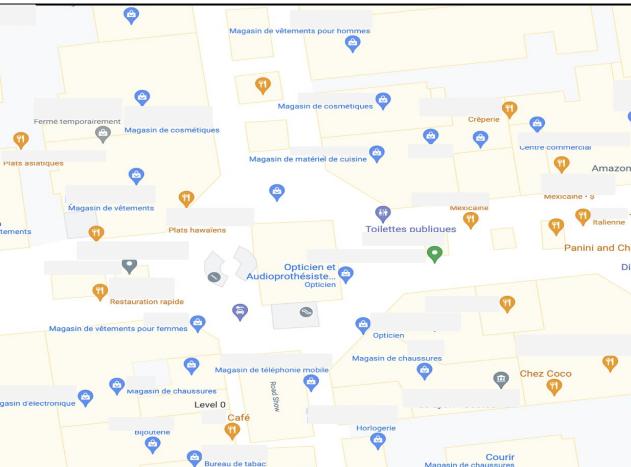


CUEL Tom  
Candidat n°14904

# Principes et Erreurs de mesures du système GPS

# La Précision et l'Utilisation du GPS en Ville

Les Services



L'autonomie des Voitures



Le Sport



Quelle est la précision d'un  
capteur GPS de sport dans un  
environnement urbain ?

I) Principe de fonctionnement du système GPS

II) Outil de modélisation de données

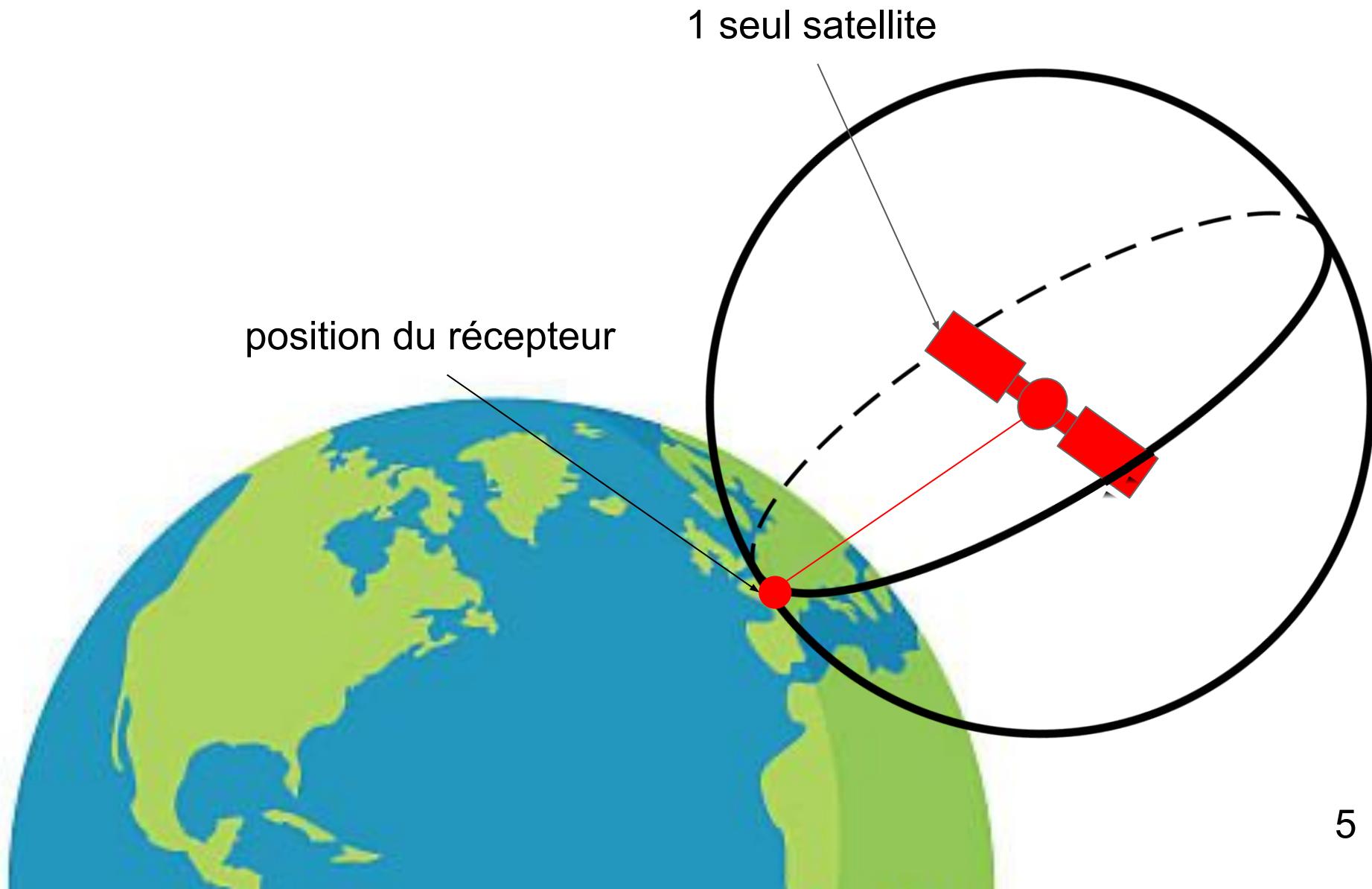
III) Observation des erreurs en statique

a) montre et capteur vélo

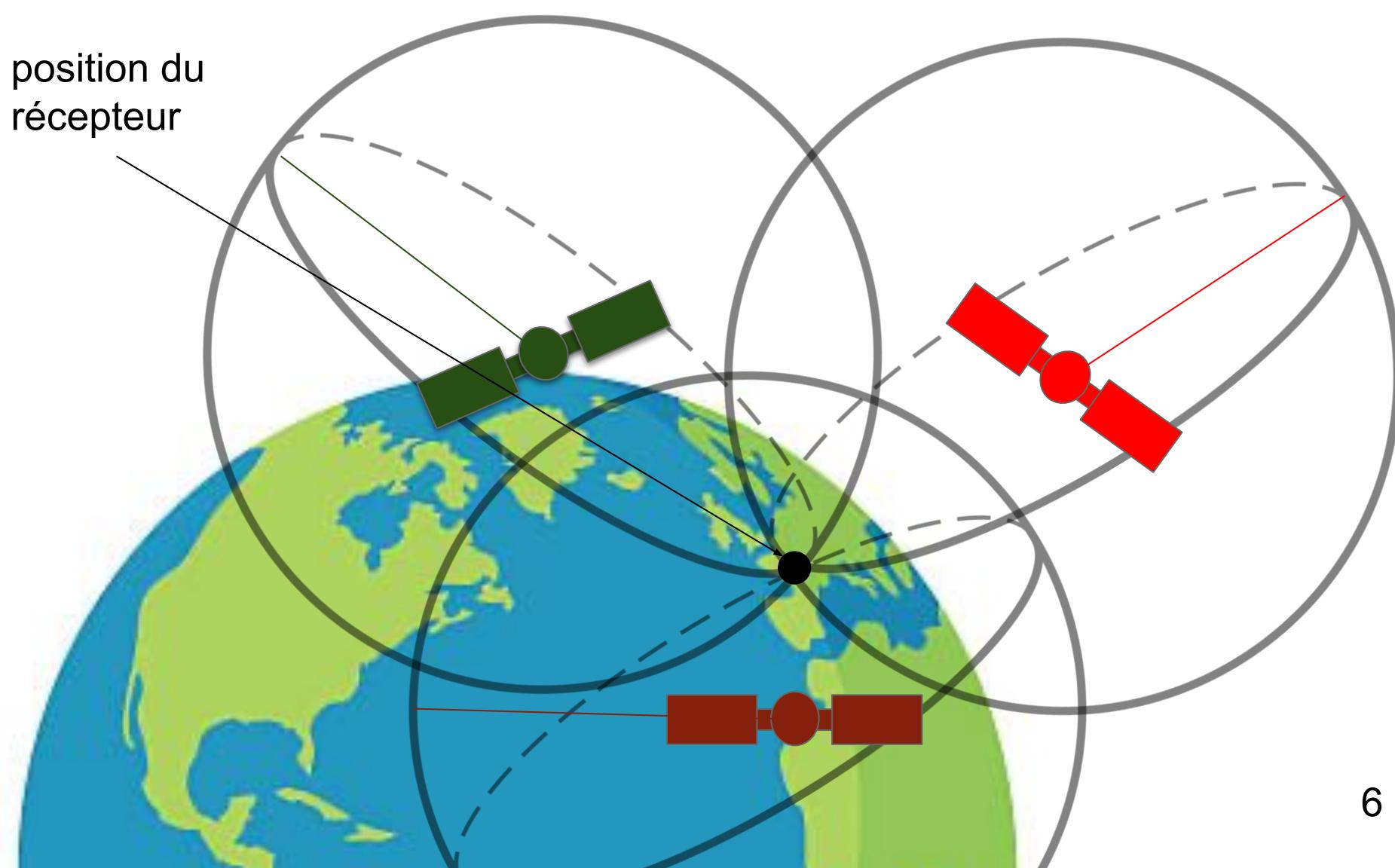
b) les effets d'obstacles

IV) En dynamique

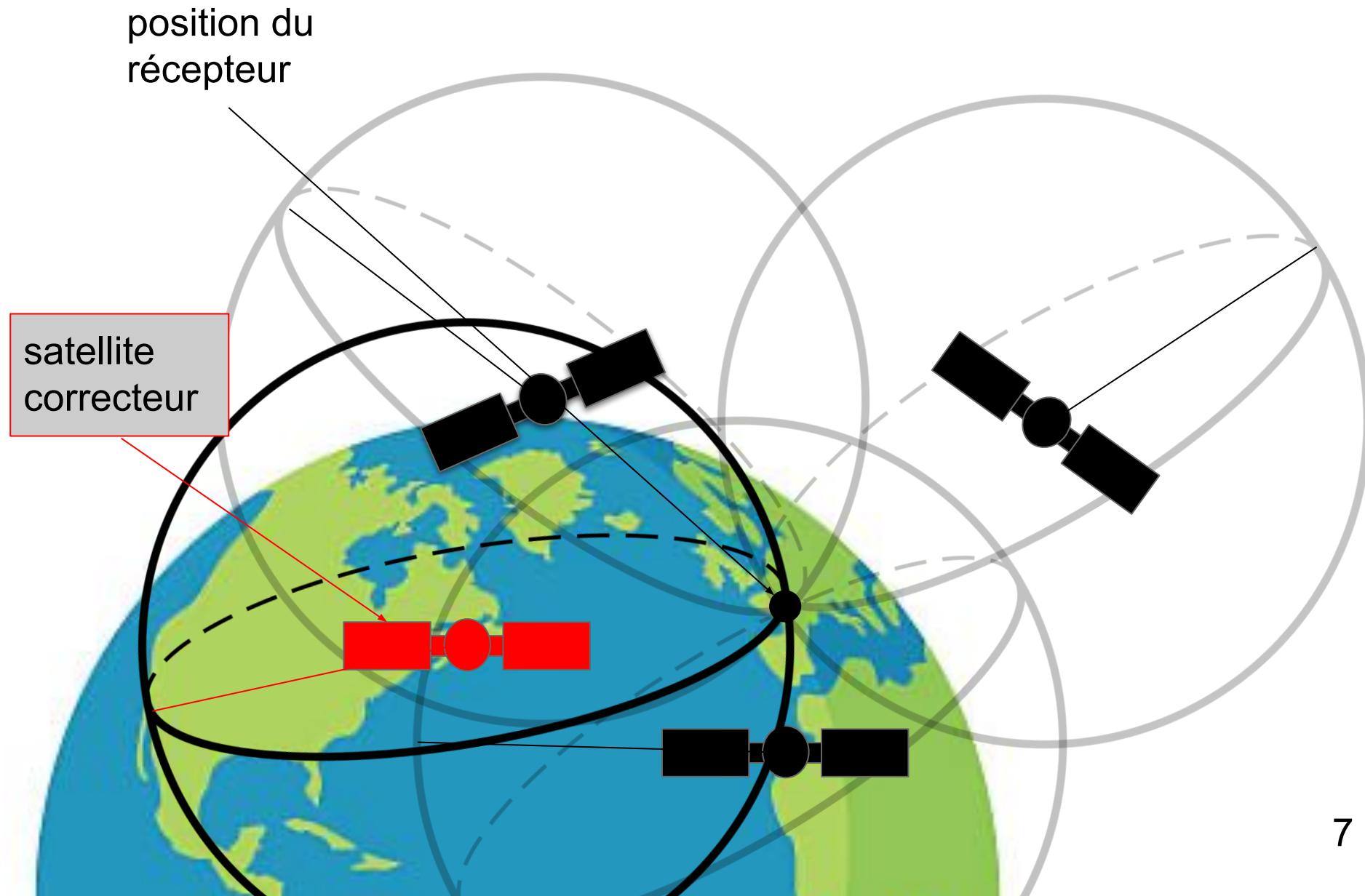
# Principe de fonctionnement du système GPS : trilateration



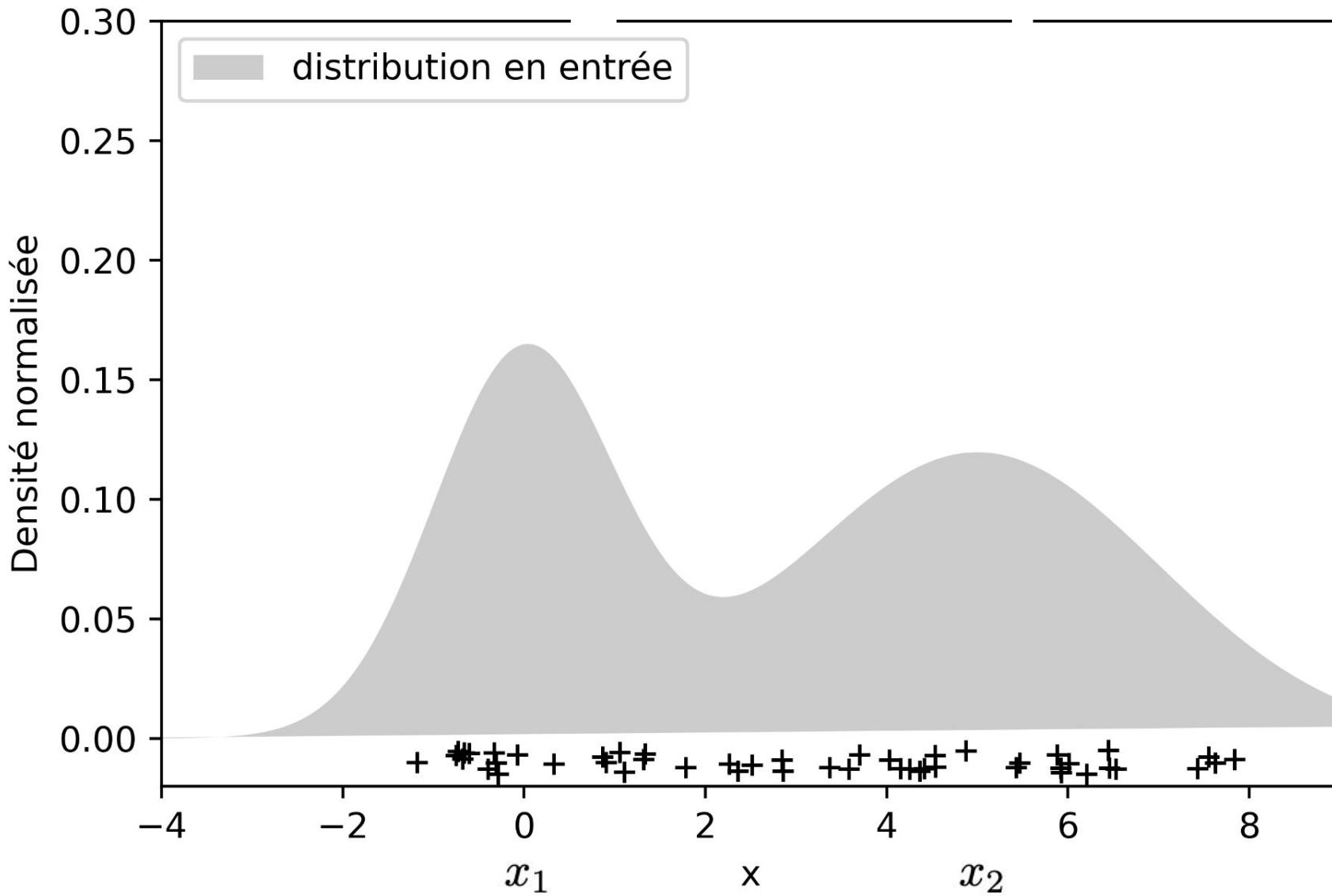
# Principe de fonctionnement du système GPS : trilateration



# Principe de fonctionnement du système GPS : trilateration



# une distribution d'entrée



# L'estimation par noyau (KDE)

→ méthode d'estimation de la densité de probabilité d'une variable aléatoire

Noyau : fonction, notée  $K$ , positive, intégrable à valeurs réelles vérifiant :

→ Symétrie :

$$K(-u) = K(u)$$

→ Normalisation :

$$\int_{-\infty}^{+\infty} K(u)du = 1$$

densité de probabilité

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

le noyau gaussien

# La modélisation de l'estimation

pour choisir une échelle adaptée aux données

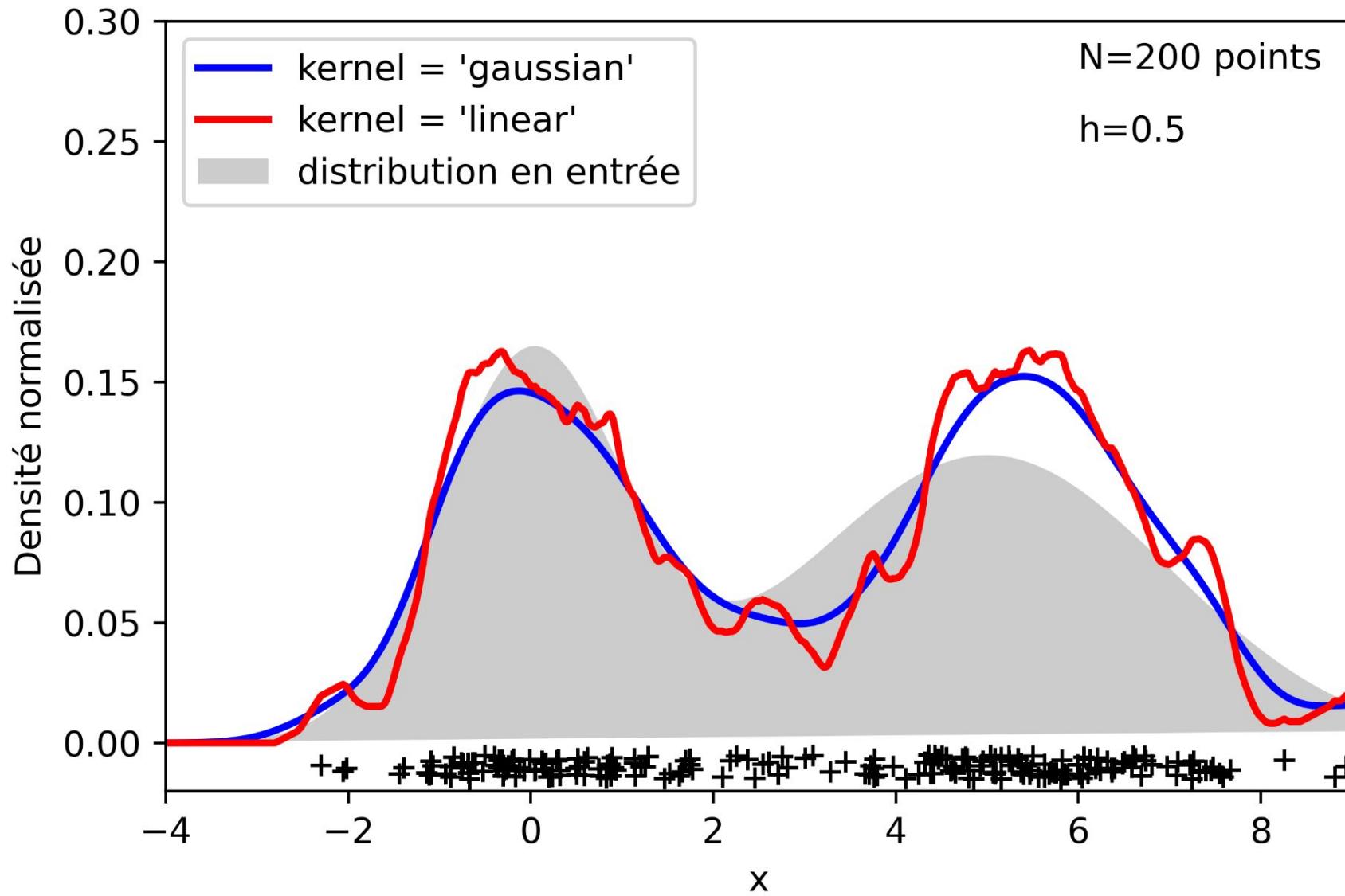
$$K_i^*(x) = \frac{1}{h} K \left( \frac{x - x_i}{h} \right) h > 0$$

centré sur  $x_i$ , d'écart-type  $h$

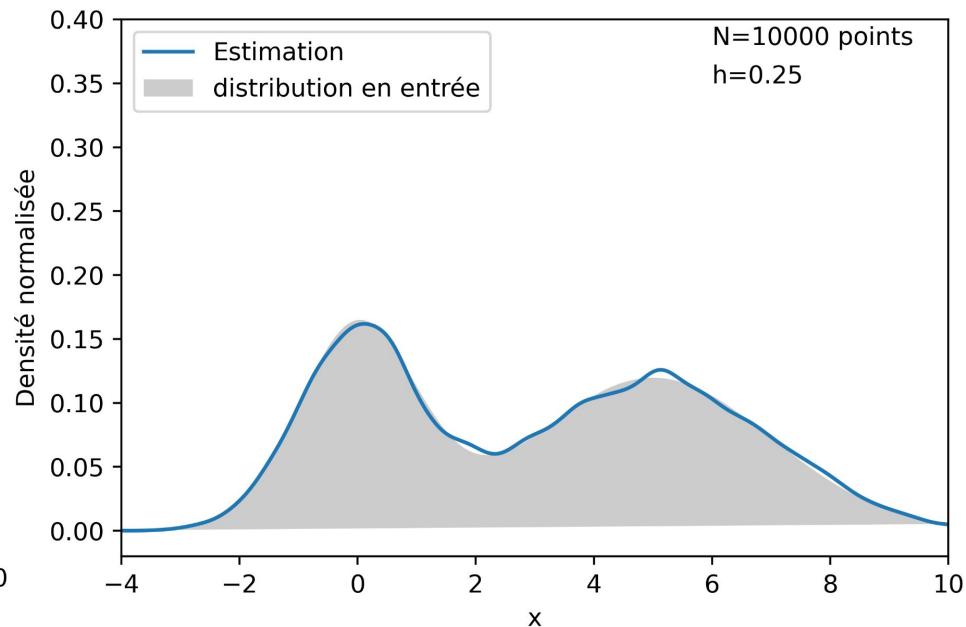
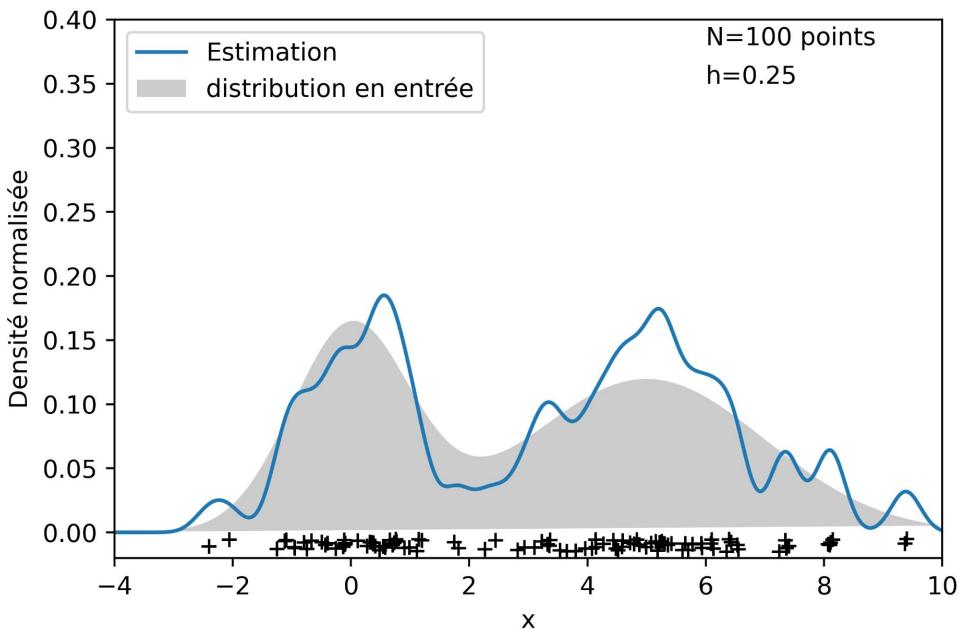
Fonction d'estimation :

