Mardi 8 décembre 2021

Compte-rendu : Filtre de Kalman

Poursuite de cible mobile

Alexandre Chaussard

Antoine Klein

Introduction :

Dans ce TP, il sera question de la poursuite d'une cible à partir de mesures bruitées. Nous nous intéresserons à deux modèles de trajectoire : un modèle linéaire et gaussien, lequel nous permettra de comprendre le filtre de Kalman, puis un modèle où l'observation n'est plus linéaire mais les lois toujours gaussiennes, pour lequel nous utiliserons le filtre de Kalman étendu.

1. Poursuite de cible linéaire avec loi gaussienne.

Considérons le système d'état suivant :

$$X_k = \underbrace{\begin{pmatrix} 1 & T_e & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_e \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{F}_k} X_{k-1} + U_k, \quad (1)$$

$$Y_k = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{H}_k} X_k + V_k, \quad (2)$$

$$X_0 \sim \mathcal{N}(\mathbf{m}_{0|0}, \mathbf{P}_{0|0}), \quad (3)$$

Le modèle ainsi construit caractérise un mouvement rectiligne uniforme à un bruitage gaussien près ; cela justifie l'hypothèse linéaire.

Nous garderons la donnée d'un vecteur X de dimension 4, à savoir position en X, vitesse en X, position en Y, vitesse en Y.

Des observations nous garderons un vecteur Y de dimension 2 : les coordonnées (x,y) à l'étape donnée.

Rappelons que pour un modèle linéaire et gaussien, à chaque étape, la densité de probabilité reste gaussienne. Se faisant, il nous suffit à chaque étape de connaître la moyenne et la variance.

Les équations d'états sont donc les suivantes :

1. Etape de prédiction consistant à estimer la position à l'étape $k+1$ sachant les observations jusqu'à l'étape k:
   $p_{k+1|k}(x_{k+1}|y_1, \dots, y_k) = \int N(x_{k+1}; Fx_k, Q) \times N(x_k; m_k; p_k) dx_k =$
   $N(x_{k+1}; m_{k+1|k}, p_{k+1|k})$
   Où

$$m_{k+1|k} = Fm_k$$

$$P_{k+1|k} = Q + FP_kF^T$$

2. Etape de mise à jour :

$$p_{k+1}(x_{k+1}|y_1,\dots,y_{k+1}) \propto \mathrm{N}\left(\mathrm{x}_{k+1}; m_{k+1|k}, P_{k+1|k}\right) \times N(y_{k+1}; Hx_{k+1}, R)$$

$$p_{k+1}(x_{k+1}|y_1,\dots,y_{k+1}) \propto \mathrm{N}(\mathrm{x}_{k+1}; m_{k+1}, P_{k+1})$$

Où

$$m_{k+1} = m_{k+1|k} + K(y_{k+1} - Hm_{k+1|k})$$
$$P_{k+1} = (I - KH)P_{k+1|k}$$
$$K = P_{k+1|k}H^T S^{-1}$$
$$S = KP_{k+1|k}H^T + R$$

Notons que K représente le gain de Kalman, S l'innovation et que $P_{k+1} \leq P_k$; autrement dit, la variance diminue au fur et à mesure du temps.

On simule alors une trajectoire X et des observations Y selon le modèle précédent.

Une réalisation de la simulation nous donne le graphe suivant :

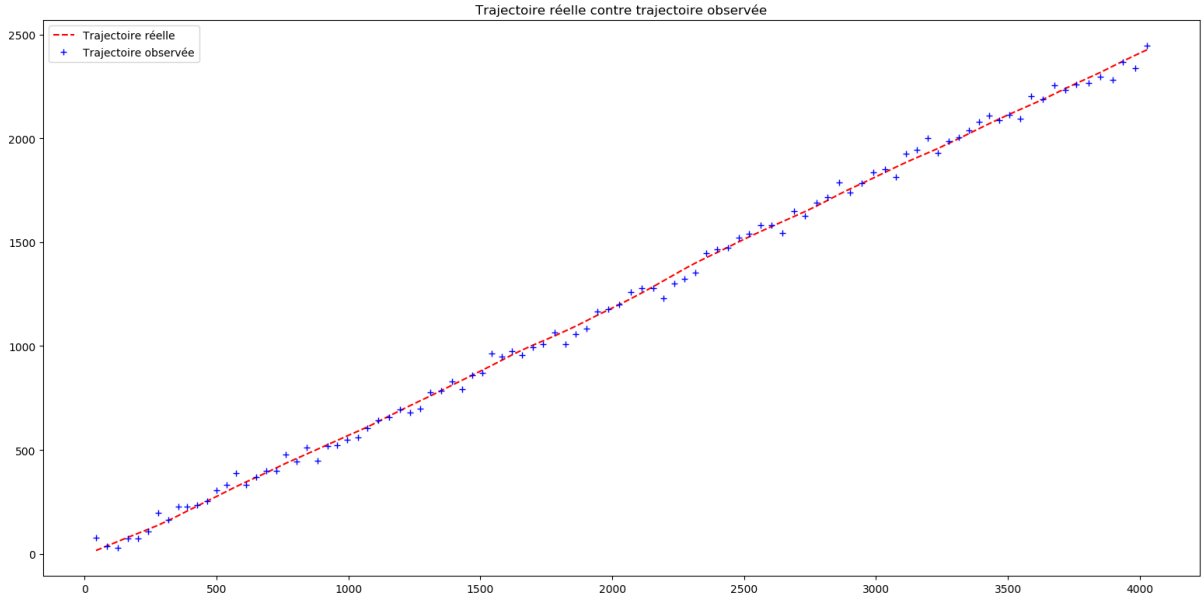

Figure 1 : Une réalisation de trajectoire et ses observations

On implémente alors le filtre de Kalman en Python (cf. annexe code). Pour donner un critère de comparaison, nous implémentons l'erreur quadratique moyenne du filtre.

On implémente également un calcul de l'erreur moyenne défini comme :

$$EM = \frac{1}{T}\sum_{k=1}^{T}\sqrt{erreur\_quadra(k)}$$

Voici l'application du filtre de Kalman sur la simulation précédente :

