SDS386D: Monte Carlo Methods in Statistics **Cesar N. Yahia**
homework 6 Prof. S.G. Walker

Take $x_0 = 1$ and then generate data $y_{1:50}$ using the model $p(x_t|x_{t-1}) = N(x_t|x_{t-1}, 1)$ and $p(y_t|x_t) = N(y_t|x_t, 1)$
First use the Kalman filter updates to estimate $p(x_{50}|y_{1:50})$. Then re-estimate using the particle filter with importance sampling density as $q(x_t|x_{0:t-1}, y_{1:t}) = p(x_t|x_{t-1})$.

The resulting estimate for $p(x_t|y_{1:t})$ is $\hat{p}(x_t|y_{1:t}) = \sum_{i=1}^{N} \tilde{w}_t^i \delta_{x_t^i}$. In other words, since $\tilde{w}$ corresponds to the probability of the sequence $x_{0:t}$ resulting from a specific initial particle, to find the probability of being in state x_t, you sum the sequence probabilities $\tilde{w}$ over all sequences that end in x_t. This is opposed to using $\delta_{x_{0:t}}$ as in the notes to find $\hat{p}(x_{0:t}|y_{1:t}) = \sum_{i=1}^{N} \tilde{w}_t^i \delta_{x_{0:t}^i}$. In practice, since x is a continuous random variable, it will not happen that 2 or more different sequences end up at the same point, so the probability of ending up in a point x_t will be the same as the probability of the sequence $x_{0:t}$ that leads to ending up in x_t.

Kalman filter:
After generating an observation from the true model $p(x_t|x_{t-1}) = N(x_t|x_{t-1}, 1)$ and $p(y_t|x_t) = N(y_t|x_t, 1)$. The Kalman filter aims to re-estimate the true state given the observations $y_{1:t}$ only. This estimate is obtained by updating the mean and variance of the true state distribution given the observations. In this case, the mean and variance fully describe $p(x_t|y_{1:t})$ since we are assuming that the probability of being in a particular state x_t is always normally distributed. The Kalman filter update equations for mean and variance based on the observations are given below:

$$\mu_t = \frac{y_t(\sigma^2 + \psi_{t-1}^2) + \tau^2 \mu_{t-1}}{\sigma^2 + \psi_{t-1}^2 + \tau^2} \tag{1}$$

$$\psi_t^2 = \frac{\tau^2(\sigma^2 + \psi_{t-1}^2)}{\sigma^2 + \psi_{t-1}^2 + \tau^2} \tag{2}$$

where μ_t and ψ_t are the updated mean and variance at time t.

Particle filter:
For the particle filter, we use an importance sampling proposal density of $q(x_t|x_{0:t-1}, y_{1:t}) = p(x_t|x_{t-1})$. The particle weight corresponding to a specific sequence of states is given by $w_t = \frac{P(x_{0:t}|y_{1:t})}{q(x_{0:t}|y_{1:t})}$. The resulting update of weights, given an importance sampling proposal that decomposes, is given by $w_{t+1} = \frac{P(y_{t+1}|x_{t+1})P(x_{t+1}|x_t)}{q(x_{t+1}|x_{0:t-1}, y_{1:t})P(y_{t+1}|y_{1:t})} w_t \propto \frac{P(y_{t+1}|x_{t+1})P(x_{t+1}|x_t)}{q(x_{t+1}|x_{0:t-1}, y_{1:t})} w_t$. Therefore, by taking $q(x_t|x_{0:t-1}, y_{1:t}) = p(x_t|x_{t-1})$, we have that $w_{t+1} \propto P(y_{t+1}|x_{t+1})w_t$. Note that in the preceding discussion the index corresponding to a specific particle sequence was not written for notational convenience. Consider the normalized weights $\tilde{w}_{t+1}^j = \frac{w_{t+1}^j}{\sum_{j=1}^{N} w_{t+1}^j}$ where N is the number of particles. The probability $\hat{p}(x_t|y_{1:t}) = \sum_{i=1}^{N} \tilde{w}_t^i \delta_{x_t^i}$.

Therefore, to implement the PF, at each iteration (1) sample x_{t+1} from $p(x_t|x_{t-1})$, (2) update weights such that $w_{t+1} \propto P(y_{t+1}|x_{t+1})w_t$, (3) normalize weights $\tilde{w}_{t+1}^j = \frac{w_{t+1}^j}{\sum_{j=1}^{N} w_{t+1}^j}$, and (4) the desired probability is $\hat{p}(x_t|y_{1:t}) = \sum_{i=1}^{N} \tilde{w}_t^i \delta_{x_t^i}$.