$$f(x) = \frac{1}{2} (K_1^*(x) + K_2^*(x))$$

# Comparaison des différents types de noyau

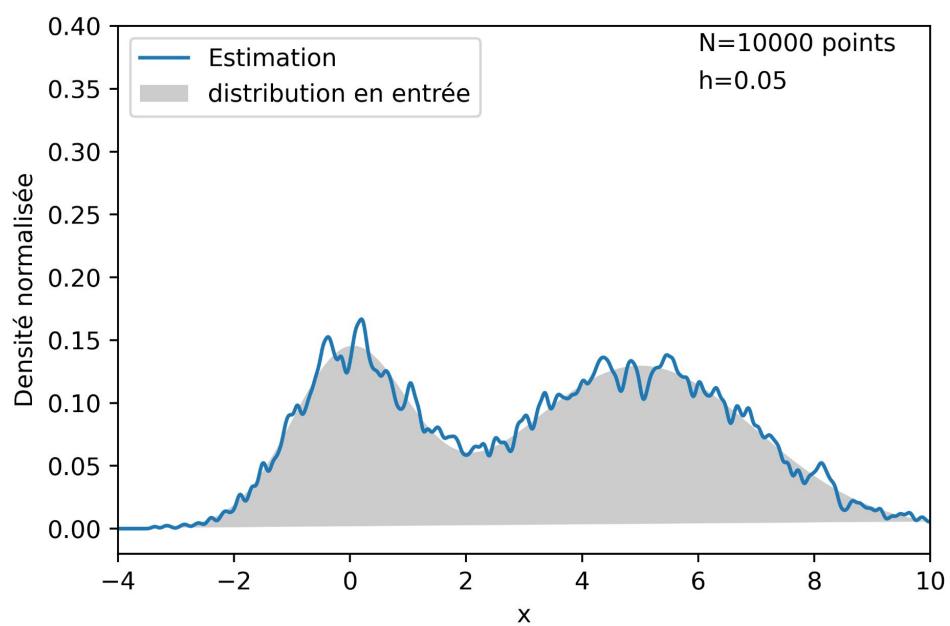
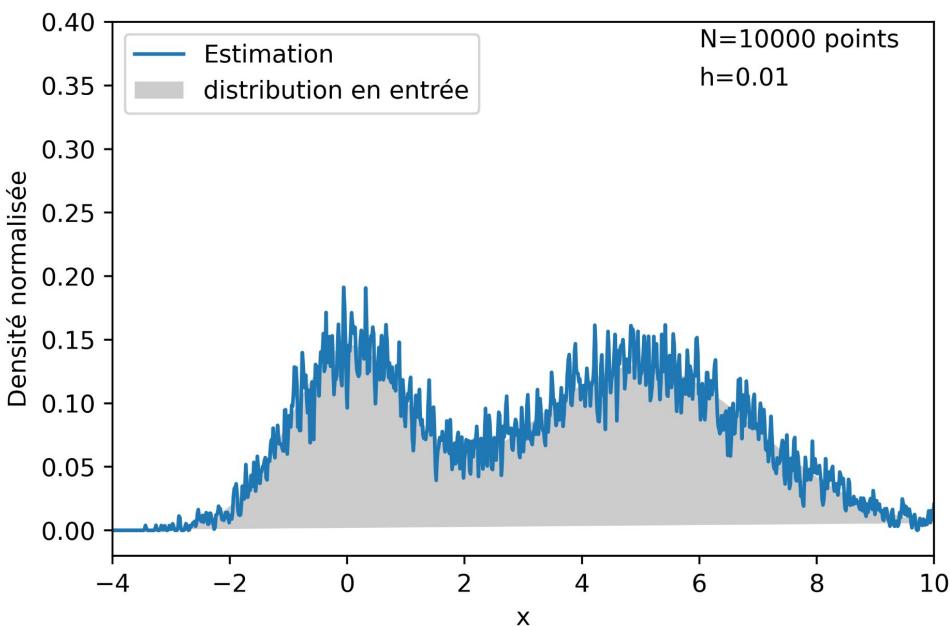


# Influence du nombre de points (N) de la distribution



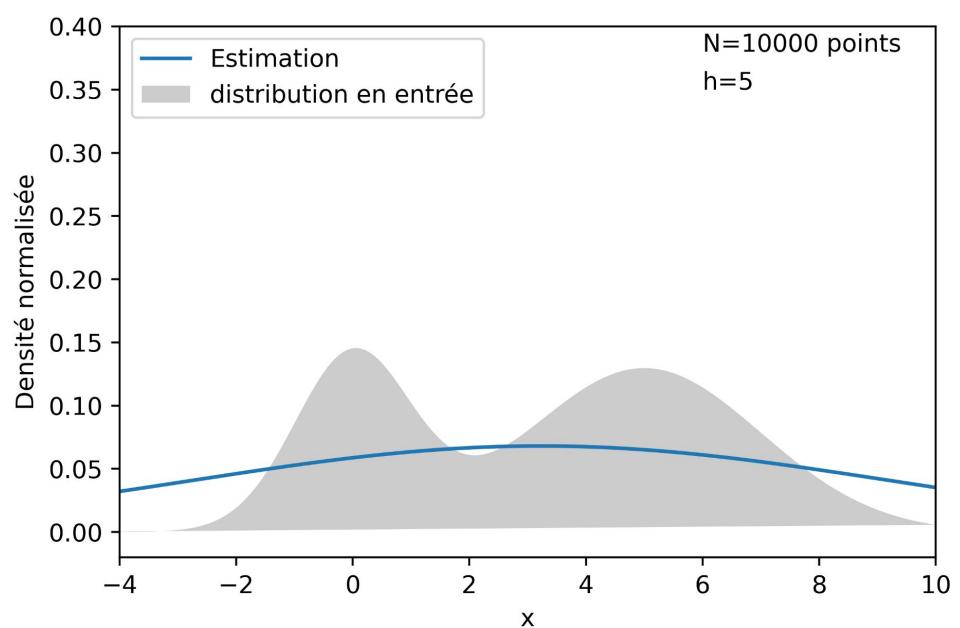
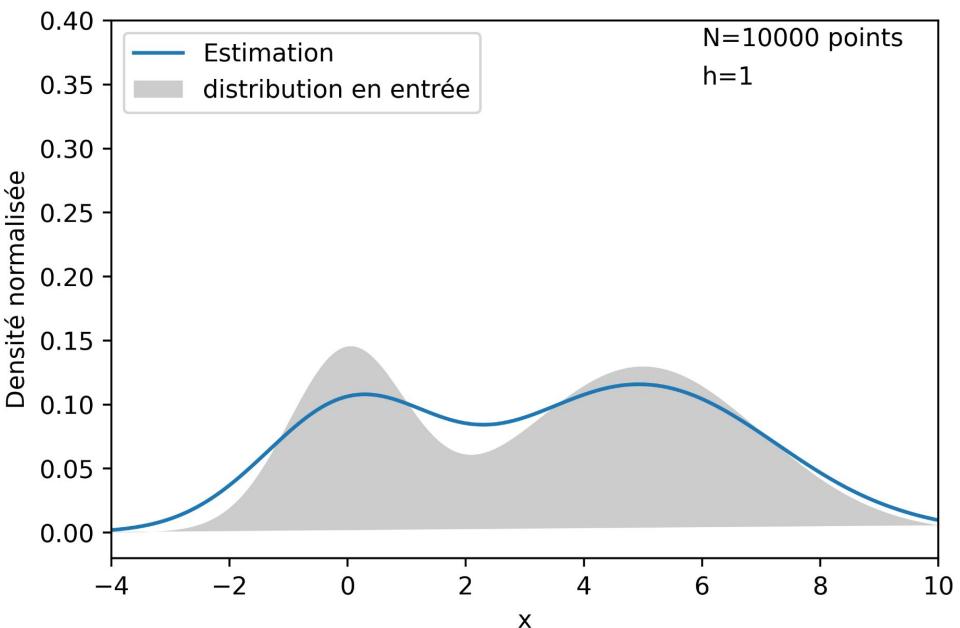
- N petit, la densité estimée écartée de l'entrée
- N grand, écart entre l'estimation et l'entrée réduit

# Influence de $h$ (paramètre de lissage)



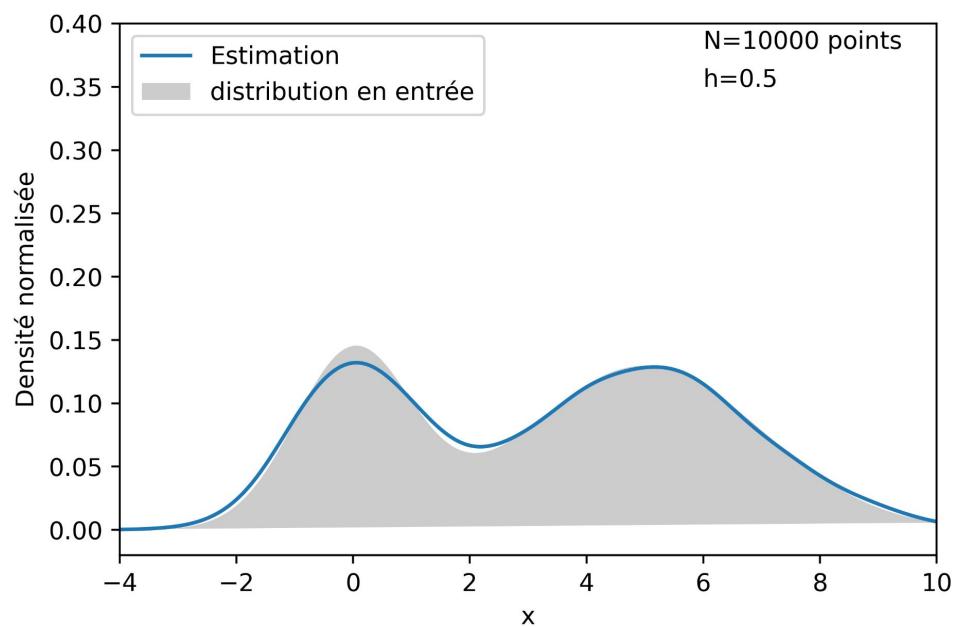
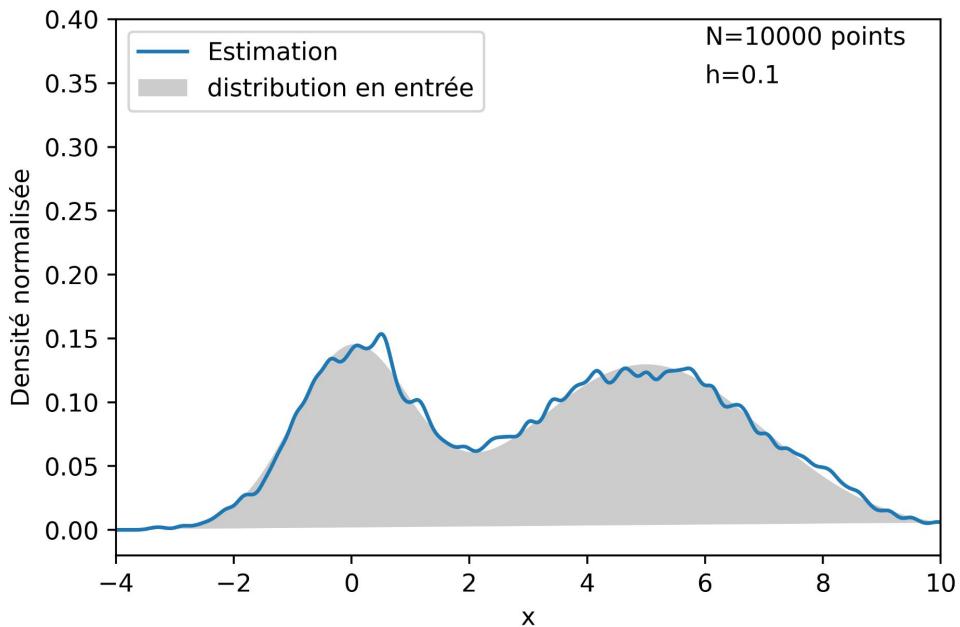
→ courbes pas assez lissées, apparition de détails artificiels

# Influence de $h$ (paramètre de lissage)



→ courbes trop lissées, majorité des caractéristiques effacées

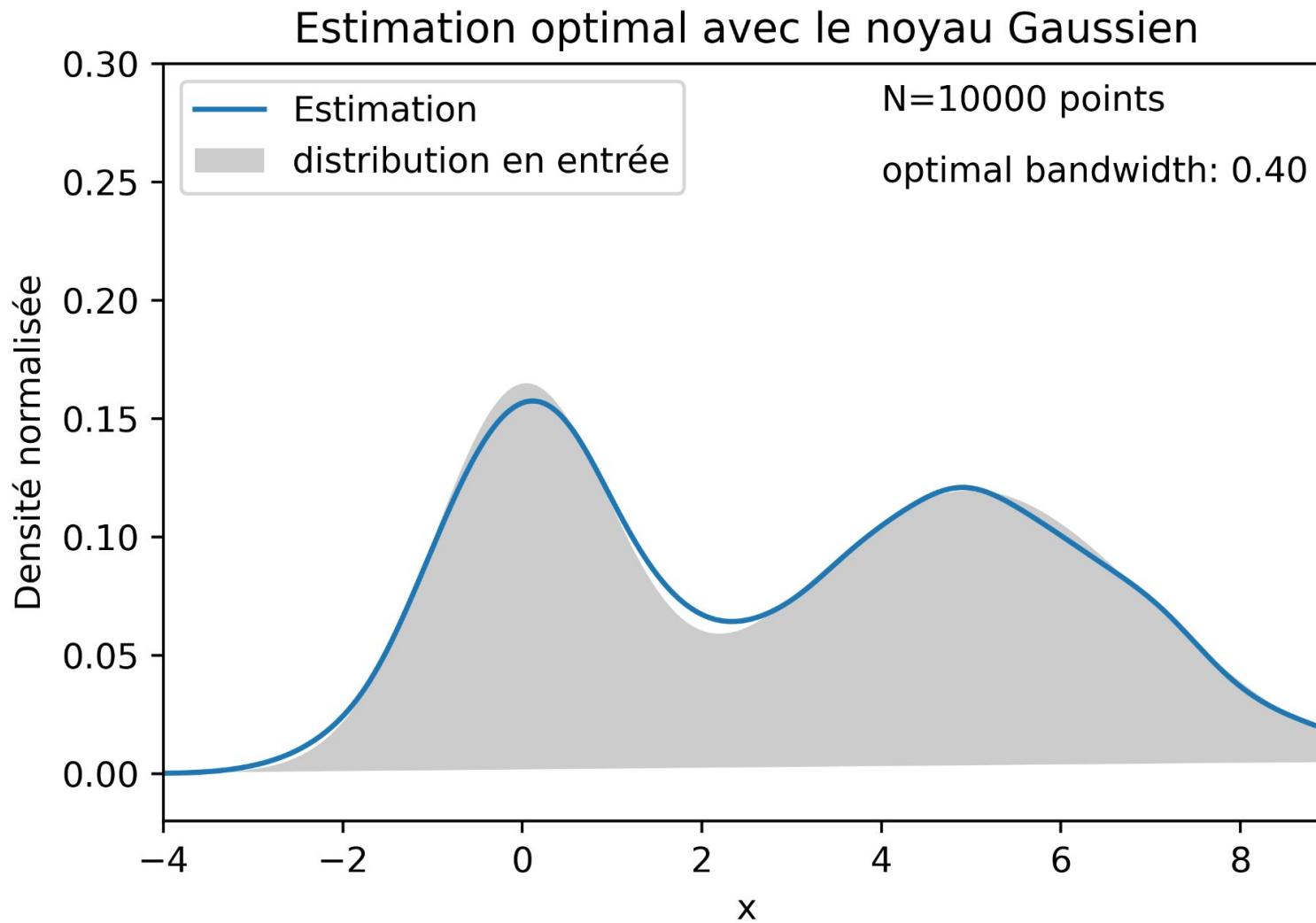
# Influence de $h$ (paramètre de lissage)



→ courbes qui suivent bien la courbe de distribution

→ nécessité de choisir une valeur équilibrée pour ce paramètre

# choix de $h$ (paramètre de lissage)



→ choix de  $h$  fait directement par le programme

# Description de l'Expérience Statique



# Résultats des mesures statiques réalisées



## GPS exp 22h

Événement: -- ▾

Type d'événement: Sans catégorie ▾

Parcours: -- ▾

Equipement: Ajouter

9.95 km

Distance

22:02:01

Durée

0 /km

Allure moyenne

45 m

Ascension totale

Comparer

Envoyer vers l'appareil

Enregistrer en tant que parcours

Définir comme RP

Exporter au format original

Exporter en TCX

Exporter en GPX

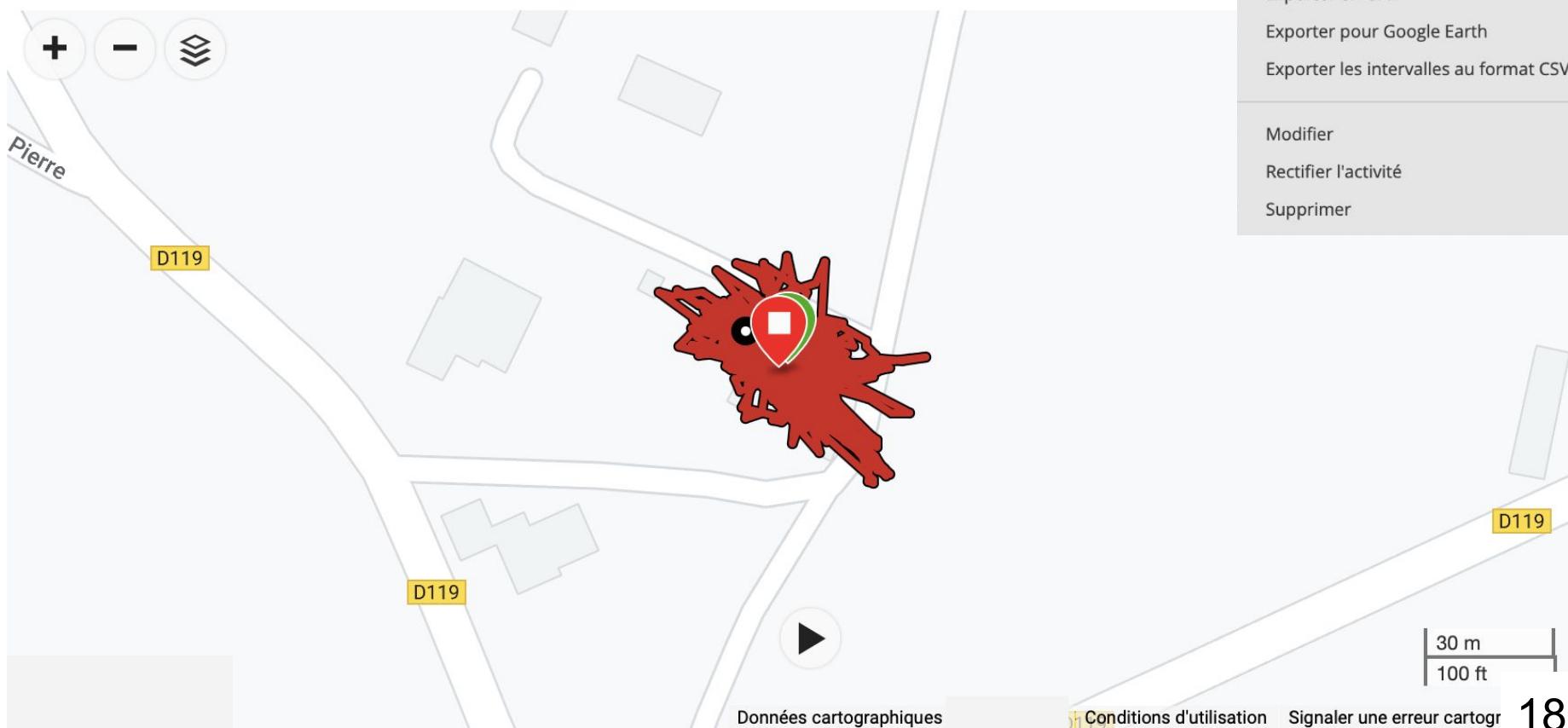
Exporter pour Google Earth

Exporter les intervalles au format CSV

Modifier

Rectifier l'activité

Supprimer



# Résultats des mesures statiques réalisées

0.06 km DISTANCE	14:40 DURÉE	-- ASCENSION TOTALE
0.66 km DISTANCE	1:22:20 DURÉE	1 m ASCENSION TOTALE
1.24 km DISTANCE	3:33:36 DURÉE	-- ASCENSION TOTALE
3.61 km DISTANCE	10:47:30 DURÉE	5 m ASCENSION TOTALE
6.02 km DISTANCE	19:32:27 DURÉE	1 m ASCENSION TOTALE
9.95 km DISTANCE	22:02:01 DURÉE	45 m ASCENSION TOTALE

