Corrigé

Corrigé Epreuve d'info banque PT 2018

[1]:
```
from math import *     # d'après le texte
import numpy as np
```

[2]:
```
#LT est la liste des temps de mesure
#LT = [0, 0.2, 0.4, 0.60001, 0.8, 1]
LTexp = [0.2 * i for i in range(50)]

#LVexp liste des vitesses mesurées (relevées sur le sujet)
LVexp = [0.00, 0.57, 2.45, 4.35, 5.52, 6.27, 7.05, 7.61, 8.17, 8.73, 9.17, 9.48, 9.
↪79, 10.11, 10.43, 10.67, 10.90, 11.13, 11.34,\
        11.57, 11.60, 11.63, 11.67, 11.70, 11.78, 11.88, 11.93, 12.03, 12.13, 12.
↪17, 12.21, 12.24, 12.28, 12.31, 12.35, 12.40,\
        12.39, 12.33, 12.27, 12.22, 12.17, 12.15, 12.14, 12.14, 12.15, 12.24, 12.
↪19, 12.04, 11.89, 11.73]
```

3.1 Analyse du déroulement de la course (20% barême)

3.1.1 Détermination de l'instant d'arrivée

Q12a

On intègre la vitesse sur l'intervalle $[t_i, t_{i+1}]$ pour déterminer la position au temps t_{i+1} :

$$x_{i+1} = x_i + \int_{t_i}^{t_{i+1}} v(t)dt$$

Q12b

On utilise la formule de la méthode des trapèzes pour l'intégrale :

$$\int_{t_i}^{t_{i+1}} v(t)dt \simeq \frac{v_{i+1} + v_i}{2}(t_{i+1} - t_i)$$

ce qui donne :

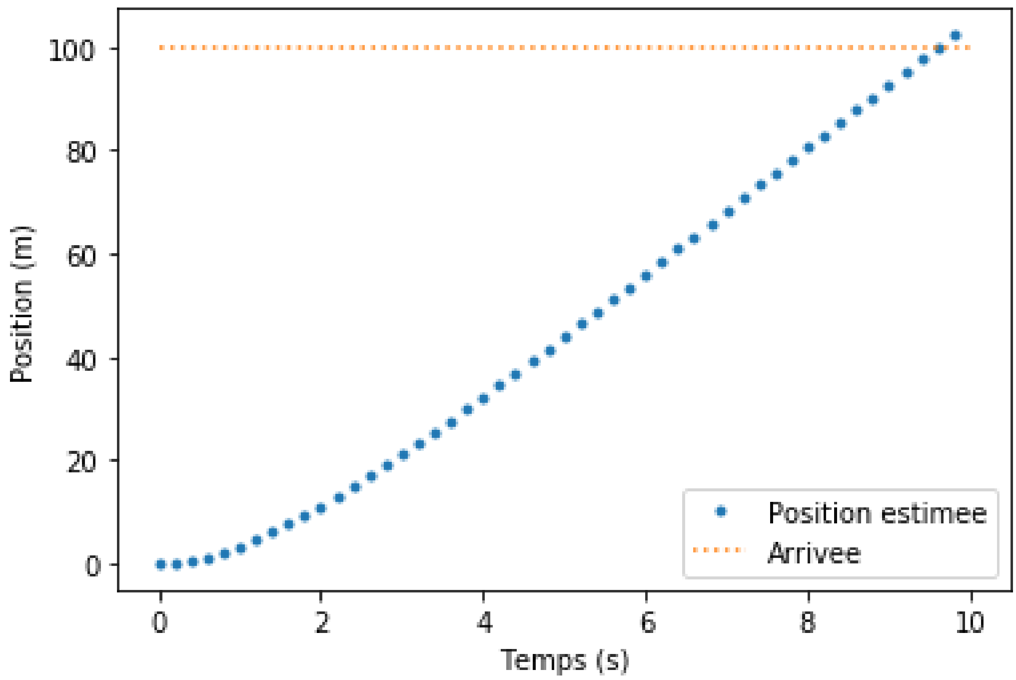
$$x_{i+1} = x_i + \frac{v_{i+1} + v_i}{2}(t_{i+1} - t_i)$$

