

Lab 5 : Kalman filtering

Jeroen Olieslagers
Richard-John Lin

Center for Data Science

10/04/2022



Analytical vs empirical

Analytical = using information given by the model WITHOUT data

Empirical = using information given by the data WITHOUT model assumptions

$$\gamma(h) = \mathbb{E}[x_t x_{t-h}]$$

$$\hat{\gamma}(h) = \sum_{t=h+1}^T (x_t - \bar{x})(x_{t-h} - \bar{x})$$

$$\rho(h) = \frac{\mathbb{E}[x_t x_{t-h}]}{\mathbb{E}[x_t x_t]} = \frac{\gamma(h)}{\gamma(0)}$$

$$\hat{\rho}(h) = \frac{\sum_{t=h+1}^T (x_t - \bar{x})(x_{t-h} - \bar{x})}{\sum_{t=1}^T (x_t - \bar{x})(x_t - \bar{x})} = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}$$

Analytical MA ACF

$$v_t = \frac{w_t + w_{t-1} + w_{t-2}}{3}$$

$$\begin{aligned}\gamma_v(0) &= \mathbb{E}[v_t \times v_t] = \mathbb{E}\left[\left(\frac{w_t + w_{t-1} + w_{t-2}}{3}\right)^2\right] \\ &= \frac{1}{9} \left(\mathbb{E}[w_t^2 + w_{t-1}^2 + w_{t-2}^2] + \text{cross terms} \right) \\ &= \frac{1}{9} \left(\sigma^2 + \sigma^2 + \sigma^2 \right) = \frac{\sigma^2}{3}\end{aligned}$$

Analytical MA ACF

$$v_t = \frac{w_t + w_{t-1} + w_{t-2}}{3}$$

$$\begin{aligned}\gamma_v(1) &= \mathbb{E}[v_t \times v_{t-1}] \\ &= \mathbb{E}\left[\left(\frac{w_t + w_{t-1} + w_{t-2}}{3}\right)\left(\frac{w_{t-1} + w_{t-2} + w_{t-3}}{3}\right)\right] \\ &= \frac{1}{9}\left(\mathbb{E}[w_{t-1}^2 + w_{t-2}^2] + \text{cross terms}\right) \\ &= \frac{1}{9}\left(\sigma^2 + \sigma^2\right) = \frac{2\sigma^2}{9}\end{aligned}$$

Analytical MA ACF

$$v_t = \frac{w_t + w_{t-1} + w_{t-2}}{3}$$

$$\begin{aligned}\gamma_v(2) &= \mathbb{E}[v_t \times v_{t-2}] \\ &= \mathbb{E}\left[\left(\frac{w_t + w_{t-1} + w_{t-2}}{3}\right)\left(\frac{w_{t-2} + w_{t-3} + w_{t-4}}{3}\right)\right] \\ &= \frac{1}{9}\left(\mathbb{E}[w_{t-2}^2] + \text{cross terms}\right) \\ &= \frac{1}{9}\left(\sigma^2\right) = \frac{\sigma^2}{9}\end{aligned}$$

Analytical MA ACF

$$v_t = \frac{w_t + w_{t-1} + w_{t-2}}{3}$$

$$\begin{aligned}\gamma_v(3+) &= \mathbb{E}[v_t \times v_{t-3+}] \\ &= \mathbb{E}\left[\left(\frac{w_t + w_{t-1} + w_{t-2}}{3}\right)\left(\frac{w_{t-3+} + w_{t-4+} + w_{t-5+}}{3}\right)\right] \\ &= \frac{1}{9}(\text{cross terms}) \\ &= \frac{1}{9}(0) = 0\end{aligned}$$

Normalization

The implementation from statsmodels (which is very commonplace) is to normalize the ACF by dividing by the variance ($= \gamma(0)$):

$$\rho(h) = \frac{\rho(h)}{\rho(0)}$$

$$\rho_v(0) = \frac{\rho_v(0)}{\rho_v(0)} = 1$$

$$\rho_v(1) = \frac{\rho_v(1)}{\gamma_v(0)} = \frac{2}{3}$$

$$\rho_v(2) = \frac{\gamma_v(2)}{\gamma_v(0)} = \frac{1}{3}$$

$$\rho_v(3+) = \frac{\gamma_v(3+)}{\gamma_v(0)} = 0$$

Motivation

Kalman filtering are widely used in signal processing, to extract a long series from noisy and / or incomplete measurements.