# Extraction des données

```
<Position>
  <LatitudeDegrees>46.17750618606806</LatitudeDegrees>
  <LongitudeDegrees>4.680427573621273</LongitudeDegrees>
</Position>
<AltitudeMeters>325.20001220703125</AltitudeMeters>
<DistanceMeters>17.420000076293945</DistanceMeters>
<Extensions>
  <ns3:TPX>
    <ns3:Speed>0.0</ns3:Speed>
    <ns3:RunCadence>0</ns3:RunCadence>
  </ns3:TPX>
</Extensions>
</Trackpoint>
<Trackpoint>
  <Time>2022-05-01T07:33:45.000Z</Time>
  <Position>
    <LatitudeDegrees>46.177517250180244</LatitudeDegrees>
    <LongitudeDegrees>4.680402260273695</LongitudeDegrees>
  </Position>
  <AltitudeMeters>325.20001220703125</AltitudeMeters>
  <DistanceMeters>19.73999977118164</DistanceMeters>
  <Extensions>
    <ns3:TPX>
      <ns3:Speed>0.0</ns3:Speed>
      <ns3:RunCadence>0</ns3:RunCadence>
    </ns3:TPX>
  </Extensions>
</Trackpoint>
<Trackpoint>
  <Time>2022-05-01T07:33:58.000Z</Time>
  <Position>
    <LatitudeDegrees>46.1775179207325</LatitudeDegrees>
    <LongitudeDegrees>4.680405529215932</LongitudeDegrees>
  </Position>
  <AltitudeMeters>325.20001220703125</AltitudeMeters>
  <DistanceMeters>20.479999542236328</DistanceMeters>
  <Extensions>
    <ns3:TPX>
      <ns3:Speed>0.0</ns3:Speed>
      <ns3:RunCadence>0</ns3:RunCadence>
    </ns3:TPX>
  </Extensions>
</Trackpoint>
<Trackpoint>
```

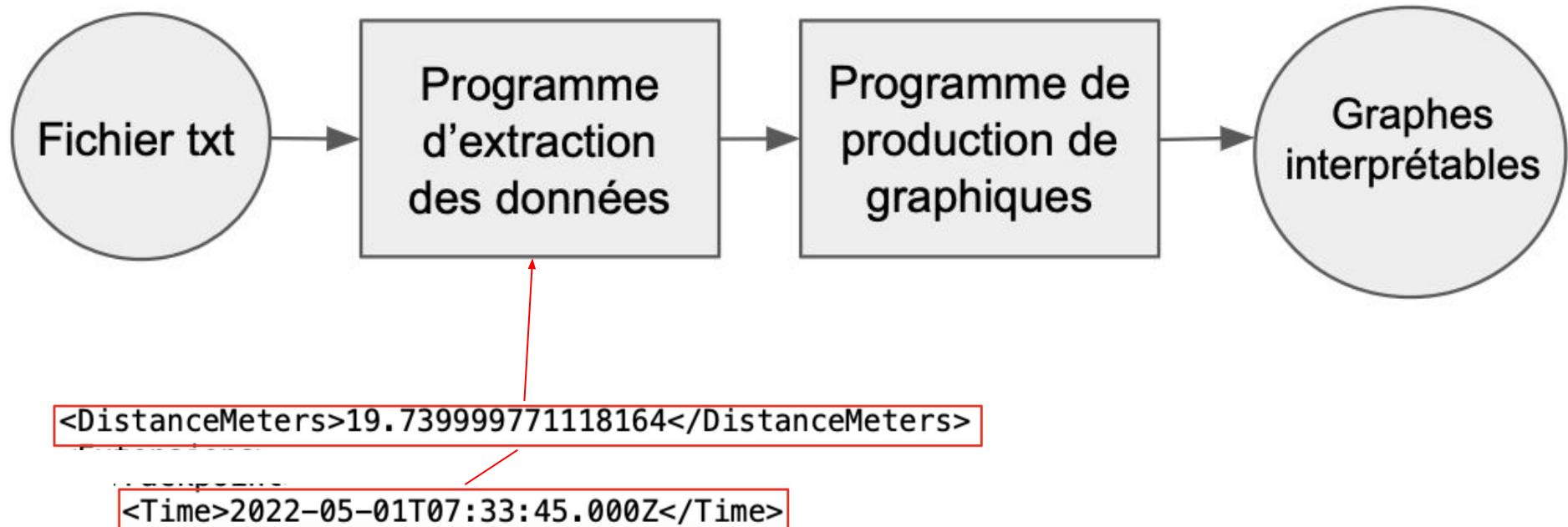


Format type des TCX étudiés

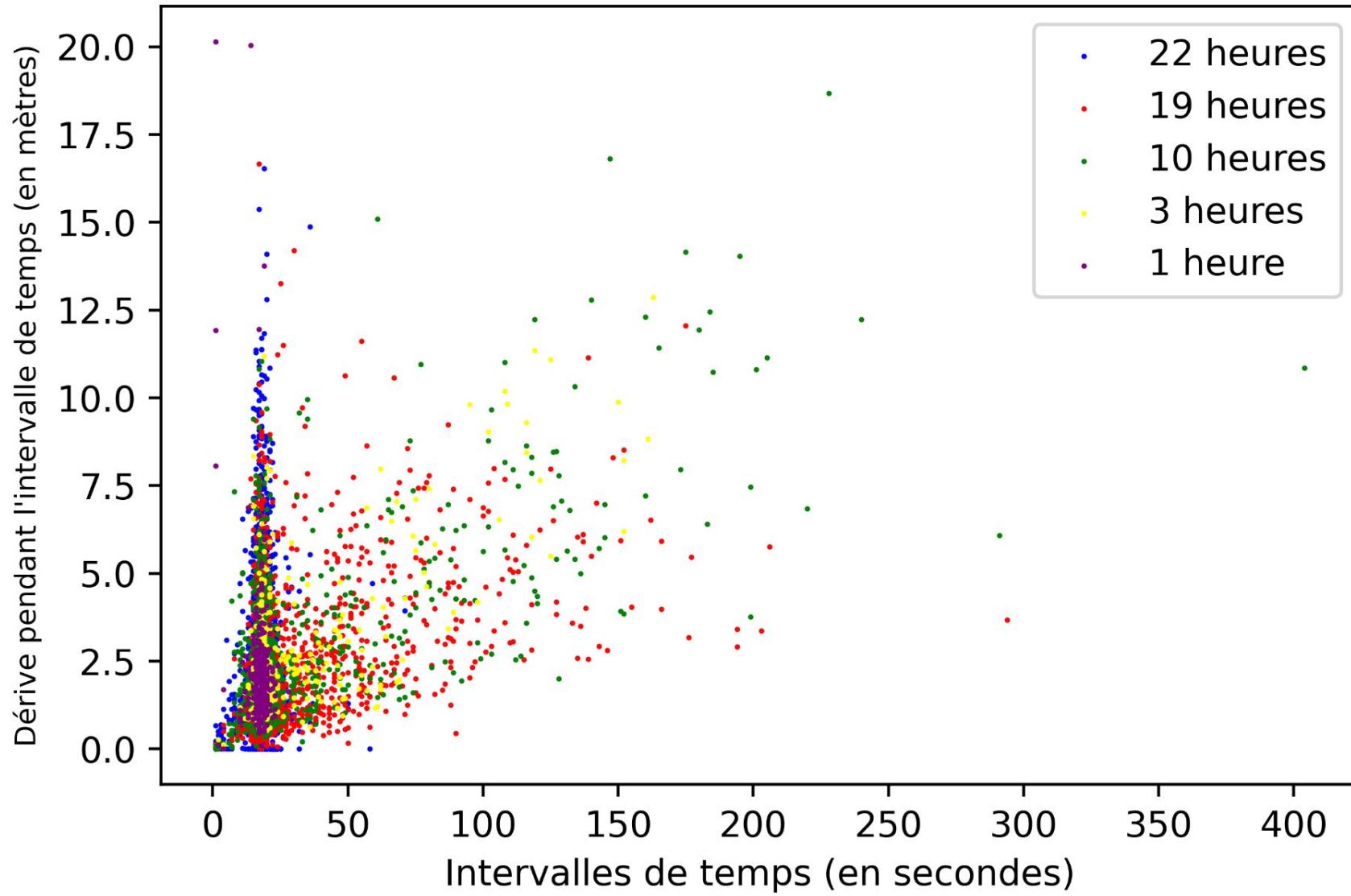
*sur 70 000 lignes*

des milliers de données  
→ Compatible avec l'étude de  
l'outil statistique précédent

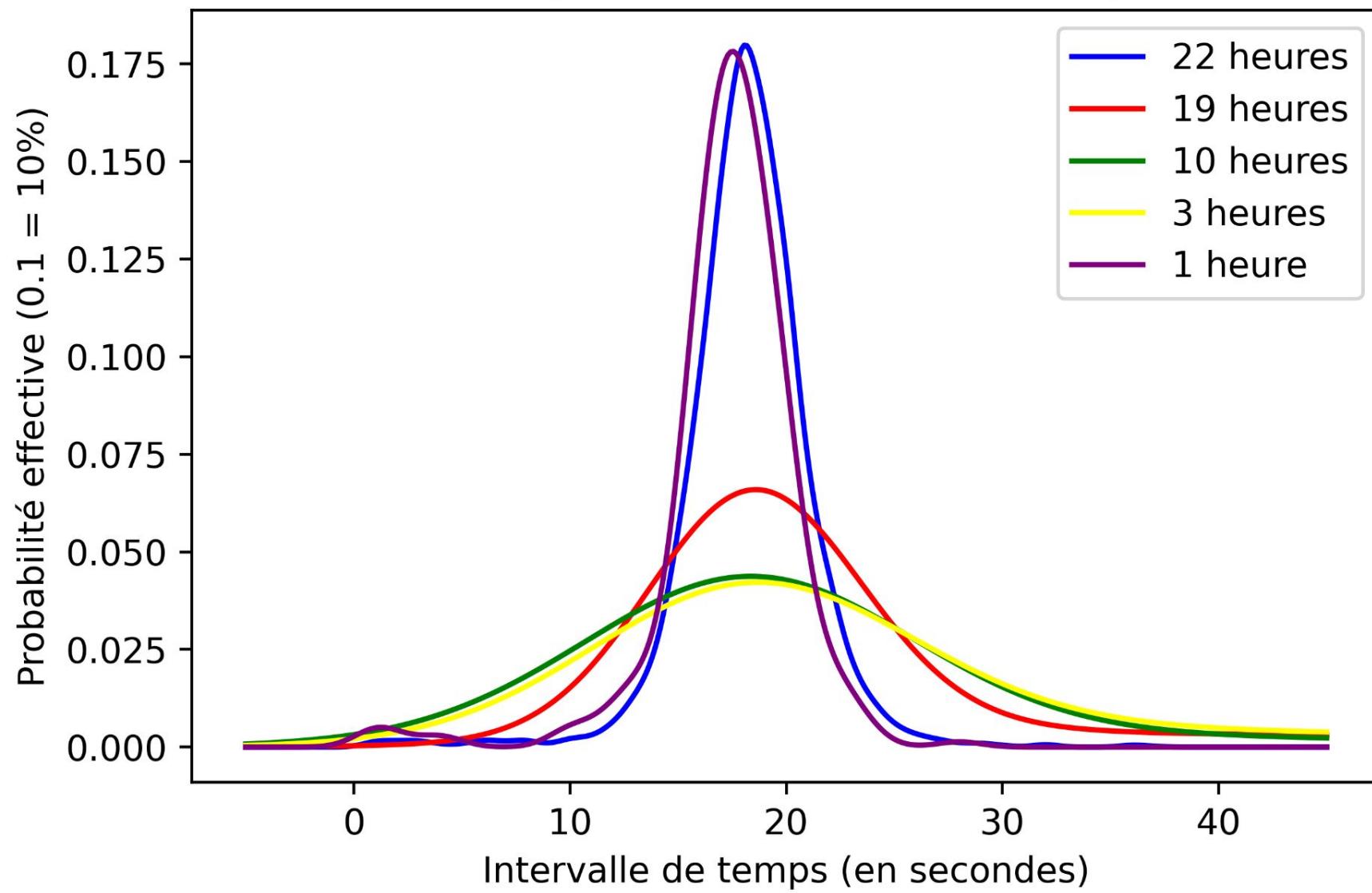
# Extraction des données



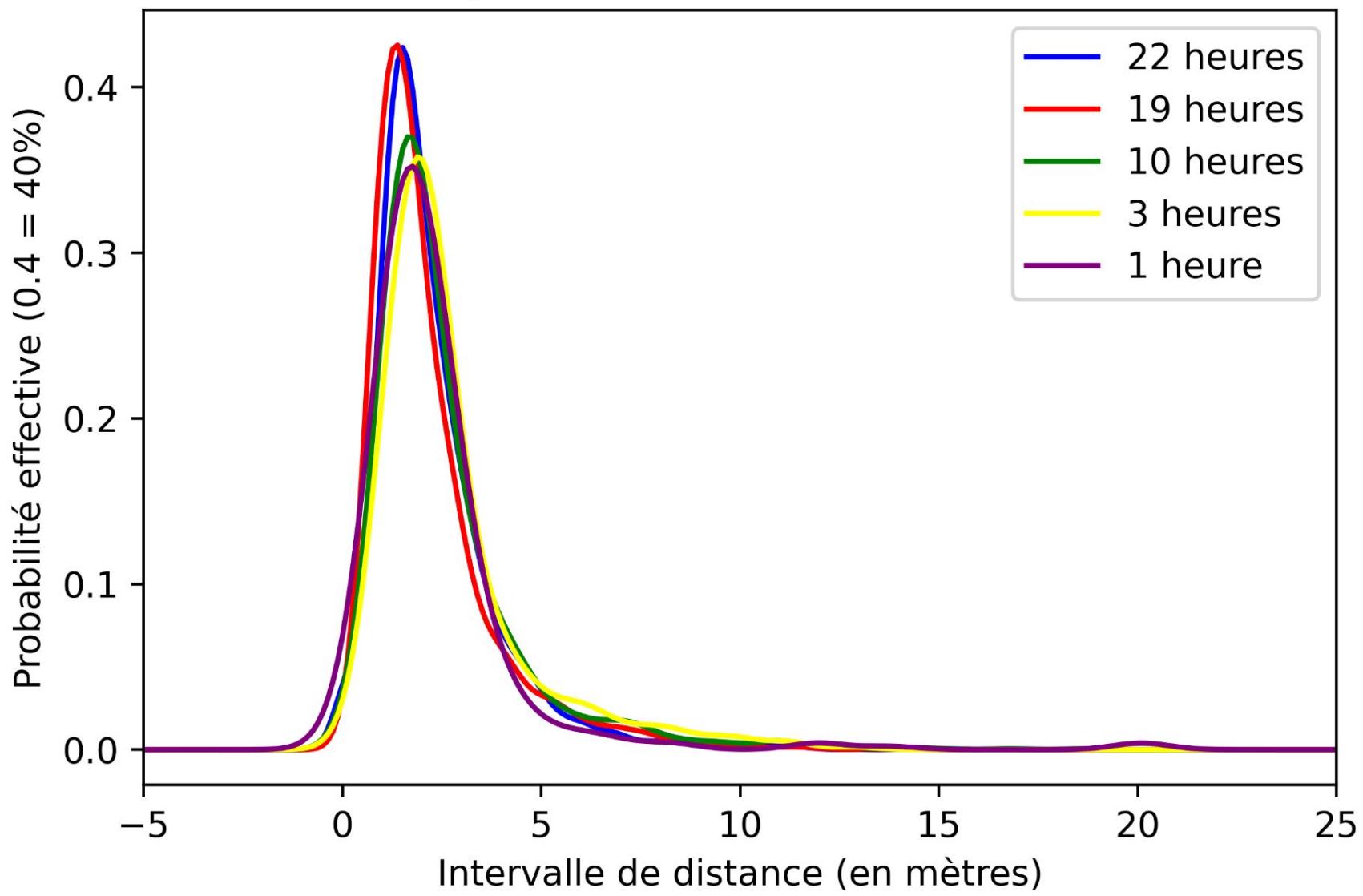
## Répartition des intervalles de distances selon l'intervalle de temps



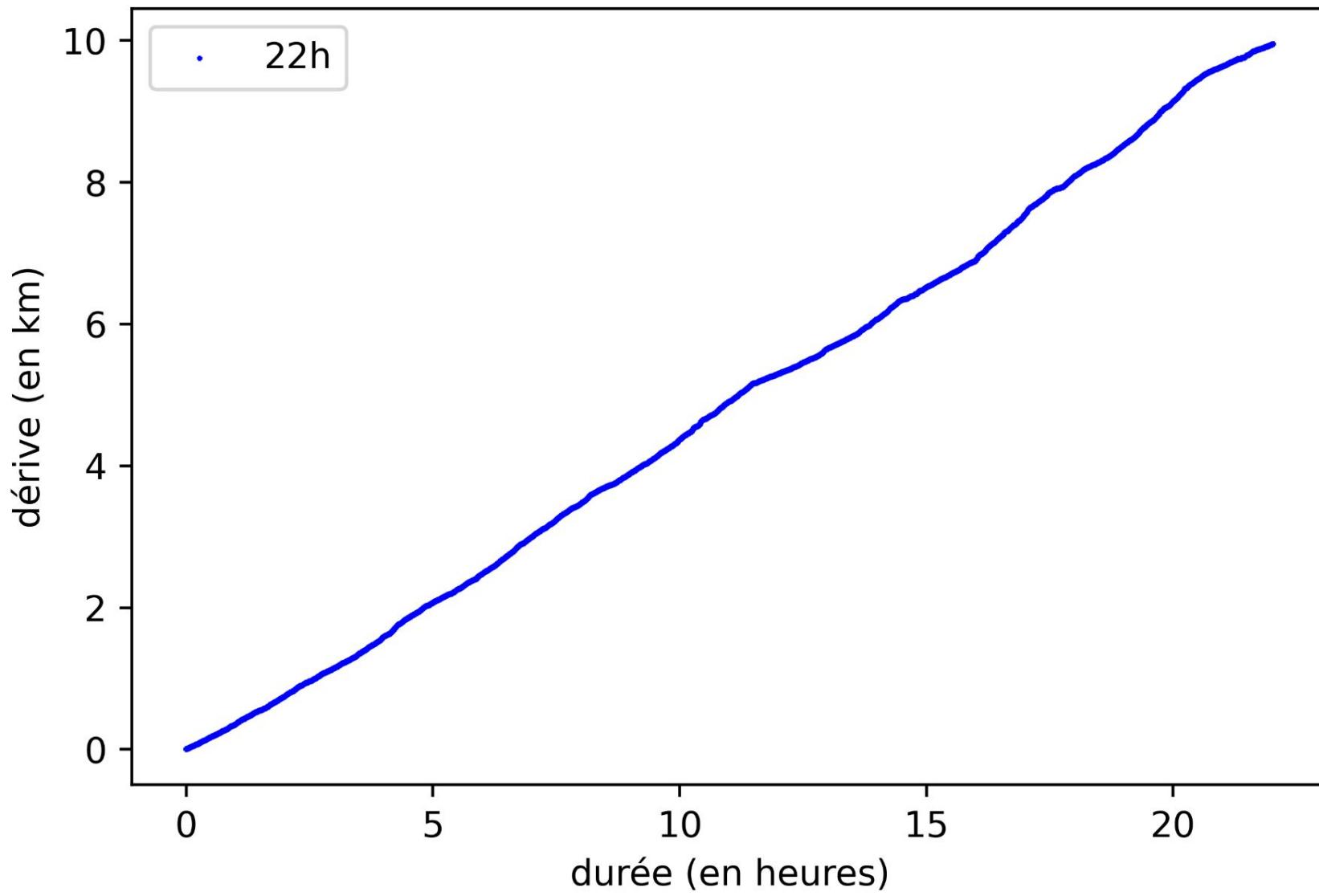
## densité statistique des intervalles de temps mesurés



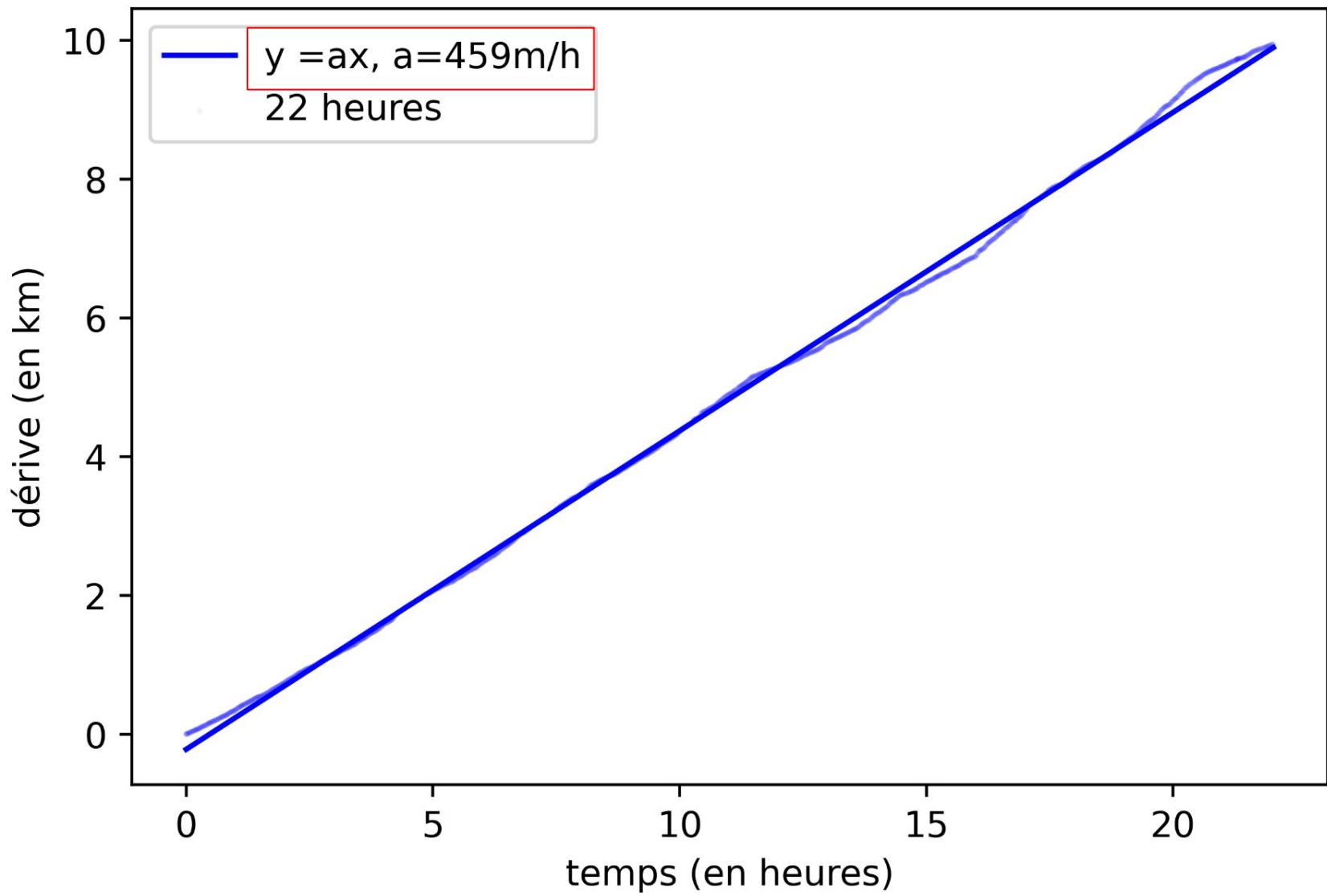
## densité statistique des intervalles de distances mesurés