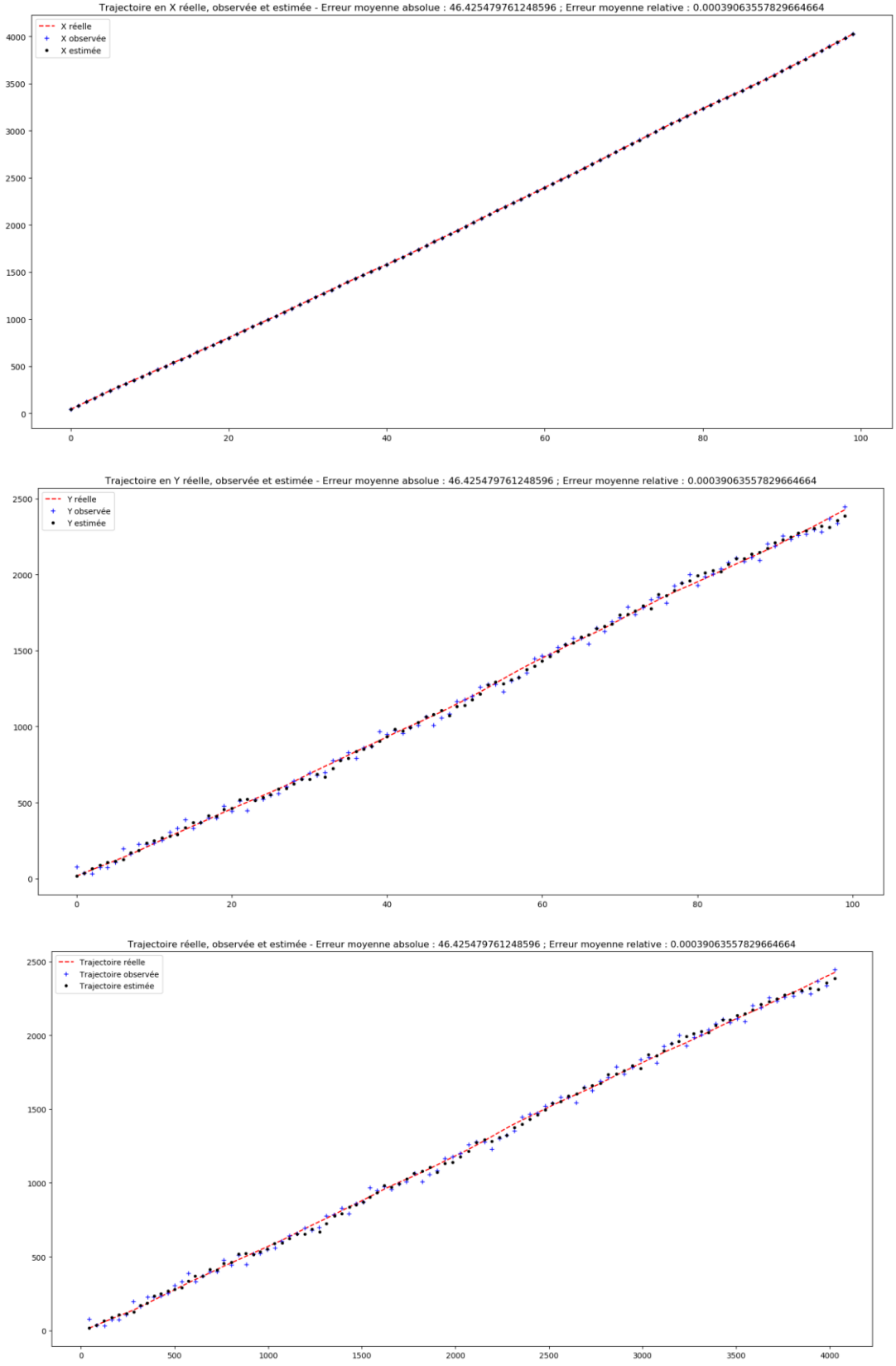

Figure 2 : Une réalisation de trajectoire, de ses observations et de son estimation

On remarque toutefois que l'erreur moyenne calculée est absolue et dépend de l'ordre de grandeur des valeurs prises par la trajectoire. Nous nous proposons donc de définir une autre métrique relative qui nous permettra de faire des comparaisons :

$$ER = \frac{\frac{1}{T}\sum_{k=1}^{T}\sqrt{erreur_{quadra(k)}}}{\sum_{k=1}^{T} Py}$$

On s'intéresse maintenant à l'influence du bruit sur le système, en faisant varier le bruit de processus noté σ_Q, le bruit selon l'abscisse x noté σ_{px}, le bruit selon l'ordonnée noté σ_{py} :

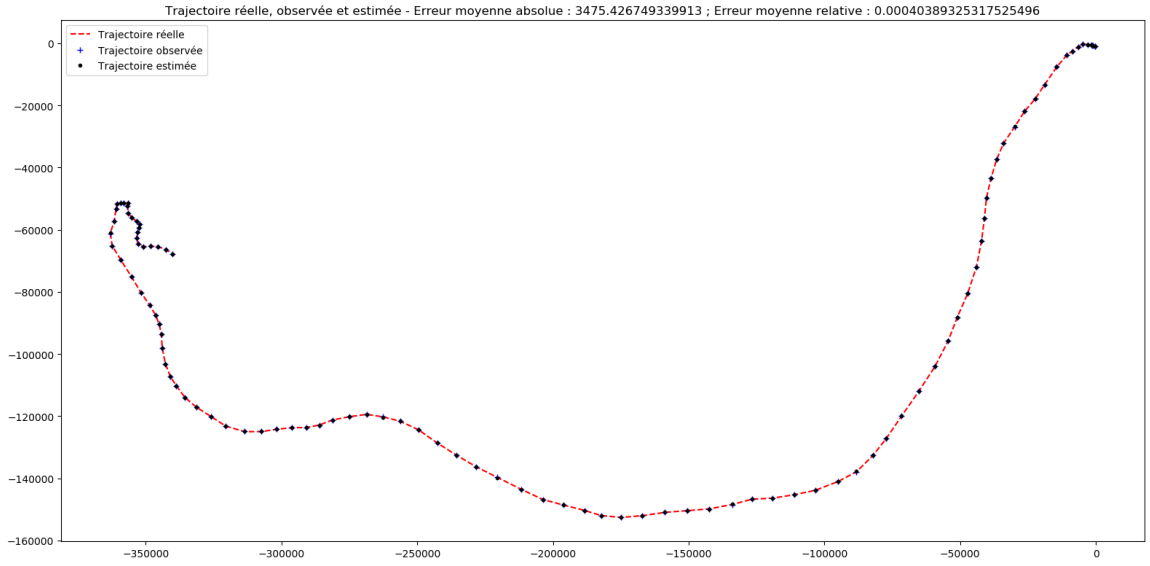

Figure 3a. : $\sigma_Q = 1000, \sigma_{px} = 1, \sigma_{py} = 30$ (bruit de processus contre bruits de mesures)

