 vs 0.1138
```