- ▶ Used for the first time during the Apollo mission in the 1960s
- ▶ Tracking noisy targets : object detection, financial time series, monitoring chemical reactions, noise removal from cardiac signal...

Intuition : g-h filter

The g-h filter considers a position and a velocity, and predicts :

$$x_k = \hat{x}_{k-1} + \Delta T \hat{v}_{k-1}$$

$$v_k = \hat{v}_{k-1}.$$

Given measurements z_k , the filter updates the values of the estimations, with fixed weights g and h :

$$\hat{x}_k = x_k + g(z_k - x_k)$$

$$\hat{v}_k = v_k + \frac{h}{\Delta T}(z_k - x_k).$$

g-h filter

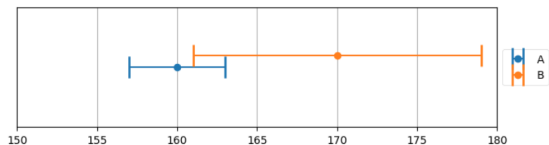


Figure 1: Parameter estimation given confidence bounds

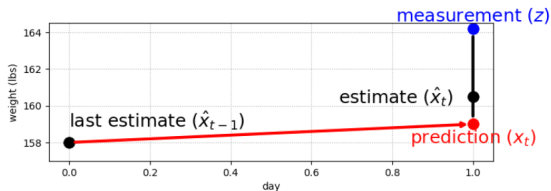
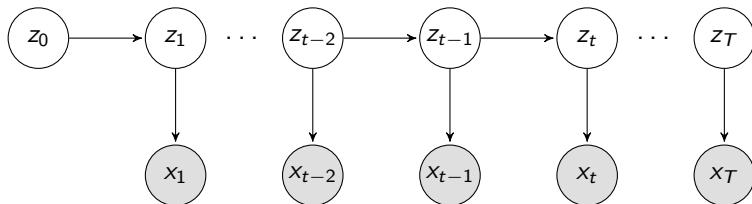


Figure 2: Estimation

Problem formulation : LDS



$$z_t = Az_{t-1} + w_t$$

$$x_t = Cz_t + v_t,$$

with :

$$w_t \sim \mathcal{N}(0, Q), \quad v_t \sim \mathcal{N}(0, R), \quad z_0 \sim \mathcal{N}(\mu_0, \Sigma).$$

It is usual to assume some structure of the noise, e.g. Q is often chosen to be diagonal.

Filtering

The aim of filtering is to characterize the distribution at the present time, given all the past observations.

For an LDS :

- Prediction : $P(z_i | x_{1:i}) \sim \mathcal{N}(\mu_{i|i-1}, \Sigma_{i|i-1})$.