Q12c

[3]:
```
def inte(Lv, LT):
    """renvoie la liste des positions estimées, avec des listes"""
    LX = [0]
    for i in range(len(LT)-1):
        LX.append(LX[-1] + (Lv[i + 1] + Lv[i]) / 2 * (LT[i + 1] - LT[i]))
    return LX
LXexp = inte(LVexp, LTexp)
```

Q13

[4]:
```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(LTexp, LXexp, '.')
plt.plot([0, 10], [100, 100], ':')
plt.xlabel('Temps (s)')
plt.ylabel('Position (m)')
plt.legend(['Position estimee', 'Arrivee'], loc=4)
plt.show()
```



Q14a

On a une relation affine entre d et T :

$$d = X_{iA-1} + \frac{X_{iA} - X_{iA-1}}{t_{iA} - t_{iA-1}} \cdot (T - t_{iA-1})$$

$$\Leftrightarrow T = t_{iA-1} + (d - X_{iA-1}) \cdot \frac{t_{iA} - t_{iA-1}}{X_{iA} - X_{iA-1}}$$

Q14b

[5]:
```
def arrivee(LX, LT, d):
    """calcule l'instant d'arrivée à la distance d, par interpolation linéaire,
    avec while"""
    if LX[-1] < d:
        return False   # si le coureur n'atteint pas l'arrivée
    i = 0
    while LX[i] < d:   # on détermine entre quelles mesures, le coureur
        i += 1         # passe la ligne d'arrivée
    return LT[i - 1] + (d - LX[i - 1]) * (LT[i] - LT[i - 1]) / (LX[i] - LX[i - 1])
```

Q14c

[6]:
```
arrivee(LXexp, LTexp, 100)
```

[6]: 9.600084674005078

3.1.2 Délimitation des phases de la course

[7]:
```
def f(LV, LT):
    LY = []
    for k in range(len(LT) - 1):
        LY.append((LV[k + 1] - LV[k]) / (LT[k + 1] - LT[k]))
    return LY
```

Q15

Lorsqu'on applique la fonction f aux listes LVexp et LTexp on estime l'accélération par dérivation numérique de la vitesse. La dérivée $\frac{dv}{dt}$ est approximée par le taux d'accroîssement :

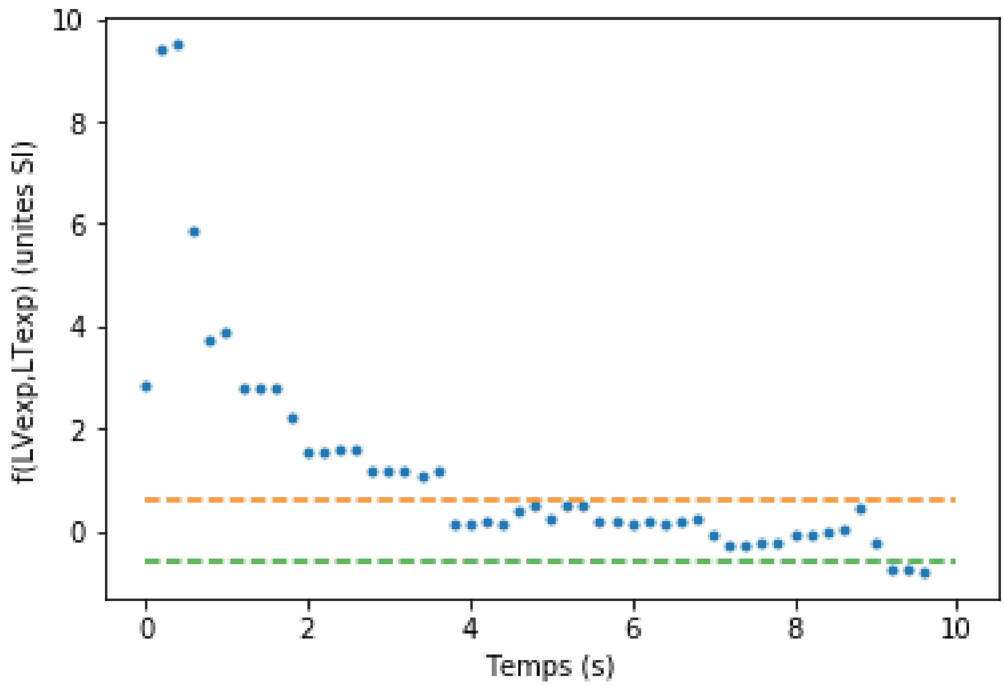
$$\frac{dv(t_i)}{dt} \simeq \frac{v_{i+1} - v_i}{t_{i+1} - t_i}$$

