Statistics for Data Science
Wintersemester 2024/25
Solutions to the 12th exercise

1. We have a linear measurement model

$$y = \begin{pmatrix} 2 \\ 1 \end{pmatrix} x + \eta$$

with additive Gaussian noise $\eta \sim \mathcal{N}(0, \gamma^2 I_2)$ and a Gaussian prior $x \sim \mathcal{N}(0, 2)$ such that $x \perp \eta$. This is precisely the linear Gaussian setting, meaning that the posterior is Gaussian with known mean and covariance. Given the observation $y = (1, 2)^{\mathrm{T}}$, the posterior is

$$f(x|y = (1,2)^{\mathrm{T}}) \propto \exp\bigg(-\frac{1}{2\sigma^2_{\mathrm{post}}}(x - \mu_{\mathrm{post}})^2 \bigg),$$

where

$$\sigma^2_{\mathrm{post}} = \left(2^{-1} + \begin{pmatrix} 2 & 1 \end{pmatrix} \gamma^{-2} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \end{pmatrix} \right)^{-1} = \frac{1}{\frac{1}{2} + 5\gamma^{-2}}$$

and

$$\mu_{\mathrm{post}} = \sigma^2_{\mathrm{post}} \bigg(\begin{pmatrix} 2 & 1 \end{pmatrix} \gamma^{-2} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} + 2^{-1} \cdot 0 \bigg) = \frac{1}{\frac{1}{2} + 5\gamma^{-2}} \cdot \frac{4}{\gamma^2} = \frac{4}{\frac{1}{2}\gamma^2 + 5}.$$

Clearly

$$\mu_{\mathrm{post}} \xrightarrow{\gamma \downarrow 0} \frac{4}{5} \quad \text{and} \quad \sigma_{\mathrm{post}} \xrightarrow{\gamma \downarrow 0} 0,$$

so the posterior converges weakly to the Dirac distribution

$$\delta_{\frac{4}{5}}(B) = \begin{cases} 1 & \text{if } \frac{4}{5} \in B, \\ 0 & \text{otherwise,} \end{cases} \qquad B \subset \mathbb{R},$$

as $\gamma \downarrow 0$.

2. (a) Since $x_0 \sim \mathcal{N}(1, 1)$ and $\xi \sim \mathcal{N}(0, 1)$,

$$x_1 = x_0 + \xi \sim \mathcal{N}(1, 2),$$

where we made use of the fact that an affine transformation of two Gaussian random variables is Gaussian as well as the identities $\mathbb{E}[x_0 + \xi] = \mathbb{E}[x_0] + \mathbb{E}[\xi] = 1 + 0 = 1$ and $\mathrm{Var}(x_0 + \xi) = \mathrm{Var}(x_0) + \mathrm{Var}(\xi) = 1 + 1 = 2$ (note that this latter equality is valid since we assumed $x_0 \perp \xi$).

(b) Since $x_1 \sim \mathcal{N}(1, 2)$ and $y_1 = 1$, we can obtain the mean and covariance of $x_1|y_1 = 1$ using the Kalman filter formulas. In this case, the Kalman gain is given by

$$K = 2 \cdot 1 \cdot (1 \cdot 2 \cdot 1 + 1)^{-1} = 2 \cdot \tfrac{1}{3} = \tfrac{2}{3}.$$

Therefore the analysis mean is given by

$$m_1 = 1 + \tfrac{2}{3} \cdot (1 - 1) = 1$$

and the analysis variance is

$$\sigma_1^2 = 2 - \tfrac{2}{3} \cdot 1 \cdot 2 = \tfrac{2}{3}.$$

Hence we conclude that

$$x_1 | y_1 = 1 \sim \mathcal{N}(1, \tfrac{2}{3}).$$

3. The update formulae for the Kalman filter algorithm can be summarized as follows.

Given: Initial distribution for $x_0 \sim \mathcal{N}(m_0, C_0)$, where $m_0 \in \mathbb{R}^d$ and $C_0 \in \mathbb{R}^{d \times d}$ symmetric, positive definite.
for $j = 0, 1, 2, \ldots$**, do**

Prediction step:

$$\begin{aligned} \widehat{m}_{j+1} &= M m_j \\ \widehat{C}_{j+1} &= M C_j M^{\mathrm{T}} + \Sigma \end{aligned}$$

Correction step:

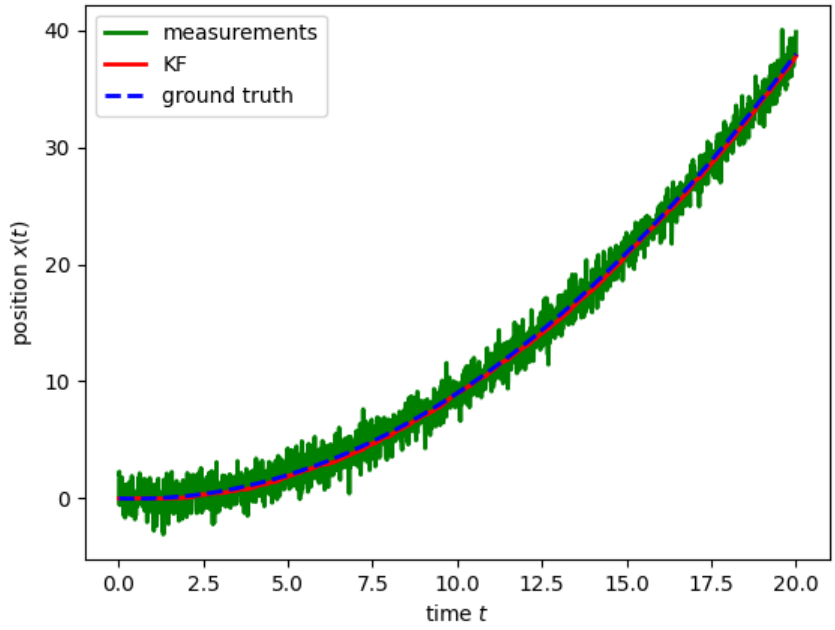
$$\begin{aligned} K_{j+1} &= \widehat{C}_{j+1} H^{\mathrm{T}} (H \widehat{C}_{j+1} H^{\mathrm{T}} + \Gamma)^{-1} \\ m_{j+1} &= \widehat{m}_{j+1} + K_{j+1}(y_{j+1} - H \widehat{x}_{j+1}) \\ C_{j+1} &= \widehat{C}_{j+1} - K_{j+1} H \widehat{C}_{j+1} \end{aligned}$$

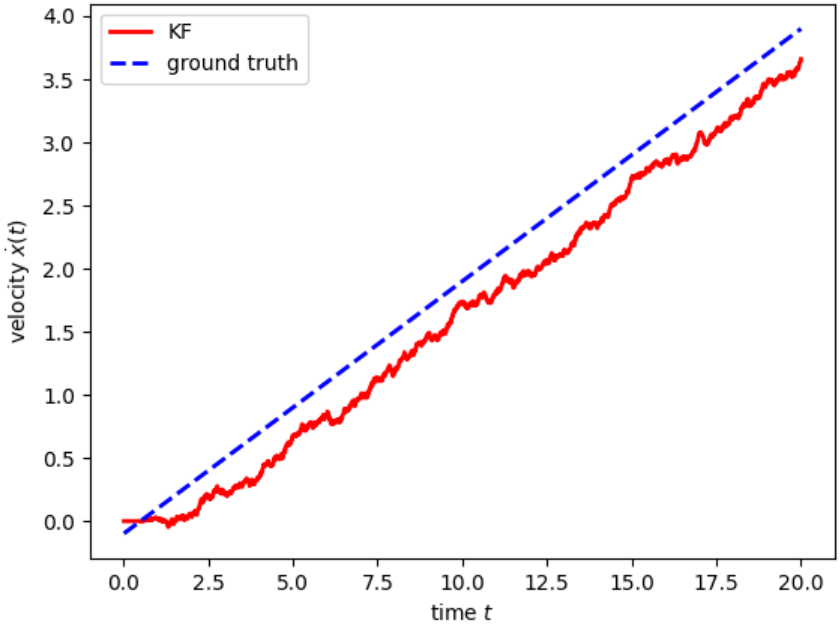
end for
Output: Predicted distributions $\mathcal{N}(\widehat{m}_{j+1}, \widehat{C}_{j+1})$ and filtering distributions for $x_{j+1} | y_1, \ldots, y_{j+1} \sim \mathcal{N}(m_{j+1}, C_{j+1})$, $j = 0, 1, 2, \ldots$

Solution using Python

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

# Initial conditions
xfilt = np.zeros((2,1)) # initial state
Cfilt = np.zeros((2,2)) # no uncertainty in the initial state

# Initialize the problem set-up
dt = .01 # time step
M = np.array([[1,dt],[0,1]])
H = np.array([[1,0]])
Sigma = np.array([[.25*dt**4,.5*dt**3],[.5*dt**3,dt**2]]) # innovation term
Gamma = 1 # observation noise level
```

```
true = lambda t: .1*(t**2-t) # true path
t = np.arange(dt,20+dt,dt) # discretize the time interval
n = t.size # number of time steps
meas = true(t) + np.sqrt(Gamma)*np.random.normal(size=t.shape) # generate noisy
                                                               # measurement data
state = []