$$\mu_{i|i-1} = A\mu_{i-1|i-1}$$

$$\Sigma_{i|i-1} = A\Sigma_{i-1|i-1}A^\top + Q.$$

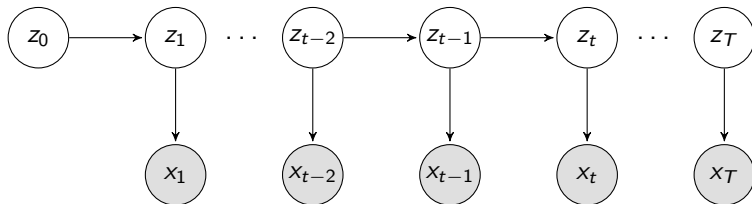
- Update : Include observation x_n

$$\mu_{i|i} = \mu_{i|i-1} + K_i (x_i - C\mu_{i|i-1})$$

$$\Sigma_{i|i} = (I - K_i C)\Sigma_{i|i-1}$$

$$K_i = \Sigma_{i|i-1} C^\top \left(C\Sigma_{i|i-1} C^\top + R \right)^{-1}$$

Proof



$$p(z_{1:i}, x_{1:i}) = p(z_{1:i-1}, x_{1:i})p(z_i|z_{i-1})p(x_i|z_i)$$

$$p(z_{1:i}|x_{1:i}) = p(z_{1:i-1}|x_{1:i-1}) \frac{p(z_i|z_{i-1})p(x_i|z_i)}{p(x_i|x_{1:i-1})}$$

where

$$p(x_i|x_{1:i-1}) = \int p(z_{1:i-1}|x_{1:i-1})p(z_i|z_{i-1})p(x_i|z_i) dz_{i-1} dz_i.$$

Proof

$$\begin{aligned}p(z_{1:i}, x_{1:i}) &= p(z_{1:i-1}, x_{1:i})p(z_i|z_{i-1})p(x_i|z_i) \\p(z_{1:i}|x_{1:i}) &= p(z_{1:i-1}|x_{1:i-1})\frac{p(z_i|z_{i-1})p(x_i|z_i)}{p(x_i|x_{1:i-1})}\end{aligned}$$

where

$$p(x_i|x_{1:i-1}) = \int p(z_{1:i-1}|x_{1:i-1})p(z_i|z_{i-1})p(x_i|z_i) dz_{i-1} dz_i.$$

Often, we write the equation only depending on the last variable, marginalizing out $x_{1:i-1}$.

$$p(z_i|x_{1:i}) = \frac{p(x_i|z_i)p(z_i|x_{1:i-1})}{p(x_i|x_{1:i-1})}, \quad (1)$$

and

$$p(z_i|x_{1:i-1}) = \int p(z_i|z_{i-1})p(z_{i-1}|x_{1:i-1}) dz_{i-1}. \quad (2)$$

Notes

- ▶ Equation (2) is called the prediction and (1) is called the update.
- ▶ Up to now, we have not used any assumptions besides the Markov structure.
- ▶ In the general case the integral is not tractable, but for gaussians, we can derive an analytical solution !

Smoothing

The value predicted is not the best possible as we discard information. We now aim to compute the distribution for a given time, with all the observations.

Knowing the values of the filtering, we only need to further sweep "backwards" to include the future observations.

In an LDS :

$$\begin{aligned}\mu_{i|T} &= \mu_{i|i} + F_i(\mu_{i+1|T} - \mu_{i+1|i}) \\ \Sigma_{i|T} &= F_i(\Sigma_{i+1|T} - \Sigma_{i+1|i})F_i^\top + \Sigma_{i|i} \\ F_i &= \Sigma_{i|i}A^\top\Sigma_{i+1|i}^{-1}\end{aligned}$$

Proof

The joint distribution can be written :

$$\begin{aligned} p(z_{1:T}|x_{1:T}) &= p(z_T|x_{1:T}) \prod_{i=1}^{T-1} p(z_i|z_{i+1}, x_{1:T}) \\ &= p(z_T|x_{1:T}) \prod_{i=1}^{T-1} p(z_i|z_{i+1}, x_{1:i}). \end{aligned} \quad (3)$$

With Bayes' rule :

$$p(z_i|z_{i+1}, x_{1:i}) = \frac{p(z_{i+1}|z_i)p(z_i|x_{1:i})}{p(z_{i+1}|x_{1:i})}$$

Then, the quantity $p(z_i|x_{1:i})$ was computed in the forward pass, and $p(z_{i+1}|x_{1:n})$ can be obtained by marginalizing the backward pass.