Q16a

La longueur de f(LVexp,LTexp) vaut $n-1$ car la formule précédente utilise deux échantillons successifs de position pour calculer une vitesse, alors avec n échantillons de position, on calcule $n-1$ vitesses.

Q16b

[8]:
```
plt.figure()
plt.plot(LTexp[:-1], f(LVexp, LTexp), '.')    # on a enlevé la dernière valeur de␣
 ↪LTexp
'''Pour avoir la figure du sujet'''          # pour avoir le même nombre de points
LY = f(LVexp, LTexp)
moyenne = sum(LY) / len(LY)
    #variante    moyenne = np.array(LY).mean()
borne = 0.5 * moyenne
plt.plot([0, 10], [borne, borne], '--')
plt.plot([0, 10], [-borne, -borne], '--')
plt.xlabel('Temps (s)')
plt.ylabel('f(LVexp,LTexp) (unites SI)')
plt.show()
```



Q17

[9]:
```
def instants(LY, LT):
    """renvoie le temps de début de la phase à vitesse constante, et le temps de
    début de la phase de décélération"""
    i = 0
    moyenne = sum(LY) / len(LY)
    #variante    moyenne = np.array(LY).mean()
    borne = 0.5 * moyenne
    # on utilise une valeur absolue pour |Y|< 50%.moyenne
    while (abs(LY[i]) > borne) and (i < len(LY)):    # tant qu'on est pas rentré␣
↪dans
        i += 1                                       # le tube ni allé à la fin
    if i == len(LY):    # on est allé à la fin sans rentrer dans le tube
        vcons = -1
    else:               # on est rentré dans le tube avant d'avoir atteint la fin
        vcons = LT[i]   # i est le premier indice pour lequel on est dedans
    j = len(LY) - 1    # on prend le dernier indice
    while (LY[j] < -borne) and (j > 0):    # Tant qu'on est au dessus de la borne␣
↪inf
        j = j - 1                          # du tube et qu'on est pas remonté au␣
↪début
    if j == len(LY) - 1:    # on est arrivé au début sans trouver
        vdec = -1
    else:                   # on est sorti par en dessous du tube
        vdec = LT[j + 1]
    return vcons, vdec      # on renvoie les valeurs de temps
```

[10]:
```
instants(f(LVexp, LTexp), LTexp[:-1])   # on enlève la dernière valeur de LTexp
```

[10]: (3.8000000000000003, 9.200000000000001)

Q18

L'algorithme est constitué de 2 boucles successives qui s'exécutent au maximum n fois. La complexité est donc linéaire, en $\mathcal{O}(n)$ si la liste LY est de longueur n

3.2 Modélisation dynamique de la course (15% du barème total)

$$a(t) = A + Bv(t) + Cv(t)^2$$

3.2.1 Identification et validation du modèle ### Q19

[11]:
```
#import numpy as np
LAexp = f(LVexp, LTexp)
P = np.polyfit(LVexp[:-1], LAexp, 2)
C, B, A = P[0], P[1], P[2]
```