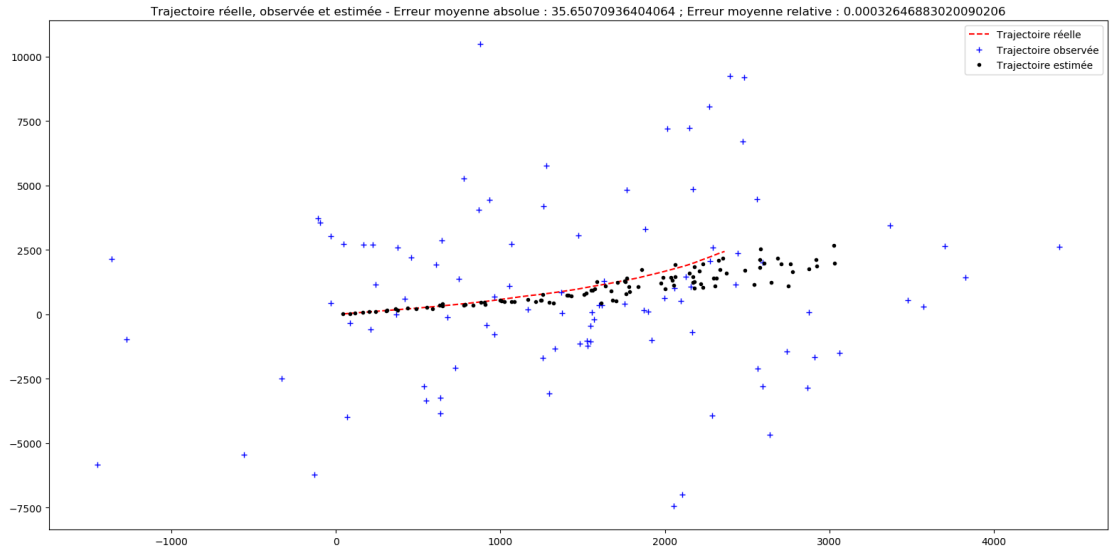

Figure 3b. : $\sigma_Q = 1, \sigma_{px} = 1000, \sigma_{py} = 3000$

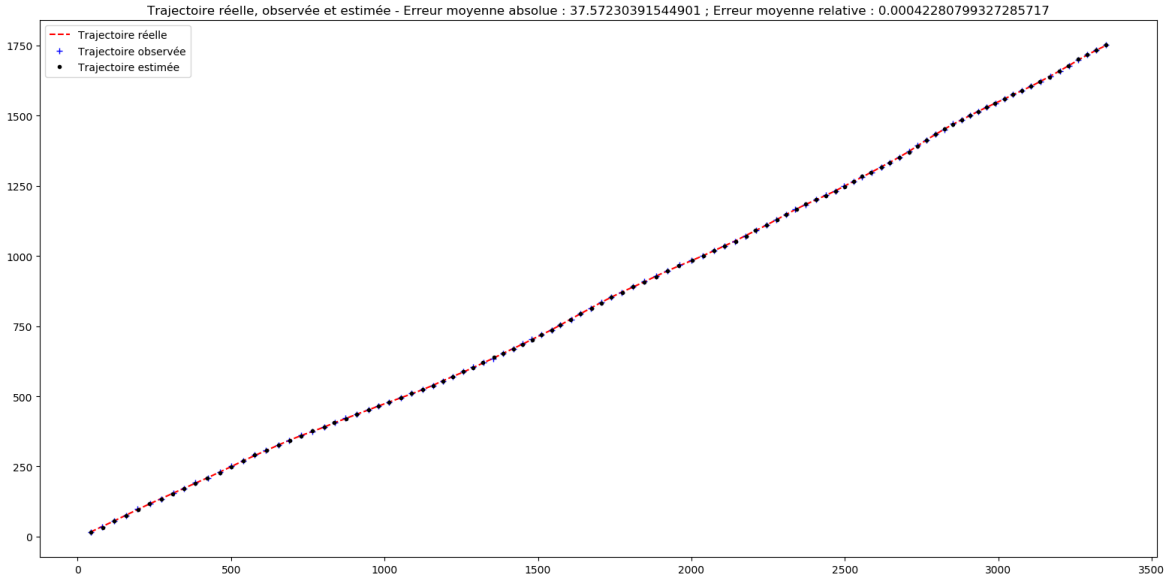

Figure 3c. : $\sigma_Q = 1, \sigma_{px} = 1, \sigma_{py} = 1$

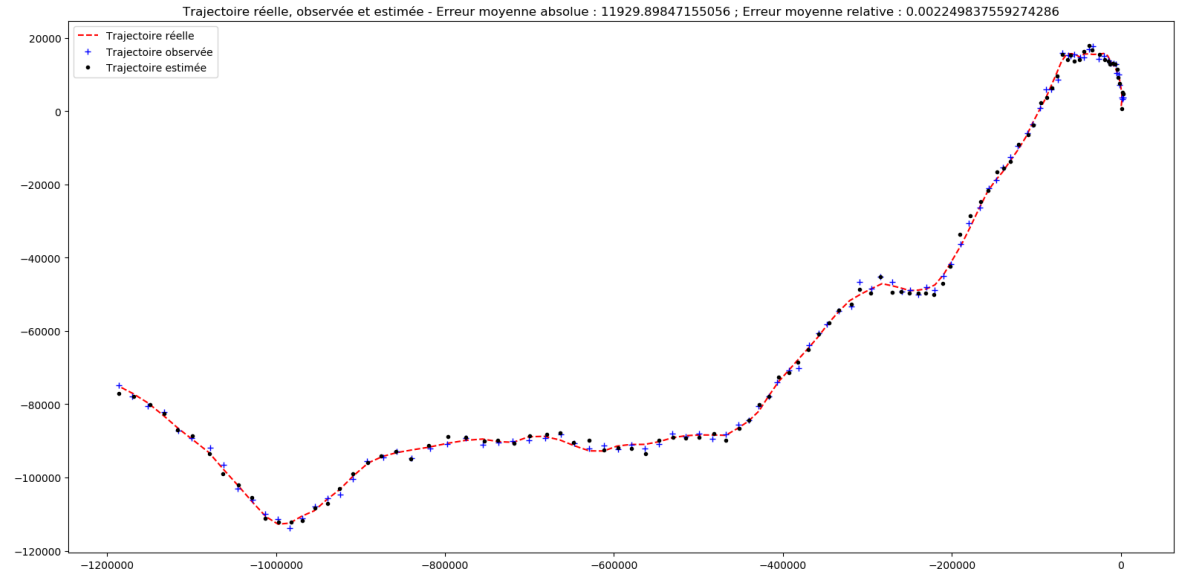

Figure 3d. : $\sigma_Q = 1000, \sigma_{px} = 1000, \sigma_{py} = 1000$

Si les bruits $\boldsymbol{\sigma_{px}}$ et $\boldsymbol{\sigma_{py}}$ caractérisent les incertitudes de mesure et $\boldsymbol{\sigma_q}$ la fluctuation de la trajectoire, nous remarquons que le filtre de Kalman est plus sensible, à changement de bruit comparable, aux fluctuations de trajectoire qu'aux incertitudes de mesure. Cela peut s'expliquer par le fait qu'une trajectoire fortement fluctuante tend à s'éloigner d'un modèle linéaire, hypothèse du filtre.

2. Application pratique

On s'intéresse à une application du filtre de Kalman pour la poursuite d'un avion de ligne et d'un avion de voltige.

Toutefois, lors de la poursuite, il est possible que l'on perde de vue la cible en raison d'un nuage par exemple. Dans ce cas, l'observation retournée indique une erreur.

Afin de prendre en compte cette absence de détection, on cherche à estimer la x_k sachant $y_{0:k-1}$.

Comme le processus $\{x_n, y_n\}$ est gaussien par hypothèse, toutes les lois marginales et les lois conditionnelles sont gaussiennes. On a alors :

$$p(x_k|y_{0:k-1}) \sim N(m_{k|k-1}, P_{k|k-1}) \qquad \text{(A)}$$

Où :

$$m_{k|k-1} = Fm_{k-1|k-1}$$

$$P_{k|k-1} = Q + FP_{k-1|k-1}F^T$$

L'estimation de x_k sachant $y_{0:k-1}$ est alors donnée par un tirage selon la loi (A).

Pour compléter nos prédictions, nous pourrions attendre une nouvelle observation, afin de corriger les prédictions effectuées avec du lissage.