Proof

With Bayes' rule :

$$p(z_i | z_{i+1}, x_{1:i}) = \frac{p(z_{i+1} | z_i) p(z_i | x_{1:i})}{p(z_{i+1} | x_{1:i})}$$

Marginalizing out all variables but one in 3, we obtain :

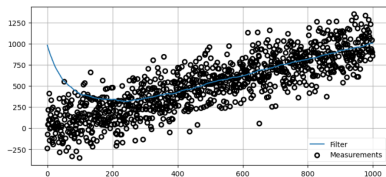
$$p(z_i | x_{1:T}) = p(z_i | x_{1:i}) \int \frac{p(z_{i+1} | z_i) p(z_{i+1} | x_{1:T})}{p(z_{i+1} | x_{1:i})} dz_{i+1},$$

and

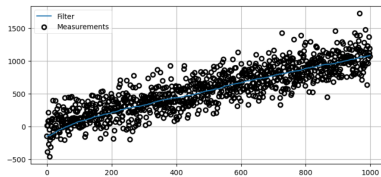
$$\begin{aligned} p(z_{i+1} | x_{1:T}) &= \int p(z_{i+1}, z_i | x_{1:i}) dz_i \\ &= \int p(z_{i+1} | z_i) p(z_i | x_{1:i}) dz_i. \end{aligned}$$

Bad initialisations

Although with the right parameters and with enough data, a Kalman filter can recover from a bad initial guess, under noisy signals it may take a long time. If better initial conditions are known, for example the first data point, they can reduce this transition period significantly.



(a) Bad initial conditions



(b) Better initial conditions

Extensions

Kalman Filter does not work well with non linearities :

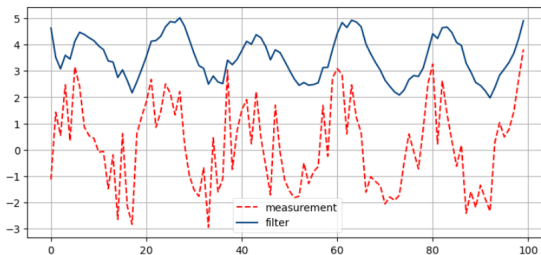


Figure 4: Kalman filtering on $2 \sin\left(\frac{i}{3}\right) + w_i$, $w_i \sim \mathcal{N}(0, 1.2)$

Extensions

- ▶ Unscented kalman filter
Uses sigma points to efficiently approximate a distribution
- ▶ Extended kalman filters
Linearize the non linear equations at the point of the current estimate, then apply the linear Kalman filter

References

- ▶ [Kalman and Bayesian Filters in Python](#), Roger R. Labbe
- ▶ A Tutorial on Particle Filtering and Smoothing : Fifteen years later, Arnaud Doucet and Adam M. Johansen

The dataset

Data

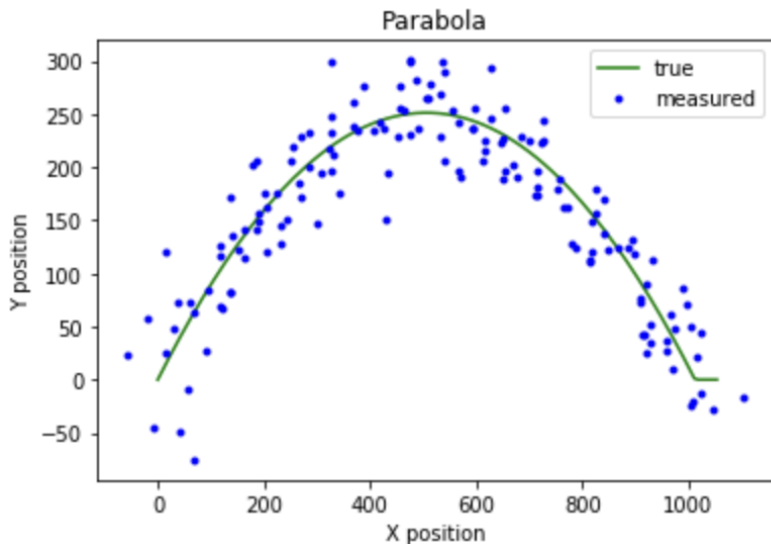
We will use a common physics problem with a twist. This example will involve firing a ball from a cannon at a 45-degree angle at a velocity of 100 units/sec. We have a camera that will record the ball's position (pos_x, pos_y) from the side every second. The positions measured from the camera (\hat{pos}_x, \hat{pos}_y) have significant measurement error.