The code for (1) sampling from model, (2) Kalman filter, and (3) particle filter is shown below:

```
"""
This code is for the sequential Monte Carlo
problem in assignment 6

@cnyahia
"""

import numpy.random as nprand
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import norm


# define the sampling from p(xt|xt-1)
def pxx(x_prev, stdev):
        """
        implements sampling P(xt|xt-1)=N(xt-1,var)
        This is normal around xt with variance of 1
        :param x_prev: previous sample
        :param stdev: standard deviation
        :return: sample
        """
        sample = nprand.normal(loc=x_prev, scale=stdev)
        return sample


# define the sampling for p(yt|xt)
def pyx(x, stdev):
        """
        implements sampling P(yt|xt)=N(yt|xt,1)
        :param x: current value of x
        :param stdev: observation density standard dev.
        :return: sample
        """
        sample = nprand.normal(loc=x, scale=stdev)
        return sample


# define update of mean from Kalman filter equations
def update_mean(observation, prev_mean, prev_var, pred_var, obs_var):
        """
        update of mean in KF equations for P(xt|y1:t)
        :param observation: observation yt
        :param prev_mean: previous mean of P(xt|y1:t)
        :param prev_var: previous var of P(xt|y1:t)
        :param pred_var: prediction variance of P(xt|xt-1)
```

```
        :param obs_var: observation variance of P(yt|xt)
        :return: new_mean
        """
        numerator = observation * (pred_var + prev_var) + obs_var * prev_mean
        denom = pred_var + prev_var + obs_var
        new_mean = float(numerator) / denom
        return new_mean


# define update of variance for Kalman filter equations
def update_var(prev_var, pred_var, obs_var):
        """
        update variance in KF equations for P(xt|y1:t)
        :param prev_var: previous variance of P(xt|y1:t)
        :param pred_var: prediction variance of P(xt|xt-1)
        :param obs_var: observation variance of P(yt|xt)
        :return: new_var
        """
        numerator = obs_var * (pred_var + prev_var)
        denominator = pred_var + prev_var + obs_var
        new_var = float(numerator)/denominator
        return new_var


# implement algorithm
if __name__ == '__main__':
        # actual initial value
        x = 1

        y_samples = list()  # generated observations from model
        # model parameters
        pred_var = 1
        obs_var = 1

        # KF initial distribution
        mean = 1
        var = 1

        # generate initial samples for PF
        particles = list()
        particle_weights = list()
        normalized_weights = list()
        N = 100000  # number of particles
        for particle in range(1, N+1):
                particles.append(nprand.normal(loc=1, scale=1))
                # initialize with uniform weight
                particle_weights.append(1.0 / N)
```

```
for idx in range(1, 50 + 1):
        # Generate observations
        x = pxx(x, np.sqrt(pred_var))
        y = pyx(x, np.sqrt(obs_var))
        y_samples.append(y)

        # Kalman filter
        mean = update_mean(y, mean, var, pred_var, obs_var)
        var = update_var(var, pred_var, obs_var)

        # particle filter
        # get new particles at t+1 using proposal
        for key, particle in enumerate(particles):
                particles[key] = pxx(particle, np.sqrt(pred_var))

        # update particle weights
        for key, particle in enumerate(particles):
                particle_weights[key] = particle_weights[key] *
                                norm.pdf(y, particle, np.sqrt(obs_var))

        # re-normalize particles
        sum_weights = sum(particle_weights)
        normalized_weights = [float(p_weight)/sum_weights for
                                        p_weight in particle_weights]

# the distribution P(x50|y1:50) is normal(mean, var)
# mean, var what results at the end from updates above
print("... plotting Kalman filter density ...")
print('mean: ', mean)
print('std: ', np.sqrt(var))
# plot Kalman filter resulting density
x_range = np.arange(mean-10, mean+10, 0.001)
plt.plot(x_range, norm.pdf(x_range, mean, np.sqrt(var)), 'g')
# plot normalized weights from particle filter
print("... plotting PF results ...")
plt.bar(particles, normalized_weights, width=0.65)
plt.title("density from KF and PF updates")
plt.xlabel("x")
plt.ylabel("density f(x)")
plt.tight_layout()
plt.savefig('KfPf.png')
plt.show()
```

The results are given below, a discussion follows:

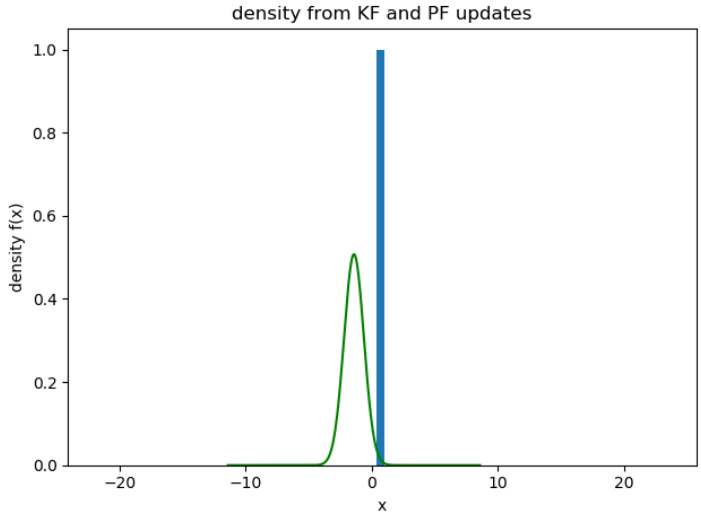

Figure 1: comparison of Kalman filter and particle filter density estimates for $N = 1000$