On implémente alors l'estimateur précédent pour les absences de détection, puis on obtient les graphes suivants :

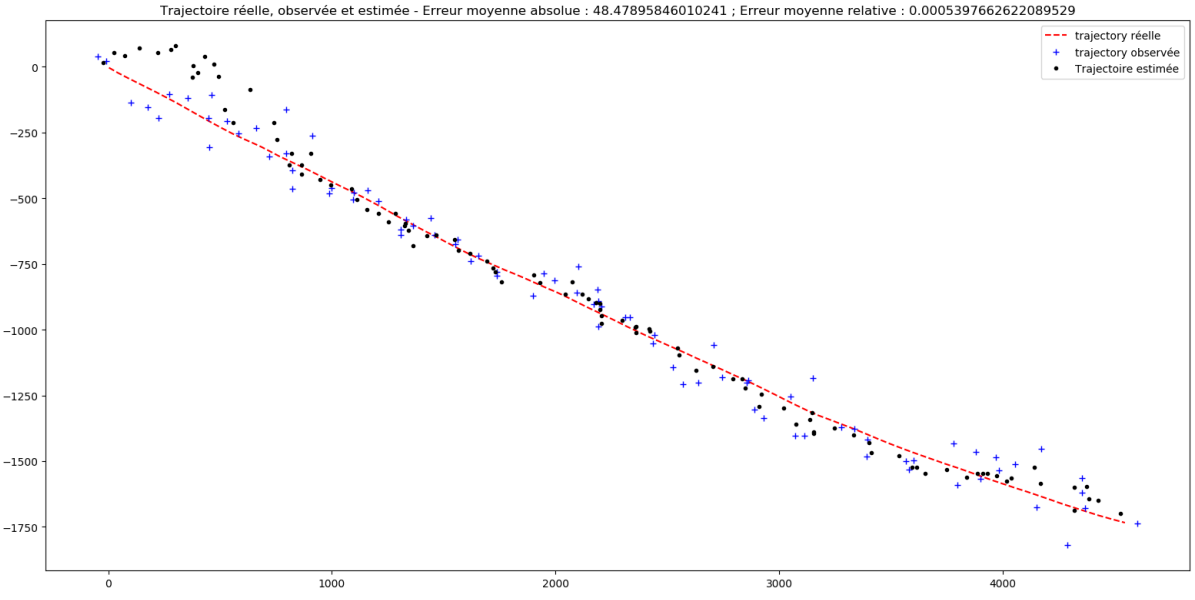

Figure 4 : Poursuite de cible dans le cas de l'avion de ligne

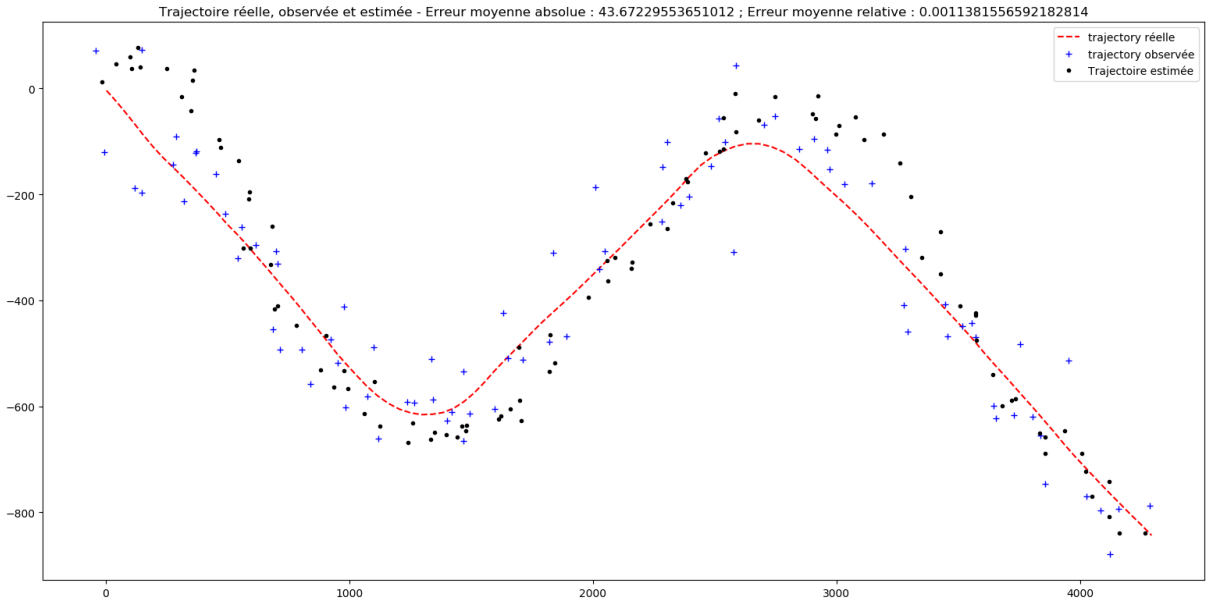

Figure 5 : Poursuite de cible dans le cas de l'avion de voltige

Si cela ne saute pas aux yeux ni n'apparait dans l'erreur absolue, le modèle commet dix fois plus d'erreur dans le cas de l'avion de voltige que dans celui de ligne. La raison est simple, dans ses virages, l'avion dévie de son mouvement rectiligne uniforme, l'hypothèse linéaire devient non pertinente. En quelque sorte, le filtre a la fâcheuse tendance d'estimer l'avion dans la suite de mon mouvement (telle l'inertie) et donc d'avoir un temps de retard dans les phases de voltiges.

3. Poursuite Angle-Distance

Dans certaines situations comme le radar, les coordonnées de la cible observée ne sont pas données dans le repère cartésien mais dans le repère polaire.

La modélisation précédente doit donc être adaptée à l'observation polaire.

Commençons par écrire la transformation polaire/cartésienne :

$$r = \sqrt{\mathrm{p_x^2} + p_y^2}$$

$$\Theta = \arctan(\frac{p_y}{p_x})$$

Où p_x, p_y représentent les coordonnées cartésiennes de l'observation, r, Θ les coordonnées polaires de l'observation.

La transformation cartésienne/polaire est une transformation non linéaire. Il faut donc remplacer le terme Hx_k du filtre de Kalman précédent par le terme non linéaire suivant :

$$H(x_k) = \begin{bmatrix} \Theta \\ r \end{bmatrix}$$

$$o\grave{u}\ r, \Theta\ renvoient\ aux\ transformations\ précédentes$$

La matrice R prend également de nouvelles valeurs :

$$R = \begin{bmatrix} \sigma_{angle}^2 & 0 \\ 0 & \sigma_{dist}^2 \end{bmatrix} o\grave{u}\ \sigma_{angle} = \frac{\pi}{180}\ et\ \sigma_{dist} = 10$$

Dès lors, les nouvelles équations de modèle sont les suivantes :

$$x_k = F_k x_{k-1} + U_k$$

$$y_k = H(x_k) + V_k$$

Où $U_k \sim N(0_{4\times1}, Q)$ et $V_k \sim N(0_{2\times1}, R)$

Étant donné la nature non linéaire de la transformation H, on ne peut plus utiliser le filtre de Kalman classique. Remarquons que la trajectoire reste linéaire, seule l'observation ne l'est plus. Nous allons donc utiliser le filtre de Kalman étendu.