Latent Variable $z = [pos_x, pos_y, V_x, V_y]$

Observed Variable $x = [\hat{pos}_x, \hat{pos}_y, \hat{V}_x, \hat{V}_y]$

Reference: <http://greg.czerniak.info/guides/kalman1/>

The dataset



Gaussian review

Review on Gaussian marginal and conditional distributions

Assume

$$z = [x^T y^T]^T$$
$$z = \begin{bmatrix} x \\ y \end{bmatrix} \sim N \left(\begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix} \right)$$

then the marginal distributions are

$$x \sim N(a, A)$$
$$y \sim N(b, B)$$

and the conditional distributions are

$$x|y \sim N(a + CB^{-1}(y - b), A - CB^{-1}C^T)$$
$$y|x \sim N(b + C^T A^{-1}(x - a), B - C^T A^{-1}C)$$

important take away: given the joint Gaussian distribution we can derive the conditionals

Linear Dynamical System review

Review on Linear Dynamical System

Latent variable:

$$z_n = Az_{n-1} + w$$

Observed variable:

$$x_n = Cz_n + v$$

Gaussian noise terms:

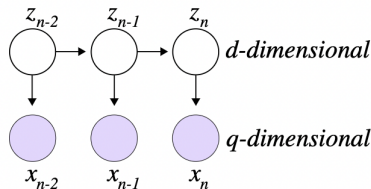
$$w \sim N(0, \Gamma)$$

$$v \sim N(0, \Sigma)$$

$$z_0 \sim N(\mu_0, \Gamma_0)$$

As a consequence, z_n , x_n and their joint distributions are Gaussian so we can easily compute the marginals and conditionals.

Linear Dynamical System review



right now n depends only on what was one time step back $n - 1$ (Markov chain)

Given the graphical model of the LDS we can write out the joint probability for both temporal sequences:

$$P(\mathbf{z}, \mathbf{x}) = P(z_0) \prod_{n=1 \dots N} P(z_n | z_{n-1}) \prod_{n=0 \dots N} P(x_n | z_n)$$

all probabilities are implicitly conditioned on the parameters of the model

Filtering

Kalman


We want to infer the latent variable z_n given the observed variable x_n .

$$P(z_n | x_1, \dots, x_n, x_{n+1}, \dots, x_N) \sim N(\hat{\mu}_n, \hat{V}_n)$$

Forward: Filtering

obtain estimates of latent by running the filtering from $n = 0, \dots, N$

prediction given latent space parameters



$$z_n^{pred} \sim N(\mu_n^{pred}, V_n^{pred})$$

$$\mu_n^{pred} = A\mu_{n-1}$$

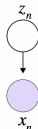
this is the prediction for z_n obtained simply by taking the expected value of z_{n-1} and projecting it forward one step using the transition probability matrix A

$$V_n^{pred} = AV_{n-1}A^T + \Gamma$$

same for the covariance taking into account the noise covariance Γ

Filtering

correction (innovation) from observation



project to observational space:

$$x_n^{pred} \sim N(C\mu_n^{pred}, CV_n^{pred}C^T + \Sigma)$$

correct prediction by actual data:

$$z_n^{innov} \sim N(\mu_n^{innov}, V_n^{innov})$$

$$\mu_n^{innov} = \mu_n^{pred} + K_n(x_n - C\mu_n^{pred})$$

$$V_n^{innov} = (I - K_nC)V_n^{pred}$$

Kalman gain matrix:

$$K_n = V_n^{pred}C^T(CV_n^{pred}C^T + \Sigma)^{-1}$$

we use the latent-only prediction to project it to the observational space and compute a correction proportional to the error $x_n - CAz_{n-1}$ between prediction and data, coefficient of this correction is the Kalman gain matrix