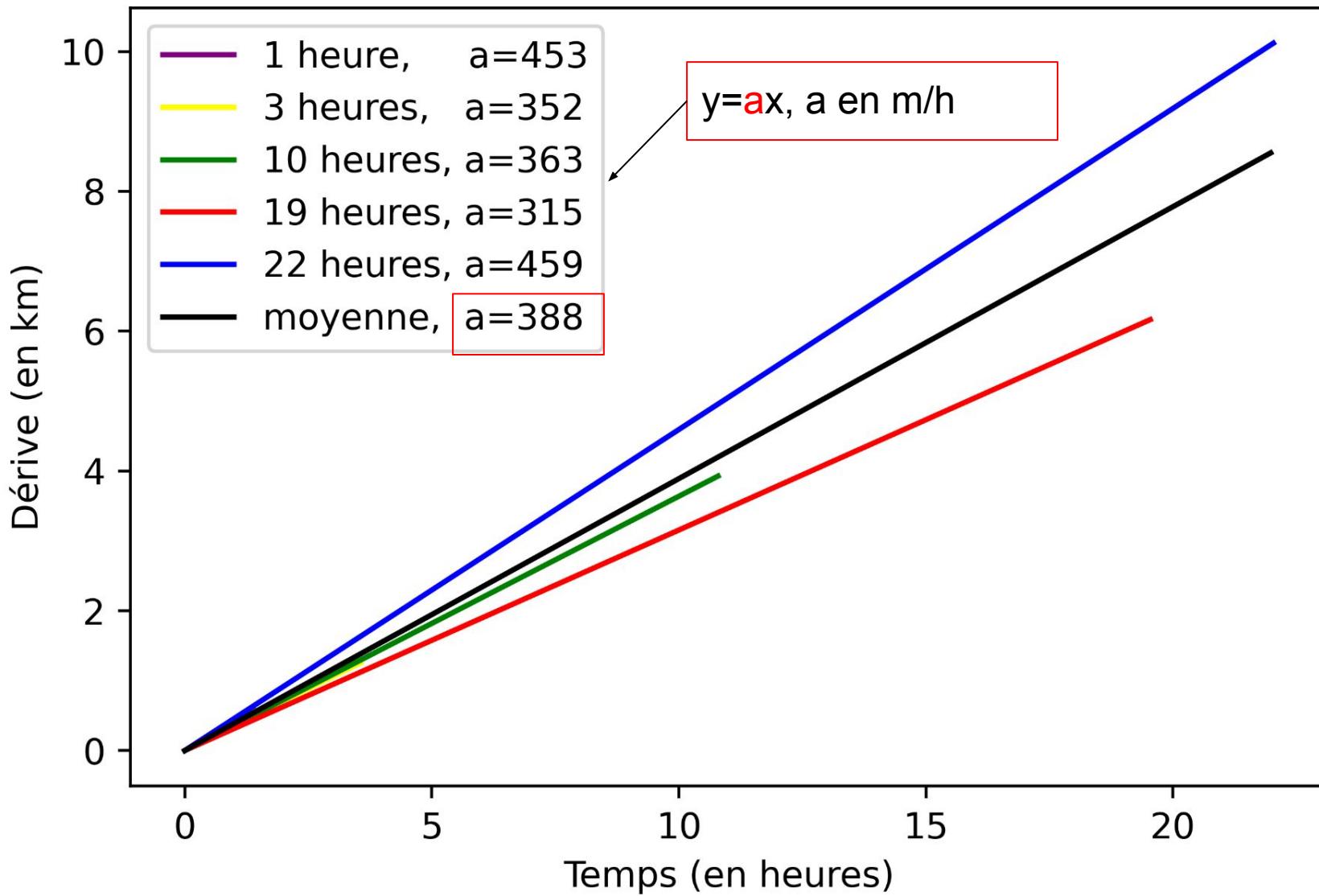
## Dérives du capteur GPS en statique



## régression linéaire et courbe réelle



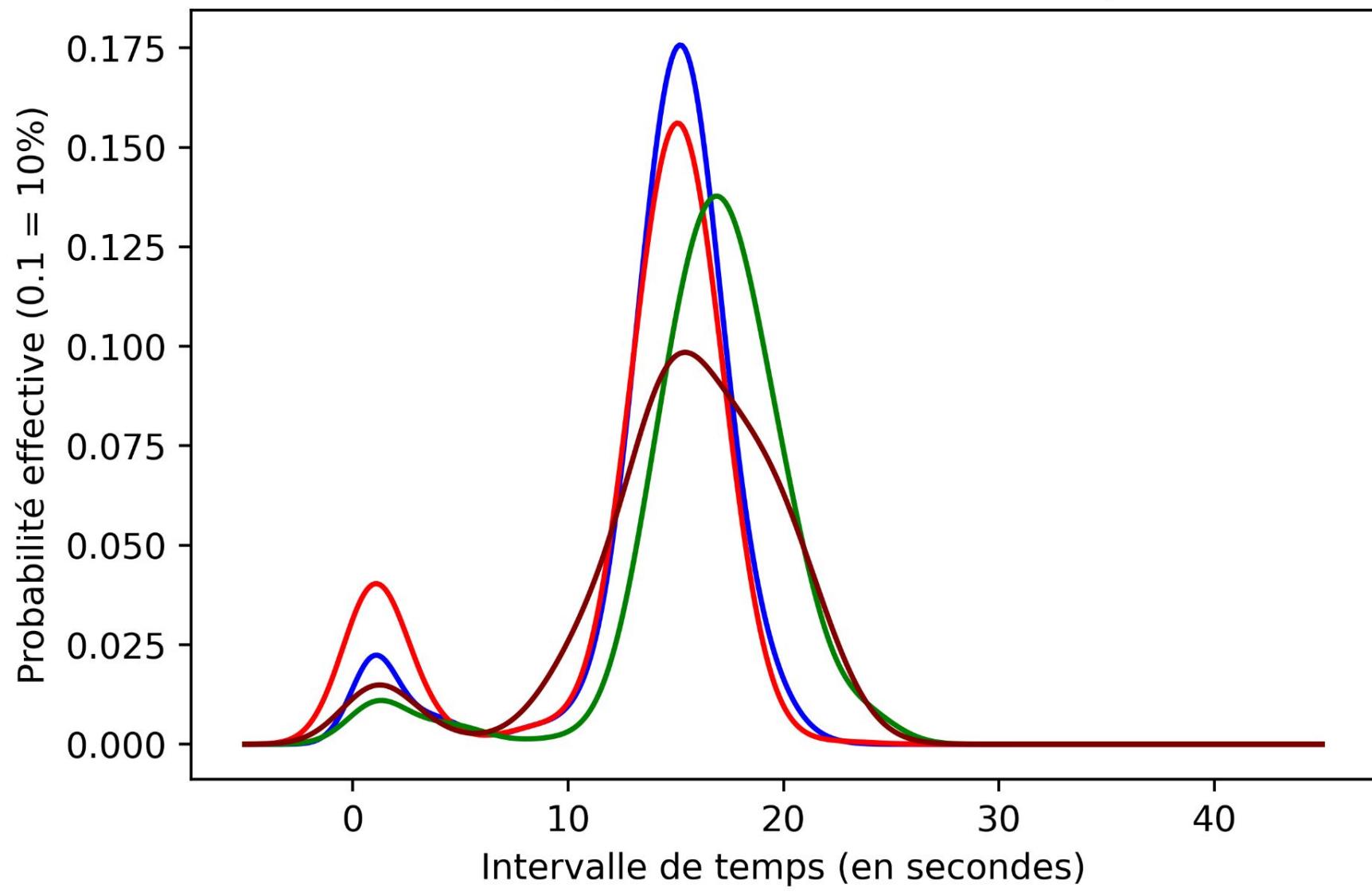
## Régressions linéaires et la moyenne



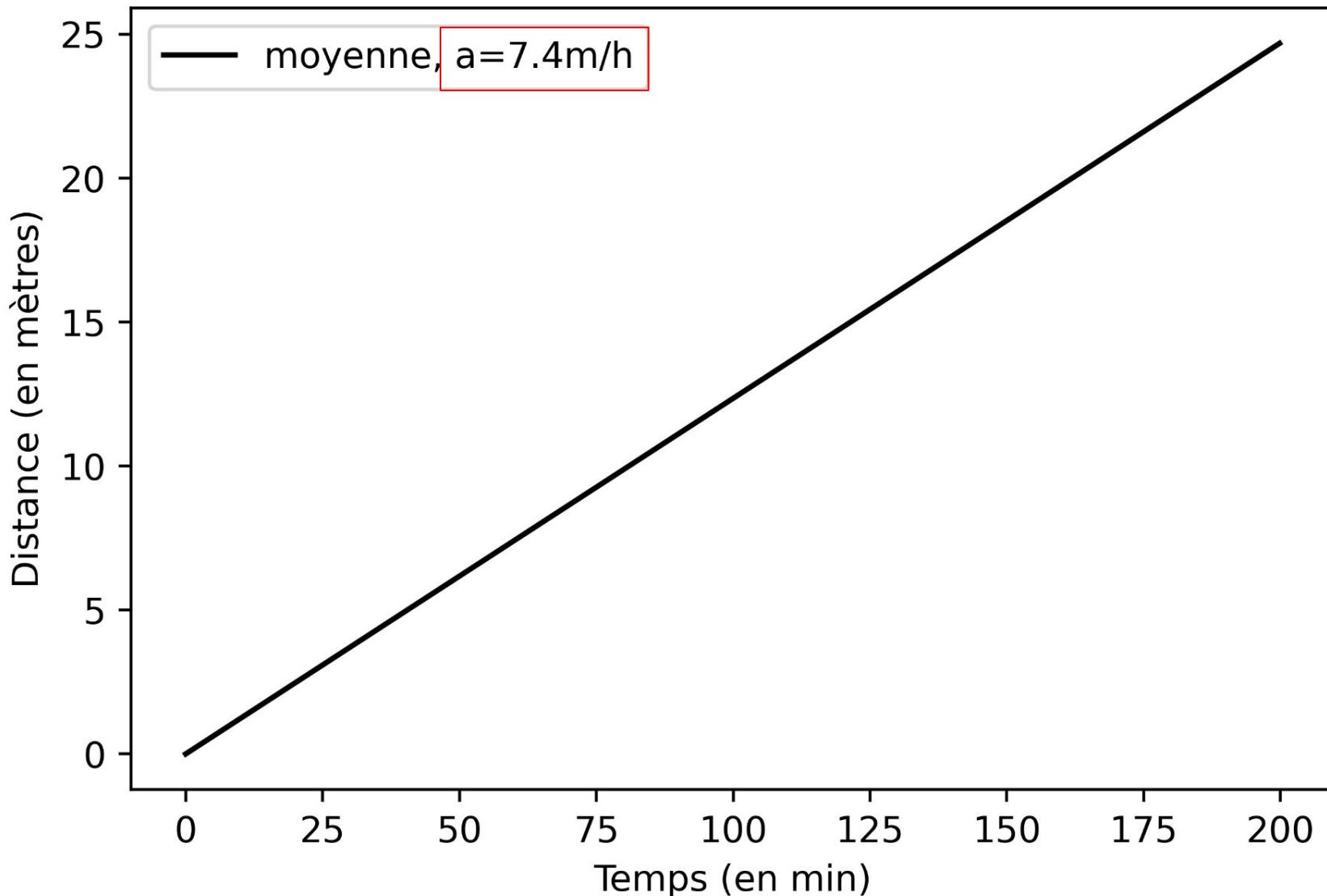
# Expérience complémentaire GPS vélo en statique



## Densité statistique des intervalles de temps mesurés



## régression linéaire moyenne

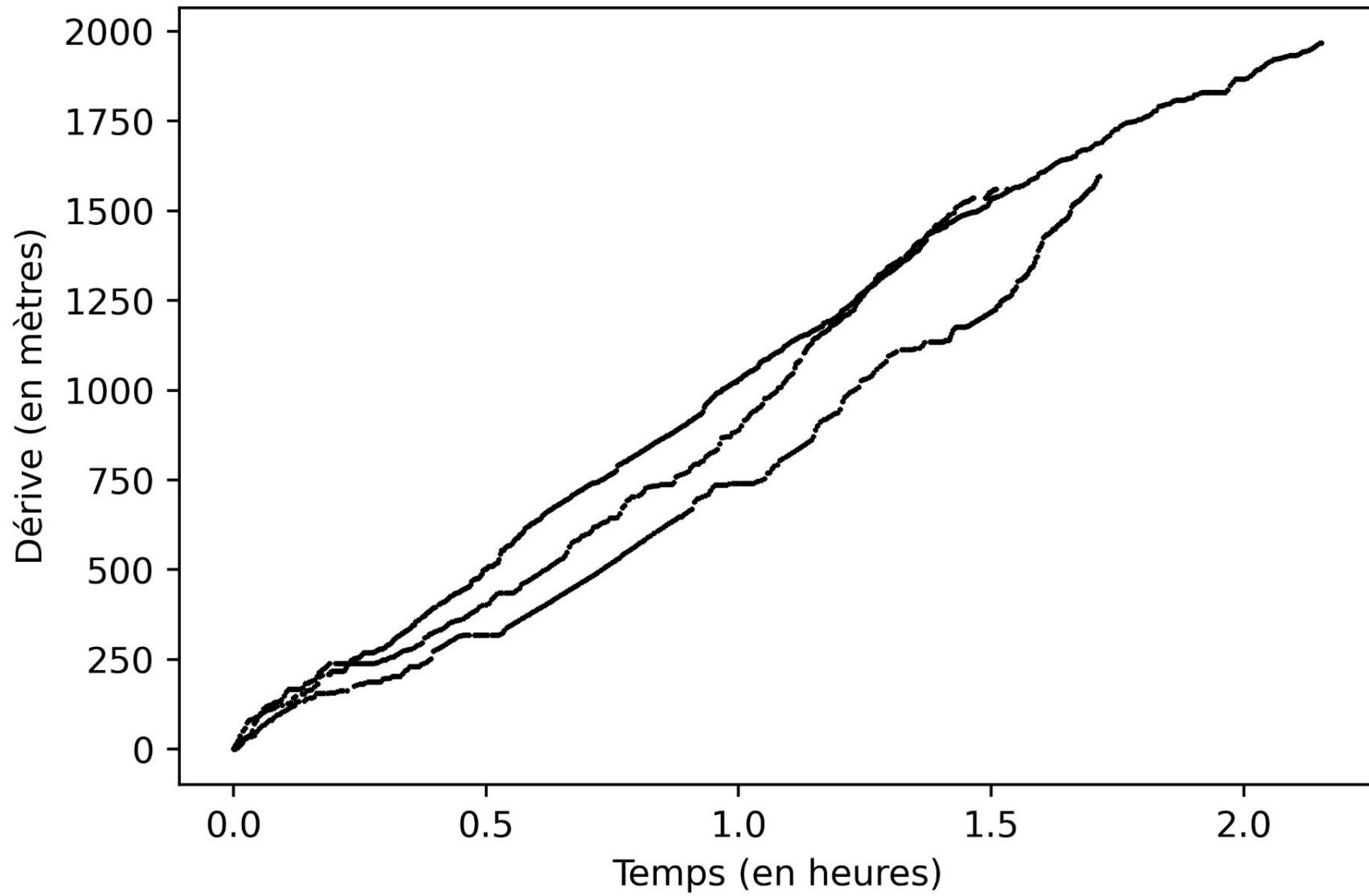


- dérive 50 fois plus faible
- mise en évidence de la présence d'un algorithme de détection de l'immobilité sur ce type de GPS

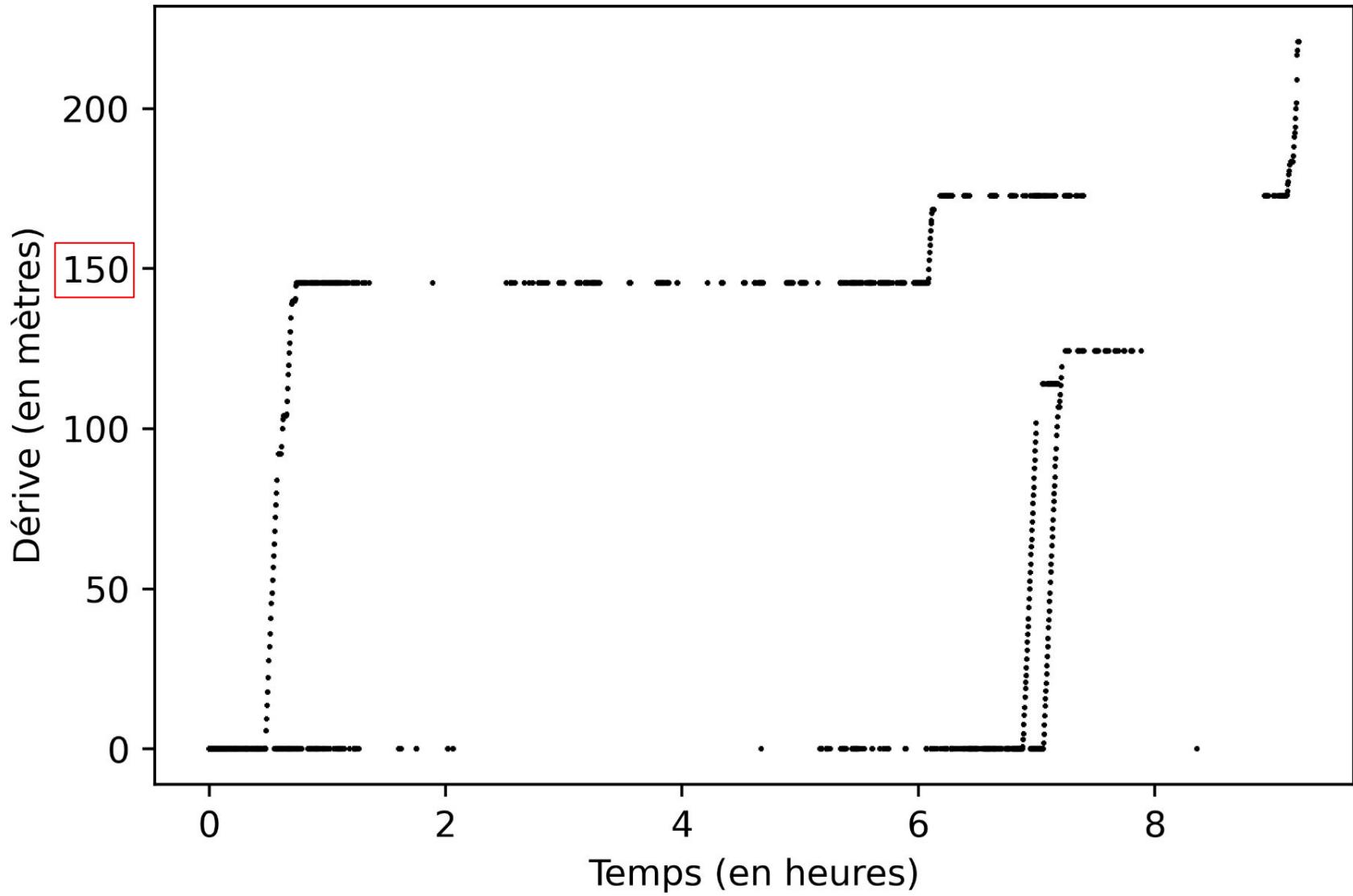
# Expérience statique de comparaison entre des volets ouverts et fermés : simulation d'un obstacle



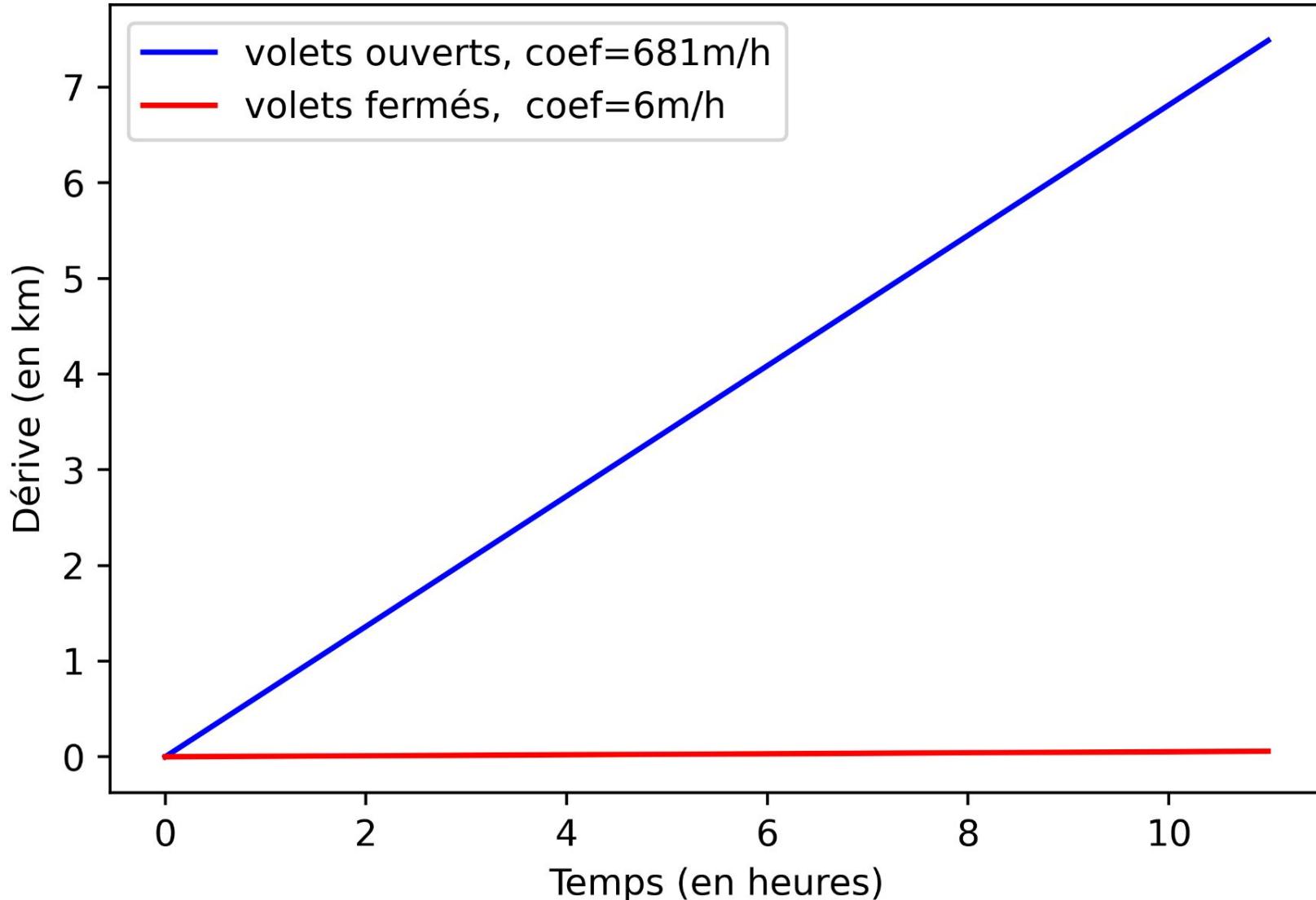
## Dérives en statique en intérieur, volets ouverts



