Informatique : cahier de vacances

Préambule

Pour les manipulations de fichiers, on rappelle les fonctions suivantes :

- `fichier=open(nom_fichier,'r')` : ouvre le fichier en mode lecture
- `fichier.close()` : ferme le fichier, après une ouverture avec `open`
- `ligne=fichier.readline()` : retourne une chaîne de caractère contenant la ligne suivante de `fichier`
- `s.split(sep)` : découpe la chaîne de caractère `s` selon le séparateur `sep`. Par exemple si `s` vaut `''42,21,3''`, alors `s.split('','')` renvoie la liste `['']42'','21'','3'']`
- `float(s)` : retourne un flottant correspondant à la valeur contenue dans la chaîne `s`.

Pour le tracé de graphiques et le module numpy, on se reportera utilement au résumé de cours "Tracé de graphes".

—Semaine 1 —

Ne pas oublier de se reporter au préambule

Lundi

I Ecrire un programme python qui renvoie l'incertitude élargie sur a dans la formule suivante $a = \left(\frac{GMT^2}{4\pi^2}\right)^{1/3}$, avec $T = 88\text{jour} \pm 40\text{mn}$, $G = (6,67430 \pm 10^{-5}).10^{-11}\text{m}^3\text{kg}^{-1}\text{s}^{-2}$ et $M = (1,989 \pm 0,001).10^{30}\text{kg}$

II Ecrire une fonction `prod_mat(M1:np.array,M2:np.array)->np.array` qui renvoie la matrice résultat du produit des matrices `M1` et `M2`.

Mardi

I Coder une fonction `max_rec(L:list)->float`, récursive, qui retourne le maximum d'une liste `L`

II On a codé une fonction prenant en entrée une date au format `(j,m)`, où `j` correspond au jour du mois, et `m` au numéro du mois (avec janvier correspondant à 1). Donner un partitionnement des tests d'entrée en précisant à chaque fois un exemple. *(MPSI*)* : préciser le comportement du programme dans chaque cas du partitionnement.

Mercredi

I Un transpondeur d'avion envoie des messages codés sur 128bits avec la structure suivante :

- chaque message commence par une marque de début de 6bits
- les 24 bits suivants sont utilisés pour le numéro d'identification de l'avion.
- chaque message se termine par 4bits de contrôle et une marque de fin de 6bits

Les autres informations sont des entiers codés en binaire :

- l'altitude, entre 2000 et 66000 pieds
- la vitesse ascensionnelle, entre -5000 à 5000 pieds/minute

Peut-on envoyer toutes ces informations en un message ?

II Coder une fonction `dichotomie(L:list,e:str)->int` qui recherche `e` par dichotomie dans une liste `L` a priori triée de façon croissante.

Jeudi

I On donne la fonction récursive suivante :

```
def fibo(n:int)->int:
        if n<=1:
            return n
        else:
            return fibo(n-1)+fibo(n-2)
```

Evaluer la complexité asymptotique de cette fonction. Commentaire.

II *(Maths)* : montrer par récurrence sur n que le graphe complet de taille n non orienté contient $\frac{(n-1)n}{2}$ arêtes.

Vendredi

I On considère un point dans l'espace décrit par ses trois coordonnées $(x, y, z) \in \mathbb{Z}^3$. Initialement, ce point est en $(0, 0, 0)$ et se déplace selon la règle suivante : à chaque étape, seule une composante varie, soit de $+1$, soit de -1, avec la même probabilité. En utilisant `randint` du module `random`, écrire un programme stockant la position du mobile initiale et après chacun de ses N déplacements dans trois liste `liste_x`, `liste_y`, `liste_z`. On pourra tracer la trajectoire à l'aide du code suivant :

```
from matplotlib import pyplot as plt
ax = plt.figure().add_subplot(projection='3d')
ax.plot(liste_x,liste_y,liste_z)
plt.show()
```

II Expliquer en l'appliquant à un exemple (simple) l'algorithme suivant :

```
def tri_bulle(L:list)->None:
    n=len(L)
    for i in range(n):
        for j in range(n-i-1):
            if L[j]>=L[j+1]:
                L[j],L[j+1]=L[j+1],L[j]
```

et en donner la complexité asymptotique en fonction de `n`.

