Épreuve d'Informatique et Modélisation
Banque PT Session 2018

UPSTI

Corrigé édité à partir du fichier .ipynb

```
In [1]: from math import *    # d'après le texte

In [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

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

Q12b

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

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

Q12c

```
In [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)
```

Variante avec numpy

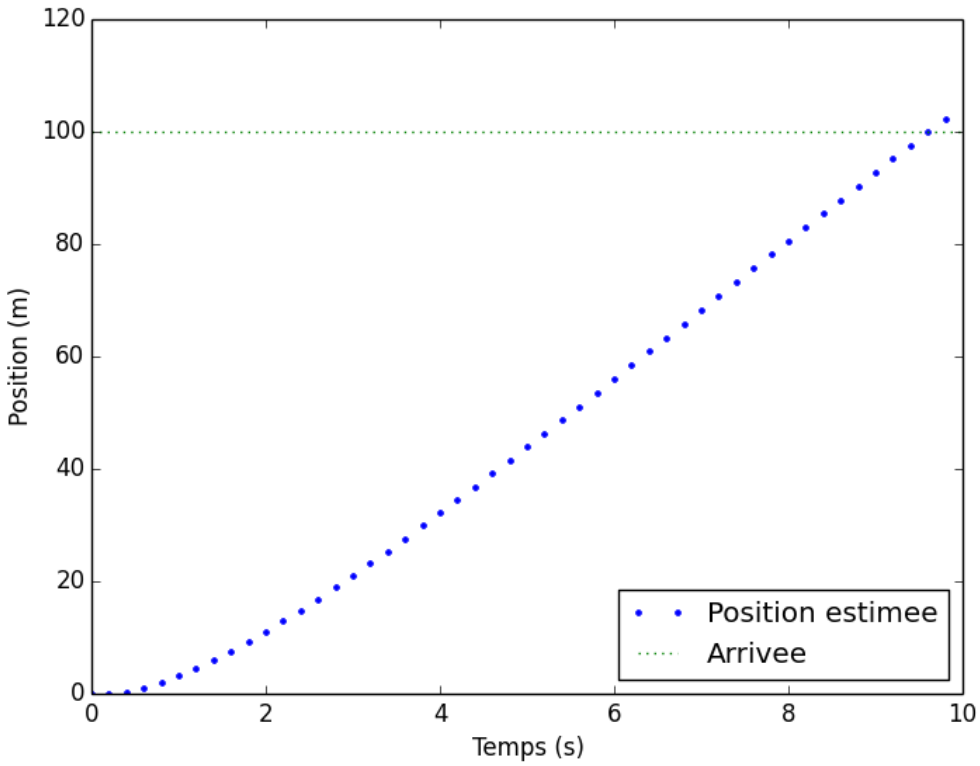
```
In [4]: import numpy as np
        def inte(Lv, LT):
            """renvoie la liste des positions estimées, avec numpy"""
            LV = np.array(Lv)
            return np.concatenate(([0], np.cumsum((LV[1:] + LV[:-1]) / 2 \
                    * np.diff(LT))))
```

Autre variante avec integrate