Q20a

La méthode d'Euler explicite est :

$$v(t_{i+1}) \simeq v(t_i) + (t_{i+1} - t_i)\frac{dv}{dt}(t_i)$$

Ce qui donne ici :

$$v_{i+1} = v_i + a_i(t_{i+1} - t_i) = v_i + \left(A + Bv_i + Cv_i^2\right)(t_{i+1} - t_i)$$

On peut rappeler la méthode d'Euler implicite (hors-programme) :

$$v(t_{i+1}) \simeq v(t_i) + (t_{i+1} - t_i)\frac{dv}{dt}(t_{i+1})$$

Et l'équation différentielle fournit une relation entre $v(t_{i+1})$ et $\dfrac{dv}{dt}(t_{i+1})$ ce qui permet de poursuivre le calcul.

Q20b

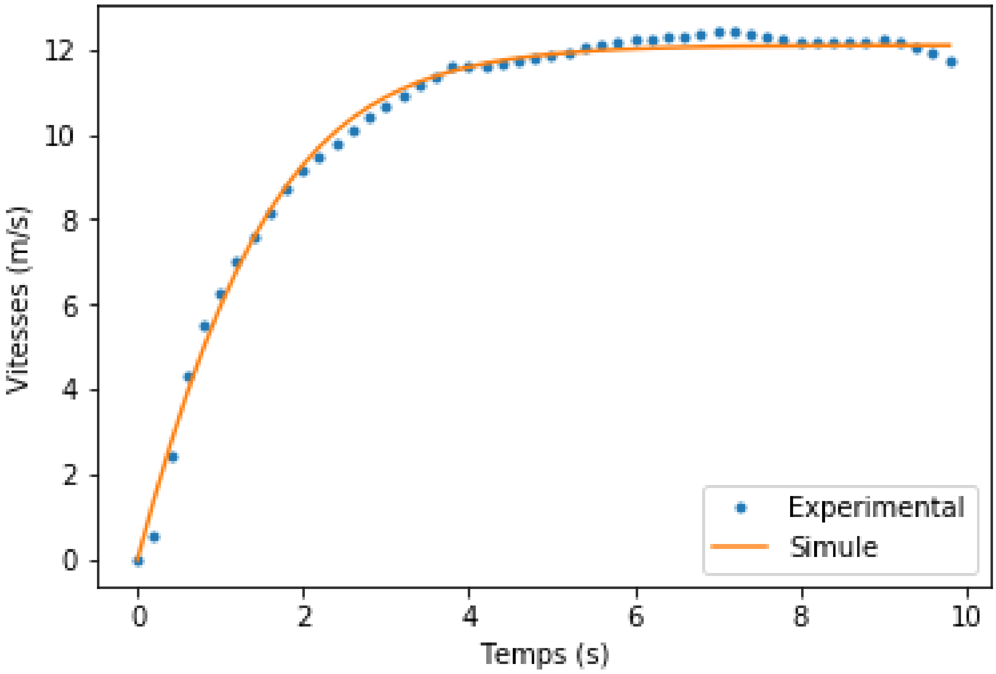
[12]:
```
def simu(LT, P):
    """Renvoie la liste des vitesse simulées instant de LT"""
    v0 = 0.0
    v = v0
    V = [v0]
    for i in range(len(LT) - 1):
        v += (P[2] + P[1] * v + P[0] * v**2) * (LT[i + 1] - LT[i])
        V.append(v)
    return V
```

Variante numpy

[13]:
```
def simu2(LT, P):
    """Renvoie le vecteur des vitesse simulées aux instants de LT"""
    n = len(LT)
    V = np.zeros(n)
    v0 = 0.0        # inutile ici
    V[0] = v0    # vu qu'on a une CI nulle
    for i in range(n - 1):     # pas besoin de remplir la première
        V[i+1] = V[i] + (P[2] + P[1] * V[i] + P[0] * V[i]**2) * (LT[i + 1] - LT[i])
    return V
```