—Semaine 2 —

Ne pas oublier de se reporter au préambule

Lundi

I Écrire un programme permettant d'afficher la représentation graphique de la solution sur $[0; 2, 5]$ de l'équation différentielle suivante :

$$\left(x + e^{f(x)}\right).\frac{df}{dx}(x) = 1$$

avec $f(0) = 0$. On prendra un pas de $0,0001$.

II Donner la série de `assert` qu'il faut placer en début de la fonction du II, semaine 1, lundi pour vérifier que les deux matrices sont bien carrées et de même taille.

Mardi

I Écrire une fonction `max3(L:list)->list` qui renvoie les valeurs des 3 plus grands nombres distincts d'une liste. *(MPSI*)* : le faire en un seul parcours de la liste.

II Quelle ligne faut-il écrire pour importer les fonctions `cos` et `sqrt` ainsi que les variables `pi` et `e` ?

Mercredi

I On considère la fonction $f : \mathbb{R}_+ \to \mathbb{R}; x \mapsto x - e^x + 3$. Montrer rapidement que :

- cette fonction est strictement décroissante sur $\mathbb{R}_+$
- cette fonction admet un zéro sur $\mathbb{R}_+$

En déduire et coder une méthode pour déterminer la valeur de ce zéro à 10^{-7} près. On utilisera la fonction `exp` du module `math`

II Donner la valeur du plus grand nombre entier signé que l'on puisse représenter sur 16bits, puis son ordre de grandeur en puissance de 10.

Jeudi

I Coder une fonction `test_tri(L:list)->bool` récursive qui vérifie si une liste de nombres `L` est bien triée dans l'ordre croissant.

II Utiliser python pour trouver le résultat de $\sum_{k=6}^{570} \frac{k+1}{k-5}$

Vendredi

I Coder et utiliser le tri rapide pour ordonner la liste fournie de façon croissante et en trouver le 437ème élément une fois la liste ordonnée.

II On fournit une structure de type file de priorité, contenant des chaines de caractères, de type `Fp` avec les fonctions suivantes :

- `pile_vide()->Fp` : crée une file de priorité
- `pop(F:Fp)->(str,int)` : enlève un élément de la file ayant la priorité la plus basse et le retourne, avec sa priorité
- `top(F:Fp)->(str,int)` : retourne l'élément ayant la priorité la plus basse, avec sa priorité
- `insert(F:Fp,e:str,p:int->None` : insère dans la file l'élément `e` avec la priorité `p`
- `est_vide(F:Fp)->bool` : retourne `True` si la file est vide, `False` sinon.

On dispose également d'un graphe `G`, de type `graphe` **non pondéré**, et de la fonction `voisin(G:graphe,s:str)->list[str]` retournant la liste des voisins de `s`, définis comme les sommets accessibles en un seul arc/une seule arête depuis `s`.

Recoder entièrement l'algorithme de Dijkstra pour trouver le plus court chemin d'un sommet `s0` à un sommet `sf` de `G`, sous la forme d'une fonction `dijkstra(G:graphe,s0:str,sf:str->list[str],int`, qui retourne le chemin ainsi que son poids, défini ici comme le nombre d'arêtes qu'il faut utiliser pour aller de `s0` à `sf`.

—Semaine 3 —

Ne pas oublier de se reporter au préambule

Lundi

I Écrire une fonction `stats(L)` qui retourne la moyenne, la médiane et l'écart-type d'une liste de nombres `L`, sous la forme d'un dictionnaire.

II Ecrire un programme python qui retourne un tableau d'entiers bidimensionnel $N \times N$ rempli de zéros :

- en utilisant `numpy`
- sans utiliser `numpy`

puis utiliser `randint` du module `random` pour passer une case aléatoire de ce tableau à 1.

Mardi

I On considère l'algorithme suivant :

```
def truc(x,n):
    if n==0:
        return 1
    elif n==1:
        return x
    else:
        if n%2==0:
            return truc(x**2,n//2)
        else:
            return x*truc(x**2,(n-1)//2)
```

Prouver la finitude de cet algorithme.

II Simulation d'une martingale

On considère un joueur qui s'essaie à la martingale à la roulette. A chaque tour :

- si il gagne, il s'arrête de jouer
- sinon, il recommence en doublant sa mise

A la roulette, on a une probabilité de gagner de $\frac{18}{37}$, et si on gagne on récupère le double de sa mise. On commence le programme de la façon suivante :

```
from random import random
def martingale():
    mise=1
    gain=0
    while True:
        gain-=...
        if ...:
            return ...
        else:
            mise*=...
```

Compléter le programme

Mercredi

I On s'intéresse à un graphe orienté et non pondéré, représenté par sa matrice d'adjacence `G:np.array`, et dont les n sommets sont numérotés de 0 à $n-1$. Coder les fonctions suivantes :

- `degrep(G:np.array,s:int)->int` : donne le degré sortant de `s`
- `degrem(G:np.array,s:int)->int` : donne le degré entrant de `s`
- `voisins(G:np.array,s:int->list[int]` : retourne la liste des voisins de `s`
- `nb_boucles(G:np.array)->int` : retourne le nombre de boucles dans `G`

II On s'intéresse à une liste de listes `L` dont chaque ligne est une liste à 3 éléments, décrivant un point par sa longitude, latitude et altitude, dans cet ordre. Cette liste de points décrit une randonnée, et on définit le *dénivelé positif* par la somme des différences d'altitudes entre deux points successifs lorsque cette différence est positive, et le *dénivelé négatif* la valeur absolue de la somme des différences d'altitudes lorsque cette différence est négative.

Coder une fonction `deniveles(L)` qui retourne un tuple contenant le dénivelé positif et le dénivelé négatif de la randonnée.

Jeudi

I Ecrire un programme python permettant de calculer $\int_{-1}^{1} \frac{\sin(x)}{x} dx$ par méthode des rectangles médians. On trouvera les contournement mathématiques nécessaires pour 0.

II Donner la valeur en base 10 du nombre signé représenté sur un octet par

1	0	1	0	0	1	1	0

Vendredi

I On considère un fichier `suivi_rando.csv` qui contient :

- en première ligne le nom des colonnes : `lat,long,height,time`
- pour les lignes d'après la latitude en degré, longitude en degré, altitude en mètre et temps en secondes, par exemple `45.461516,6.44461,1315.221,1597496965`

Coder une fonction `importe(nom_fichier)`, qui retourne une liste de listes de quatre flottants (latitude, longitude, altitude, temps).

II On considère un graphe non pondéré et non orienté représenté par sa matrice d'adjacence `G`, à n sommets numérotés de 0 à $n-1$. On note $G^(K) = \prod_{k=1}^{K} G$, et $G_{i,j}^K$ le coefficient i, j de la matrice G^K

II.1 Que représente la valeur $G_{i,j}$?

II.2 Démontrer la proposition suivante : $G_{i,j}^2 \neq 0 \Leftrightarrow$ il existe un chemin de taille longueur 2 du sommet i au sommet j.

II.3 Généraliser sans démonstration la proposition précédente à $G_{i,j}^K$

II.4 Que représente alors, pour un i donné, $G_{i,i}^K$

On suppose codée la fonction II, semaine 1, lundi.

II.5 Quelle est la longueur maximale d'un cycle dans G ?

II.6 En déduire une fonction `detecte_cycle(G:np.array)->int` qui renvoie le plus petit cycle existant dans G, et -1 si il n'y en a pas.

—Semaine 4 —

Ne pas oublier de se reporter au préambule

Lundi

I On veut résoudre par la méthode d'Euler explicite l'équation différentielle d'inconnue y et de variable x, pour $x \in [0, 10[$ avec 5000 points :

$$\frac{dy}{dx} = 5(2-y)^2$$

Écrire le code correspondant, avec $y(0) = 0$, et tracer y en fonction de x.

II On définit la trace d'une matrice carrée M par la somme des coefficients de sa diagonale. Coder une fonction `trace(M)` retournant la trace de la matrice :

- de façon itérative
- de façon récursive

Mardi

I Ecrire un programme qui renvoie la liste de tous les vendredi 13 d'aujourd'hui au 31 décembre 2099, sous la forme d'une fonction `vendredi13(j:int,d:int,m:int,a:int)->list[str]`, où `j` représente le numéro d'aujourd'hui dans la semaine (0 pour lundi et 6 pour dimanche), `d` le numéro d'aujourd'hui dans le mois, `m` le numéro du mois (1 pour janvier et 12 pour décembre), et `a` le numéro de l'année actuelle. On fournit les lignes suivantes :

```
mois=[31,28,31,30,31,30,31,31,30,31,30,31]
if a%4==0 and a%100!=0) or a%400==0:
    mois[1]=29
else:
    mois[1]=28
```

qu'on essaiera de comprendre au préalable. Les éléments de la liste de retour seront au format `str(d)+"/"+str(m)+"/"+str(a)` (et devront logiquement tous commencer par 13).

Les seuls arguments d'entrée possibles sont les éléments de la date actuelle (par exemple à l'heure où l'auteur écrit ces lignes : `vendredi12(1,24,6,2025)`. On informe pour vérification que le 13 novembre 2099 sera un vendredi...

II On dispose d'une liste de tâches `L`, utilisant la même ressource. Chaque tâche est caractérisée par un tuple à trois éléments : son numéro, entier positif, son moment de début, entier et son moment de fin, entier. La liste `L` peut par exemple commencer par :

```
[(34,253,460),(4,324,6324),(12,1234,4501)...]
```

On cherche dans un premier temps à trier cette liste, en place, par **moment de fin croissant**. On rappelle l'algorithme du tri par insertion sur une liste d'entiers :

```
def tri_insertion(l:list[int])->None:
    for i in range(1,len(l)):
        e=l[i]
        j=i
        while l[j-1]>e and j>0:
            l[j]=l[j-1]
            j-=1
        l[j]=e
```

Adapter cet algorithme pour qu'il trie `L` comme demandé.

Mercredi

I On dispose d'un graphe orienté pondéré représenté par la liste `A` de ses arêtes, entre des sommets numérotés de `0` à `n-1`. Chaque arête est un tuple de la forme `i,j,p`, où `i` est le sommet de départ, `j` le sommet d'arrivée et `p` le poids de l'arête `i,j`. Ecrire un programme qui, connaissant `A` et `n`, retourne la matrice d'adjacence associée à ce graphe. On commencera comme suit :

```
def Madj(A:list[tuple[int]],n:int):
    M=np.zeros((n,n),dtype=int)
    ...
```

II On dispose d'un fichier `scores.csv` contenant N lignes, chaque ligne contenant, séparés par des virgules, le numéro d'un participant à une course et le temps qu'il a mis pour terminer cette course. Ecrire les instructions pour obtenir une liste de liste, nommée `l_score` contenant ces deux éléments (i.e. un tableau de $N \times 2$). Le fichier ne contient pas de ligne de titre.

Jeudi

I *Rappel des bases de la norme IEEE 754, simple précision : on code sur 32 bits avec 1 bit de signe s, 8 bits d'exposants e et 23 bits de mantisse ma. Le décalage d de l'exposant est de $2^{8-1}-1 = 127$.*

L'exposant est un nombre non signé, entre 0 et $2^8 - 1$, puis décalé pour les exposants négatifs. La mantisse est calculée par $ma = \sum_{i=1}^{23} b_i * 2^{-i}$ où i est la place du bit ($i = 1$ pour le bit de poids fort de ma.