# Kalman filter
for iter in range(n):
    y = meas[iter]
    # Prediction
    xpred = M@xfilt # predicted mean
    Cpred = M@Cfilt@M.T + Sigma # predicted covariance
    # Correction
    K = Cpred@H.T@np.linalg.inv(H@Cpred@H.T+Gamma) # Kalman gain
    xfilt = xpred + K@(y-H@xpred) # filtered (posterior) mean
    Cfilt = Cpred - K@H@Cpred # filtered (posterior) covariance
    # Store the state
    state.append(xfilt)
state = np.array(state)

# Visualize the results
fig,ax = plt.subplots()
ax.plot(t,meas,'g',linewidth=2,label='measurements')
ax.plot(t,state[:,0],'r',linewidth=2,label='KF')
ax.plot(t,true(t),'b--',linewidth=2,label='ground truth')
ax.legend(loc='best')
ax.set_xlabel('time $t$')
ax.set_ylabel('position $x(t)$')
plt.show()

fig,ax = plt.subplots()
ax.plot(t,state[:,1],'r',linewidth=2,label='KF')
ax.plot(t,.1*(2*t-1),'b--',linewidth=2,label='ground truth')
ax.legend(loc='best')
ax.set_xlabel('time $t$')
ax.set_ylabel('velocity $\dot{x}(t)$')
plt.show()
```





4. The implementation is otherwise the same as the regular Kalman filter, with the following differences:

(i) Instead of updating the analytical formulas for the means and covariances of the states, we begin by drawing a sample of size $N = 100$ from the initial distribution $x_0 \sim \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0 & 0\\0 & 0\end{bmatrix}\right)$ which is just the list $x_0^{(j)} = \begin{bmatrix}0\\0\end{bmatrix}$, $j = 1, \ldots, N$, since we assume to have no uncertainty about the initial position of the particle). We set $k = 0$.

(ii) (*Prediction*) For each particle $x_k^{(j)}$, $j = 1, \ldots, N$, we compute the predicted locations at the next time step using the evolution model:

$$\widehat{x}_{k+1}^{(j)} = Mx_k^{(j)} + \xi_{k+1}^{(j)}, \quad \xi_{k+1}^{(j)} \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \Sigma), \quad j = 1, \ldots, N.$$

We compute the *sample covariance* of the prediction:

$$\widehat{C}_{k+1} = \frac{1}{N} \sum_{j=1}^{N} (\widehat{x}_{k+1}^{(j)} - \widehat{m}_{k+1})(\widehat{x}_{k+1}^{(j)} - \widehat{m}_{k+1})^{\mathrm{T}},$$

where $\widehat{m}_{k+1} = \frac{1}{N} \sum_{j=1}^{N} \widehat{x}_{k+1}^{(j)}$.

(iii) (*Correction*) We compute the Kalman gain using the sample covariance obtained in the prediction step:

$$K_{k+1} = \widehat{C}_{k+1} H^{\mathrm{T}} (H \widehat{C}_{k+1} H^{\mathrm{T}} + \Gamma)^{-1}$$