Filtering

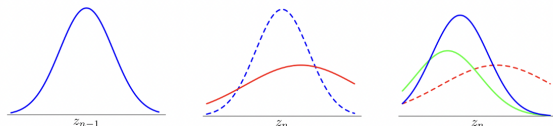


Figure 13.21 The linear dynamical system can be viewed as a sequence of steps in which increasing uncertainty in the state variable due to diffusion is compensated by the arrival of new data. In the left-hand plot, the blue curve shows the distribution $p(\mathbf{z}_{n-1}|\mathbf{x}_1, \dots, \mathbf{x}_{n-1})$, which incorporates all the data up to step $n-1$. The diffusion arising from the nonzero variance of the transition probability $p(\mathbf{z}_n|\mathbf{z}_{n-1})$ gives the distribution $p(\mathbf{z}_n|\mathbf{x}_1, \dots, \mathbf{x}_{n-1})$, shown in red in the centre plot. Note that this is broader and shifted relative to the blue curve (which is shown dashed in the centre plot for comparison). The next data observation \mathbf{x}_n contributes through the emission density $p(\mathbf{x}_n|\mathbf{z}_n)$, which is shown as a function of \mathbf{z}_n in green on the right-hand plot. Note that this is not a density with respect to \mathbf{z}_n and so is not normalized to one. Inclusion of this new data point leads to a revised distribution $p(\mathbf{z}_n|\mathbf{x}_1, \dots, \mathbf{x}_n)$ for the state density shown in blue. We see that observation of the data has shifted and narrowed the distribution compared to $p(\mathbf{z}_n|\mathbf{x}_1, \dots, \mathbf{x}_{n-1})$ (which is shown in dashed in the right-hand plot for comparison).

from Bishop (2006), chapter 13.3

if measurement noise is small and dynamics are fast -> estimation will depend mostly on observed data

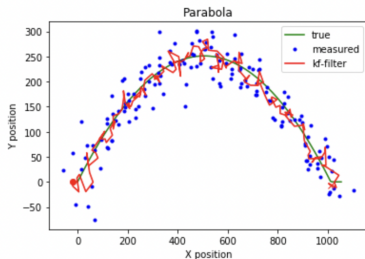
Filtering

Kalman Filter to predict true (latent) trajectory from observed variable using Pykalman API

```

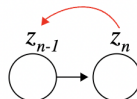
kf = KalmanFilter(n_dim_state=data.shape[1], n_dim_obs=data.shape[1])
# fit the model (use EM algorithm to estimate the parameters, we will not worry about this for now)
kf.em(data, n_iter=6)
# Kalman filtering
filtered_state_means, filtered_state_covariances = kf.filter(data)
fig = plot_kalman(x,y,nx,ny, filtered_state_means[:,0], filtered_state_means[:,1], "r-", "kf-filter")

```



Smoothing

Backward: Smoothing



obtain estimates by propagating from x_n back to x_1 using results of forward pass (μ_n^{innov} , V_n^{innov} , V_n^{pred})

$$N(z_n | \mu_n^{smooth}, V_n^{smooth})$$

$$\mu_n^{smooth} = \mu_n^{innov} + J_n(\mu_{n+1}^{smooth} - A\mu_n^{innov})$$

$$V_n^{smooth} = V_n^{innov} + J_n(V_{n+1}^{smooth} - V_{n+1}^{pred})J_n^T$$

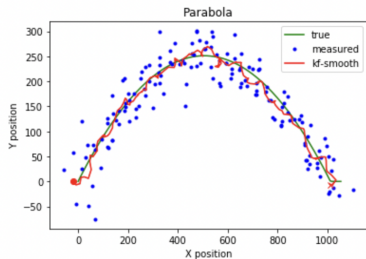
$$J_N = V_n^{innov} A^T (V_{n+1}^{pred})^{-1}$$