Tracé

[14]:
```
plt.figure()
plt.plot(LTexp, LVexp, '.', label='Experimental')
plt.plot(LTexp, simu(LTexp, P), '-', label='Simule')
plt.xlabel('Temps (s)')
plt.ylabel('Vitesses (m/s)')
plt.legend(loc=4)
plt.show()
```



[15]:
```
LVsim = simu(LTexp, P)
N, D = 0, 0
for i in range(len(LVexp)):
    N = N + (LVsim[i] - LVexp[i])**2
    D = D + LVexp[i]**2
print(sqrt(N/D))
```

```
0.021856970388307535
```

Q21

La quantité affichée est :

$$\sqrt{\frac{\sum_{i=0}^{n-1}(v_{sim}(i)-v_{exp}(i))^2}{\sum_{i=0}^{n-1}(v_{exp}(i))^2}}$$

On évalue la racine du rapport entre l'écart au carré entre les deux estimations et la mesure expérimentale au carré prise pour référence.

3.2.2 Exploitation du modèle pour estimer les efforts sur le coureur

$$a_i = f_i + P_1 v_i + P_0 v_i^2$$

Q22

Dans la boucle for, les $k^{\text{ème}}$ composantes des listes n'existent pas, les listes sont à ce moment d'indice maximum $k-1$

[16]:
```
#Modifications proposées
def composantes(LV, LA, P):
    a, b = P[0], P[1]
    LA0, LA1, LA2 = [], [], []
    for k in range(len(LA)):
        LA2.append(a*LV[k]**2 )
        LA1.append(b*LV[k])
        LA0.append(LA[k] - LA1[k] - LA2[k])
    return (LA0, LA1, LA2)

# la solution LA1 = LA1 + b*LV[k] fonctionne mais est
# moins rapide et coûteuse en mémoire sur des listes,
# elle serait à utiliser sur un array
```

Les 3 composantes recherchées sont données dans cet ordre :

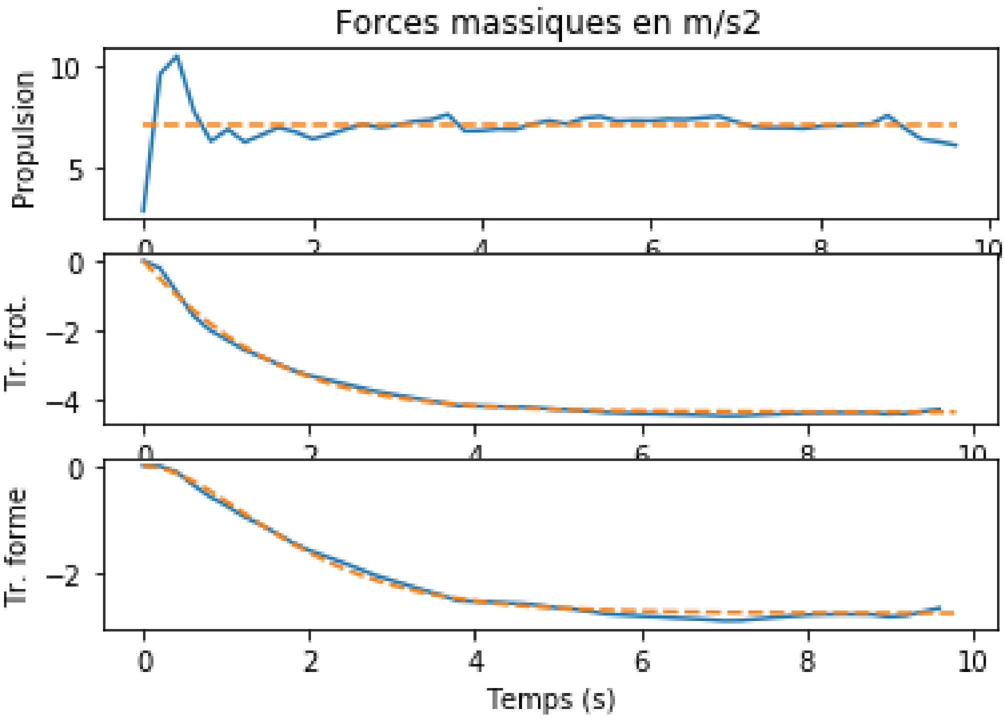
f_i, force massique de propulsion ;

