SISEA — Filtrage de Kalman et modèles de Markov cachés

TP 1 : Filtre de Kalman (implémentation en Python)

jeudi 12 novembre 2020, via Zoom

L'objectif de ce TP est juste de mettre en œuvre le filtre de Kalman dans un cas simple afin de bien comprendre les différentes étapes de l'algorithme et leur articulation.

On considère un système linéaire gaussien dont le vecteur d'état bi–dimensionnel est noté X_k. Pour décrire l'évolution de l'état entre deux instants successifs, on dispose d'un modèle incertain (dont le rôle est de propager les informations et les incertitudes associées)

$$X_k = c\,R\,X_{k-1} + \sqrt{1-c^2}\,W_k \qquad \text{avec} \qquad R = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}$$

où la condition initiale X_1 est un vecteur aléatoire gaussien centré de matrice de covariance identité, et où les vecteurs aléatoires $(W_2, \cdots, W_k, \cdots)$ sont indépendants de la condition initiale X_1 et forment un bruit blanc gaussien centré de matrice de covariance identité. Ici, le coefficient c est compris (strictement) entre 0 et 1 et la matrice R (une matrice de rotation) vérifie $R\,R^* = I$.

(i) **Démontrer qu'à chaque instant l'état caché X_k est un vecteur aléatoire gaussien centré de matrice de covariance identité.**

Une réalisation particulière (mais inconnue) de la suite $(X_1, \cdots, X_n)$ produit une suite d'observations $(Y_1, \cdots, Y_n)$, selon un des scénarios décrit plus bas, et il s'agira dans chaque cas d'estimer l'état caché au vu des observations ainsi produites (et disponibles).

Pour le besoin des représentations graphiques qui permettront de se faire une idée qualitative du comportement du filtre de Kalman dans chacun des scénarios étudiés, la réalisation particulière de la suite des états cachés (inconnue en principe, et qu'il s'agit d'estimer) est fournie dans le fichier `"etat_cache.mat"` où elle est stockée sous la forme d'une matrice à 2 lignes et n colonnes. Pour récupérer ces données

```
x = io.loadmat('etat_cache.mat')['x']
n = x.shape[1]
```

et pour les visualiser

```
truePlot = plt.figure()
plt.plot(x[0,:],x[1,:],'b-')
plt.draw()
```

La liste des modules Python nécessaires est donnée à la fin de l'énoncé.

On peut aussi représenter, sous la forme d'une animation bi–dimensionnelle, la suite des états cachés $(X_1, \cdots, X_k)$, quand k varie de 1 à n

```
trueanimPlot = plt.figure()
for k in range(1,n):
   plt.clf()
   plt.plot(x[0,1:k],x[1,1:k],'-b')
   plt.plot(x[0,k-1],x[1,k-1],'.b') # why is this offset needed here?
   plt.draw()
   plt.pause(0.1)
```

Les valeurs numériques à utiliser pour les constantes c, α et ε, et pour le nombre n d'observations, sont données ci–dessous.

$$\boxed{c = 0.9 \qquad \alpha = -\pi/8 \qquad \varepsilon = 0.1 \qquad n = 100}$$

Observation des deux composantes simultanément

Dans ce premier scénario, chacune des deux composantes de l'état est observée dans un bruit blanc gaussien additif : l'observation recueillie est bi–dimensionnelle et décrite par

$$Y_k = H\, X_k + \varepsilon\, V_k \qquad \text{avec} \qquad H = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

où les vecteurs aléatoires $(V_1, \cdots, V_k, \cdots)$ sont indépendants de la condition initiale X_1 et des vecteurs aléatoires $(W_2, \cdots, W_k, \cdots)$, et forment un bruit blanc gaussien centré de matrice de covariance identité.

(ii) **Lire la suite d'observations $(Y_1, \cdots, Y_n)$ dans le fichier `"observation.mat"` où elle est stockée sous la forme d'une matrice à 2 lignes et n colonnes. Pour récupérer ces données**

```
y = io.loadmat('observation.mat')['y']
```

(iii) **Programmer le filtre de Kalman, c'est–à–dire calculer l'estimateur $\widehat{X}_k$ et la matrice de covariance d'erreur P_k pour l'estimation récursive (des deux composantes) de l'état caché X_k au vu des observations passées $(Y_1, \cdots, Y_k)$.**

Représenter sur deux graphiques différents

- **la suite des estimateurs et la suite des états cachés en fonction du temps,**

pour la première composante dans un premier graphique, et pour la deuxième composante dans un deuxième graphique.

Représenter sous la forme d'une animation bi–dimensionnelle

- **la suite des états cachés $(X_1, \cdots, X_k)$,**

- **l'ellipse de confiance à 90% définie par la matrice de covariance d'erreur P_k autour de l'estimateur $\widehat{X}_k$,**

quand k varie de 1 à n.

On trouvera en fin d'énoncé des indications pour le calcul et la représentation graphique d'une ellipse de confiance.