This gives us the final estimate for z_n .

$$\begin{aligned}\hat{\mu}_n &= \mu_n^{smooth} \\ \hat{V}_n &= V_n^{smooth}\end{aligned}$$

Smoothing

```
# Kalman smoothing
smoothed_state_means, smoothed_state_covariances = kf.smooth(data)
fig = plot_kalman(x,y,nx,ny, smoothed_state_means[:,0], smoothed_state_means[:,1], "r-", "kf-smooth")
```



Main exercise

Kalman Filter Implementation

In this part of the exercise, you will implement the Kalman filter. Specifically, you need to implement the following method:

- filter: assume learned parameters, perform the forward calculation
- smooth: assume learned parameters, perform both the forward and backward calculation

Exercise class

```
class MyKalmanFilter:
    """
    Class that implements the Kalman Filter
    """
    def __init__(self, n_dim_state=2, n_dim_obs=2):
        """
        @param n_dim_state: dimension of the laten variables
        @param n_dim_obs: dimension of the observed variables
        """
        self.n_dim_state = n_dim_state
        self.n_dim_obs = n_dim_obs
        self.transition_matrices = np.eye(n_dim_state)
        self.transition_offsets = np.zeros(n_dim_state) # you can ignore this one, not used
        self.transition_covariance = np.eye(n_dim_state)
        self.observation_matrices = np.eye(n_dim_obs, n_dim_state)
        self.observation_covariance = np.eye(n_dim_obs)
        self.observation_offsets = np.zeros(n_dim_obs) # you can ignore this one, not used
        self.initial_state_mean = np.zeros(n_dim_state)
        self.initial_state_covariance = np.eye(n_dim_state)
```

TODO 1

```
def filter(self, X):
    """
    Method that performs Kalman filtering
    @param X: a numpy 2D array whose dimension is [n_example, self.n_dim_obs]
    @output: filtered_state_means: a numpy 2D array whose dimension is [n_example, self.n_dim_state]
    @output: filtered_state_covariances: a numpy 3D array whose dimension is [n_example, self.n_dim_state, self.n_dim_state]
    """

    # validate inputs
    n_example, observed_dim = X.shape
    assert observed_dim==self.n_dim_obs

    # create holders for outputs
    filtered_state_means = np.zeros( (n_example, self.n_dim_state) )
    filtered_state_covariances = np.zeros( (n_example, self.n_dim_state, self.n_dim_state) )

    #####
    # TODO: implement filtering #
    #####

    return filtered_state_means, filtered_state_covariances
```

TODO 2

```
def smooth(self, X):
    """
    Method that performs the Kalman Smoothing
    @param X: a numpy 2D array whose dimension is [n_example, self.n_dim_obs]
    @output: smoothed_state_means: a numpy 2D array whose dimension is [n_example, self.n_dim_state]
    @output: smoothed_state_covariances: a numpy 3D array whose dimension is [n_example, self.n_dim_state,
    """

    # TODO: implement smoothing

    # validate inputs
    n_example, observed_dim = X.shape
    assert observed_dim==self.n_dim_obs

    # run the forward path
    mu_list, v_list = self.filter(X)

    # create holders for outputs
    smoothed_state_means = np.zeros( (n_example, self.n_dim_state) )
    smoothed_state_covariances = np.zeros( (n_example, self.n_dim_state, self.n_dim_state) )

    #####
    # TODO: implement smoothing #
    #####

    return smoothed_state_means, smoothed_state_covariances
```