and assimilate the data y_{k+1} into our model by using the formula (compare with the regular Kalman filter / linear Gaussian setting)

$$x_{k+1}^{(j)} = \widehat{x}_{k+1}^{(j)} + K_{k+1}(y_{k+1} - H\widehat{x}_{k+1}^{(j)}), \quad j = 1, \ldots, N.$$

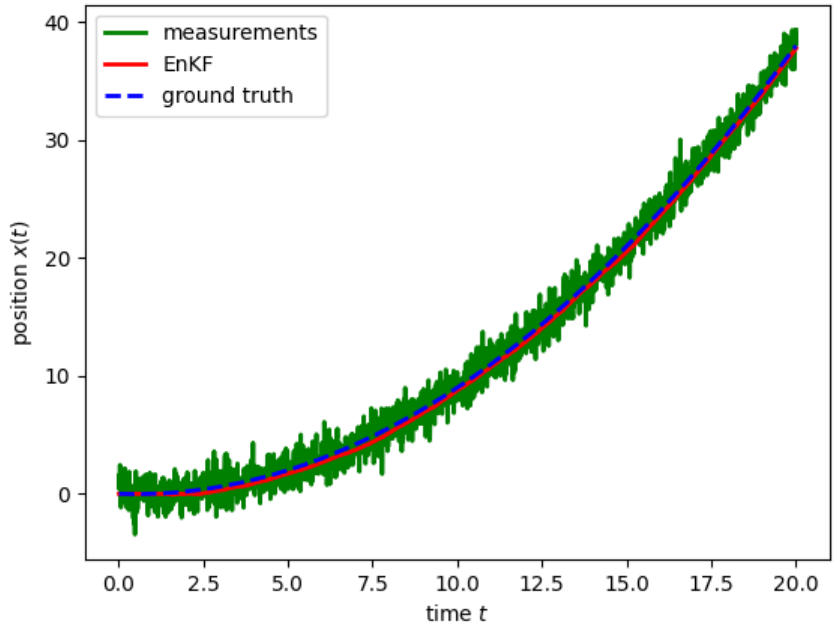
Set $k \leftarrow k + 1$ and return to step (ii).

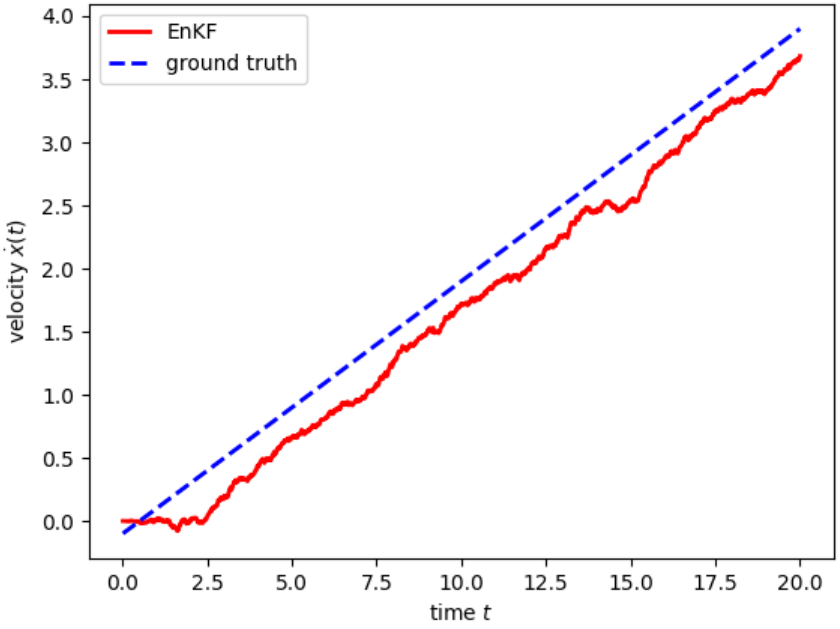
The output of the algorithm are the ensembles $\{x_k^{(j)}\}_{j=1}^N$, $k = 0, \ldots, 2000$. We plot the ensemble averages $(t_k, \overline{x_k})$ for $k = 1, \ldots, 2000$.

Solution using Python

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

# Initialize the problem set-up
dt = .01 # time step
M = np.array([[1,dt],[0,1]])
H = np.array([[1,0]])
Sigma = np.array([[.25*dt**4+1e-10,.5*dt**3],[.5*dt**3,dt**2+1e-10]]) # innovation term
Gamma = 1 # observation noise level
true = lambda t: .1*(t**2-t) # true path
t = np.arange(dt,20+dt,dt) # discretize the time interval
n = t.size # number of time steps
meas = true(t) + np.sqrt(Gamma)*np.random.normal(size=t.shape) # generate noisy
                                                               # measurement data

N = 100 # ensemble size
xfilt = np.zeros((2,N))
state = []
# Kalman filter
```

```
for iter in range(n):
    y = meas[iter]
    # Prediction
    xi = np.random.multivariate_normal((0,0),Sigma,size=N)
    xpred = M@xfilt+xi.T
    #mpred = np.mean(xpred,axis=1)
    Cpred = np.cov(xpred)
    # Correction
    K = Cpred@H.T@np.linalg.inv(H@Cpred@H.T+Gamma) # Kalman gain
    xfilt = xpred + K@(y-H@xpred) # filtered (posterior) mean
    # Store the state
    state.append(xfilt)
res = np.mean(np.array(state),axis=2)

# Visualize the results
fig,ax = plt.subplots()
ax.plot(t,meas,'g',linewidth=2,label='measurements')
ax.plot(t,res[:,0],'r',linewidth=2,label='KF')
ax.plot(t,true(t),'b--',linewidth=2,label='ground truth')
ax.legend(loc='best')
ax.set_xlabel('time $t$')
ax.set_ylabel('position $x(t)$')
plt.show()

fig,ax = plt.subplots()
ax.plot(t,res[:,1],'r',linewidth=2,label='KF')
ax.plot(t,.1*(2*t-1),'b--',linewidth=2,label='ground truth')
ax.legend(loc='best')
ax.set_xlabel('time $t$')
ax.set_ylabel('velocity $\dot{x}(t)$')
plt.show()
```


measurements
EnKF
ground truth
position $x(t)$
time t
0
10
20
30
40
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


EnKF
ground truth
velocity $\dot{x}(t)$
time t
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0