## Dérives en statique en intérieur, volets fermés

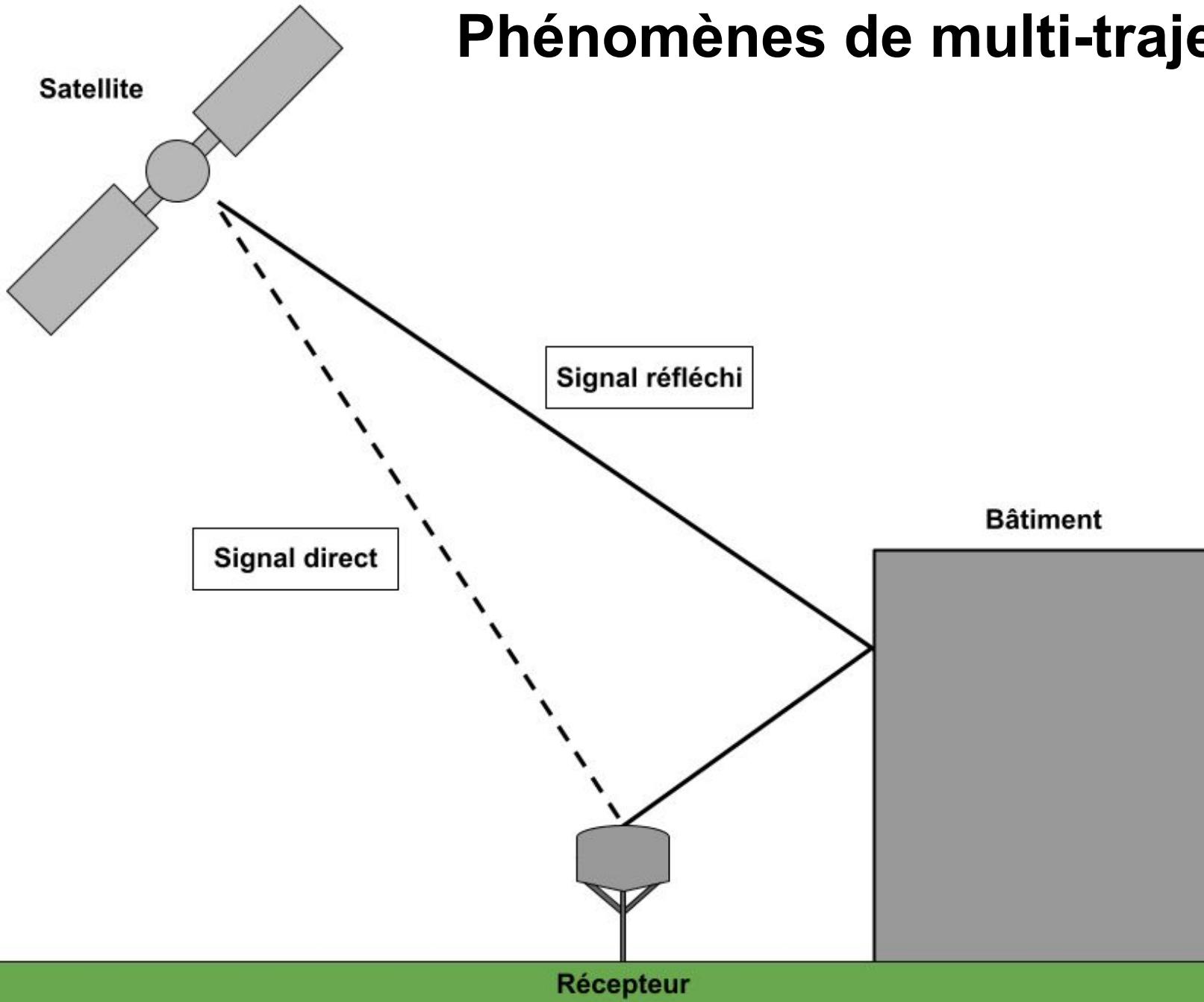


## régressions linéaires et leur moyenne



- erreurs presque doublées avec les volets ouverts
- pas de dérive volets fermés : difficulté à établir la connexion

# Phénomènes de multi-trajets



# en statique dans une cage en PVC



le capteur à une  
hauteur variable  $h$

$h=0$

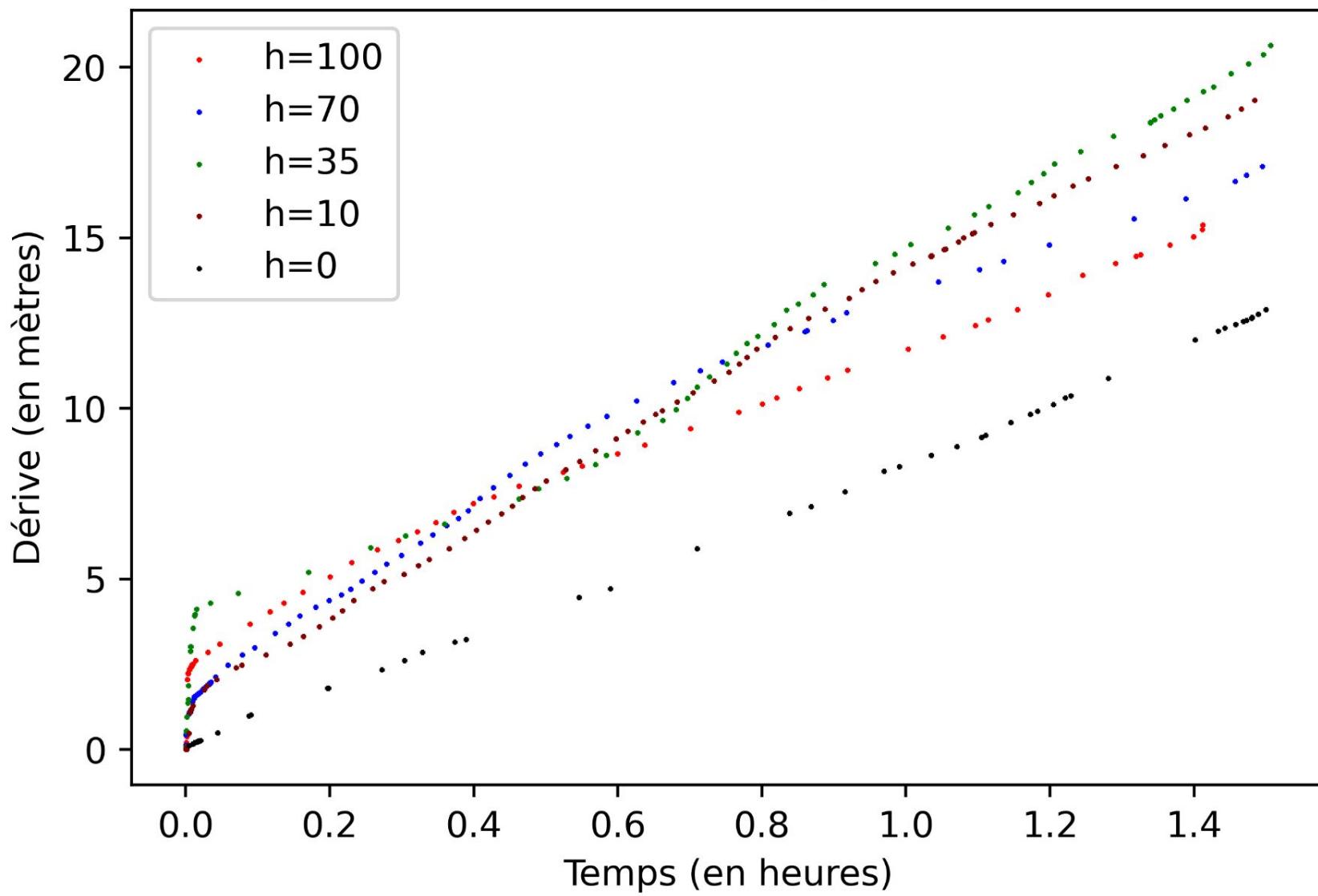
$h=10$

$h=35$

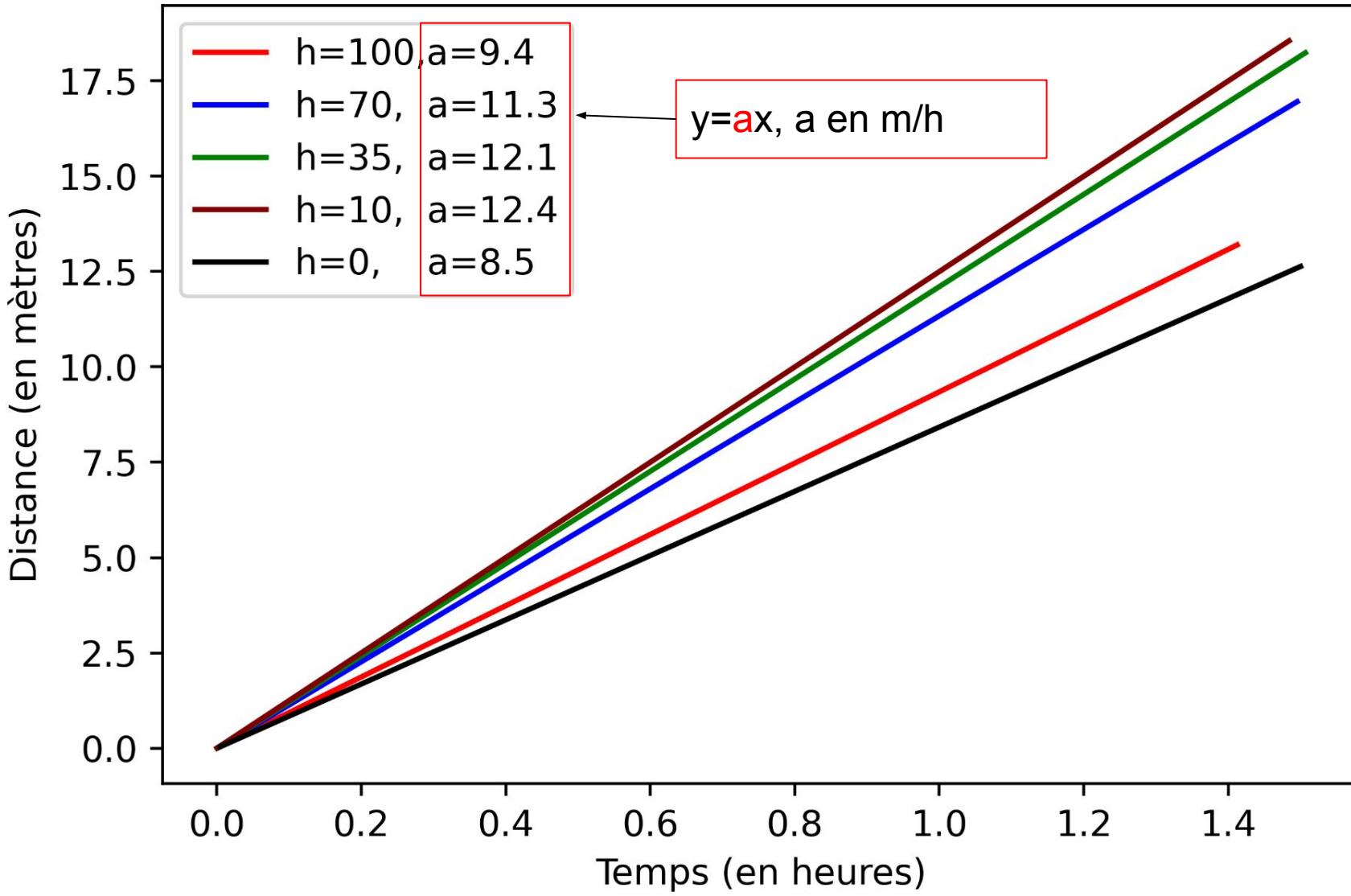
$h=70$

$h=100$

## Dérives en statique, PVC



## régressions linéaires dans le PVC



→ pas d'effets notables

# Description de l'Expérience Dynamique

⇒ Garmin 935 XT, 735 XT, EDGE 1000

Mesure réelle avec un décamètre



1 tour = 253,5 m



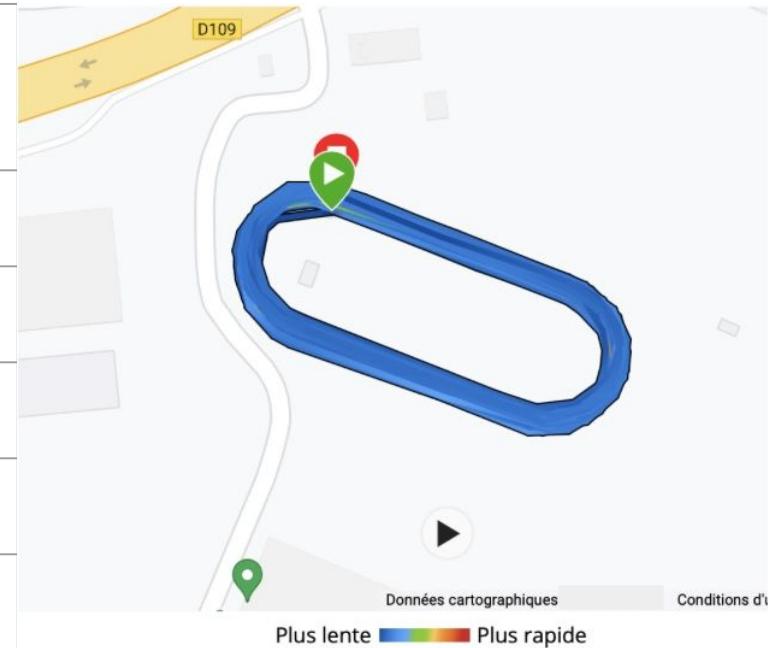
3 expériences avec 4 capteurs



25, 50 et 75 tours

# Résultats Brutes de l'Expérience Dynamique

Appareil de mesure :	Distance (km) : 50 tours en 40 min
Garmin Edge 1040	12,67
Garmin 935 XT	12,83
Garmin 735 XT	12,78
Compteur mécanique	12,80
Données réelles*	12,6750



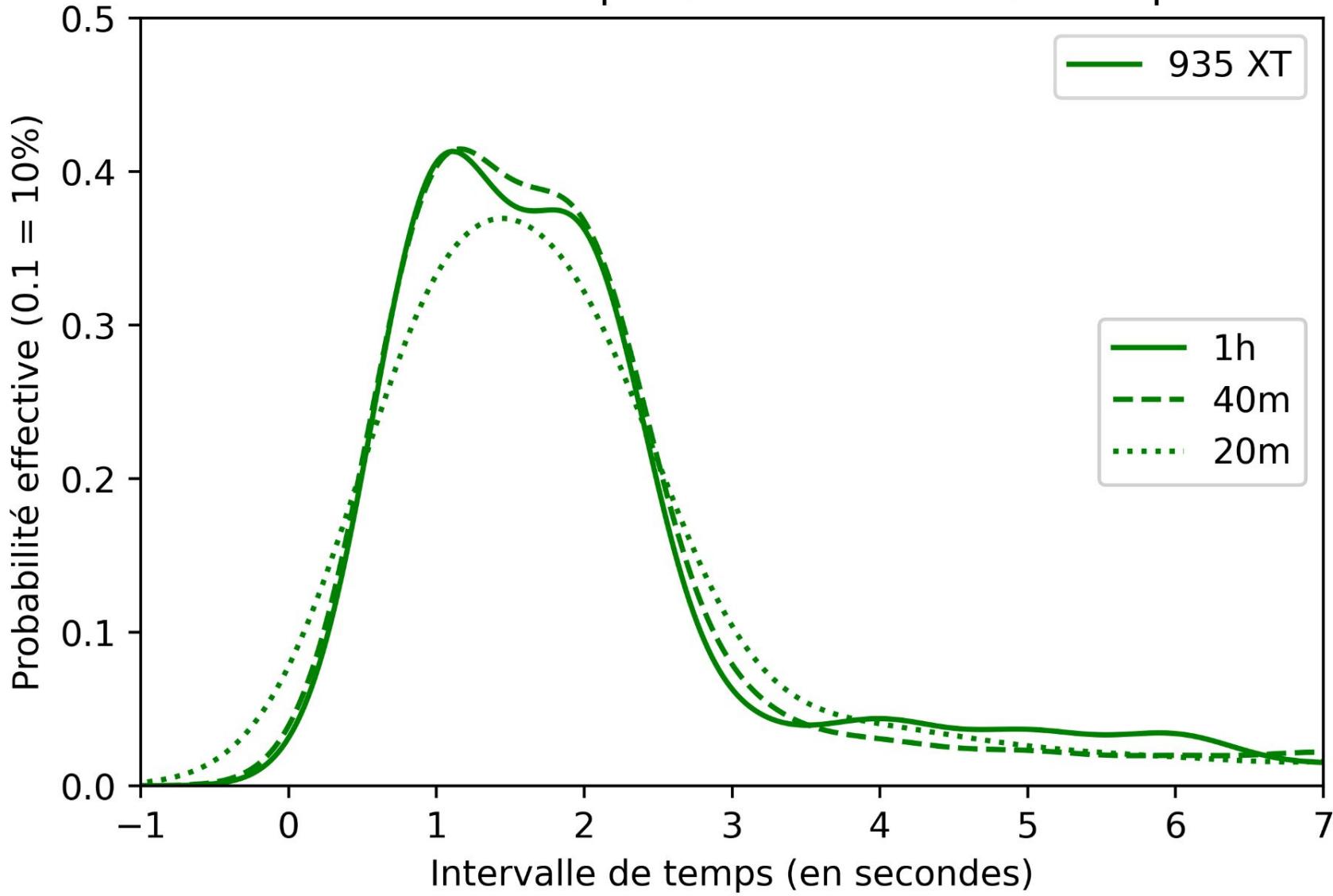
\*base de 253,5 mètres par tour, calcul d'une vitesse moyenne de référence pour observer les écarts

# Résultats Brutes de l'Expérience Dynamique

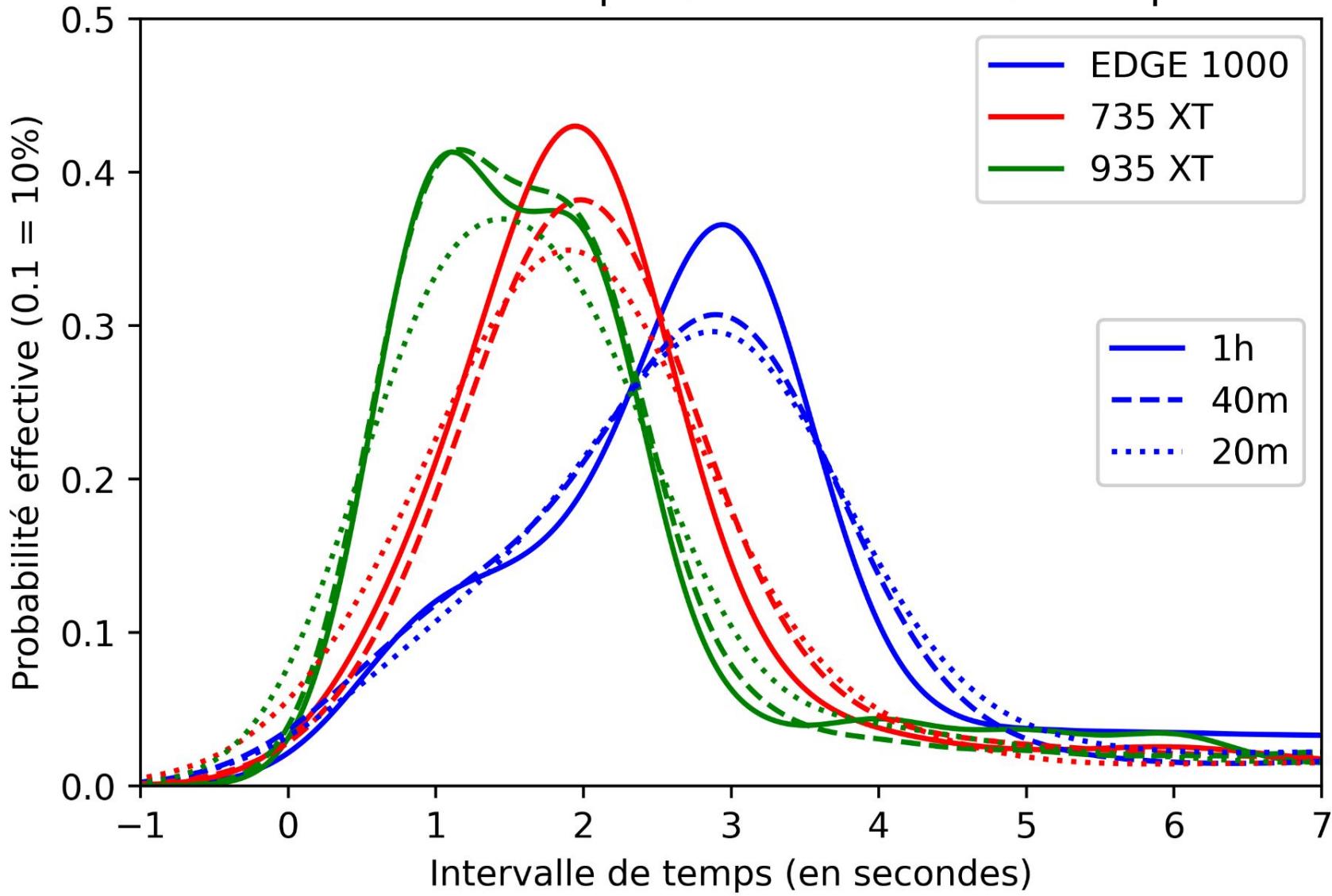
<b>Appareil de mesure</b>	<b>Distance (km) : 25 tours en 20 minutes</b>	<b>Distance (km) : 50 tours en 40 minutes</b>	<b>Distance (km) : 75 tours en 1 heure</b>
Garmin Edge 1040	6,33	12,67	19,03
Garmin 935 XT	6,38	12,83	19,17
Garmin 735 XT	6,36	12,78	19,21
Compteur Mécanique	6,40	12,80	19,30
Données Réelles*	6,3375	12,6750	19,0125

\* base de 253,5 mètres par tour, on peut calculer une vitesse moyenne de référence pour observer les écarts

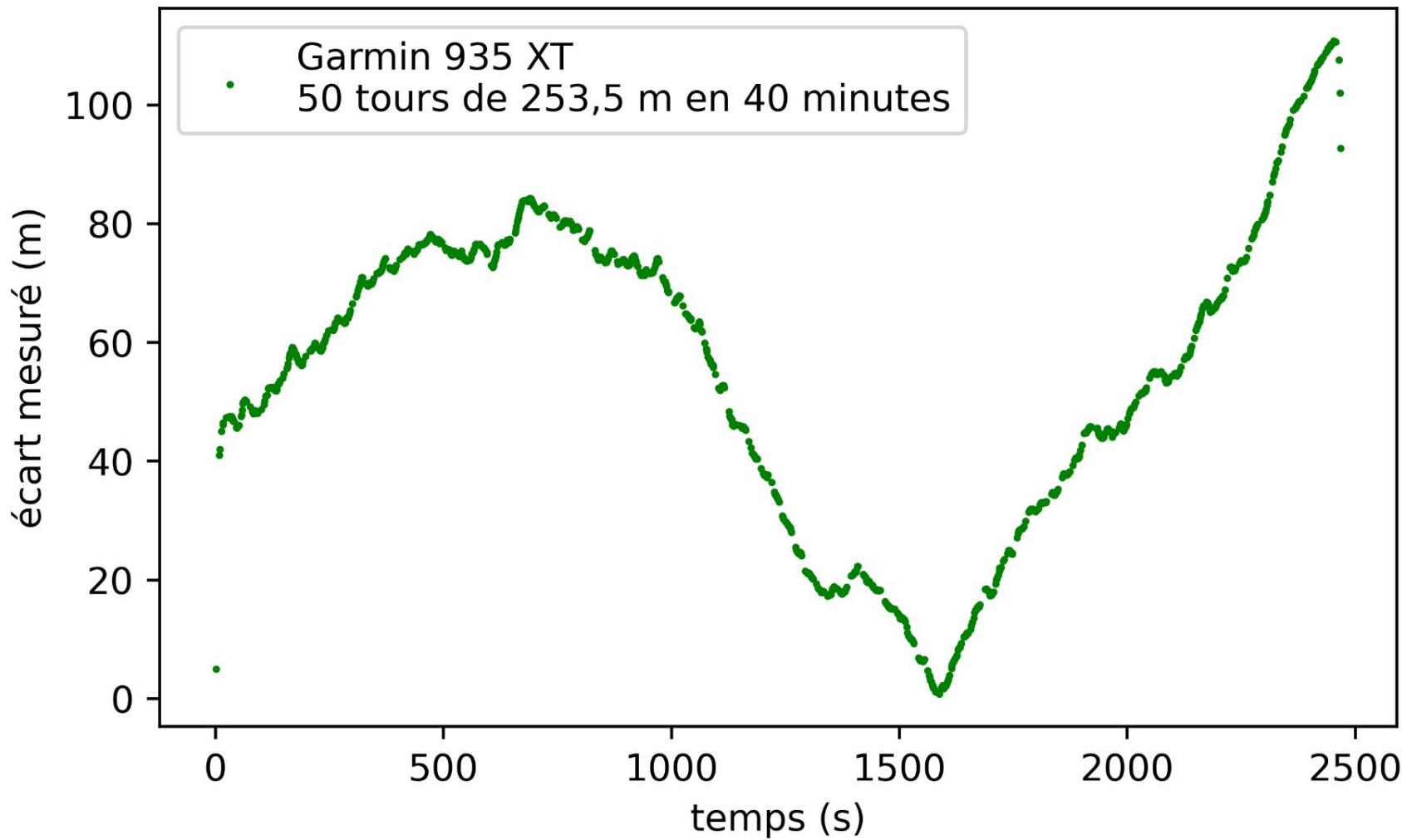
## Densité statistique des intervalles de temps



