DS n°1 d'informatique

Durée : 1h30

Le langage de programmation de cette épreuve est Python. Tous les programmes, à créer ou à compléter, peuvent être commentés, au sein du programme, au moyen du symbole #, ou entre triples guillemets """ """.

Le sujet présent comporte 3 pages. Les réponses doivent être rédigées sur copie et sur le Document Réponse prévu. Pensez à rendre ce document réponse avec votre copie !

Les exercices 2 et 3 seront rédigés sur des copies séparées.

Exercice 1 : Méthodes numériques

On propose d'étudier ici l'équation différentielle non linéaire suivante, modélisant la chute d'un corps ponctuel de masse *m*, entre les temps t_0 et t_f, soumis à un champ de pesanteur indépendant de l'altitude, ainsi qu'à une force de frottement proportionnelle au carré de sa vitesse.
On admet alors que, si l'origine est placée au sol, et si l'axe de chute de (*Oz*) est orienté vers le haut, alors l'altitude $z(t)$du corps au temps *t* vérifie la relation :

$$(E)\begin{cases} z(t_0) = z_0 \\ z'(t_0) = 0 \\ \forall t\ [t_0, t_f], \quad mz''(t) = C(z'(t))^2 - mg \end{cases}$$

où l'on prendra $g = 9{,}81\ m/s^2$, $m = 80\ kg$ et C un coefficient de frottement (en kg/m) dépendant de l'altitude et de la résistance du corps en chute libre.
Dans ce problème, on suppose que ***C* est constant** pour la durée de la chute.
Les scripts à programmer seront précédés des lignes suivantes :

```
import maplotlib.pyplot as pl
g,m = 9.81,80
```

1. **Approximation de la vitesse par le schéma numérique d'Euler explicite à <u>pas constant h</u>**

   Programmer une fonction **euler(t0,tf,z0,C,h)** ayant pour paramètres 5 réels positifs t_0, t_f, z_0, C et h, et un entier naturel *N,* et qui retourne le couple (T, V) des deux tableaux :
   - T : discrétisation à pas constant de l'intervalle des temps ;
   - V : liste des valeurs approchées de $z'(t)$, pour tout $t \in T$, obtenues par la méthode d'Euler explicite, sur l'intervalle $[t_0, t_f]$, pour le pas constant *h*.

2. **Approximation de l'altitude par la méthode de Simpson**

   On souhaite à présent déterminer une valeur approchée de l'altitude $z(t_f)$. On admet que :

$$z(t_f) \approx z_0 + \frac{h}{6}\left(\sum_{k=0}^{N-2} v_k + 4v_{k+1} + v_{k+2}\right)$$

   où $N \in \mathbb{N}^*$ et $V = [v_0, \ldots, v_N]$ est la liste renvoyée par la fonction du **1**.

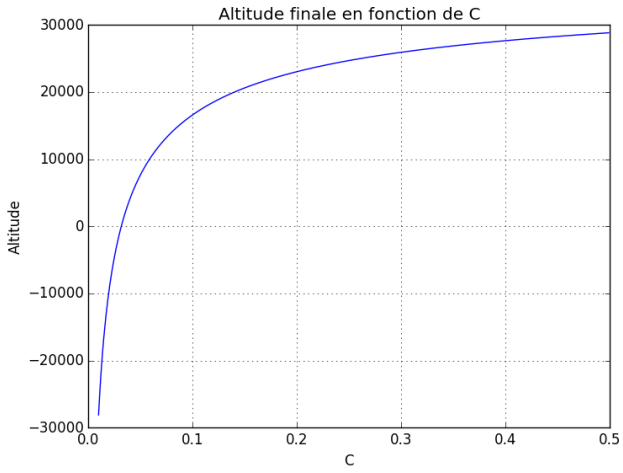
   Programmer alors une fonction **altitude(t0,tf,z0,C,h)** de paramètres t_0, t_f, z_0 C et h, et qui retourne une valeur approchée de $z(t_f)$ grâce à la méthode de Simpson.

3. ***Détermination du coefficient C pour un cas concret***

> *« Félix Baumgartner, parachutiste et sauteur extrême autrichien, s'est élancé, le 14 octobre 2012, d'un ballon à hélium, sans vitesse initiale, d'une altitude de 39 km.*
> *Sa chute libre a duré 4 minutes et 19 secondes. A cet instant, il était à 2500 m d'altitude et a ouvert son parachute*. »

a) Ecrire une fonction **altitudeFB(C)** de paramètre le coefficient de frottement *C,* et qui retourne l'altitude théorique de Félix Baumgartner, au moment où il ouvre son parachute (on prendra ici h = 1 seconde).

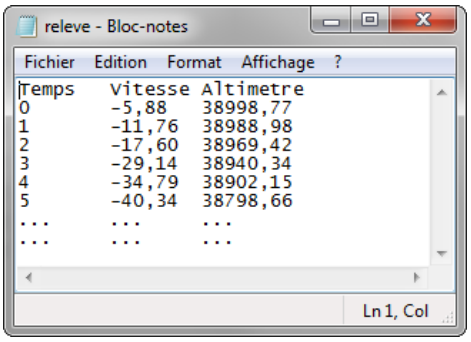
b) Ecrire des instructions permettant d'afficher le graphe suivant, représentant l'altitude **altitudeFB(C)** en fonction de **C** :



c) Ecrire une fonction **coefficient()** (sans paramètre) et qui retourne une valeur approchée du coefficient **C** lors de la chute libre de Felix Baumgartner.
On utilisera la méthode de dichotomie.

Afin de valider la valeur de C obtenue par le modèle, on décide de comparer l'évolution de l'altitude théorique au cours du temps pour C = 0,037 kg/m issue des équations précédentes avec les relevés de l'altimètre de Félix Baumgartner.

L'altimètre permet d'obtenir le fichier *releve.txt* ci-contre. Dans ce fichier brut de mesures : la première colonne correspond aux instants en secondes, la deuxième colonne à la vitesse en m/s et la troisième colonne à l'altitude en mètres. Le séparateur de ces colonnes est une tabulation (opérateur \t sous Python), le caractère de retour à la ligne est l'opérateur \n.



4. Fonction de formalisation

Chaque ligne de ce fichier est une chaîne de caractères de la forme :

'*temps* \t *vitesse* \t *altitude* \n'

Ecrire une fonction **donnees_ligne(chaine)** renvoyant la chaîne de caractère **chaine** dans laquelle :
- chaque caractère \t a été remplacé par une virgule (','),
- et chaque caractère \n de fin de chaîne a été supprimé.

5. Ecrire alors une suite d'instructions qui permet :
- d'ouvrir le fichier releve.txt en mode lecture sous une variable notée *f* ;
- de lire successivement les lignes utiles ;
- de construire 3 listes : ● **Ltemps** contenant les relevés des temps de la première colonne, sous forme d'entier.
  ● **Lvitesse** contenant les relevés des vitesses de la 2e colonne, sous forme de flottant.
  ● **Laltitude** contenant les relevés des altitudes de la 3e colonne, sous forme de flottant**.**

6. On veut comparer les altitudes de la liste **Laltitude** avec les altitudes obtenues dans la question **2.**

On définit alors en Python :

Z = [altitude(t0,t,z0,C,h) for t in Ltemps]

où les paramètres ont été remplacés par la données numériques relatives à la chute de Felix Baumgartner.

Ecrire alors une commande permettant de retourner l'erreur maximale que l'on commet en approchant les données de la liste **Laltitude** par les données de **Z.**

Exercice2: Piles

Notation Polonaise Inverse

L'objectif est d'écrire un programme pour évaluer des expressions écrites en notation polonaise inverse. Les expressions en NPI seront représentées par des tableaux contenant des entiers et des caractères. Par exemple, « 2 3 + 7×» sera représenté par le tableau [2, 3, '+', 7, '*']. Le calcul effectué sera le suivant : (2+3)*7 Principes du programme : une pile vide est créée ; le tableau est parcouru de gauche à droite et chaque nombre rencontré est empilé. Si l'élément rencontré est un opérateur, les deux opérandes sont désempilés et le résultat du calcul est empilé. 1. Nous utilisons une pile à capacité finie.

Un fichier nommé fichier_piles dans lequel sont écrites les fonctions creer_pile(c), empiler(p,x), depiler(p),pile_vide(p),sommet_pile(p), inverser_pile(p) vues en cours.

Les tableaux sont représentés par des objets de type liste.

1. Ecrire une fonction calc_npi(exp) qui prend en argument une liste notée exp. Nous ne considérons ici que les opérateurs "+" et "*". Le corps de cette fonction se compose ainsi :
   (a) création d'une pile p dont la capacité maximale est la longueur de l'expression arithmétique;
   (b) parcourt avec une boucle for des éléments de exp, de gauche à droite;
   (c) si l'élément courant est un opérateur (caractère '+' ou '*'), on dépile les deux opérandes x et y de p et on empile x + y ou x * y , selon la valeur de c : Sinon, c est un nombre et on l'empile dans p :
   (d) quand on sort de la boucle for , il ne reste plus qu'à dépiler la valeur finale res de l'expression
   (e) vérifier que la pile p est bien vide :

2. Ecrire le déroulement des différentes opérations de la fonction calc_npi pour vérifier le fonctionnement de votre fonction sur l'exemple suivant : calc_npi([3, 4, '+', 5, '*']).
3. Ecrire la ligne de code permettant d'afficher le résultat de calc_npi([3, 4, '+', 5, '*'])

Exercice3: Piles

Gestion d'un ascenseur

On souhaite gérer l'ascenseurs d'un hôpital de 5 ´étages (numérotes de 0 à 4). Les appels vont s'empiler de la façon suivante :

- lorsqu'un appel s'effectue du niveau 0 (niveau des urgences) il passe en priorité absolue sur l'ascenseur.
- si l'appel est sur le chemin de l'ascenseur en mouvement, il passe en priorité et l'ascenseur s'arrête à cet ´étage.
- dans les autres cas, chaque appel est enregistré à la 'queue'. Les appels se font via les touches 0 à 4 du clavier. Dans ce cas le déplacement se fera de manière classique (dernier appelé, dernier servi)

Les variables utiles sont :

- "appel" : variable globale de type entier contenant le numéro du dernier étage appelé.
- "pile" (variable globale) : vecteur comportant les appels ascenseur.
- "sens" (variable globale) : entier donnant le sens de déplacement de l'ascenseur (0 pas de déplacement, 1 montée, -1 descente).
- "position" (variable globale) : entier donnant la position de l'ascenseur.

On utilisera les au maximum les fonctions de « fichier_piles » de l'exercice précédent.

1. Ecrire une fonction gestion_pile(pile,appel) qui prend en argument l'appel de chaque étage et la pile « pile » déjà créée. Cette fonction modifie la pile en cours mais ne retourne rien. Une fonction « déplacement_ascenseur » (non étudiée) déplace l'ascenseur avec le sommet de « pile »