```
In [5]: from scipy import integrate
        def inte2(Lv, LT):
            """renvoie la liste des positions estimées, avec integrate"""
            return np.concatenate(([0], integrate.cumtrapz(Lv, LT)))
```

Q13

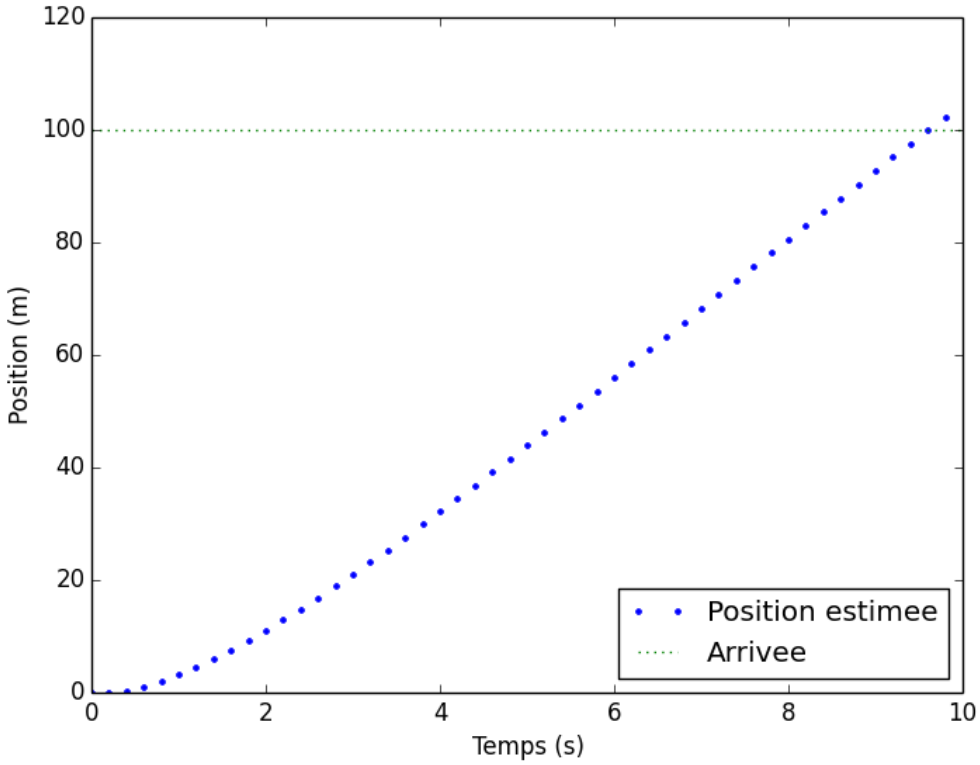
```
In [6]: import matplotlib.pyplot as p
        p.figure()
        p.plot(LTexp, LXexp, '.')
        p.plot([0, 10], [100, 100], ':')
        p.xlabel('Temps (s)')
        p.ylabel('Position (m)')
        p.legend(['Position estimee', 'Arrivee'], loc=4)
        p.show()
```



Variante plus propre mais ne correspondant pas à l'annexe du sujet sur la bibliothèque matplotlib :

```
In [7]: #import matplotlib.pyplot as p
        p.figure()
        p.plot(LTexp, LXexp, '.', label = 'Position estimee')
```

```
p.plot([0, 10], [100, 100], ':', label = 'Arrivee')
p.xlabel('Temps (s)')
p.ylabel('Position (m)')
p.legend(loc=4)
p.show()
```



Q14a

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

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

Q14b

```
In [8]: 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
            i = 0
```

```
            while LX[i] < d:
                i += 1
            return LT[i - 1] + (d - LX[i - 1]) * (LT[i] - LT[i - 1]) / (LX[i] - LX[i - 1])
```

Variante

```
In [9]: def arrivee2(LX, LT, d):
            """calcule l'instant d'arrivée à la distance d, par interpolation
            linéaire, avec enumerate"""
            for i, v in enumerate( LX ):
                if v > d:
                    return LT[i - 1] + (d - LX[i - 1]) * (LT[i] - LT[i - 1]) /
                            (LX[i] - LX[i - 1])
            return False
```

Q14c

```
In [10]: arrivee(LXexp, LTexp, 100)
```

```
Out[10]: 9.600084674005078
```

3.1.2 Délimitation des phases de la course

```
In [11]: 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 :