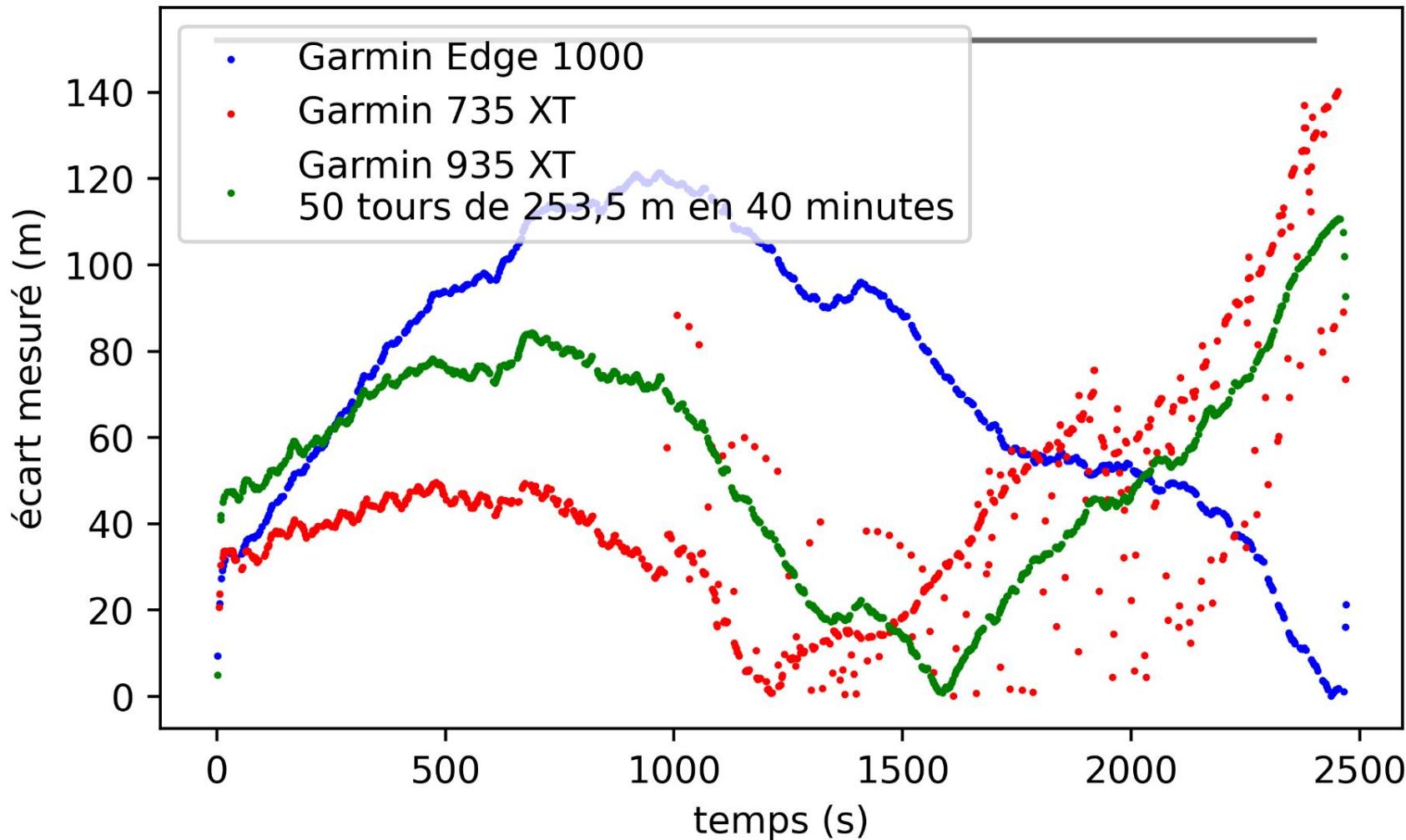
## Densité statistique des intervalles de temps



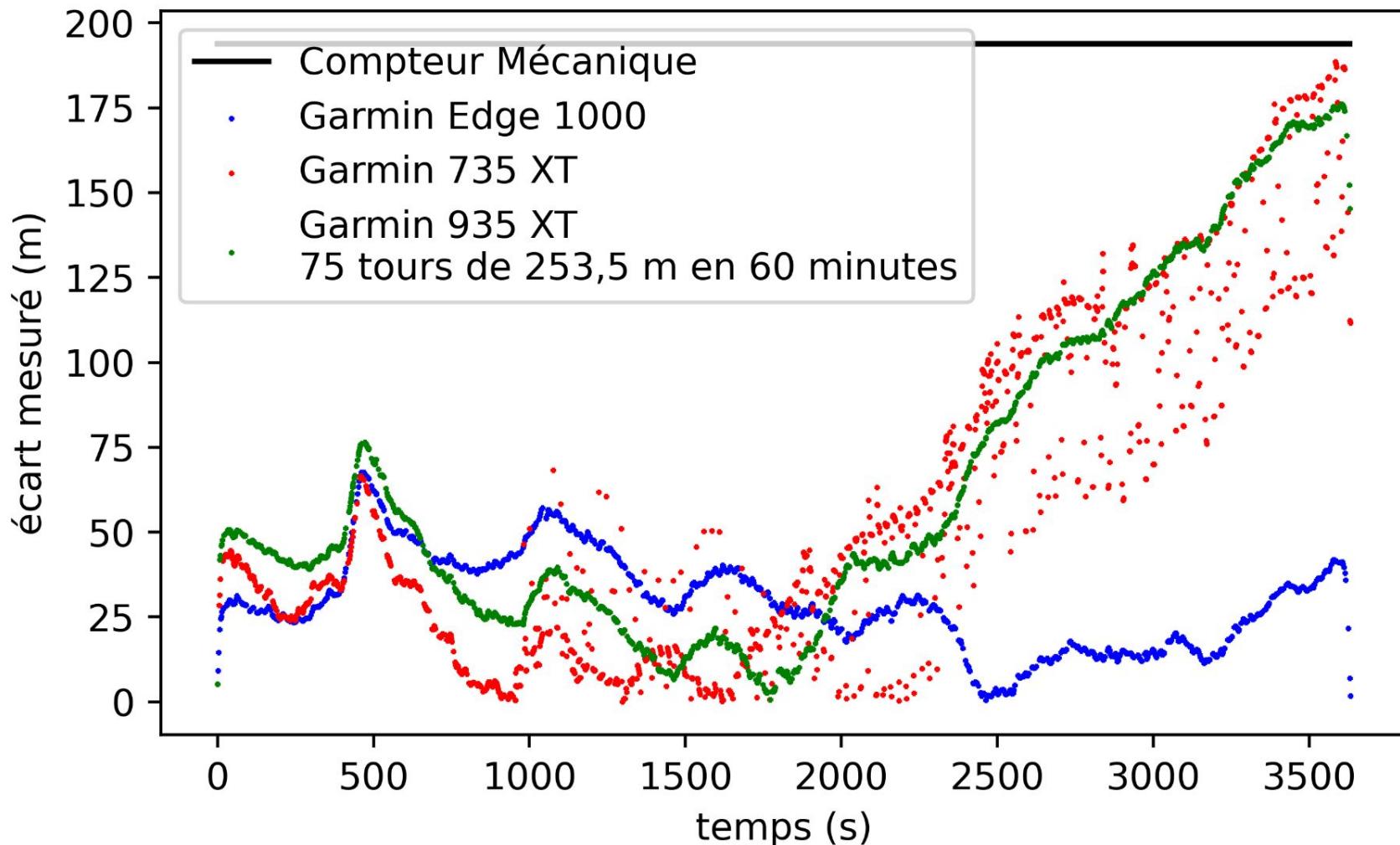
## Écart absolu entre la mesure GPS et la distance réelle parcourue



## Écart absolu entre la mesure GPS et la distance réelle parcourue

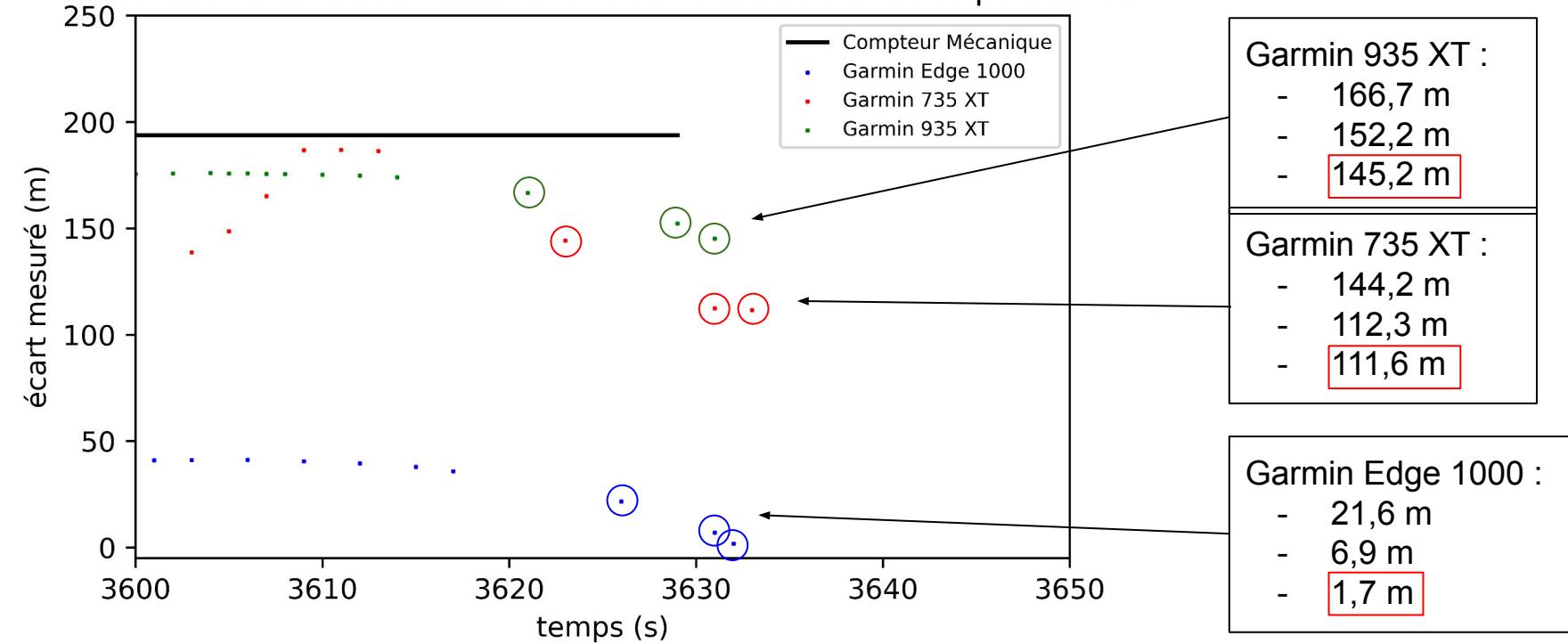


# Expérience de 60 min :



# Trois derniers écarts mesurés sur l'expérience de 60 min

Écart absolu entre la mesure GPS et la distance réelle parcourue



→ erreur de 0.8% pour la montre 935 XT

- Le système GPS reste précis à 99,2 %
- Une dérive en statique dû à différents facteurs
- Une première approche calculatoire trop limitée

# programme de tracé de graphique KDE

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
from sklearn.neighbors import KernelDensity

# générer l'échantillon à partir de deux lois normales :
N = 1000 # mettre le nombre de points voulu
h = 0.25 # valeur de la bandwidth

X = np.concatenate(
    (np.random.normal(0, 1, int(0.4 * N)), np.random.normal(5, 2, int(0.6 * N))))[:, np.newaxis]

# préparer les points où on calculera la densité
X_plot = np.linspace(-5, 10, 1000)[:, np.newaxis]

# préparation de l'affichage de la vraie densité, qui est celle à partir de laquelle les données
# ont été générées (voir plus haut)
# la pondération des lois dans la somme est la pondération des lois dans l'échantillon généré (voir plus haut)
true_dens = 0.4 * norm(0, 1).pdf(X_plot[:, 0]) + 0.6 * norm(5, 2).pdf(X_plot[:, 0])

# estimation de densité par noyaux gaussiens
kde = KernelDensity(kernel='gaussian', bandwidth=h).fit(X)

# calcul de la densité pour les données de X_plot :
density = np.exp(kde.score_samples(X_plot))

# affichage : vraie densité et estimation :
fig = plt.figure()
ax = fig.add_subplot(111) #format du graphique
ax.fill(X_plot[:,0], true_dens, fc='black', alpha=0.2, label='distribution en entrée')
ax.plot(X_plot[:,0], density, '-', label="Estimation")
ax.legend(loc='upper left')

#affichage du nombre de points de la mesure :
ax.text(6, 0.38, "N={0} points".format(N))
ax.text(6, 0.35, "h={0}".format(h))

#afficher la distribution des points en bas du graphique : afficher que pour N=10 ou N=100 sinon trop chargé
ax.plot(X[:, 0], -0.005 - 0.01 * np.random.random(X.shape[0]), "+k")

#limites du graphique :
ax.set_xlim(-4, 10)
ax.set_ylim(-0.02, 0.4)

#Ajout de la légende :
ax.set_xlabel("x")
ax.set_ylabel("Densité normalisée")

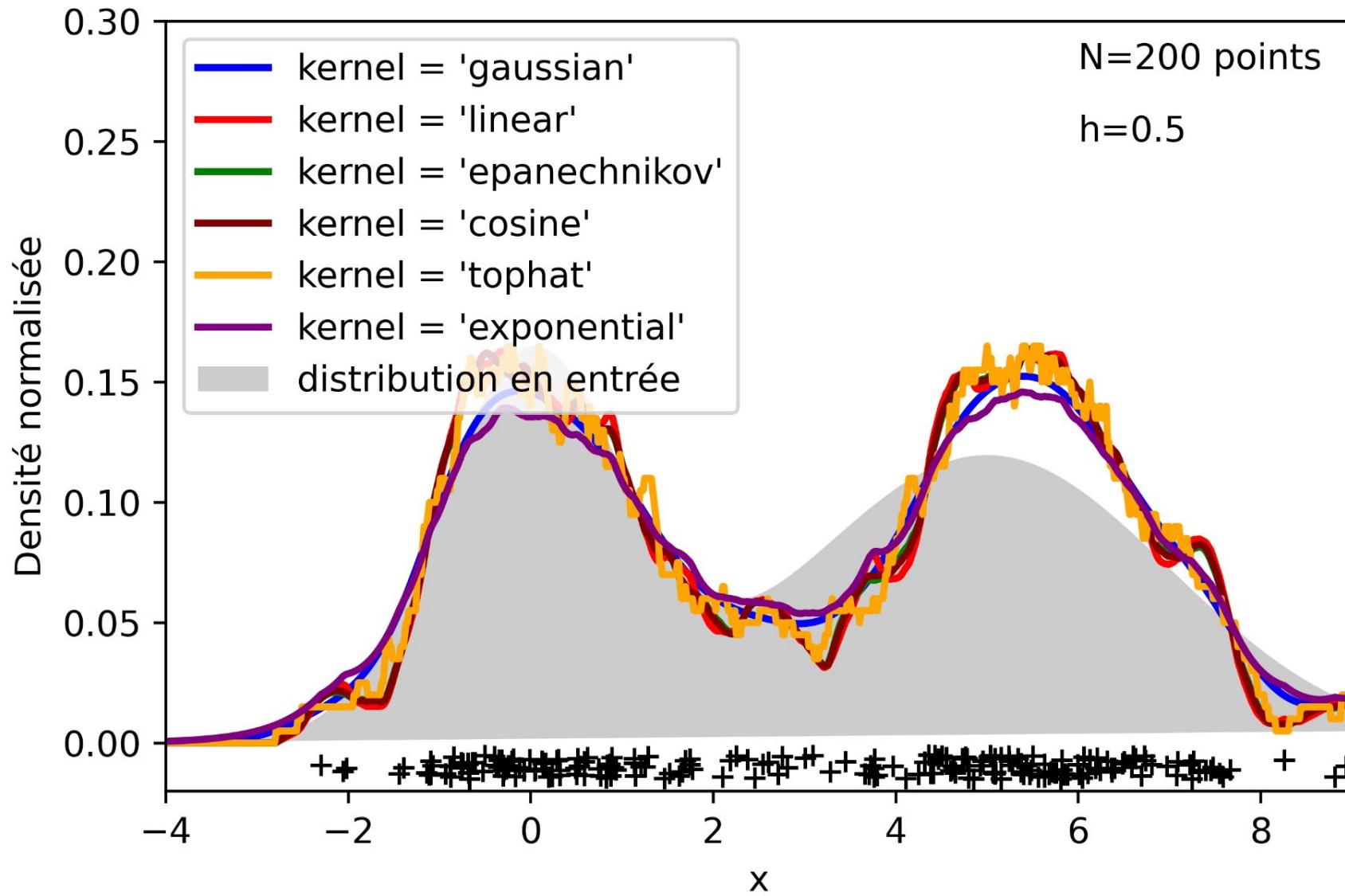
#sauvegarde du fichier :
plt.savefig('2 pics (écart 2 sur le pic 2), h=0.25, N=1000.png', dpi=500)

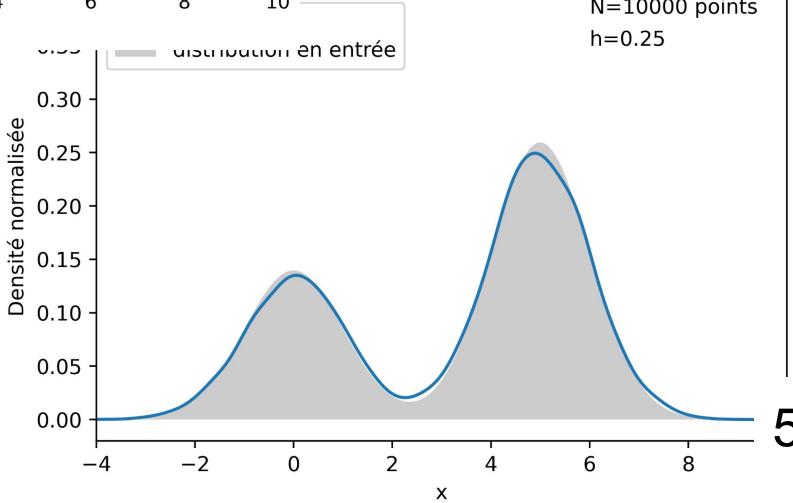
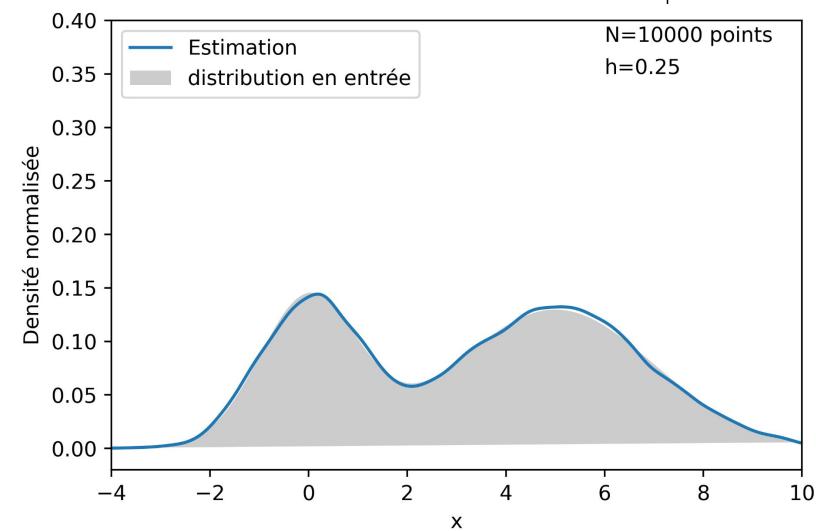
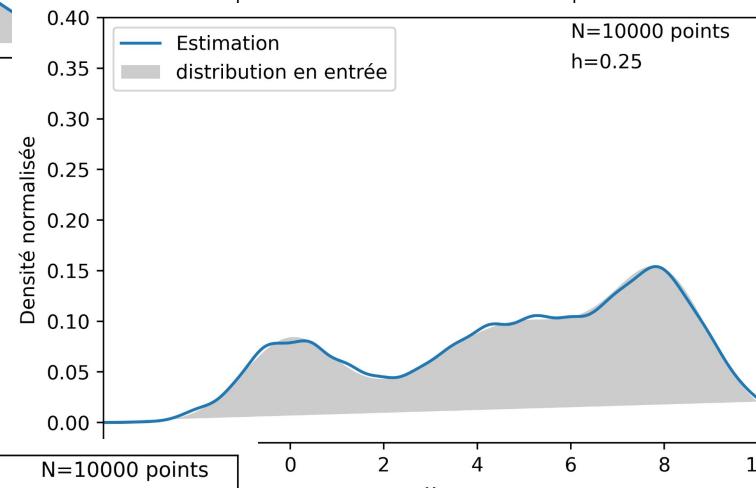
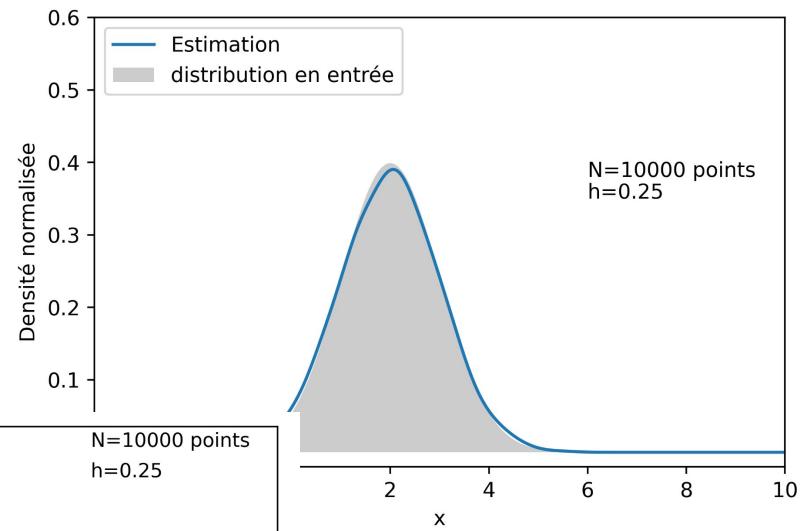
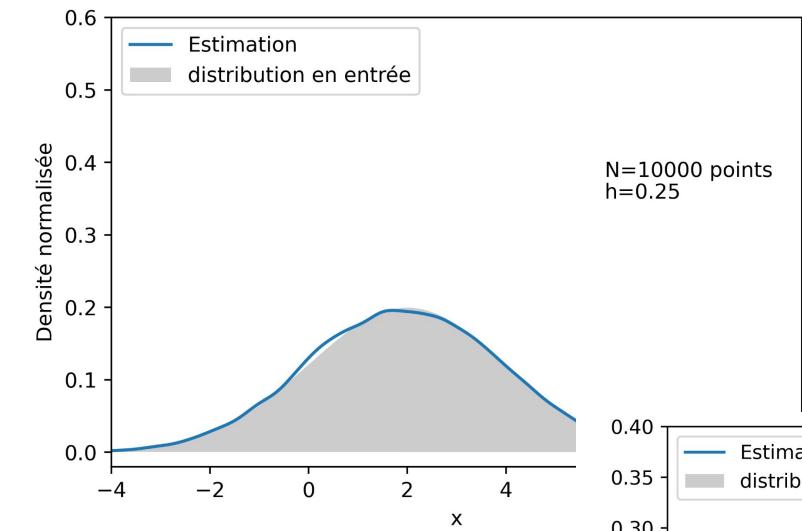
bandwidth = np.arange(0.1, 1, .05) #parcours de toutes les valeurs de h de 0.05 à 2 avec un pas de 0.05
kde = KernelDensity(kernel='gaussian')
grid = GridSearchCV(kde, {'bandwidth': bandwidth}) #fait toutes les combinaisons des paramètres
grid.fit(X)

fig = plt.figure()
ax = fig.add_subplot(111) #format du graphique
kde = grid.best_estimator_ #meilleur modèle
log_dens = kde.score_samples(X_plot)
ax.plot(X_plot[:, 0], np.exp(log_dens), color='blue', lw=2, linestyle='-', label="kernel = 'gaussian'")
ax.plot(X_plot[:, 0], np.exp(log_dens), color='red', lw=2, linestyle='--', label="kernel = 'linear'")
```

pour tracer différents noyaux en même temps

# annexe différents types de noyaux





# programme extraction des données