Pour ce faire, on écrit $H(.) = \begin{bmatrix} f(.) \\ g(.) \end{bmatrix}$, de sorte que :

$$f(x) = \arctan(\frac{p_y}{p_x})$$

$$g(x) = \sqrt{p_x^2 + p_y^2}$$

On linéarise ensuite f et g autour du vecteur $\hat{x}_{k|k-1} = F\hat{x}_{k-1|k-1}$ qui correspond à la prédiction de la position à l'instant k sachant l'estimation précédente. Pour cela, on introduit la matrice jacobienne $\widetilde{H}$ suivante :

$$\widetilde{H}(x) = \widetilde{H}(p_x, p_y) = \begin{bmatrix} -\dfrac{p_y}{p_x^2(1+\frac{p_y^2}{p_x^2})} & 0 & \dfrac{1}{p_x(1+\frac{p_y^2}{p_x^2})} & 0 \\ \dfrac{p_x}{\sqrt{p_x^2+p_y^2}} & 0 & \dfrac{p_y}{\sqrt{p_x^2+p_y^2}} & 0 \end{bmatrix} = \begin{bmatrix} \nabla_x f(x) \\ \nabla_x g(x) \end{bmatrix}$$

En linéarisant f (resp. g) autour du vecteur $\hat{x}_{k|k-1}$, il vient par formule de Taylor :

$$f(x_k) \approx f(\hat{x}_{k|k-1}) + \nabla_x f(\hat{x}_{k|k-1})(x_k - \hat{x}_{k|k-1})$$

D'où $H(x) = \begin{bmatrix} f(x) \\ g(x) \end{bmatrix} = \begin{bmatrix} f(\hat{x}_{k|k-1}) \\ g(\hat{x}_{k|k-1}) \end{bmatrix} + \widetilde{H}(\hat{x}_{k|k-1})(x - \hat{x}_{k|k-1})$

Ce qui donne une nouvelle écriture de l'équation (2) :

$$y_k = \begin{bmatrix} f(\hat{x}_{k|k-1}) \\ g(\hat{x}_{k|k-1}) \end{bmatrix} + \widetilde{H}(\hat{x}_{k|k-1})(x - \hat{x}_{k|k-1}) + v_k \qquad (8)$$

Dès lors, on passe de l'évaluation de fonction H() à un produit matriciel avec la matrice résultante de la linéarisation de H(). On construit alors un filtre de Kalman étendu en utilisant l'écriture (8) :

1. Etape de prédiction consistant à prédire la position à l'étape $k+1$ sachant les observations jusqu'à l'étape k:

$$m_{k+1|k} = Fm_k$$

$$P_{k+1|k} = Q + FP_kF^T$$

2. Etape de mise à jour :

$$p_{k+1}(x_{k+1}|y_1, \dots, y_{k+1}) \propto \mathrm{N}\left(\mathrm{x}_{k+1}; m_{k+1|k}, P_{k+1|k}\right) \times N\left(y_{k+1}; \tilde{H}\left(\hat{x}_{k+1|k}\right), R\right)$$

$$p_{k+1}(x_{k+1}|y_1, \dots, y_{k+1}) \propto \mathrm{N}(\mathrm{x}_{k+1}; m_{k+1}, P_{k+1})$$

Où

$$m_{k+1} = m_{k+1|k} + K(y_{k+1} - \tilde{H}\left(\hat{x}_{k|k-1}\right)m_{k+1|k})$$

$$P_{k+1} = (I - K\tilde{H}\left(\hat{x}_{k+1|k}\right))P_{k+1|k}$$

$$K = P_{k+1|k}\tilde{H}\left(\hat{x}_{k+1|k}\right)^T S^{-1}$$

$$S = KP_{k+1|k}\tilde{H}\left(\hat{x}_{k+1|k}\right)^T + R$$

On implémente alors le filtre de Kalman étendu, qu'on applique à une trajectoire générée et des observations radar de celle-ci :

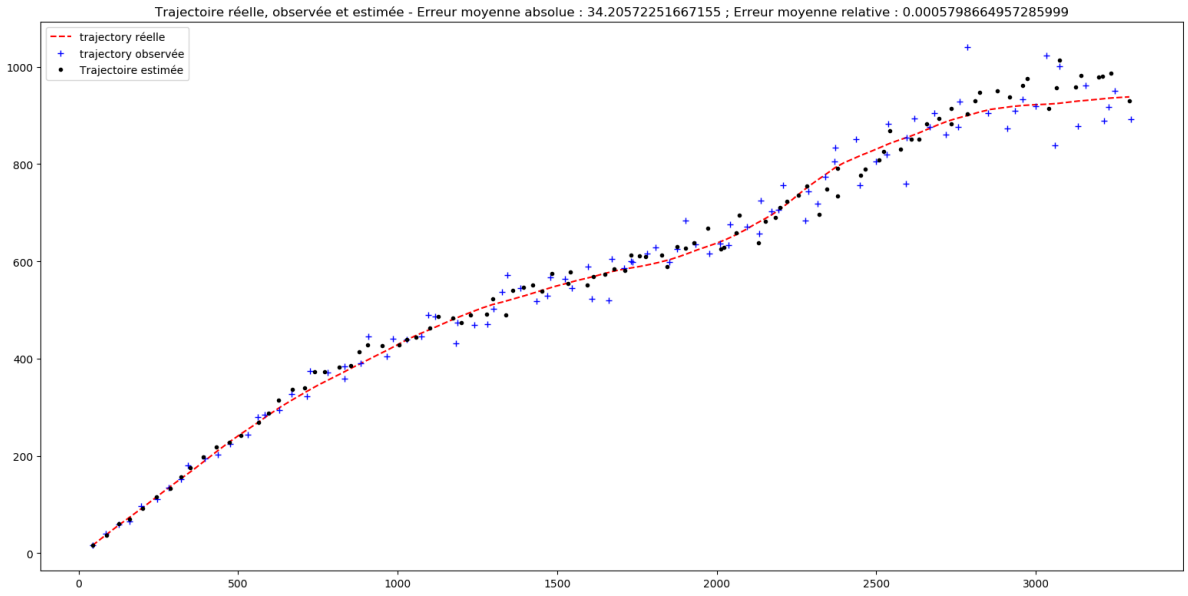

Figure 6 : Poursuite de cible cartésienne par le filtre de Kalman étendu

On constate alors une tendance hétéroscédastique de la trajectoire observée et estimée par rapport à la trajectoire réelle. (Le bruit va croissant avec le temps puisqu'il est proportionnel au jacobien). Cette tendance s'explique par le modèle polaire des observations alors que la trajectoire affichée est sur le modèle cartésien de la trajectoire réelle.