Observation d'une seule composante

Dans ce second scénario, seule la première composante de l'état est observée dans un bruit blanc gaussien additif : l'observation recueillie est mono–dimensionnelle et décrite par

$$Y_k = H\, X_k + \varepsilon\, V_k \qquad \text{avec} \qquad H = \begin{pmatrix} 1 & 0 \end{pmatrix}$$

où les variables aléatoires $(V_1, \cdots, V_k, \cdots)$ sont indépendants de la condition initiale X_1 et des vecteurs aléatoires $(W_2, \cdots, W_k, \cdots)$, et forment un bruit blanc gaussien centré de variance 1 (on pourrait évidemment considérer aussi le scénario symétrique dans lequel seule la deuxième composante de l'état serait observée).

(iv) **Lire la suite d'observations $(Y_1, \cdots, Y_n)$ dans le fichier `"observation_1.mat"` où elle est stockée sous la forme d'une matrice à 1 lignes et n colonnes (on pourrait aussi obtenir directement cette suite à partir de la suite d'observations bi–dimensionnelles lue en (ii)). Pour récupérer ces données**

```
y_1 = io.loadmat('observation_1.mat')['y_1']
```

(v) **Programmer le filtre de Kalman, c'est–à–dire calculer l'estimateur $\widehat{X}_k$ et la matrice de covariance d'erreur P_k pour l'estimation récursive (des deux composantes) de l'état caché X_k au vu des observations passées $(Y_1, \cdots, Y_k)$.**

Représenter les résultats obtenus, de la même manière qu'à la question (iii).

Pour être complet, on peut considérer aussi le scénario symétrique, dans lequel seule la deuxière composante de l'état est observée dans un bruit blanc gaussien additif : l'observation recueillie est mono–dimensionnelle et décrite par

$$Y_k = H\, X_k + \varepsilon\, V_k \qquad \text{avec} \qquad H = \begin{pmatrix} 0 & 1 \end{pmatrix}$$

où les variables aléatoires $(V_1, \cdots, V_k, \cdots)$ sont indépendants de la condition initiale X_1 et des vecteurs aléatoires $(W_2, \cdots, W_k, \cdots)$, et forment un bruit blanc gaussien centré de variance 1.

(vi) **Lire la suite d'observations $(Y_1, \cdots, Y_n)$ dans le fichier `"observation_2.mat"` où elle est stockée sous la forme d'une matrice à 1 lignes et n colonnes (on pourrait aussi obtenir directement cette suite à partir de la suite d'observations bi–dimensionnelles lue en (ii)). Pour récupérer ces données**

```
y_2 = io.loadmat('observation_2.mat')['y_2']
```

(vii) **Programmer le filtre de Kalman, c'est–à–dire calculer l'estimateur $\widehat{X}_k$ et la matrice de covariance d'erreur P_k pour l'estimation récursive (des deux composantes) de l'état caché X_k au vu des observations passées $(Y_1, \cdots, Y_k)$.**

Représenter les résultats obtenus, de la même manière qu'à la question (iii).

Les modules nécessaires sont

```
import numpy as np # calcul numerique
import matplotlib.pyplot as plt # fonctions graphiques a la MATLAB
import scipy.io as io # fonctions pour l'ouverture des fichiers .mat de MATLAB
import math
```

Ellipse de confiance

Par définition, la région de confiance de niveau $0 \leq p \leq 1$ pour un vecteur aléatoire gaussien bi–dimensionnel X de moyenne μ et de matrice de covariance Σ est la plus petite région D telle que la probabilité que X appartienne à D soit égale à p. Cette région est délimitée par une courbe de niveau de la densité de probabilité du vecteur aléatoire X, c'est–à–dire par une courbe de niveau de la forme quadratique $x \mapsto (x-\mu)^* \, \Sigma^{-1} \, (x-\mu)$. En introduisant la décomposition de Cholesky $\Sigma = U \, U^*$ de la matrice de covariance, on remarque que

$$(x-\mu)^* \, \Sigma^{-1} \, (x-\mu) = a \quad \text{si et seulement si} \quad |U^{-1} \, (x-\mu)|^2 = a \ ,$$

d'où la représentation paramétrique

$$x(s) = \mu + \sqrt{a} \, U \begin{pmatrix} \cos s \\ \sin s \end{pmatrix} \qquad \text{avec } 0 \leq s \leq 2\pi,$$

pour la courbe de niveau. Pour choisir le niveau a, on remarque que

$$\mathbb{P}[X \in D] = \mathbb{P}[(X-\mu)^* \, \Sigma^{-1} \, (X-\mu) \leq a] = \mathbb{P}[\, |U^{-1} \, (x-\mu)|^2 \leq a] = \mathbb{P}[Z \leq a] \ ,$$

où la variable aléatoire $Z = |U^{-1} \, (X-\mu)|^2$ suit la loi du χ^2 à deux degrés de liberté. On choisit donc a comme le quantile de niveau p de cette loi, ce qui donne par exemple $a \approx 4.6052$ pour une probabilité $p = 0.9$. Pour la représentation graphique, on pourra utiliser

```
def ellipse_conf(fmean,fcov):
    a = 4.6052
    npoints = 100
    t = np.linspace(0,2*math.pi,npoints)
    U = np.linalg.cholesky(fcov)
    round = np.zeros((2,npoints))
    round[0,:] = np.cos(t)
    round[1,:] = np.sin(t)
    ellipse = math.sqrt(a)*np.dot(U,round)
    ellipse = ellipse+np.tile(fmean,(1,npoints))
    plt.plot(ellipse[0,:],ellipse[1,:],'-k')
```