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

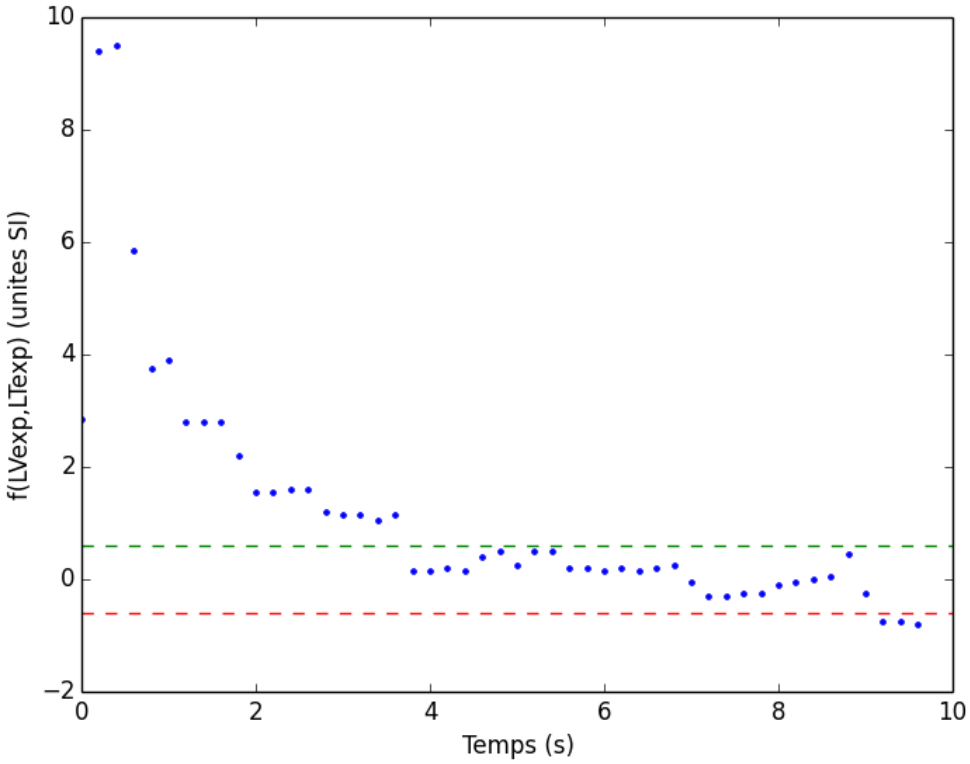
Q16a

La longueur de f(LVexp,LTexp) vaut $n - 1$.

Q16b

```
In [13]: p.figure()
         p.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
p.plot([0, 10], [borne, borne], '--')
p.plot([0, 10], [-borne, -borne], '--')
p.xlabel('Temps (s)')
p.ylabel('f(LVexp,LTexp) (unites SI)')
p.show()
```



Q17

```
In [16]: 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
             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 a été à la fin sans rentrer dans le tube
                 vcons = -1
```

```
            else:                # on est rentré dans le tube avant d'avoir été à 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
```

```
In [16]: instants(f(LVexp, LTexp), LTexp[:-1])   # on enlève la dernière valeur
                                                 de LTexp
```

```
Out[16]: (3.8000000000000003, 9.200000000000001)
```

```
In [17]: np.array(f(LVexp,LTexp)).mean()
```

```
Out[17]: 1.1969387755102041
```

Q18

L'algorithme est constitué de 2 boucles successives qui s'executent 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

```
In [18]: #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

Méthode d'Euler explicite

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

Q20b

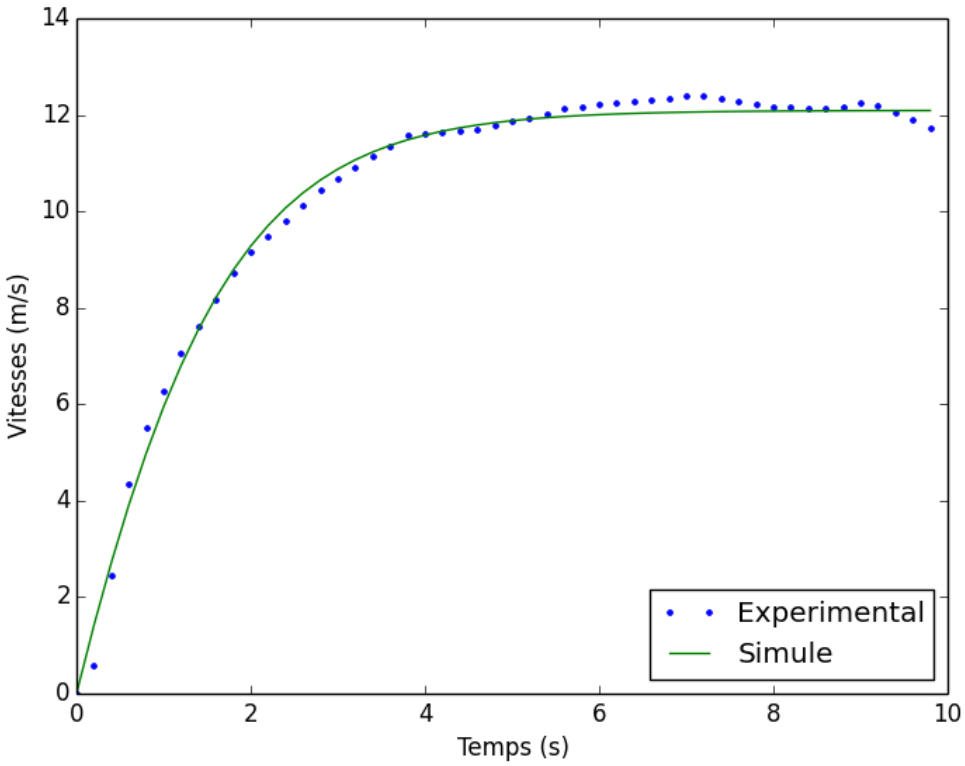
```
In [19]: def simu(LT, P):
             """Renvoie la liste des vitesse simulées instant de LT"""
             v0 = 0
             v = v0
             V = [v0]
             for i in range(len(LT) - 1):
                 v += (P[2] + P[1] * v + P[0] * v**2) * (LT[i + 1] - LT[i])
                 V.append(v)
             return V
```

Variante numpy

```
In [20]: def simu2(LT, P):
             """Renvoie l'array des vitesse simulées instant de LT"""
             n = len(LT)
             V = np.zeros(n)
             v0 = 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

```
In [23]: p.figure()
         p.plot(LTexp, LVexp, '.', label='Experimental')
         p.plot(LTexp, simu(LTexp, P), '-', label='Simule')
         p.xlabel('Temps (s)')
         p.ylabel('Vitesses (m/s)')
         p.legend(loc=4)
         p.show()
```




```
In [24]: 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))
```

```
0.02185697038830751
```

Q21

La quantité affichée est :

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

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 $k-1$

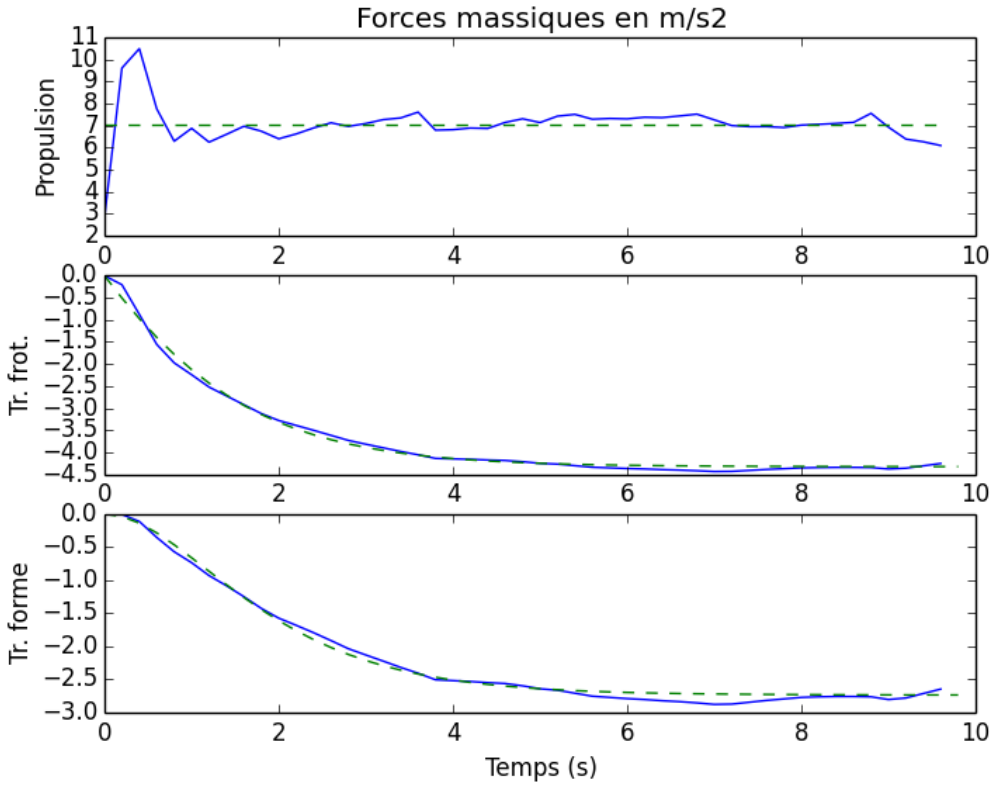
```
In [84]: #Modifications proposées
         def composantes(LV, LA, P):
             a, b = P[0], P[1]
             LA0, LA1, LA2 = [], [], []
             for k in range(len(LA)):
                 LA2.append(a*LV[k]**2 )
                 LA1.append(b*LV[k])
                 LA0.append(LA[k] - LA1[k] - 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 :

1. f_i, force massique de propulsion ;
2. $P_1 v_i$, force de traînée de frottement ;
3. $P_0 v_i^2$, force de traînée de forme.

```
In [26]: LA0exp, LA1exp, LA2exp = composantes(LVexp, LAexp, P)
         p.figure()
         p.subplot(311)
         p.plot(LTexp[:-1], LA0exp, '-')
         p.plot([0, LTexp[-2]], [A, A], '--' )
         p.ylabel('Propulsion')
         p.title('Forces massiques en m/s2')
         p.subplot(312)
         p.plot(LTexp[:-1], LA1exp, '-')
         p.plot(LTexp, B * np.array(simu(LTexp, P)), '--' )
         p.ylabel('Tr. frot.')
         p.subplot(313)
         p.plot(LTexp[:-1], LA2exp, '-')
         p.plot(LTexp, C * np.array(simu(LTexp, P))**2, '--' )
         p.ylabel('Tr. forme')
         p.xlabel('Temps (s)')
         p.show()
```



3.2.3 Bilan énergétique de la course

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

Q23

```
In [27]: 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
```

```
In [28]: 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"\
               +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

3.3.2 Traitement des données récupérées

```
In [29]: # Pour tester les fonctions suivantes
         T = [['Bolt', 'Usain', 'finale 100 m championnats du monde 2009', '2009-08-16', 9.58],\
              ['Gay', 'Tyson', 'finale 100 m championnats du monde 2009', '2009-08-16', 9.71],\
              ['Powell', 'Asafa', 'finale 100 m championnats du monde 2009', '2009-08-16', 9.84]
              ['Bailey', 'Daniel', 'finale 100 m championnats du monde 2009', '2009-08-16', 9.93
              ['Thompson', 'Richard', 'finale 100 m championnats du monde 2009', '2009-08-16', 9
              ['Chambers', 'Dwain', 'finale 100 m championnats du monde 2009', '2009-08-16', 10.
              ['Burns', 'Marc', 'finale 100 m championnats du monde 2009', '2009-08-16', 10.00],
              ['Patton', 'Darvis', 'finale 100 m championnats du monde 2009', '2009-08-16', 10.3
              ['Bolt', 'Usain', 'finale 100 m JO 2012', '2012-08-05', 9.63],\
              ['Blake', 'Yohan', 'finale 100 m JO 2012', '2012-08-05', 9.75],\
              ['Gatlin', 'Justin', 'finale 100 m JO 2012', '2012-08-05', 9.79],\
              ['Bailey', 'Ryan', 'finale 100 m JO 2012', '2012-08-05', 9.88],\
              ['Churandy', 'Martina', 'finale 100 m JO 2012', '2012-08-05', 9.94],\
              ['Thompson', 'Richard', 'finale 100 m JO 2012', '2012-08-05', 9.98],\
              ['Powell', 'Asafa', 'finale 100 m JO 2012', '2012-08-05', 11.99],\
              ['Blake', 'Yohan', 'finale 100 m championnats du monde 2011', '2011-08-28', 9.92],
              ['Dix', 'Walter', 'finale 100 m championnats du monde 2011', '2011-08-28', 10.05],
              ['Collins', 'Kim', 'finale 100 m championnats du monde 2011', '2011-08-28', 10.10]
```

```
['Lemaitre', 'Christophe', 'finale 100 m championnats du monde 2011', '2011-08-28'
['Bailey', 'Daniel', 'finale 100 m championnats du monde 2011', '2011-08-28', 10.2
['Vicaut', 'Jimmy', 'finale 100 m championnats du monde 2011', '2011-08-28', 10.27
['Carter', 'Nesta', 'finale 100 m championnats du monde 2011', '2011-08-28', 10.95
['Bolt', 'Usain', 'finale 100 m championnats du monde 2013', '2013-08-11', 9.77],\
['Gatlin', 'Justin', 'finale 100 m championnats du monde 2013', '2013-08-11', 9.85
['Carter', 'Nesta', 'finale 100 m championnats du monde 2013', '2013-08-11', 9.95]
['Bailey-Cole', 'Kemar', 'finale 100 m championnats du monde 2013', '2013-08-11',
['Ashmeade', 'Nickel', 'finale 100 m championnats du monde 2013', '2013-08-11', 9.
['Rodgers', 'Mike', 'finale 100 m championnats du monde 2013', '2013-08-11', 10.04
['Lemaitre', 'Christophe', 'finale 100 m championnats du monde 2013', '2013-08-11'
['Dasaolu', 'James', 'finale 100 m championnats du monde 2013', '2013-08-11', 10.2
```

3.3.2.1 Evolution des performances au fil du temps

Q28

```
In [30]: import datetime
         def nb_jours(date1, date2):    # cette fonction n'est pas demandée
                                          dans le sujet
                 """Renvoie le nombre entier de jours séparant les deux dates"""
                 date1 = str(date1[0])+"-"+str(date1[1])+"-"+str(date1[2])
                 date2 = str(date2[0])+"-"+str(date2[1])+"-"+str(date2[2])
                 DATETIME_FORMAT = "%Y-%m-%d"
                 from_dt = datetime.datetime.strptime(date1, DATETIME_FORMAT)
                 to_dt = datetime.datetime.strptime(date2, DATETIME_FORMAT)
                 timedelta = to_dt - from_dt
                 diff_day = timedelta.days
                 return diff_day

In [31]: def performances(nom, prenom, T):
             jours = []
             temps = []
             date0 = [2000, 1, 1]
             for i in range(len(T)):
                 if T[i][0].lower() == nom.lower() and \
                                 T[i][1].lower() == prenom.lower():
                     annee, mois, jour = T[i][3].split('-')
                     date = [int(annee), int(mois), int(jour)]
                     jours.append(nb_jours(date0, date))
                     temps.append(T[i][4])
             return (jours, temps)

         print(performances('bolt', 'usain', T))

([3515, 4600, 4971], [9.58, 9.63, 9.77])
```

3.3.2.2 Extraction des dix meilleurs temps

Q29

```
In [32]: def top10(T):
             for k in range(10):
                 mini = T[k][4]
                 for i in range(k + 1, len(T)):
                     if T[i][4] < mini:
                         temp = T[k]
                         T[k] = T[i]
                         T[i] = temp
                         mini = T[k][4]
             return T[:10]

         print(top10(T))
```

```
[['Bolt', 'Usain', 'finale 100 m championnats du monde 2009', '2009-08-16', 9.58], ['Bolt', 'Usa
```

Q30a

La complexité est en $\mathcal{O}(n)$, on réalise 10 boucles de n boucles, soit $10n$ boucles.

Q30b

Les algorithmes de tri performants ont une complexité en $\mathcal{O}(n \log n)$, pour un n grand ils sont donc moins rapides qu'une recherche des 10 plus petites valeurs.

Q30c

Le tri par sélection de l'ensemble de la liste a une complexité en $\mathcal{O}(n^2)$. Il est donc moins performant que les tris proposés si l'on trie toute la liste.