Voici un graphe représentant la même situation que le cas précédent, mais tracée selon les coordonnées polaires :

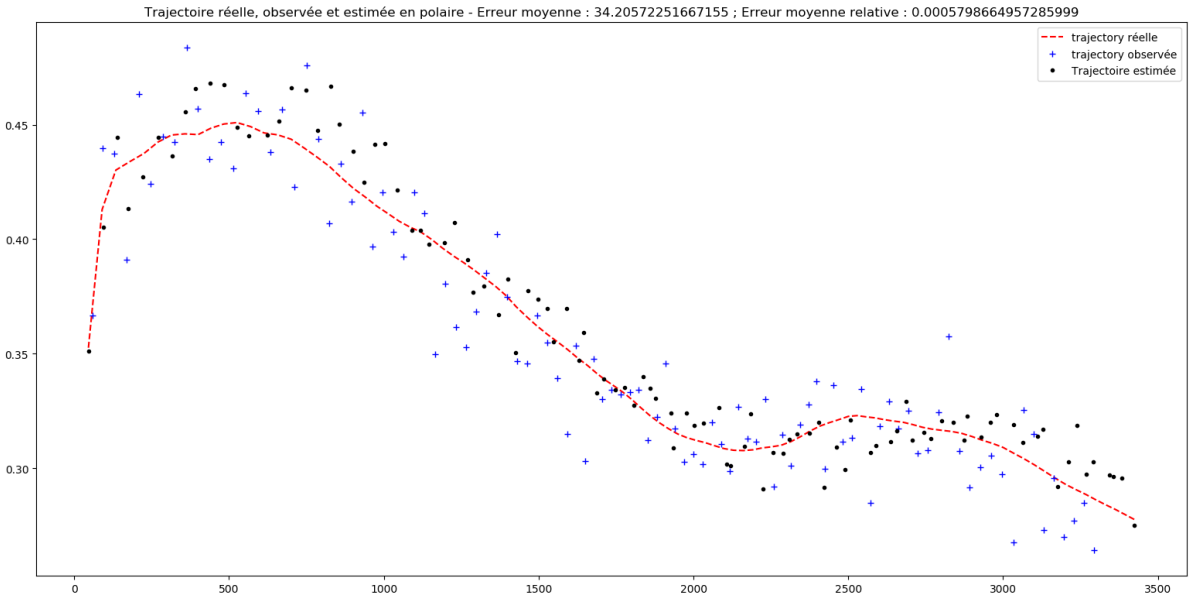

Figure 6 : Poursuite de cible polaire par le filtre de Kalman étendu

Cette fois, on constate bien que les observations et les estimations sont homoscédastiques par rapport à la trajectoire réelle.

On dégage alors la remarque suivante : Un bruit polaire constant va croissant en cartésien. Réciproquement, un bruit constant en cartésien va décroissant en polaire. Un bruit de 1° ne représente pas la même incertitude en cartésien si l'objet est à 1m ou 1km. En bref, une mesure angulaire fine est nécessaire pour bien estimer un objet lointain !

Dès lors, le filtre de Kalman étendu s'applique bien à tout type de trajectoires observées pour lesquelles la fonction non linéaire est à minima dérivable afin d'utiliser le développement de Taylor à l'ordre 1 (cela pose un véritable problème pour des fonctions hautement non linaires). Malgré tout, dans le cas non linéaire le filtre de Kalman ne donne au mieux qu'une approximation de la trajectoire estimée puisqu'on effectue un développement limité sur le modèle observé.

4. Extension à la poursuite de plusieurs cibles

Si l'on souhaite poursuivre plusieurs cibles simultanément à l'aide du filtre de Kalman, plusieurs problèmes peuvent apparaître :

- Il peut y avoir une dépendance entre les trajectoires (notamment dans le cas de courbes de poursuite, où l'un suit l'autre).
- Obstructions mutuelles : Si l'un vient à passer devant l'autre, bloquant les mesures
- Problème de perspective, c'est-à-dire estimer la distance séparant l'objet de la caméra, surtout si les objets ne sont pas à la même distance de l'objectif : induit une erreur de mesure.

Afin de pallier ces différentes situations, on peut appliquer les propositions suivantes :

- Ajouter de la dépendance entre les cibles (modèle de poursuite, trajectoire commune...)
- Estimer au préalable la distance séparant les objets de la caméra (problème de perspective)
- Joindre le vecteur des observations en un vecteur de dimension 4 avec les coordonnées de chacun des objets : on réduit un problème de filtrage de deux trajectoires en un problème de filtrage d'une trajectoire de dimension 2.
- Suivre une région de l'espace qui encadre l'objet à suivre. Ceci n'étant plus linéaire, on pourra réaliser un filtrage particulaire.

5. Conclusion

Ainsi, nous avons réalisé des poursuites de cibles au travers d'un système d'état modélisé par le filtre de Kalman. Ce modèle linéaire et gaussien est particulièrement adapté aux trajectoires rectilignes et uniformes, c'est-à-dire pour le cas d'un avion de ligne, bien moins pour le cas d'un avion de voltige. Dans le cas d'observations polaires, ces dernières ne sont plus linéaires, ce pourquoi nous avons linéarisé la fonction d'observation pour réaliser le filtrage de Kalman étendu. Dès lors, un bruit constant en cartésien va décroissant en polaire. Pour réaliser des estimations précises de trajectoires lointaines, nous comprenons maintenant qu'il faut des mesures angulaires très précises.

Code (Annexe)

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

# ----------- 2

T_e = 1
T = 100
sigma_Q = 1
sigma_px = 1
sigma_py = 30

x_init = np.array([[3, 40, -4, 20]]).transpose()
x_kalm = x_init
P_kalm = np.eye(4)

F = np.array([[1, T_e, 0, 0],
              [0, 1, 0, 0],
              [0, 0, 1, T_e],
              [0, 0, 0, 1]])
Q = sigma_Q ** 2 * np.array([[T_e ** 3 / 3, T_e ** 2 / 2, 0, 0],
                             [T_e ** 2 / 2, T_e, 0, 0],
                             [0, 0, T_e ** 3 / 3, T_e ** 2 / 2],
                             [0, 0, T_e ** 2 / 2, T_e]])
H = np.array([[1, 0, 0, 0],
              [0, 0, 1, 0]])
R = np.array([[sigma_px ** 2, 0],
              [0, sigma_py ** 2]])
```

```
# ----------- 3

def creer_trajectoire(F, Q, x_init, T):
    traj = []

    x_previous = x_init.copy()
    for i in range(0, T):
        U = np.random.multivariate_normal([0, 0, 0, 0], Q).reshape((4, 1))
        real = np.matmul(F, x_previous) + U
        x_previous = real.copy()
        traj.append(real)

    return np.array(traj).transpose().reshape(4, T)


vecteur_x = creer_trajectoire(F, Q, x_init, T)
```

```
# ----------- 4

def creer_observations(H, R, vecteur_x, T):
    obs = []
    X_t = vecteur_x.copy().transpose()

    for i in range(0, T):
        X = np.array(X_t[i]).reshape(4, 1)
        V = np.random.multivariate_normal([0, 0], R).reshape((2, 1))
        Y = np.matmul(H, X) + V
        obs.append(Y)

    return np.array(obs).transpose().reshape(2, T)


vecteur_y = creer_observations(H, R, vecteur_x, T)
```

```python
# ---------- 5

def tracer(vecteur_x, vecteur_y):
    x_x = vecteur_x[0]
    y_x = vecteur_x[2]
    x_y = vecteur_y[0]
    y_y = vecteur_y[1]

    fig, ax = plt.subplots()
    ax.plot(x_x, y_x, 'r--', label="Trajectoire réelle")
    ax.plot(x_y, y_y, 'b+', label="Trajectoire observée")
    ax.legend()

    plt.title("Trajectoire réelle contre trajectoire observée")
    plt.show()


tracer(vecteur_x, vecteur_y)
```

```python
# ---------- 6

def filtre_de_kalman(F, Q, H, R, y_k, x_kalm_prec, P_kalm_prec):
    # Partie prédiction

    x_kalm_prediction = np.matmul(F, x_kalm_prec)
    P_kalm_prediction = Q + np.matmul(np.matmul(F, P_kalm_prec), F.transpose())

    # Partie mise à jour

    S = np.matmul(np.matmul(H, P_kalm_prediction), H.transpose()) + R
    K = np.matmul(np.matmul(P_kalm_prediction, H.transpose()), np.linalg.inv(S))

    X_kalm_k = x_kalm_prediction + np.matmul(K, y_k.reshape(2, 1) - np.matmul(H, x_kalm_prediction))
    P_kalm_k = np.matmul(np.eye(4) - np.matmul(K, H), P_kalm_prediction)

    return [X_kalm_k, P_kalm_k]
```

```python
# ---------- 7

x_est = []
x_kalm_prec = x_kalm
P_kalm_prec = P_kalm
for y_k in vecteur_y.transpose():
    [X_kalm_k, P_kalm_k] = filtre_de_kalman(F, Q, H, R, y_k, x_kalm_prec, P_kalm_prec)

    x_kalm_prec = X_kalm_k
    P_kalm_prec = P_kalm_k

    x_est.append(np.random.multivariate_normal(X_kalm_k.reshape(1, 4).tolist()[0], P_kalm_k))

x_est = np.array(x_est)
```

```python
# ---------- 8

def err_quadra(k, vecteur_x, x_est):
    diff = (vecteur_x.reshape(T, 4)[k] - x_est.reshape(T, 4)[k])

    err = np.dot(diff, diff)
    # err = np.matmul((vecteur_x[k] - x_est.transpose()[k]).transpose().reshape(T, 1), (vecteur_x.transpose()[k] - x_est.tra
    return err


def erreur_moyenne(vecteur_x, x_est, T):
    sum = 0

    for i in range(0, T):
        sum = err_quadra(i, vecteur_x, x_est) ** .5

    return sum / (T)

def erreur_moyenne_propose(vecteur_x, x_est, T):
    sum = 0

    for i in range(0, T):
        sum = err_quadra(i, vecteur_x, x_est) ** .5

    return sum / (T*max_y_value(vecteur_x))

def max_y_value(vecteur_x):
    return abs(sum(abs(vecteur_x[2])))
```

```python
# ---------- 9 & 10

def tracer_estime(vecteur_x, vecteur_y, x_est, T):
    x_x = vecteur_x[0]
    y_x = vecteur_x[2]
    x_y = vecteur_y[0]
    y_y = vecteur_y[1]

    fig, ax = plt.subplots()
    ax.plot(x_x, y_x, 'r--', label="Trajectoire réelle")
    ax.plot(x_y, y_y, 'b+', label="Trajectoire observée")
    ax.plot(x_est.transpose()[0], x_est.transpose()[2], "k.", label="Trajectoire estimée")
    ax.legend()

    plt.title("Trajectoire réelle, observée et estimée - Erreur moyenne absolue : " + str(erreur_moyenne(vecteur_x, x_est, T)
    plt.show()

    #abscisse = []
    #for i in range(0, len(x_x)):
    #    abscisse.append(i)

    #fig, ax = plt.subplots()
    #ax.plot(abscisse, x_x, 'r--', label="X réelle")
    #ax.plot(abscisse, x_y, 'b+', label="X observée")
    #ax.plot(abscisse, x_est.transpose()[0], "k.", label="X estimée")
    #ax.legend()
    #plt.title("Trajectoire en X réelle, observée et estimée - Erreur moyenne absolue : " + str(
    #    erreur_moyenne(vecteur_x, x_est, T)) + " ; Erreur moyenne relative : " + str(
    #    erreur_moyenne_propose(vecteur_x, x_est, T)))
    #plt.show()

    #abscisse = []
    #for i in range(0, len(x_y)):
    #    abscisse.append(i)

    #fig, ax = plt.subplots()
    #ax.plot(abscisse, y_x, 'r--', label="Y réelle")
    #ax.plot(abscisse, y_y, 'b+', label="Y observée")
    #ax.plot(abscisse, x_est.transpose()[2], "k.", label="Y estimée")
    #ax.legend()

    #plt.title("Trajectoire en Y réelle, observée et estimée - Erreur moyenne absolue : " + str(
    #    erreur_moyenne(vecteur_x, x_est, T)) + " ; Erreur moyenne relative : " + str(
    #    erreur_moyenne_propose(vecteur_x, x_est, T)))
    #plt.show()


tracer_estime(vecteur_x, vecteur_y, x_est, T)
```

```python
######---------- | Application | ----------######

# ---------- 1

# On peut faire une moyenne entre la dernière valeur captée et la précédante

# ---------- 2

import scipy.io as scipyio


def filtre_de_kalman_avion(F, Q, H, R, y_k, x_kalm_prec, P_kalm_prec):
    # Partie prédiction

    x_kalm_prediction = np.matmul(F, x_kalm_prec)
    P_kalm_prediction = Q + np.matmul(np.matmul(F, P_kalm_prec), F.transpose())

    # Partie mise à jour

    S = np.matmul(np.matmul(H, P_kalm_prediction), H.transpose()) + R
    K = np.matmul(np.matmul(P_kalm_prediction, H.transpose()), np.linalg.inv(S))

    X_kalm_k = x_kalm_prediction + np.matmul(K, y_k.reshape(2, 1) - np.matmul(H, x_kalm_prediction))
    P_kalm_k = np.matmul(np.eye(4) - np.matmul(K, H), P_kalm_prediction)

    return [X_kalm_k, P_kalm_k, x_kalm_prediction, P_kalm_prediction]


vecteur_x_avion_ligne_dic = scipyio.loadmat("vecteur_x_avion_ligne.mat")
vecteur_x_avion_voltige_dic = scipyio.loadmat("vecteur_x_avion_voltige.mat")
vecteur_y_avion_ligne_dic = scipyio.loadmat("vecteur_y_avion_ligne.mat")
vecteur_y_avion_voltige_dic = scipyio.loadmat("vecteur_y_avion_voltige.mat")
```

```
vecteur_x_avion_ligne = []
for value in vecteur_x_avion_ligne_dic.values():
    if type(value) != np.ndarray:
        continue
    for y in value:
        vecteur_x_avion_ligne.append(y)
vecteur_x_avion_ligne = np.array(vecteur_x_avion_ligne)

vecteur_x_avion_voltige = []
for value in vecteur_x_avion_voltige_dic.values():
    if type(value) != np.ndarray:
        continue
    for y in value:
        vecteur_x_avion_voltige.append(y)
vecteur_x_avion_voltige = np.array(vecteur_x_avion_voltige)

vecteur_y_avion_ligne = []
for value in vecteur_y_avion_ligne_dic.values():
    if type(value) != np.ndarray:
        continue
    for y in value:
        vecteur_y_avion_ligne.append(y)
vecteur_y_avion_ligne = np.array(vecteur_y_avion_ligne)

vecteur_y_avion_voltige = []
for value in vecteur_y_avion_voltige_dic.values():
    if type(value) != np.ndarray:
        continue
    for y in value:
        vecteur_y_avion_voltige.append(y)
vecteur_y_avion_voltige = np.array(vecteur_y_avion_voltige)
```