$P_1 v_i$, force de traînée de frottement ;

$P_0 v_i^2$, force de traînée de forme.

[17]:
```
LA0exp, LA1exp, LA2exp = composantes(LVexp, LAexp, P)
plt.figure()
plt.subplot(311)
plt.plot(LTexp[:-1], LA0exp, '-')
plt.plot([0, LTexp[-2]], [A, A], '--' )
plt.ylabel('Propulsion')
plt.title('Forces massiques en m/s2')
plt.subplot(312)
plt.plot(LTexp[:-1], LA1exp, '-')
plt.plot(LTexp, B * np.array(simu(LTexp, P)), '--' )
plt.ylabel('Tr. frot.')
```

```
plt.subplot(313)
plt.plot(LTexp[:-1], LA2exp, '-')
plt.plot(LTexp, C * np.array(simu(LTexp, P))**2, '--' )
plt.ylabel('Tr. forme')
plt.xlabel('Temps (s)')
plt.show()
```



3.2.3 Bilan énergétique de la course

$$W = \int_0^T a(t)v(t)dt$$

Attention : ici $a(t)$ désigne une force massique (et pas l'accélération).

Q23

[18]:
```
def travail(LA, LV, LT, t):
    """Calcul du travail massique de la 'force'"""
    w, i = 0, 0
    while LT[i] < t:
        w += (LT[i + 1] - LT[i]) * LA[i] * LV[i]
        i += 1
    return w
```

[19]:
```
w1 = travail(LA0exp, LVexp, LTexp, arrivee(LXexp, LTexp, 100))
w2 = travail(LA1exp, LVexp, LTexp, arrivee(LXexp, LTexp, 100))
w3 = travail(LA2exp, LVexp, LTexp, arrivee(LXexp, LTexp, 100))
print("Pour le 100 m d'Usain Bolt, on obtient un travail massique de " +
      str(round(w1,0)) +" J/kg pour la propulsion,\n"\
```

```python
    + str(round(w2,0)) +  " J/kg pour la traînée de frottement et " +
    str(round(w3,0)) +" J/kg pour la traînée de forme")
```

```
Pour le 100 m d'Usain Bolt, on obtient un travail massique de 715.0 J/kg pour la
propulsion,
-407.0 J/kg pour la traînée de frottement et -245.0 J/kg pour la traînée de
forme
```

[ ]:

[ ]:

3.3 Stockage et mise en forme des données (25% du barême total)

3.3.1 Mise en oeuvre de la base de données

Q24

Chaque coureur ne participe qu'une fois à chaque épreuve, le couple {id_coureur, id_epreuve} est unique et peut donc constituer une clé primaire.

Q25

SELECT nom, date FROM epreuves WHERE distance = 100

Q26

Le requête proposée donne la liste des instants d'arrivée, instants initiaux de la phase à vitesse constante, instants initiaux de la phase de décélération et travaux massiques pour tous les "100 m" enregistrés courus en moins de 12 s.

Q27

SELECT c.nom, c.prenom, e.nom, e.date, p.temps FROM coureurs c JOIN performances AS p ON c.id = p.id_coureur JOIN epreuves AS e ON p.id_epreuve = e.id WHERE distance = 100

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: