EPREUVE D'INFORMATIQUE ET MODELISATION

Durée : 2 h

Si au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, d'une part il le signale au chef de salle, d'autre part il le signale sur sa copie et poursuit sa composition en indiquant les raisons des initiatives qu'il est amené à prendre.

L'usage de calculatrices est interdit.

La **présentation**, la lisibilité, l'orthographe, la qualité de la **rédaction, la clarté et la précision** des raisonnements entreront pour une **part importante** dans **l'appréciation des copies**. En particulier, les résultats non justifiés ne seront pas pris en compte. Les candidats sont invités à encadrer les résultats de leurs calculs.

Analyse d’un 100 m à l’aide d’un cinémomètre à effet Doppler-Fizeau

1. Introduction

Le *100 m* est une épreuve d’athlétisme consistant à courir sur une distance de 100 m en ligne droite en une durée la plus faible possible. A très haut niveau, il est couru en moins de 10 secondes pour les hommes et 11 secondes pour les femmes. Le record du monde est actuellement détenu par l’athlète jamaïcain Usain Bolt, qui l’a couru en 9,58 secondes aux championnats du monde de Berlin en 2009.

Un *cinémomètre* est un appareil capable de mesurer la vitesse relative d’une cible (« *relative* » signifiant ici « *dans un référentiel lié à l’appareil* »). Le plus souvent, la mesure est réalisée à distance par émission d’ondes électromagnétiques qui se réfléchissent sur la cible et frappent un récepteur. Il existe deux façons d’exploiter ce phénomène pour identifier la vitesse relative de la cible :

- soit en mesurant le délai entre émission et réception, et donc la distance appareil-cible (télémétrie), à intervalles de temps réguliers,
- soit en mesurant l’écart des fréquences respectives des ondes émise et reçue (effet Doppler-Fizeau).

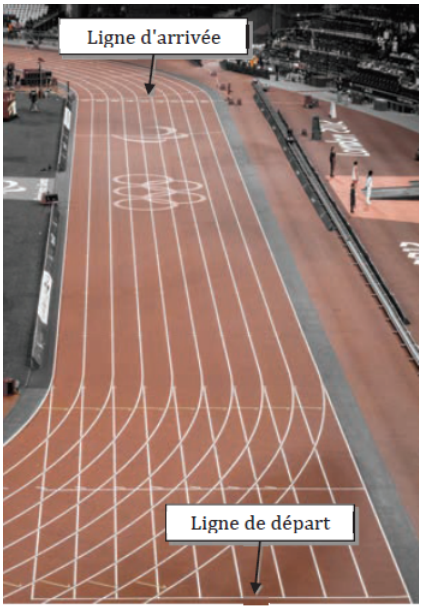

Figure 1 : une piste olympique de 100 m

Etant données la configuration rectiligne de la piste (**Figure 1**) et la faible durée d’une course, le 100 m se prête bien à l’utilisation d’un tel appareil pour analyser les performances des coureurs. Ainsi, au cours de la dernière décennie, plusieurs épreuves prestigieuses de 100 m ont fait l’objet de mesures cinémométriques.

Ce sujet traite de l’utilisation d’un cinémomètre à effet Doppler-Fizeau pour analyser les performances d’athlètes courant un 100 m. Il propose des algorithmes visant à exploiter les mesures du cinémomètre pour analyser les performances de la course et traiter les résultats obtenus.

2. Traitement des données recueillies

Nous nous intéressons à présent au traitement informatique des données fournies par le cinémomètre. L’objectif de cette partie est de proposer des algorithmes et des solutions permettant de :

- analyser le déroulement de la course (partie 2.1),
- identifier les efforts aérodynamiques et propulsifs exercés sur le coureur (partie 2.2),
- traiter les données ainsi recueillies pour fournir des indications aux entraîneurs (partie 2.3).

Ces parties, ainsi que leurs sous-parties, sont indépendantes.

Les fonctions et programmes demandés seront réalisés dans le langage Python. On supposera que le module `math` a été importé. Il est fortement conseillé d'utiliser des noms de variables ne laissant aucune ambiguïté et de commenter le code dès que vous le jugez nécessaire : un algorithme correct, même avec des erreurs de code, peut rapporter des points.

2.1. Analyse du déroulement de la course

Une fois l'acquisition terminée, les données stockées dans le microcontrôleur des jumelles sont transférées sur un ordinateur au moyen d'une liaison USB, puis importées dans un environnement Python. Les résultats des mesures sont alors stockés dans deux listes :

- `LT`, qui contient les **instants de mesure t_i** exprimés en secondes,
- et `LVexp`, qui contient les **vitesses mesurées $v_i = v(t_i)$** exprimées en mètres par seconde.

2.1.1. Détermination de l'instant d'arrivée

On souhaite dans un premier temps utiliser les vitesses mesurées pour déterminer l'instant auquel le coureur franchit la ligne d'arrivée.

Pour cela, on propose d'estimer les positions successives du coureur à l'aide de la méthode des trapèzes. On note $x_i = x(t_i)$ la valeur de la position estimée au i-ème instant de mesure. La position initiale x_0 sera prise nulle.

Q.1. a. Donner la relation entre x_{i+1}, x_i et $\int_{t_i}^{t_{i+1}} v(t)\,\mathrm{d}t$.

b. Donner l'expression approchée de $\int_{t_i}^{t_{i+1}} v(t)\,\mathrm{d}t$ en fonction de v_i, v_{i+1}, t_i et t_{i+1}. En déduire une estimation de x_{i+1} en fonction de x_i, v_i, v_{i+1}, t_i et t_{i+1}.

c. Ecrire une fonction `inte(LV, LT)` prenant pour arguments une liste `LV` de vitesses et une liste `LT` d'instants de mesure, et renvoyant la liste des positions estimées.

Les positions obtenues à partir de la liste `LVexp` sont placées dans la liste `LXexp`. Leur évolution en fonction du temps est représentée **Figure 10** pour le 100 m d'Usain Bolt. Sur ce graphe, on a également tracé une ligne horizontale en pointillés à la position correspondant à la distance entre la ligne d'arrivée et la jumelle, qui vaut ici 100 m.

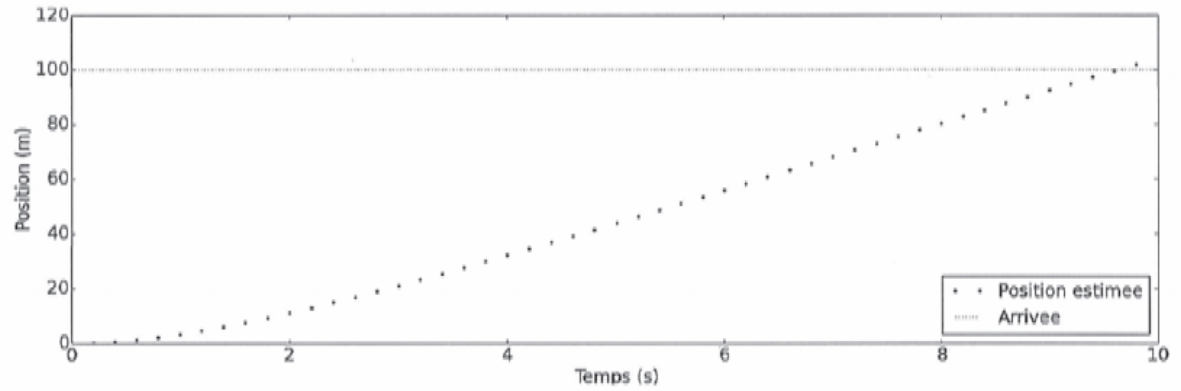

Figure 10 : positions estimées expérimentalement pour le 100 m d'Usain Bolt

On donne en annexe un extrait de la documentation du module `matplotlib.pyplot`. On suppose que ce module a été importé par la commande suivante : `import matplotlib.pyplot as plt`

Q.2. A l'aide de la documentation en annexe, donner la suite d'instructions permettant d'obtenir le tracé de la Figure 10 : positions estimées repérées par des marqueurs points, droite horizontale à 100 m allant de 0 à 10 s en pointillés ("*dotted line*"), étiquettes sur les deux axes, légende. On rappelle que pour tracer une droite, il suffit de deux points.

Pour estimer l'instant d'arrivée du coureur avec une résolution inférieure à l'intervalle de temps séparant deux mesures, on propose de procéder par interpolation linéaire. Si l'on appelle d la distance entre la ligne d'arrivée et la jumelle, le principe est le suivant (**Figure 11**) :

- on recherche dans un premier temps le premier indice i_A pour lequel $x_{iA} \geq d$,
- puis, en se basant sur l'hypothèse que la position évolue de manière affine entre t_{iA-1} et t_{iA}, on détermine l'instant d'arrivée T tel que $x(T) = d$.

Q.3. a. Donner l'expression de l'instant d'arrivée T en fonction de x_{iA}, x_{iA-1}, t_{iA}, t_{iA-1} et d.

b. Ecrire une fonction `arrivee(LX, LT, d)` prenant pour arguments la liste `LX` des positions estimées et la liste `LT` des instants de mesure, et

renvoyant l'instant d'arrivée à la distance `d`**, déterminé selon le principe ci-dessus. Si** `d` **n'est jamais atteinte, la fonction renverra** `False`**.**

c. Donner l'instruction affichant à l'écran l'instant de l'arrivée à 100 m à partir des listes `LXexp` **et** `LT`**.**

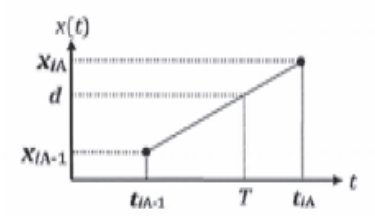

Figure 11 : principe du repérage de l'instant d'arrivée

2.1.2. Délimitation des phases de la course

Si l'on met de côté les phénomènes liés à la poussée sur les *starting-blocks*, que les mesures par jumelles laser ne permettent pas d'étudier, on peut distinguer trois phases dans la course : accélération, vitesse quasi-constante et décélération. L'objectif de cette partie est de délimiter ces trois phases.

On propose pour cela de s'appuyer sur la fonction suivante, dont la documentation a été effacée :

```
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
```

Q.4. Lorsqu'on applique cette fonction aux listes `LVexp` **et** `LTexp`**, quelle grandeur physique cela permet-il d'estimer ? Justifier la réponse. Quelle est l'approximation utilisée par cette fonction pour réaliser cette estimation ?**

Le résultat de `f(LVexp, LTexp)` est représenté **Figure 12** en fonction des instants de mesure. On a également repéré la *bande à ±50 % de la valeur moyenne* de `f(LVexp, LTexp)`.

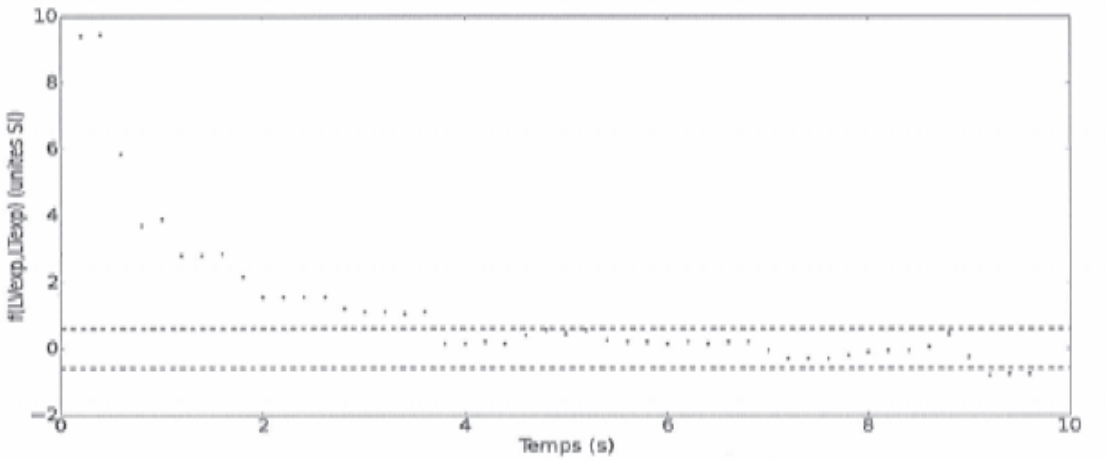

Figure 12 : `f(LVexp,LTexp)` en fonction du temps

Q.5. a. Si la longueur de `LVexp` **et** `LTexp` **vaut** `n`**, que vaut la longueur de** `f(LVexp, LTexp)` **?**

b. On rappelle que les deux listes passées en argument de la fonction `plot` doivent être de même longueur. Donner une instruction permettant de tracer `f(LVexp, LTexp)` en fonction du temps (on demande la courbe seule, sans format particulier ni légendes).

En notant `LY` = `f(LVexp, LTexp)`, on propose les définitions suivantes :

- la *phase à vitesse constante* débute au premier instant où `LY` entre dans la bande à ± 50 % de la valeur moyenne de `LY` (on supposera qu'il n'y entre pas forcément par au-dessus) ;
- la *phase de décélération* débute au premier instant où `LY` sort de la bande à ± 50 % par en-dessous et reste en –dessous de cette bande jusqu'à la fin de la mesure.

Par exemple, sur la **Figure 12**, la *phase à vitesse constante* débute à 3,8 s et la *phase de décélération* à 9,2 s. Mais, s'il y avait eu un dernier point dans la bande à ± 50 %, alors la phase de décélération n'aurait pas pu être définie pour cette mesure.

Pour le calcul de la valeur moyenne de `LY`, on accepte une simple moyenne arithmétique (il n'est donc pas demandé d'exprimer cette moyenne par une intégrale, ni d'utiliser d'autres listes que `LY`).

Q.6. Ecrire une fonction `instants(LY, LT)` prenant pour arguments une liste `LY` obtenue par `f` et une liste `LT` d'instants de mesure, et renvoyant le couple d'instants définis ci-dessus. Si un instant (voire les deux !) n'est pas trouvé, on lui attribuera la valeur -1.

Q.7. Donner, en justifiant la réponse, la complexité de `instants(LY, LT)` en fonction de la longueur des données si l'on suppose la liste `LY` quelconque.

2.2. Modélisation dynamique de la course

Les mesures de la vitesse peuvent également être employées pour construire un modèle dynamique de la course. On fait pour cela l'hypothèse que le coureur est soumis à trois forces longitudinales :

- une *force propulsive* que l'on suppose *constante* en première approximation,
- une *traînée de frottement* (proportionnelle à la vitesse),
- et une *traînée de pression*, également appelée *de forme* (proportionnelle au carré de la vitesse).

Le principe fondamental de la dynamique permet alors d'écrire la relation suivante entre la vitesse v et l'accélération a :

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

où A, B et C sont trois coefficients constants.

L'objectif de cette partie est, à partir des vitesses et accélérations mesurées pendant la course :

- *d'identifier A, B et C et de valider le modèle,*
- *d'exploiter le modèle pour estimer la traînée subie par le coureur tout au long de la course et en déduire la force propulsive qu'il a exercée,*
- *et d'estimer les travaux respectifs de chacune des trois composantes ci-dessus.*

2.2.1. Identification et validation du modèle

Pour identifier les coefficients du modèle, on dispose des mêmes listes qu'à la partie précédente :

- `LT`, qui contient les **instants de mesure t_i** exprimées en secondes,
- et `LVexp`, qui contient les **vitesses mesurées $v_i = v(t_i)$** exprimées en mètres par seconde.

A partir de ces listes, on en a construit une troisième : LAexp, qui contient les **accélérations estimées aux instants de mesure $a_i = a(t_i)$**, exprimées en $m.s^{-2}$. Indépendamment des résultats obtenus dans la partie précédente, on supposera dans cette partie que les trois listes sont de même longueur.

On propose d'utiliser la fonction polyfit du module numpy qui calcule, par la méthode des moindres carrés, les coefficients de la fonction polynomiale passant « au mieux » par un ensemble de points donnés. Un extrait de la documentation de cette fonction est donné ci-dessous.

```
polyfit(x, y, deg)

permet d'obtenir un polynôme à partir d'un ensemble de points (x,y).

Arguments d'entrée : x et y, deux listes de même longueur
                     deg, entier, degré du polynôme
Argument de sortie : p, liste des coefficients polynomiaux, la
                         puissance la plus élevée en premier.
```

On suppose pour cette question que numpy a été importé par la commande : **import** numpy **as** np.

Q.8. Donner l'instruction permettant d'identifier les coefficients du modèle à partir des listes LAexp et LVexp et de les placer dans une liste P. Préciser, pour chaque terme de P, s'il s'agit de *A*, *B* ou *C* tels que ces coefficients sont définis dans l'équation ci-dessus.

On souhaite valider le modèle en résolvant numériquement l'équation ci-dessus et en comparant les vitesses simulées aux vitesses mesurées. On se propose pour cela d'utiliser la méthode d'Euler explicite. On rappelle que les instants de mesure ne sont pas espacés de façon rigoureusement régulière.

Q.9. a. En appliquant la méthode d'Euler explicite, déterminer la relation de récurrence permettant de calculer v_{i+1} en fonction de v_i, t_{i+1}, t_i et les coefficients *A*, *B* et *C*.

b. Ecrire une fonction simu(LT, P) prenant pour arguments une liste LT d'instants et la liste P des coefficients précédemment obtenue, et renvoyant la liste des vitesses simulées aux instants de LT. La vitesse initiale v_0 sera choisie nulle.

On donne **Figure 13** le résultat de cette simulation pour les instants de mesure, superposé aux vitesses mesurées, toujours pour les mesures issues du 100 m d'Usain Bolt.

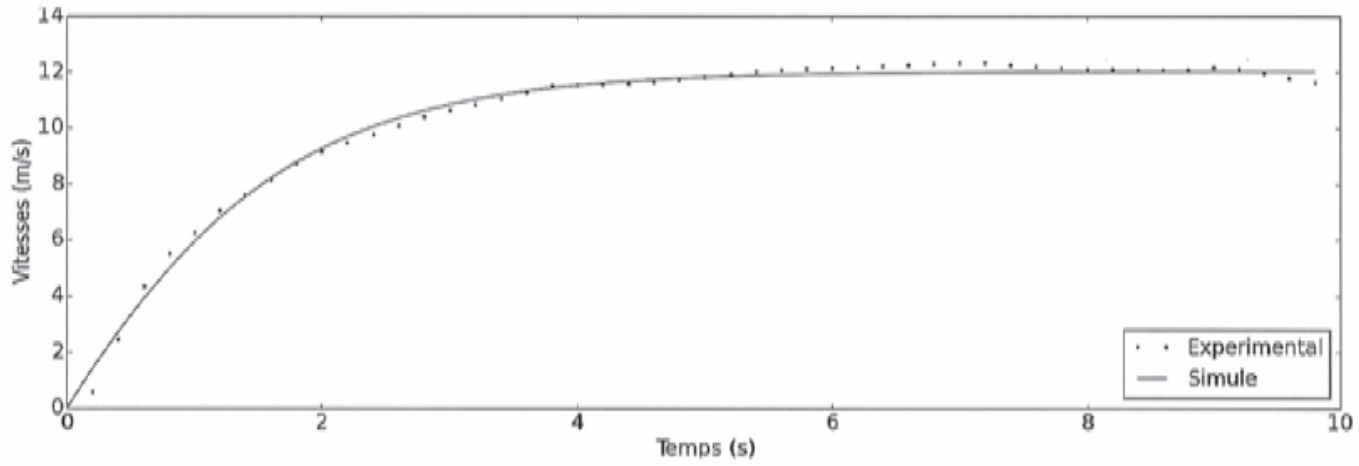

Figure 13 : comparaison des vitesses mesurées et des vitesses simulées grâce au modèle proposé

Les courbes semblant qualitativement proches, on souhaite valider quantitativement le modèle. On exécute pour cela le bloc d'instructions suivant :

```
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))
```

Q.10. Donner l'expression de la quantité affichée à l'écran par ces instructions et expliquer en quoi cette quantité mesure, quantitativement, l'écart entre les mesures et la simulation.

On se donne un seuil de 5 %. Le bloc d'instructions affiche 0,022, soit 2,2 %. On valide donc le modèle.

2.2.2. Exploitation du modèle pour estimer les efforts sur le coureur

On souhaite à présent exploiter le modèle pour estimer les efforts exercés sur le coureur tout au long de la course. L'idée est d'injecter les vitesses v_i et accélérations a_i mesurées dans la relation précédente, en conservant les coefficients des termes en v et en v^2 mais en considérant cette fois le terme « constant » comme inconnu et variable (l'hypothèse d'une force propulsive constante est en effet très discutable).

En introduisant les coefficients du polynôme P obtenu précédemment, cela s'écrit :

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

où f_i est un nouveau terme « propulsif », inconnu et variable, que l'on souhaite identifier à partir de v_i et a_i.

On dispose toujours des listes `LT`, `LVexp` et `LAexp` définies à la partie 2.1, ainsi que de la liste `P` des coefficients. Pour identifier f_i ainsi que les termes de frottement et de pression, un candidat propose le programme suivant :

```
def composantes(LV,LA,P):
    a, b = P[0], P[1]
    LA0, LA1, LA2 = [], [], []
    for k in range(len(LA)):
        LA2[k] = a*LV[k]**2
        LA1[k] = LA1 + b*LV[k]
        LA0[k] = LA[k] – LA1[k] – LA2[k]
```

```
    return (LA0,LA1,LA2)
```

mais ce programme ne produit malheureusement pas le résultat escompté. A la place, le message suivant apparaît lorsque l'on tente d'appeler cette fonction :

```
IndexError: list assignment index out of range
```

Q.11. Expliquer quelle est l'erreur dans ce programme. Proposer une modification de la / des ligne(s) incriminées afin de corriger l'erreur. Indiquer alors dans quel ordre sont renvoyées les trois composantes recherchées.

Ces composantes sont représentées **Figure 14**, en trait plein pour les mesures et en pointillés pour le modèle. On y distingue notamment, sur la courbe du terme propulsif, la poussée initiale ainsi que la « baisse de régime » finale due à la fatigue. On notera qu'il s'agit de *forces massiques*, homogènes à des accélérations.

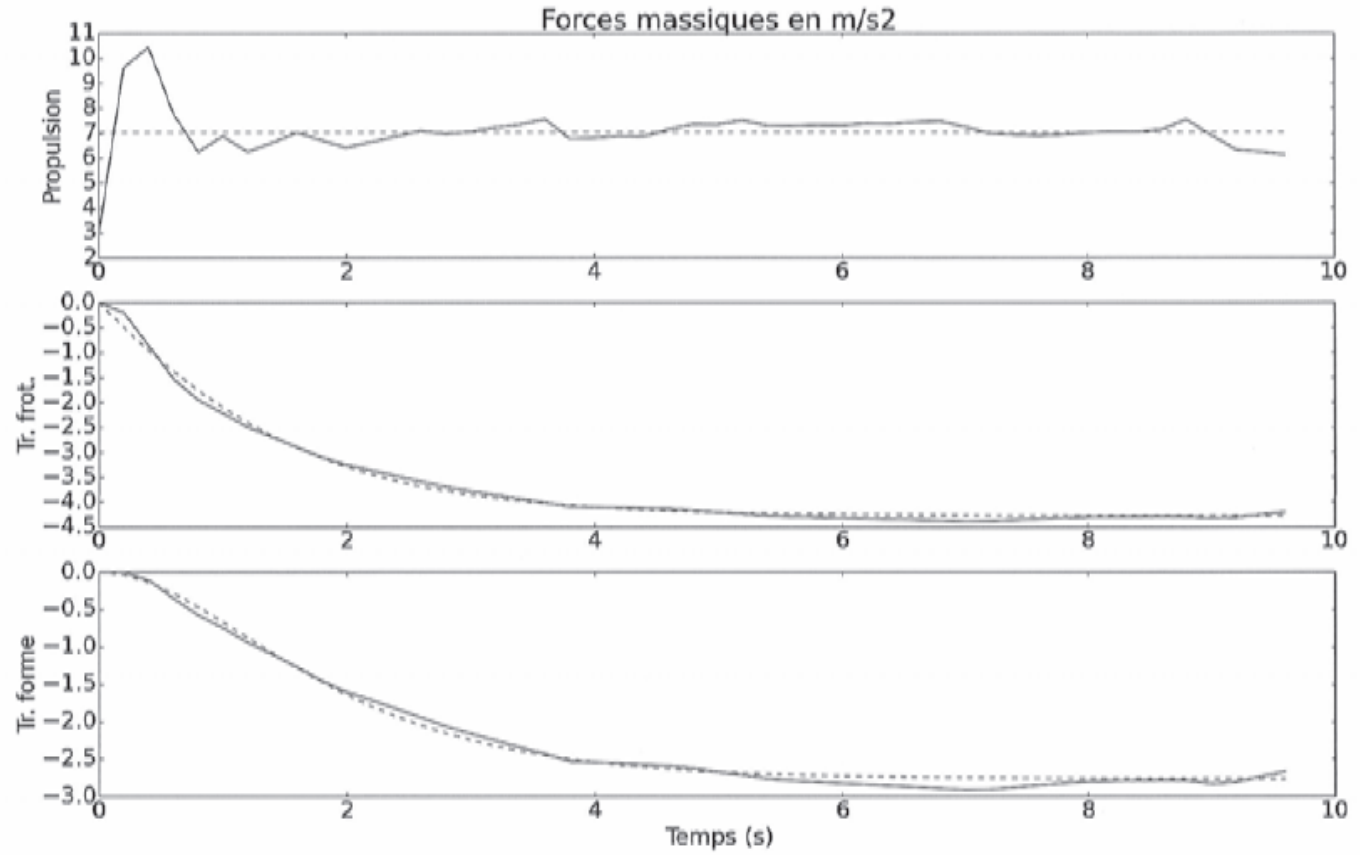

Figure 14 : forces massiques de propulsion, de traînée de frottement et de traînée de forme

2.2.3. Bilan énergétique de la course

Afin de réaliser un bilan énergétique de la course, on souhaite estimer les travaux respectifs de ces composantes (là encore, il s'agit de travaux massiques) définis par :

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

où $a(t)$ est la force massique considérée (l'une des trois composantes) et $v(t)$ la vitesse.

On dispose pour cela de l'ensemble des données calculées précédemment : les instants de mesure `LTexp`, les vitesses mesurées `LVexp`, les accélérations expérimentales `LAexp` et les trois composantes `LA0exp`, `LA1exp` et `LA2exp`. On rappelle que dans cette partie, toutes ces listes sont de même longueur.

Q.12. Ecrire une fonction `travail`, dont les arguments d'entrée sont à définir mais incluent au moins une force massique `LA`, qui retourne un flottant égal au travail massique de cette « force » tout au long de la course. Cette fonction pourra appeler n'importe quelle fonction définie dans ce sujet conformément à son en-tête, qu'elle ait été codée ou non.

Pour le 100 m d'Usain Bolt, on obtient un travail massique d'environ 700 J/kg pour la force de propulsion, - 400 J/kg pour la traînée de frottement et – 240 J/kg pour la traînée de forme.

2.3. Stockage et mise en forme des données

Une équipe professionnelle d'athlétisme souhaite acquérir un cinémomètre afin d'aider ses coureurs à s'entraîner plus efficacement. Pour cela, elle envisage de déterminer, après chaque course, les performances abordées dans les parties précédentes (instant d'arrivée, limites des phases d'accélération et de décélération, travaux des trois composantes) puis de les stocker dans une base de données afin de suivre l'évolution des performances de chaque coureur et de fournir aux entraîneurs des données synthétiques susceptibles de les guider dans la définition des programmes d'entraînement.

L'objectif de cette partie est :

- *de mettre en œuvre la base de données en écrivant quelques requêtes couramment utilisées,*
- *et d'élaborer des programmes mettant en forme les données extraites de la base afin de fournir des indications synthétiques aux entraineurs.*

2.3.1. Mise en œuvre de la base de données

Une analyse des besoins des entraîneurs a conduit à la définition d'une base de données constituée de trois tables, dont les attributs respectifs sont partiellement définis dans le **Tableau 1**.

Table `coureurs`		
Attribut	Type	Description
`id`	Entier	Identifiant du coureur (clé primaire)
`nom`	Chaîne	Nom du coureur
`prenom`	Chaîne	Prénom du coureur
Table `epreuves`		
Attribut	Type	Description
`id`	Entier	Identifiant de l'épreuve (clé primaire)
`nom`	Chaîne	Nom précis de l'épreuve
`date`	Chaîne	Date de l'épreuve au format `'AAAA-MM-JJ'`
`distance`	Flottant	Distance courue (m)
Table `performances`		
Attribut	Type	Description
`id_coureur`	Entier	Identifiant du coureur
`id_epreuve`	Entier	Identifiant de l'épreuve
`temps`	Flottant	Temps, ou instant d'arrivée (s)
`inst_cte`	Flottant	Instant initial da la phase à vitesse constante (s)
`inst_dec`	Flottant	Instant initial de la phase de décélération (s)
`travail`	Flottant	Travail massique de la force de propulsion (J/kg)

Tableau 1 : définition partielle de la base de données

On précise que pour la table `performances`, la clé primaire est constituée du couple formé par les deux identifiants `id_coureur` et `id_epreuve`.

Q.13. Justifier que ce couple d'identifiants constitue bien une clé primaire pour cette table.

Q.14. Ecrire une requête SQL renvoyant les noms et dates de toutes les épreuves de « 100 m » enregistrées dans la base.

Q.15. Que fait la requête suivante ?

```
SELECT p.temps, p.inst_cte, p.inst_dec, p.travail
FROM performances p
JOIN epreuves e ON e.id = p.id_epreuve
WHERE e.distance = 100 AND temps <= 12
```

Q.16. Ecrire une requête SQL renvoyant les noms et prénoms des coureurs, les noms et dates des épreuves, et les temps réalisés pour tous les « 100 m » enregistrés dans la base.

2.3.2. Traitement des données récupérées

L'environnement Python utilisé permet, grâce à des fonctions non étudiées ici, d'envoyer des requêtes SQL à la base de données et d'en récupérer les résultats. Ceux-ci peuvent ensuite être traités pour fournir des données synthétiques aux entraîneurs.

Dans cette partie, on suppose que le résultat de la requête précédente a été placé dans une liste de listes nommée `T`, dont chaque ligne `T[i]` est construite de la façon suivante :

```
[nom_coureur, prenom_coureur, nom_epreuve, date_epreuve, temps]
```

où chaque élément ci-dessus est une chaîne, à l'exception de `temps` qui est un flottant.

2.3.2.1. Evolution des performances des coureurs au fil du temps

L'entraîneur souhaite, à partir du tableau `T`, pouvoir visualiser l'évolution des performances de chaque coureur au fil du temps. Pour cela, il faut extraire, pour un coureur donné, les temps et les dates des épreuves, puis convertir les dates en un nombre de sorte à pouvoir visualiser des graphiques.

On suppose, pour la question suivante, que chaque coureur est identifié sans ambiguïté par son nom et son prénom. Cependant, ceux-ci ne sont pas forcément tous saisis de la même façon en ce qui concerne la casse, c'est-à-dire les minuscules et les majuscules (on peut ainsi trouver `'Nom'`, `'NOM'`, `'nom'` …). Or, on souhaite que l'identification du coureur fonctionne quelle que soit la casse utilisée.

On donne ci-dessous quelques méthodes et fonctions permettant de manipuler des chaînes de caractères (toutes ne sont pas utiles). Dans ce qui suit, `s` est une chaîne.

- `list(s)` renvoie la liste des caractères de `s` ; par exemple, `list('abc')` renvoie `['a','b','c']` ;
- `s.split(c)` « découpe » `s` à chaque occurrence du caractère `c` et renvoie la liste des « morceaux » sans modifier `s` ; par exemple, `'Je suis là'.split(' ')` renvoie `['Je','suis','là']` ;

- `int(s)` convertit `s` en un entier, si `s` peut être interprétée comme telle ; par exemple, `int('1')` et `int(' 1 ')` renvoient toutes deux l'entier `1` ;
- `float(s)` fait de même pour un flottant ; par exemple, `float('1')` renvoie le flottant `1.0` ;
- `s.lower()` passe `s` en minuscules ; par exemple, `'Bonjour'.lower()` renvoie `'bonjour'` ;
- `s.upper()` passe `s` en majuscules ; par exemple, `'Bonjour'.lower()` renvoie `'BONJOUR'`.

Enfin, pour convertir les dates en nombre, on dispose d'une fonction **`nb_jours`**`(date1, date2)` qui :

- prend pour arguments deux dates qui sont des listes au format `[annee, mois, jour]`, où `annee`, `mois` et `jour` sont des entiers,
- et renvoie le nombre (entier) de jours séparant les deux dates.

On rappelle que chaque date du tableau `T` est une chaîne au format `'AAAA-MM-JJ'`.

Q.17. Ecrire une fonction `performances`(`nom, prenom, T`) prenant en arguments deux chaînes `nom` et `prenom` et le tableau `T`, qui recherche dans `T` les données relatives au coureur identifié par `nom` et `prenom` puis renvoie un couple de listes (jours, temps) de même longueur telles que, pour tout indice `i` strictement inférieur à cette longueur :

- **`jours[i]` contienne le nombre entier de jours séparant la date de l'épreuve du 1er janvier 2000 (on utilisera la fonction `nb_jours` pour réaliser le calcul) ;**
- **et `temps[i]` contienne le temps en secondes, au format flottant.**

On ne demande pas de trier les résultats. On rappelle que le coureur doit pouvoir être identifié même si la casse de `nom` et/ou `prenom` ne correspond pas à celle qui est enregistrée dans `T`.

FIN DE L'EPREUVE

3. Annexe : documentation partielle

Vous trouverez ci-dessous un extrait de la documentation officielle du module `matplotlib.pyplot`, dédié au tracé et à la mise en forme de graphiques.

3.1. Fonction plot

```
matplotlib.pyplot.plot(*args, **kwargs)
```

Plot lines and/or markers to the Axes. *args* is a variable length argument, allowing for multiple *x*,*y* pairs with an optional format string. For example, each of the following is legal:

```
plot(x, y)         # plot x and y using default line style and color
plot(x, y, 'bo')   # plot x and y using blue circle markers
plot(y)            # plot y using x as index array 0..N-1
plot(y, 'r+')      # ditto, but with red plusses
```

An arbitrary number of *x*,*y*, *fmt* groups can be specified, as in :

```
a.plot(x1, y1, 'g^', x2, y2, 'g-')
```

character	description
`'-'`	solid line style
`'--'`	dashed line style
`'-.'`	dash-dot line style
`':'`	dotted line style
`'.'`	point marker
`','`	pixel marker
`'o'`	circle marker
`'v'`	triangle_down marker
`'^'`	triangle_up marker
`'<'`	triangle_left marker
`'>'`	triangle_right marker
`'1'`	tri_down marker

'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

The following color abbreviations are supported :

character	color
‘b’	blue
‘g’	green
‘r’	red
‘c’	cyan
‘m’	magenta
‘y’	yellow
‘k’	black
‘w’	white

3.2. Fonctions annexes pour la mise en forme

En plus de la fonction `plot`, le module `matplotlib.pyplot` propose diverses fonctions dédiées à la mise en forme des graphiques. En voici quelques-unes :

- `xlabel(s)` : écrit le contenu de la chaîne `s` comme étiquette des abscisses.
- `ylabel(s)` : écrit le contenu de la chaîne `s` comme étiquette des ordonnées.
- `title(s)` : écrit le contenu de la chaîne `s` comme titre du graphique.
- `legend(L)` : donne une légende au graphique. `L` doit être une liste de chaînes : `L[0]` est la légende de la première courbe, `L[1]` de la deuxième, etc.