```
def LectureApp(string):  
  
    # Lecture du document  
    texte = open(string, "r")  
    t = texte.readlines()  
    a = ""  
  
    # On récupère le nombre de lignes  
    for i in range(len(t)):  
        a += t[i]  
  
    # Ensuite, avec toutes les fonctions suivantes, on va extraire les  
    # données qui nous intéressent.  
  
    #fonction qui cherche si le 'motif' est à la position 'i' dans le 'texte'  
    def est_ici(texte,motif,i):  
        a = 0  
        if len(motif)>len(texte):  
            return False  
  
        while a<len(motif):  
            if texte[i+a] == motif[a]:  
                a += 1  
            else:  
                return False  
        return True  
  
    #fonction qui cherche si le 'motif' est dans le 'texte'  
    def est_sous_mot(texte,motif):  
        indice = 0  
        while indice < len(texte):  
            if est_ici(texte,motif,indice) == False :  
                indice += 1  
            else:  
                return True  
        return False  
  
    #fonction qui cherche la(les) position(s) du motif 'i' dans le 'texte'  
    def position_sous_mot(texte,motif):  
        occurrence = []  
        for i in range (len(texte)):  
            if est_ici(texte, motif, i) ==True :  
                occurrence.append(i)  
        return(occurrence)  
  
    #fonction qui donne la liste contenant les différentes positions des motifs  
    # 'c' et 'b' dans le texte  
    def liste_pour_recherche_de_mot(c,b):  
        D = position_sous_mot(a,c) #position(s) du motif 'c' dans le texte 'a'  
        E = position_sous_mot(a,b) #position(s) du motif 'b' dans le texte 'a'  
        n = len(c)  
        for i in range(len(D)):  
            D[i] += n  
        return D,E  
  
    liste = liste_pour_recherche_de_mot("<DistanceMeters>","</DistanceMeters>")  
    liste2 = liste_pour_recherche_de_mot("<Time>","</Time>")
```

```
def entre_occurrence (l1,l2):  
    L = []  
    for i in range(len(l1)):  
        longueur = l2[i]-l1[i]  
        mot = ""  
        for j in range(longueur):  
            mot += a[l1[i]+j]  
        L.append(mot)  
    return L  
  
def intervalle_temps(liste):  
    L = []  
    for i in range(len(liste)-1):  
        donnees1 = liste[i].split('T')  
        donnees2 = liste[i+1].split('T')  
        donnees3 = donnees1[1].split(',')  
        donnees4 = donnees2[1].split(',')  
        donnees5 = donnees3[0].split(':')  
        donnees6 = donnees4[0].split(':')  
        if donnees1[0] == donnees2[0]:  
            L.append((int(donnees6[0])*3600+int(donnees6[1])*60+int(donnees6[2]))-(int(donnees5[0])*3600+  
        else:  
            L.append((int(donnees6[0])*3600+int(donnees6[1])*60+int(donnees6[2]))+3600*24)-(int(donnees5[0]))  
    return L  
  
def intervalle_distance(liste):  
    L = []  
    for i in range(1,len(liste)-1):  
        L.append(float(liste[i+1])-float(liste[i]))  
    return L  
  
intervalle_distance = intervalle_distance(entre_occurrence(liste[0],liste[1]))  
intervalle_temps = intervalle_temps(entre_occurrence(liste2[0],liste2[1]))  
  
p = []  
m = []  
somme = 0  
somme_bis = 0  
temoin = []  
  
#condition : ne les prendra pas en compte dans les expériences statiques :  
  
#expérience de 60min :  
if len(intervale_distance)==1037 or len(intervale_distance)==1269 or len(intervale_distance)==1576 :  
    vmoy = 5.2390465693  
    for i in range(len(intervale_temps)):  
        somme += intervalle_distance[i]  
        somme_bis += intervalle_temps[i]  
        p.append(somme)  
        m.append(somme_bis)  
        if abs(somme-(vmoy*somme_bis)) < 1000 :  
            temoin.append(abs(somme-(vmoy*somme_bis)))  
        if abs(somme-(vmoy*somme_bis)) > 1000 :  
            m.pop()  
            p.pop()  
#expérience de 40min :  
elif len(intervale_distance)==674 or len(intervale_distance)==775 or len(intervale_distance)==1121 :  
    vmoy = 5.1399026764  
    for i in range(len(intervale_temps)):  
        somme += intervalle_distance[i]  
        somme_bis += intervalle_temps[i]  
        p.append(somme)  
        m.append(somme_bis)  
        if abs(somme-(vmoy*somme_bis)) < 1000 :  
            temoin.append(abs(somme-(vmoy*somme_bis)))  
        if abs(somme-(vmoy*somme_bis)) > 1000 :  
            m.pop()  
            p.pop()  
#expérience de 20 min :  
elif len(intervale_distance)==336 or len(intervale_distance)==426 or len(intervale_distance)==518 :  
    vmoy = 5.25497512438  
    for i in range(len(intervale_temps)):  
        somme += intervalle_distance[i]  
        somme_bis += intervalle_temps[i]  
        p.append(somme)  
        m.append(somme_bis)  
        if abs(somme-(vmoy*somme_bis)) < 1000 :  
            temoin.append(abs(somme-(vmoy*somme_bis)))  
        if abs(somme-(vmoy*somme_bis)) > 1000 :  
            m.pop()  
            p.pop()
```

```

#expérience statique :
else:
    for i in range(len(intervale_distance)):
        somme += intervalle_distance[i]
        somme_bis += intervalle_temps[i]
        p.append(somme)
        m.append(somme_bis)

# On met dans une nouvelle liste les données suivantes sous forme de listes :
K=[]
# temps total
K.append(m)
# distance totale
K.append(p)
# intervalle temps
K.append(intervale_temps)
# intervalle distance
K.append(intervale_distance)
# moyenne de la distance parcourue par seconde
K.append(temoin)

return(K)

# Calcul de coefficient par régression linéaire
def coeff(L):
    tps = L[0]
    parc= L[1]
    coef, ordo= np.polyfit(tps, parc, 1)
    return(coef)

# Construction de deux listes pour exploiter la régression linéaire dans des graphiques
def plotage(L):
    coef = coeff(L)
    t=[]
    mg=[]
    for i in (L[0]):
        t.append(i)
        mg.append(i*coef)
    K=[]
    K.append(t)
    K.append(mg)
    return(K)

# Première fonction d'estimation de densité par noyau gaussien utilisée
def GAUSSIENNE(b):
    density = kde.gaussian_kde(b)
    x = np.linspace(-5,45, 400)
    y=density(x)
    return(x,y)

# Deuxième fonction d'estimation de densité par noyau gaussien utilisée
def GAUSSIENNE2(b):
    density = kde.gaussian_kde(b)
    x = np.linspace(-2,8, 400)
    y=density(x)
    return(x,y)

# Fonction annexe du calcul de KDE (car on a rencontré quelques problèmes avec le module
def kde_statsmodels_u(x, x_grid, bandwidth=3, **kwargs):
    kde = KDEUnivariate(x)
    kde.fit(bw=bandwidth, **kwargs)
    return kde.evaluate(x_grid)

# FONCTION DE LANCEMENT DU PROGRAMME

def Start():
    mesure_des_ecarts_statique()
    mesure_des_ecarts_dynamique()

```

```

def mesure_des_ecarts_statique():

    #Indication dans la console de quelle partie il s'agit
    print("MESURE DES ÉCARTS STATIQUES \n")

    #Récupération des listes de données des différents fichiers
    A = LectureApp("ACT22-FINAL.txt")
    B = LectureApp("ACT19-FINAL.txt")
    C = LectureApp("ACT10-FINAL.txt")
    D = LectureApp("ACT3-FINAL.txt")
    E = LectureApp("ACT1-FINAL.txt")

    # Informations utiles
    print(" ----- Informations Utiles ----- \n")
    print("Nombre de mesures (22h) : "+str(len(A[0])))
    print("Nombre de mesures (1h) : "+str(len(E[0])))
    cana = []+A[4]+B[4]+C[4]+D[4]+E[4]
    cana.sort()
    pana=[]+A[1]+B[1]+C[1]+D[1]+E[1]
    panatps = []+A[0]+B[0]+C[0]+D[0]+E[0]
    pana.sort()
    panatps.sort()
    (coe,ordo) = np.polyfit(panatps, pana, 1)
    print("Coefficient général : "+str(coe))
    print("Coefficient 1h : " +str(coeff(E)))
    print("Coefficient 3h : " +str(coeff(D)))
    print("Coefficient 10h : " +str(coeff(C)))
    print("Coefficient 19h : " +str(coeff(B)))
    print("Coefficient 22h : " +str(coeff(A)))
    moycoef = (coeff(A)+coeff(B)+coeff(C)+coeff(D)+coeff(E))/5
    print("Moyenne des coefficients : " + str(moycoef))

    #Affichage des courbes brutes en fonction du temps

    Z=LectureApp("ACT1-FINAL.txt")
    for i in range (len(Z[0])):
        Z[0][i] = Z[0][i]/3600
    for i in range (len(Z[1])):
        Z[1][i] = Z[1][i]/1000

    Y=LectureApp("ACT3-FINAL.txt")
    for i in range (len(Y[0])):
        Y[0][i] = Y[0][i]/3600
    for i in range (len(Y[1])):
        Y[1][i] = Y[1][i]/1000

    X=LectureApp("ACT10-FINAL.txt")
    for i in range (len(X[0])):
        X[0][i] = X[0][i]/3600
    for i in range (len(X[1])):
        X[1][i] = X[1][i]/1000

    53

```

```

# Estimation de densité par noyau (via fonction auxiliaire)
AL = GAUSSIENNE(A[3])
BL = GAUSSIENNE(B[3])
CL = GAUSSIENNE(C[3])
DL = GAUSSIENNE(D[3])
EL = GAUSSIENNE(E[3])

# Affichage
plt.clf()
plt.plot(AL[0], AL[1], color="blue", label="22 heures")
plt.plot(BL[0], BL[1], color="red", label="19 heures")
plt.plot(CL[0], CL[1], color="green", label="10 heures")
plt.plot(DL[0], DL[1], color="yellow", label="3 heures")
plt.plot(EL[0], EL[1], color="purple", label="1 heure")
plt.title("densité statistique des intervalles de distances mesurés")
plt.legend(loc=1)
plt.xlim(-5, 25)
plt.xlabel("Intervalle de distance (en mètres)")
plt.ylabel("Probabilité effective (0.4 = 40%)")
plt.savefig("ACT-GAUSSIENNE-DISTANCE.png", dpi=500)
plt.show()

# Affichage de l'estimation de densité par noyau (KDE) des intervalles de temps
# Estimation de densité par noyau (via fonction auxiliaire)
AK = GAUSSIENNE(A[2])
BK = GAUSSIENNE(B[2])
CK = GAUSSIENNE(C[2])
DK = GAUSSIENNE(D[2])
EK = GAUSSIENNE(E[2])

# Affichage
plt.clf()
plt.plot(AK[0], AK[1], color="blue", label="22 heures")
plt.plot(BK[0], BK[1], color="red", label="19 heures")
plt.plot(CK[0], CK[1], color="green", label="10 heures")
plt.plot(DK[0], DK[1], color="yellow", label="3 heures")
plt.plot(EK[0], EK[1], color="purple", label="1 heure")
plt.title("densité statistique des intervalles de temps mesurés")
plt.legend(loc=1)
plt.xlabel("Intervalle de temps (en secondes)")
plt.ylabel("Probabilité effective (0.1 = 10%)")
plt.savefig("ACT-GAUSSIENNE-TEMPS.png", dpi=500)
plt.show()

```

utilisé à des modifications près  
pour les autres types  
d'expériences en statique

```

# Graphique représentant les régressions linéaires et leur courbe associée
plt.clf()
# regression linéaire
number=coeff(V)
ah=int(str(number*1000)[:3]) #=459 m/h
plt.plot(plotage(V)[0], plotage(V)[1], color='blue', label="y =ax, a="+str(ah)+"m/h")
#courbes réelles
plt.scatter(V[0],V[1], marker='+', s=1, c='blue', alpha=0.05, label="22 heures")
plt.xlabel("temps (en heures)")
plt.ylabel("dérive (en km)")
plt.legend(loc=0)
plt.title("régression linéaire et courbe réelle")
plt.savefig("reg linéaire avec courbe réelle.png", dpi=500)
plt.show()

plt.clf()
# Régressions linéaires :
bh=(int(str(coeff(X)*1000)[:3])) #= 315 m/h
ch=(int(str(coeff(Y)*1000)[:3])) #=363 m/h
dh=(int(str(coeff(Z)*1000)[:3])) #=352 m/h
eh=(int(str(coeff(W)*1000)[:3])) #=453 m/h

plt.plot(plotage(Z)[0], plotage(Z)[1], color='purple', label="1 heure, a="+str(ceil(eh)))
plt.plot(plotage(Y)[0], plotage(Y)[1], color='yellow', label="3 heures, a="+str(ceil(dh)))
plt.plot(plotage(X)[0], plotage(X)[1], color='green', label="10 heures, a="+str(ceil(ch)))
plt.plot(plotage(W)[0], plotage(W)[1], color='red', label="19 heures, a="+str(ceil(bh)))
plt.plot(plotage(V)[0], plotage(V)[1], color='blue', label="22 heures, a="+str(ceil(ah)))

# Courbes réelles :
plt.scatter(V[0],V[1], marker='+', s=1, c='blue', alpha=0.05)
plt.scatter(W[0],W[1], marker='+', s=1, c='red', alpha=0.05)
plt.scatter(X[0],X[1], marker='+', s=1, c='green', alpha=0.05)
plt.scatter(Y[0],Y[1], marker='+', s=1, c='yellow', alpha=0.05)
plt.scatter(Z[0],Z[1], marker='+', s=1, c='purple', alpha=0.05)

plt.xlabel("Temps (en heures)")
plt.ylabel("Distance (en km)")
plt.title("régressions linéaires et courbes réelles")
plt.legend(loc=2)
plt.savefig("regs linéaires avec courbes réelles.png", dpi=500)
plt.show()

#Affichage des régressions linéaires avec la moyenne
plt.clf()
plt.plot(plotage(Z)[0], plotage(Z)[1], color='purple', label="1 heure, a="+str(ceil(eh)))
plt.plot(plotage(Y)[0], plotage(Y)[1], color='yellow', label="3 heures, a="+str(ceil(dh)))
plt.plot(plotage(X)[0], plotage(X)[1], color='green', label="10 heures, a="+str(ceil(ch)))
plt.plot(plotage(W)[0], plotage(W)[1], color='red', label="19 heures, a="+str(ceil(bh)))
plt.plot(plotage(V)[0], plotage(V)[1], color='blue', label="22 heures, a="+str(ceil(ah)))

# Calcul des valeurs moyennes
mh=moycoef*3600/1000 #en km/h
time=[]
long=[]
for i in range(1000):
    time.append(i*22/1000)
    long.append(i*22*mh/1000)
mhh=str(moycoef*3600)[:3]
plt.plot(time, long, color='black', label="moyenne, a="+str(mhh))

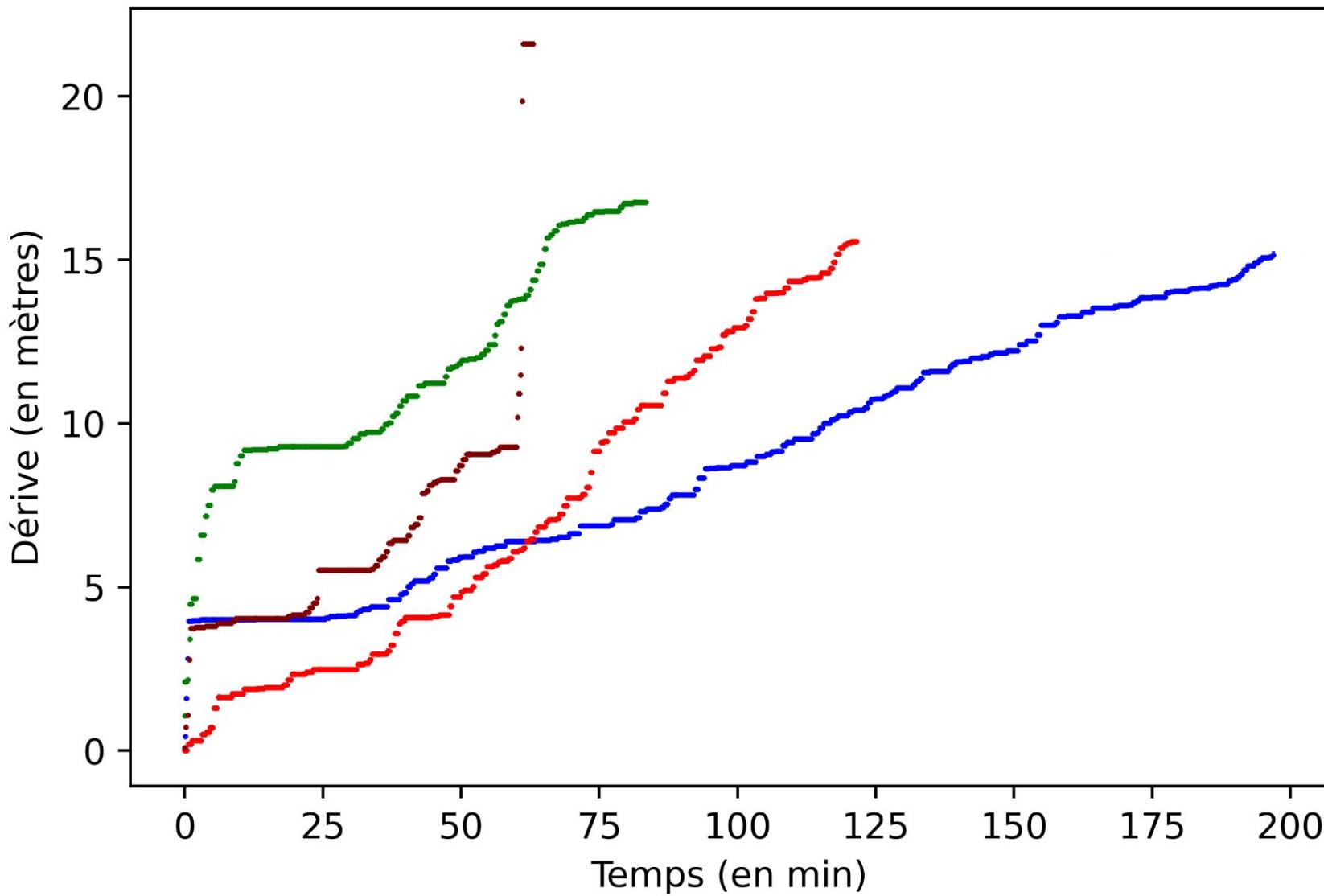
# Mise en forme
plt.xlabel("Temps (en heures)")
plt.ylabel("Dérive (en km)")
plt.title("Régessions linéaires et la moyenne")
plt.legend(loc=2)
plt.savefig("ACT-Courbes régressions linéaires + moyenne.png", dpi=500)
plt.show()

return(None)

```

# annexe capteur vélo

Dérives du capteur GPS-vélo en statique



```

def mesure_des_ecarts_dynamique():

#Indication dans la console de quelle partie il s'agit
print("\n \n MESURE DES ÉCARTS DYNAMIQUES \n")

#Récupération des listes de données des différents fichiers.
A = LectureApp("ACT-BELDYN-1h-COMPTEURGARMIN.txt")
B = LectureApp("ACT-BELDYN-1h-GARMIN735.txt")
C = LectureApp("ACT-BELDYN-1h-GARMIN935.txt")
D = LectureApp("ACT-BELDYN-40m-COMPTEURGARMIN.txt")
E = LectureApp("ACT-BELDYN-40m-GARMIN735.txt")
F = LectureApp("ACT-BELDYN-40m-GARMIN935.txt")
G = LectureApp("ACT-BELDYN-20m-COMPTEURGARMIN.txt")
H = LectureApp("ACT-BELDYN-20m-GARMIN735.txt")
I = LectureApp("ACT-BELDYN-20m-GARMIN935.txt")

# Informations utiles
print(" ----- Informations Utiles (Expérience Dynamique) ----- \n")
print("Nombre de mesures (Compteur Garmin 1h) : "+str(len(A[0])))
print("Nombre de mesures (Garmin 735 1h) : "+str(len(B[0])))
print("Nombre de mesures (Garmin 935 1h) : "+str(len(C[0])))+"\n"

# Calcul du coefficient directeur général (i.e. vitesse globale)

distance_totale=[]+A[1]+B[1]+C[1]+D[1]+E[1]+F[1]+G[1]+H[1]+I[1]
temps_total = []+A[0]+B[0]+C[0]+D[0]+E[0]+F[0]+G[0]+H[0]+I[0]
distance_totale.sort()
temps_total.sort()
(vitesse,ordo) = np.polyfit(temps_total, distance_totale, 1)
print("Coefficient général (vitesse en mètres/seconde): "+str(vitesse))

# Représentation de la répartition des intervalles de distance et de temps

# Création de l'étalement des abscisses
x_grid = np.linspace(-2, 30, 1000)

# Reset du cadre graphique
plt.clf()

# Représentation des intervalles de distance

# PLOT de la répartition de densité distance sous forme gaussiennes
AL = GAUSSIENNE(A[3])
#BL = GAUSSIENNE(B[3])
CL = GAUSSIENNE(C[3])
DL = GAUSSIENNE(D[3])
#EL = GAUSSIENNE(E[3])
FL = GAUSSIENNE(F[3])
GL = GAUSSIENNE(G[3])
#HL = GAUSSIENNE(H[3])
#IL = GAUSSIENNE(I[3])

```

```

# Pour certaines listes, on a utilisé un module annexe car nous avions des problèmes
# que nous n'avons pas réussi à identifier
BL = [x_grid, kde_statsmodels_u(B[3], x_grid, bandwidth=1)]
EL = [x_grid, kde_statsmodels_u(E[3], x_grid, bandwidth=1)]
HL = [x_grid, kde_statsmodels_u(H[3], x_grid, bandwidth=1)]
IL = [x_grid, kde_statsmodels_u(I[3], x_grid, bandwidth=1)]

# On ajoute chaque répartition gaussienne
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(CL[0], CL[1], color="green", label="935 XT")
ax.plot(FL[0], FL[1], "g-")
ax.plot(IL[0], IL[1], "g:")
maxline, = ax.plot(0, 0, '--', color='green')
minline, = ax.plot(0, 0, '---', color='green')
basicline, = ax.plot(0, 0, ':', color='green')
# On ajoute la première légende
leg1 = ax.legend(loc='upper right', fontsize=10)
# On met ensuite la deuxième légende
leg2 = ax.legend([maxline,minline,basicline],[ '1h', '40m', '20m'], loc='right', fontsize=10)
# Et on fait revenir la première légende (astuce pour légender deux fois)
ax.add_artist(leg1)
ax.add_artist(leg2)
plt.ylim(0, 0.3)
plt.xlim(-1, 28)
plt.title("Densité statistique des intervalles de distance")
plt.xlabel("Intervalle de distance (en mètres)")
plt.ylabel("Probabilité effective (0.4 = 40%)")
plt.savefig("ACT-GAUSSIENNE-DYNAMIQUE-DISTANCE-935-XT.png", dpi=500)
plt.show()
plt.clf()

fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(AL[0], AL[1], color="blue", label="EDGE 1000")
ax.plot(BL[0], BL[1], color="red", label="735 XT")
ax.plot(CL[0], CL[1], color="green", label="935 XT")
ax.plot(DL[0], DL[1], "b-")
ax.plot(EL[0], EL[1], 'r--')
ax.plot(FL[0], FL[1], 'g--')
ax.plot(GL[0], GL[1], 'b:')
ax.plot(HL[0], HL[1], 'r:')
ax.plot(IL[0], IL[1], 'g:')
# Réglages des paramètres
maxline, = ax.plot(0, 0, '--', color='blue')
minline, = ax.plot(0, 0, '---', color='blue')
basicline, = ax.plot(0, 0, ':', color='blue')
# On ajoute la première légende
leg1 = ax.legend(loc='upper right', fontsize=10)
# On met ensuite la deuxième légende
leg2 = ax.legend([maxline,minline,basicline],[ '1h', '40m', '20m'], loc='right', fontsize=10)
# Et on fait revenir la première légende (astuce pour légender deux fois)
ax.add_artist(leg1)
ax.add_artist(leg2)
plt.ylim(0, 0.3)
plt.xlim(-1, 28)

```

```

plt.title("Densité statistique des intervalles de distance")
plt.xlabel("Intervalle de distance (en mètres)")
plt.ylabel("Probabilité effective (0.4 = 40%)")
plt.savefig("ACT-GAUSIENNE-DYNAMIQUE-DISTANCE.png", dpi=500)
plt.show()

# Répartition des intervalles de temps en fonction des différentes activités et des capteurs
AK = GAUSSIENNE2(A[2])
BK = GAUSSIENNE2(B[2])
CK = GAUSSIENNE2(C[2])
DK = GAUSSIENNE2(D[2])
EK = GAUSSIENNE2(E[2])
FK = GAUSSIENNE2(F[2])
GK = GAUSSIENNE2(G[2])
HK = GAUSSIENNE2(H[2])
IK = GAUSSIENNE2(I[2])

plt.clf()
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(CK[0], CK[1], color='green', label="935 XT")
ax.plot(FK[0], FK[1], 'g--')
ax.plot(IK[0], IK[1], 'g:')
maxline, = ax.plot(0, 0, '--', color='green')
minline, = ax.plot(0, 0, '---', color='green')
basicline, = ax.plot(0, 0, ':', color='green')
# On ajoute la première légende
leg1 = ax.legend(loc='upper right', fontsize=10)
# On met ensuite la deuxième légende
leg2 = ax.legend([maxline,minline, basicline],['1h', '40m', '20m'], loc='right', fontsize=10)
# Et on fait revenir la première légende (astuce pour légendier deux fois)
ax.add_artist(leg1)
ax.add_artist(leg2)
plt.title("Densité statistique des intervalles de temps")
plt.xlim(-1, 7)
plt.ylim(0, 0.5)
plt.xlabel("Intervalle de temps (en secondes)")
plt.ylabel("Probabilité effective (0.1 = 10%)")
plt.savefig("ACT-GAUSIENNE-DYNAMIQUE-TEMPS-935-XT.png", dpi=500)
plt.show()

plt.clf()
fig = plt.figure()
ax = fig.add_subplot(111)
# On construit chaque courbe
ax.plot(AK[0], AK[1], color='blue', label="EDGE 1000")
ax.plot(BK[0], BK[1], color='red', label="735 XT")
ax.plot(CK[0], CK[1], color='green', label="935 XT")
ax.plot(DK[0], DK[1], 'b--')
ax.plot(EK[0], EK[1], 'r--')
ax.plot(FK[0], FK[1], 'g--')
ax.plot(GK[0], GK[1], 'b:')
ax.plot(HK[0], HK[1], 'r:')
ax.plot(IK[0], IK[1], 'g:')
# On fixe les paramètres
maxline, = ax.plot(0, 0, '--', color='blue')
minline, = ax.plot(0, 0, '---', color='blue')
basicline, = ax.plot(0, 0, ':', color='blue')
# On refait l'astuce de la double légende (voir plus haut)
leg1 = ax.legend(loc='upper right', fontsize=10)
leg2 = ax.legend([maxline,minline, basicline],['1h', '40m', '20m'], loc='right', fontsize=10)
ax.add_artist(leg1)
ax.add_artist(leg2)
# On met en forme le dernier graphique de répartition
plt.title("Densité statistique des intervalles de temps")
plt.xlim(-1, 7)
plt.ylim(0, 0.5)
plt.xlabel("Intervalle de temps (en secondes)")
plt.ylabel("Probabilité effective (0.1 = 10%)")
plt.savefig("ACT-GAUSIENNE-DYNAMIQUE-TEMPS.png", dpi=500)
plt.show()

# Ensuite, on vient représenter les écarts absolus des activités enregistrées par les différents capteurs
# Activité dynamique (40 minutes) et 50 tours
plt.clf()
plt.scatter(F[0],F[4], color='green', label="Garmin 935 XT \n50 tours de 253,5 m en 40 minutes", s=1)
plt.title("Écart absolu entre la mesure GPS et la distance réelle parcourue")
plt.subplots_adjust(top=0.80)

# Données du compteur mécanique (calcul fait à la fin de l'expérience le jour-même)
VitesseReelle = [5.25497512438, 5.1399026764, 5.2390465693]
VitesseMecaniqueCalculee = [5.30679934, 5.19059205, 5.30365485]

plt.legend(loc=2, fontsize=10)
plt.xlabel("temps (s)")
plt.ylabel("écart mesuré (m)")
plt.savefig("TIPE-Ecart absolu expérience dynamique (format 40min 50 tours)-735-XT.png", dpi=500)
plt.show()

# Activité dynamique (1 heure) et 75 tours
plt.clf()
plt.scatter(A[0],A[4], color='blue', label="Garmin Edge 1000", s=1, marker='+')
plt.scatter(B[0],B[4], color='red', label="Garmin 735 XT", s=1, marker='+')
plt.scatter(C[0],C[4], color='green', label="Garmin 935 XT \n75 tours de 253,5 m en 60 minutes", s=1, marker='+')
# Droite représentant l'écart absolu entre la réalité et
# les données établis par le capteur mécanique sur le format 1h
x = np.linspace(0, 3629, 4000)
y = []
b = abs((VitesseReelle[2]*3000)-(VitesseMecaniqueCalculee[2]*3000))
for i in x:
    y.append(b)
plt.plot(x,y,color="black", label="Compteur Mécanique")
plt.subplots_adjust(top=0.80)
plt.legend(loc=2, fontsize=10)
plt.xlabel("temps (s)")
plt.ylabel("écart mesuré (m)")
plt.savefig("TIPE-Ecart absolu expérience dynamique (format 1h 75 tours).png", dpi=500)
plt.show()

# Activité dynamique (40 minutes) et 50 tours
plt.clf()
plt.scatter(D[0],D[4], color='blue', label="Garmin Edge 1000", s=1)
plt.scatter(E[0],E[4], color='red', label="Garmin 735 XT", s=1)
plt.scatter(F[0],F[4], color='green', label="Garmin 935 XT \n50 tours de 253,5 m en 40 minutes", s=1)
# Droite représentant l'écart absolu entre la réalité et
# les données établis par le capteur mécanique sur le format 40 min
x1 = np.linspace(0, 2400, 3000)
y1=[]
c = abs((VitesseReelle[1]*3000)-(VitesseMecaniqueCalculee[1]*3000))
for i in x1 :
    y1.append(c)
plt.plot(x1, y1, color="black", alpha=0.6)
plt.title("Écart absolu entre la mesure GPS et la distance réelle parcourue")
plt.subplots_adjust(top=0.80)
plt.legend(loc=2, fontsize=10)
plt.xlabel("temps (s)")
plt.ylabel("écart mesuré (m)")
plt.savefig("TIPE-Ecart absolu expérience dynamique (format 40min 50 tours).png", dpi=500)
plt.show()

# Activité dynamique (20 minutes) et 25 tours
plt.clf()
plt.scatter(G[0],G[4], color='blue', label="Garmin Edge 1000", s=1)
plt.scatter(H[0],H[4], color='red', label="Garmin 735 XT", s=1)
plt.scatter(I[0],I[4], color='green', label="Garmin 935 XT \n25 tours de 253,5 m en 20 minutes", s=1)
# Droite représentant l'écart absolu entre la réalité et
# les données établis par le capteur mécanique sur le format 20 min
x2 = np.linspace(0, 1200, 2000)
y2=[]
d = abs(130)
for i in x2 :
    y2.append(d)
plt.plot(x2, y2, color="black", alpha=0.4)
plt.subplots_adjust(top=0.80)
plt.legend(loc=2, fontsize=10)
plt.xlabel("temps (s)")
plt.ylabel("écart mesuré (m)")
plt.savefig("TIPE-Ecart absolu expérience dynamique (format 20min 25 tours).png", dpi=500)
plt.show()


```

```

# Superposition des activités dynamiques en ajoutant la comparaison aux capteurs mé

# Données du compteur mécanique (calcul fait à la fin de l'expérience le jour-même)
VitesseReelle = [5.25497512438, 5.1399026764, 5.2390465693]
VitesseMecaniqueCalculee = [5.30679934, 5.19059205, 5.30365485]

# Reset du cadre graphique
plt.clf()

# Droite représentant l'écart absolu entre la réalité et
# les données établis par le capteur mécanique sur le format 1h
x = np.linspace(1450, 3629, 4000)
y = []
b = abs((VitesseReelle[2]*3000)-(VitesseMecaniqueCalculee[2]*3000))
for i in x:
    y.append(b)
plt.plot(x,y,color="black", label="Compteur Mécanique")

# Droite représentant l'écart absolu entre la réalité et
# les données établis par le capteur mécanique sur le format 40 min
x1 = np.linspace(0, 2400, 3000)
y1=[]
c = abs((VitesseReelle[1]*3000)-(VitesseMecaniqueCalculee[1]*3000))
for i in x1 :
    y1.append(c)
plt.plot(x1, y1, color="black", alpha=0.6)

# Droite représentant l'écart absolu entre la réalité et
# les données établis par le capteur mécanique sur le format 20 min
x2 = np.linspace(0, 1200, 2000)
y2=[]
d = abs(130)
for i in x2 :
    y2.append(d)
plt.plot(x2, y2, color="black", alpha=0.4)

# Représentation des écarts globaux sur toutes les activités
plt.scatter(A[0],A[4], color='blue', label="Garmin Edge 1000", s=1, marker="+")
plt.scatter(B[0],B[4], color='red', label="Garmin 735 XT", s=1, marker="+")
plt.scatter(C[0],C[4], color='green', label="Garmin 935 XT", s=1, marker="+")
plt.scatter(D[0],D[4], color='blue', s=1, marker="+", alpha=0.3)
plt.scatter(E[0],E[4], color='red', s=1, marker="+", alpha=0.3)
plt.scatter(F[0],F[4], color='green', s=1, marker="+", alpha=0.3)
plt.scatter(G[0],G[4], color='blue', s=1, marker="+", alpha=0.15)
plt.scatter(H[0],H[4], color='red', s=1, marker="+", alpha=0.15)
plt.scatter(I[0],I[4], color='green', s=1, marker="+", alpha=0.15)
plt.title("Écart absolu entre la mesure GPS et la distance réelle parcourue")
plt.subplots_adjust(top=0.88)
plt.legend(loc="upper left", fontsize=9)
plt.ylim(-5, 250)
plt.xlabel("temps (s)")
plt.ylabel("écart mesuré (m)")
plt.savefig("Ecart mesuré total.png", dpi=800)
plt.show()
plt.clf()

#faire ressortir les derniers points :
plt.plot(x,y,color="black", label="Compteur Mécanique")
plt.plot(x1, y1, color="black", alpha=0.6)
plt.plot(x2, y2, color="black", alpha=0.4)
# Représentation des écarts globaux sur toutes les activités
plt.scatter(A[0],A[4], color='blue', label="Garmin Edge 1000", s=1, marker="+")
plt.scatter(B[0],B[4], color='red', label="Garmin 735 XT", s=1, marker="+")
plt.scatter(C[0],C[4], color='green', label="Garmin 935 XT", s=1, marker="+")
plt.scatter(D[0],D[4], color='blue', s=1, marker="+", alpha=0.3)
plt.scatter(E[0],E[4], color='red', s=1, marker="+", alpha=0.3)
plt.scatter(F[0],F[4], color='green', s=1, marker="+", alpha=0.3)
plt.scatter(G[0],G[4], color='blue', s=1, marker="+", alpha=0.15)
plt.scatter(H[0],H[4], color='red', s=1, marker="+", alpha=0.15)
plt.scatter(I[0],I[4], color='green', s=1, marker="+", alpha=0.15)
plt.title("Écart absolu entre la mesure GPS et la distance réelle parcourue")
plt.subplots_adjust(top=0.88)
plt.legend(loc="upper right", fontsize=7)
plt.ylim(-5, 250)
plt.xlim(3600,3650)
plt.xlabel("temps (s)")
plt.ylabel("écart mesuré (m)")
plt.savefig("Ecart mesuré total, derniers points.png", dpi=800)
plt.show()
plt.clf()

```

```

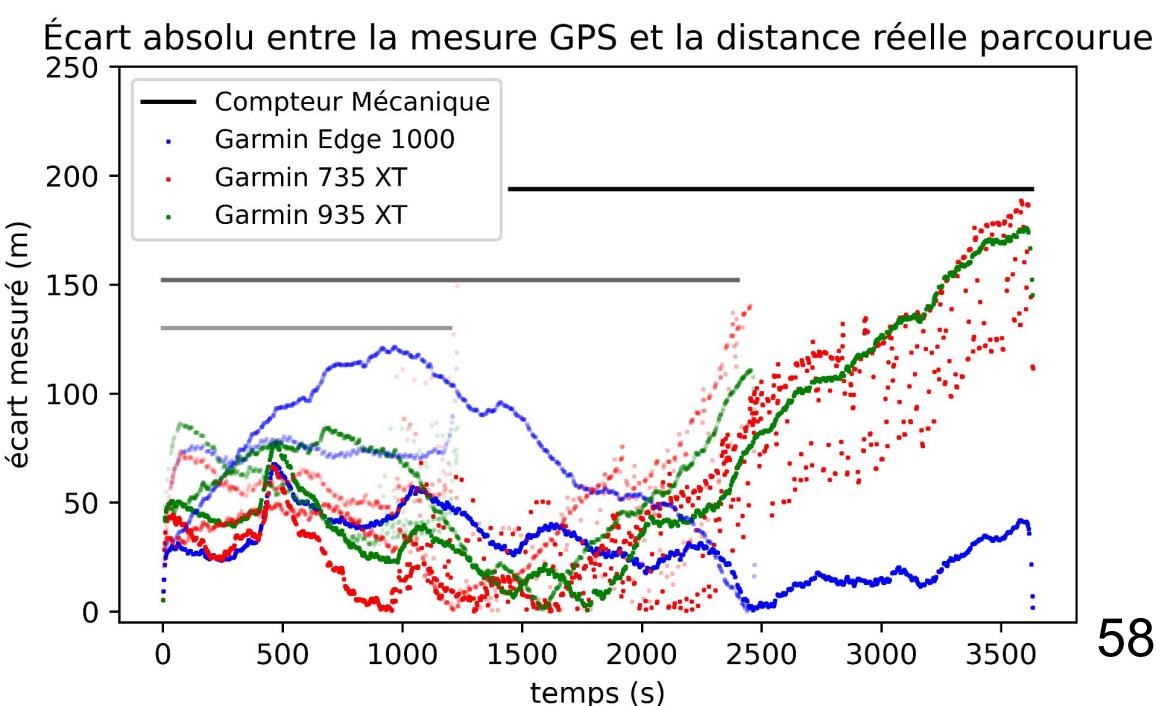
# On va ressortir les trois derniers écarts absolu mesurés entre la réalité et chaque capteur GPS
T = [A,B,C,D,E,F,G,H,I]
print("\n\n***** Trois derniers écarts mesurés en mètres *****")
# afficher les données qui nous intéressent, en les regroupant
for i in T :
    if i==A or i==B or i==C :
        duree_activite = "1 heure"
    elif i==D or i==E or i==F :
        duree_activite = "40 minutes"
    elif i==G or i==H or i==I :
        duree_activite = "20 minutes"
    if i==A or i==D or i==G :
        capteur="Garmin Edge 1000"
    elif i==B or i==E or i==H :
        capteur="Garmin 735 XT"
    elif i==C or i==F or i==I :
        capteur="Garmin 935 XT"

    if i==A or i==D or i==G :
        print("\n\n----- Activité format : "+duree_activite+" -----")
    print("\n "+capteur+ " : \n")
    for j in range(-3, 0, 1):
        print(" "+str(i[4][j])))

# On a obtenu nos quatre graphiques et des données extraites intéressantes, on retourne dans le reste
return(None)

```

## annexe résultats dynamique



# Et les incertitudes dans tout ça ?

⇒ Notre TIPE s'intéresse aux incertitudes de mesure et de calcul des données par les capteurs mais pour nos calculs et nos graphiques, nous avons mesuré et sélectionné des données issues des capteurs

## Pour les capteurs GARMIN (Edge 1000, 735 XT, 935 XT) (statique et dynamique ) :

Ordre de précision de  $10^{-15}$  m pour la distance et de  $10^{-3}$  s pour la durée et après les calculs (voir annexe), on obtient une variation des données maximales globales (incertitude relative maximale) de l'ordre de :

**0,000 000 001 % à 0,02 % donc négligeable.**

## Pour la mesure de la longueur de la piste et du tour de référence :

En considérant une précision du mm sur le décamètre, le tour mesuré en 4 mesures et une variation de notre trajectoire d'un rayon oscillant entre  $\pm 5$  cm d'un rayon moyen de référence, sur le nombre de tours maximum et la variation maximale possible, l'incertitude relative est de l'ordre de **0,03 % donc négligeable**.

## Pour le capteur Mécanique :

Deux mesures relevées, une à la fin, une au début au mètre près donc de l'ordre de 4 m d'incertitude sur les kilomètres parcourus, au final l'incertitude relative est de l'ordre de **0,02 % donc négligeable**.

## Expérience Dynamique :

### I. Données "Réelles"

Nous avons mesuré le tour du stade avec un décamètre à la précision à chaque mesure de 1 mm en 4 fois car le décamètre était de longueur 70 m et  $4 \times 70 = 280$  m, donc environ 5 mm de précision sur le tracé du tour. donc 1 tour =  $253\ 500 \pm 5$  mm.

De plus, lorsqu'on roulait en vélo, on a pu osciller de maximum 10 cm sur notre tracé, 5 cm de chaque côté du tracé donc en modélisant le stade comme un cercle, de  $253\ 500 \pm 5$  mm de périmètre alors le rayon moyen est d'environ  $40\ 346 \pm 1$  mm donc

La distance d'un tour (périmètre minimal) est :  $2\pi(40\ 346 \pm 1\text{ mm} - 5\text{ cm}) = 2\pi(40\ 340) = 253\ 464 \pm 1$  mm et la distance d'un tour (périmètre maximal) est :  $2\pi(40\ 346 \pm 1\text{ mm} + 5\text{ cm}) = 2\pi(40\ 352) = 253\ 539 \pm 1$  mm

Ainsi, la variation entre deux tours est de :  $253\ 539 - 253\ 464 = 75$  mm soit une précision pendant les tours à vélo d'environ 7,5 cm

L'expérience maximal dure 75 tours, donc on peut cumuler cette imprécision 75 fois :  $75 * 75\text{ mm} = 5\ 625\text{ mm} = 5,625$  mètre sur 75 tours et en distance totale on est à 19 km

donc  $100 * (\text{variation}) / (\text{distance totale}) = 0,02\%$ , donc nos mesures réelles sont précises à une variation de 0,02 % près et sur 25 tours, petite expérience :  $100 * (0,075 * 25) / (6338) = 0,03\%$

### II. Données par les capteurs

**Garmin 935, Garmin 735, Garmin EDGE 1000** : De manière analogue au statique :

$10^{-9}$  m de précision sur la distance

$10^{-3}$  s de précision sur le temps

et on est sur des ordres de grandeurs de durée très inférieures (1h au lieu de 24h) et de distance similaire (10/20 km maximum contre 10 km) donc les incertitudes sont tout autant négligeables dans nos calculs.

### Capteur mécanique :

Nous n'avons relevé que deux mesures de ce capteur, une au top départ et une à l'arrivée. Nous avons étalonné le capteur mécanique au mètre près donc une variation de  $\pm 1$  m donc de 2 m au départ et la même chose à l'arrivée donc une variation totale de 4m donc incertitude relative du capteur mécanique :  $100 * (4\text{ m}) / 19\ 300 = 0,02\%$  donc négligeable tout comme les autres.

⇒ Toutes les incertitudes liées à nos mesures sur les mesures des capteurs sont négligeables, on peut donc étudier les incertitudes de prise de mesure des capteurs sans prendre en compte notre impact de la mesure.