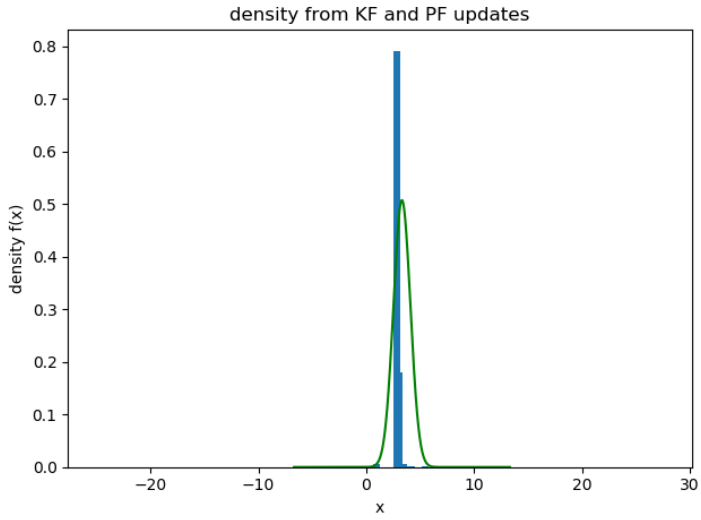

Figure 2: comparison of Kalman filter and particle filter density estimates for $N = 10,000$

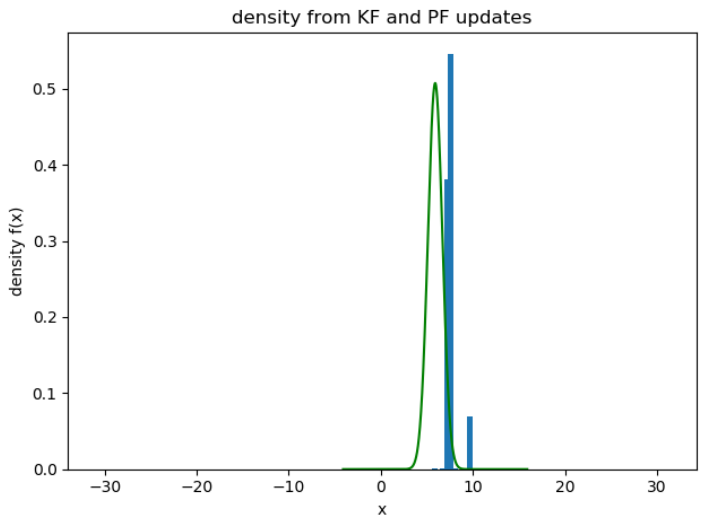

Figure 3: comparison of Kalman filter and particle filter density estimates for $N = 100,000$

Discussion:
As shown in the figures above, different number of particles N was chosen and the resulting density estimates from the particle filter (bar plot) were compared to the Kalman filter updated Gaussian density.

The particle filter sample degeneracy problem appears when the number of particles is not high. Specifically, for $N = 1000$, we can see that all but one of the weights end up being zero. This indicates that the particles for which the weights collapsed to zero were not good compared to the remaining particle that has a weight of 1. However, it may be that the remaining particle with weight 1 is itself not very good, and if we had more particles remaining at this time step t, our particle weights will be on values closer to the KF mean. In addition, having only one particle with a weight of 1 is a problem when we are trying to evaluate an integral $I_t = \int h(x_t)P(x_t|y_{1:t})dx_t \approx \sum_{j=1}^{N} h(x_t^j)\tilde{w}_t^j$ since our estimated density is concentrated at one point, and we are not able to capture the true density $P(x_t|y_{1:t})$.

When we used a larger number of initial particles, $N = 10,000$ and $N = 100,000$, a larger number of particles with positive weights survived until $t = 50$. It can be observed in the figures that for $N = 100,000$, there was a larger number of particles approximating $P(x_t|y_{1:t})$ compared to $N = 10,000$. However, handling vectors of dimension $100,000$ was not very efficient, computing the particle filter density using $N = 10,000$ was considerably faster than $N = 100,000$. Looking at the results and the computation time, I would go for $N = 10,000$ as a good number of particles.