```
if True:

    def tracer_trajectoire(vecteur_x, vecteur_y):
        x_est = []
        x_kalm_prec = x_kalm
        P_kalm_prec = P_kalm

        y_last = [0, 0]
        for y_k in vecteur_y.transpose():

            if str(y_k[0]) == "nan" or str(y_k[1]) == "nan":
                x_k = np.random.multivariate_normal(y_last[0].reshape(1, 4).tolist()[0], y_last[1])
                x_est.append(x_k)
                continue

            [X_kalm_k, P_kalm_k, m, P] = filtre_de_kalman_avion(F, Q, H, R, y_k, x_kalm_prec, P_kalm_prec)

            x_kalm_prec = X_kalm_k
            P_kalm_prec = P_kalm_k
            y_last = [m, P]
            x_est.append(np.random.multivariate_normal(X_kalm_k.reshape(1, 4).tolist()[0], P_kalm_k))

        x_est = np.array(x_est)
        tracer_estime(vecteur_x, vecteur_y, x_est, T)

    tracer_trajectoire(vecteur_x_avion_ligne, vecteur_y_avion_ligne)
    tracer_trajectoire(vecteur_x_avion_voltige, vecteur_y_avion_voltige)
```

```
######---------- | Partie 2 | ----------######

# ----------- 1

def cylindric(p_x, p_y):
    r = (p_x ** 2 + p_y ** 2) ** 0.5
    theta = np.arctan(p_y / p_x)

    return [theta, r]


# ----------- 2

def H(X):
    p_x = X[0][0]
    p_y = X[2][0]
    [theta, r] = cylindric(p_x, p_y)
    return np.array([[theta],
                     [r]])


# Que devient la loi g_k(y_k|x_k) ?

# ----------- 3

sigma_angle = np.pi / 180
sigma_dist = 10

R = np.array([[sigma_angle ** 2, 0],
              [0, sigma_dist ** 2]])
```

```
def creer_observation_radar(R, vecteur_x, T):
    obs = []
    X_t = vecteur_x.copy().transpose()

    for i in range(0, T):
        X = np.array(X_t[i]).reshape(4, 1)
        V = np.random.multivariate_normal([0, 0], R).reshape((2, 1))
        Y = H(X) + V
        obs.append(Y)

    return np.array(obs).transpose().reshape(2, T)


# ----------- 4
# Non car non linéaire

# ----------- 5

def f(X):
    return np.arctan(X[2][0] / X[0][0])


def g(X):
    return (X[0][0] ** 2 + X[2][0] ** 2) ** 0.5


def y_k(x_predic, x_k):
    vector = np.array([[f(x_predic)],
                       [g(x_predic)]])

    v_k = np.random.multivariate_normal([0, 0], R).reshape((2, 1))

    return vector + np.matmul(H(x_predic), (x_k - x_predic)) + v_k
```

```python
# ---------- 6

def H_tilde(X):
    x = X[0][0]
    y = X[2][0]

    return np.array([[-y / (x ** 2 * (1 + (y / x) ** 2)), 0, 1 / (x * (1 + (y / x) ** 2)), 0],

                     [x / ((x ** 2 + y ** 2) ** 0.5), 0, y / ((x ** 2 + y ** 2) ** 0.5), 0]])


def filtre_de_kalman_radar(F, Q, R, y_k, x_kalm_prec, P_kalm_prec):
    # Partie prédiction

    x_kalm_prediction = np.matmul(F, x_kalm_prec)
    P_kalm_prediction = Q + np.matmul(np.matmul(F, P_kalm_prec), F.transpose())

    H_tild = H_tilde(x_kalm_prediction)

    # Indication
    y_k_prime = y_k.reshape(2, 1)

    # Partie mise à jour

    S = np.matmul(np.matmul(H_tild, P_kalm_prediction), H_tild.transpose()) + R
    K = np.matmul(np.matmul(P_kalm_prediction, H_tild.transpose()), np.linalg.inv(S))

    X_kalm_k = x_kalm_prediction.reshape(4, 1) + np.matmul(K, (y_k_prime - H(x_kalm_prediction)).reshape(2, 1)).reshape(4, 1)
    P_kalm_k = np.matmul(np.eye(4) - np.matmul(K, H_tild), P_kalm_prediction)

    return [X_kalm_k, P_kalm_k]


vecteur_y = creer_observation_radar(R, vecteur_x, T)
```

```python
x_est = []
x_kalm_prec = x_kalm
P_kalm_prec = P_kalm
for y_k in vecteur_y.transpose():
    [X_kalm_k, P_kalm_k] = filtre_de_kalman_radar(F, Q, R, y_k, x_kalm_prec, P_kalm_prec)

    x_kalm_prec = X_kalm_k
    P_kalm_prec = P_kalm_k

    x_est.append(np.random.multivariate_normal(X_kalm_k.reshape(1, 4).tolist()[0], P_kalm_k))

vecteur_y_car = []
for i in range(0, T):
    r = vecteur_y[1][i]
    theta = vecteur_y[0][i]
    vecteur_y_car.append([r*np.cos(theta), r*np.sin(theta)])

x_est = np.array(x_est)
vecteur_y_car = np.array(vecteur_y_car).transpose()

tracer_estime(vecteur_x, vecteur_y_car, x_est, T)
```

```python
def tracer_estime_polaire(vecteur_x, vecteur_y, x_est, T):

    def angle(px, py):
        return np.arctan(py/px)
    def radius(px, py):
        return (px**2 + py**2)**.5

    X_pol = []
    Y_pol = []
    X_est_pol = []

    for i in range(0, len(vecteur_x[0])):
        X_pol.append([angle(vecteur_x[0][i], vecteur_x[2][i]),
                      radius(vecteur_x[0][i], vecteur_x[2][i])])
        Y_pol.append([angle(vecteur_y[0][i], vecteur_y[1][i]),
                      radius(vecteur_x[0][i], vecteur_x[1][i])])
        X_est_pol.append([angle(x_est.transpose()[0][i], x_est.transpose()[2][i]),
                      radius(x_est.transpose()[0][i], x_est.transpose()[2][i])])

    X_pol = np.array(X_pol).transpose()
    Y_pol = np.array(Y_pol).transpose()
    X_est_pol = np.array(X_est_pol).transpose()

    x_x = X_pol[1]
    y_x = X_pol[0]
    x_y = Y_pol[1]
    y_y = Y_pol[0]

    fig, ax = plt.subplots()
    ax.plot(x_x, y_x, 'r--', label="Trajectoire réelle")
    ax.plot(x_y, y_y, 'b+', label="Trajectoire observée")
    ax.plot(X_est_pol[1], X_est_pol[0], "k.", label="Trajectoire estimée")
    ax.legend()

    plt.title("Trajectoire réelle, observée et estimée en polaire - Erreur moyenne : " + str(erreur_moyenne(vecteur_x, x_est,
    plt.show()

tracer_estime_polaire(vecteur_x, vecteur_y_car, x_est, T)
```