On distingue trois types de représentations :

- *Les nombres normalisés, où l'exposant est différent de 0 et de 255, dont la valeur vaut $v = (-1)^s.m.2^{e-d}$ avec $m = ma + 1$*
- *Les nombres où l'exposant vaut 0, dont la valeur vaut $v = (-1)^s.ma.2^{-126}$ (ce sont les plus petits nombres en valeur absolue)*
- *Les nombres où l'exposant vaut 255, dont la valeur vaut ∞ si la mantisse vaut 0, NaN sinon.*

Les nombres des deux dernières catégories sont dits *dénormalisés.*

Donner la valeur représentée par 1 00010011 00110010110001000001001

II On repart du fichier `l_score` de l'exercice II de mercredi. Ecrire un programme qui détermine le minimum et le maximum des temps réalisés, puis qui trace un histogramme du nombre de participants dans un intervalle de temps donné, divisé en déciles.

Vendredi

I On repart de la liste de tâches de l'exercice II de mardi, que l'on suppose triée par moment de fin croissant. Si on appelle d_i et f_i les moments de début et de fin de la tâche i, on dit que deux tâches i et j distinctes ($i \neq j$) sont compatibles ssi $[d_i, fi[\bigcap[d_j, f_j[$ (on remarquera le caractère semi-ouvert des intervalles). On cherche alors un ordonnancement des tâches, c'est-à-dire une liste `Lo`, sous-liste de `L`, contenant le plus grand nombre de tâches compatibles. On utilise alors l'algorithme suivant :

- partant de `L` triée par moment de fin croissant, et de `i=0`
- tant que `i` désigne une tâche de `L`
- si `L[i]` est compatible avec la dernière tâche dans `Lo` ou si `Lo` est vide, on ajoute `L[i]` à `Lo`
- on incrémente `i` de 1

Coder cet algorithme et expliquer pourquoi il s'agit d'un algorithme glouton.

II Résoudre par dichotomie l'équation $\frac{\sin x}{x} = \frac{1}{2}$, sur $]0, \pi[$ à 10^{-4} près

—Semaine 5 —

Ne pas oublier de se reporter au préambule

Lundi

I "Retrouver" numériquement par méthode des rectangles médians que $\lim_{x\to+\infty} \int_{-x}^{x} e^{-y^2} dy = \sqrt{\pi}$. Jusqu'à quelle valeur de x faut-il aller pour avoir une différence entre les deux membres inférieure à 10^{-9} ?

II A l'issue d'une simulation numérique, on dispose d'un tableau `numpy`, appelé `resultats`, de N lignes et de 4 colonnes, contenant respectivement pour divers instants t les valeurs de $x(t)$, $\frac{dx}{dt}(t)$, $y(t)$ et $\frac{dy}{dt}(t)$, où x et y sont les coordonnées cartésiennes classiques. Ecrire un programme qui trace la trajectoire, c'est-à-dire y en fonction de x.

Mardi

I Ecrire une fonction `liste_adj(M:np.array)->list`, retournant à partir de la matrice d'adjacence d'un graphe pondéré la liste d'adjacence de ce graphe, telle que `L[i]` contienne la liste des voisins de `i`, qui sera une liste de tuples `(j,p)`, où `j` est le voisin et `p` le poids de l'arête.

II Soit une matrice M carrée de taille N, dont les lignes sont numérotées par i et les colonnes par j, $(i,j) \in [\![0, N-1]\!]^2$. $m_{i,j}$ désigne alors l'élément de M situé ligne i, colonne j. On appelle $M^{(i)}$ la sous matrice de M, de taille $N-1$, obtenue en supprimant la colonne 0 et la ligne i de M. On rappelle alors que le déterminant *det* de M est égal à :

- $M_{0,0}$ si $N = 1$
- $\sum_{i=0}^{N-1} (-1)^i m_{i,0} det\left((M^{(i)}\right)$

Coder une fonction `det(M)`, récursive, calculant le déterminant de M selon l'algorithme précédent.

Mercredi

I Donner la valeur du plus grand nombre dénormalisé, en puissance de 2, puis approximativement en puissance de 10, suivant la norme IEEE 74 sur 32 bits (voir semaine précédente)

II Démontrer la finitude et évaluer la complexité temporelle de l'algorithme de calcul du déterminant d'hier.

Jeudi

I On définit l'objet `pile`, noté `P` comme un objet informatique régi par les règles suivantes :

- il existe une pile vide, et la fonction pour la créer est `pile_vide()`.
- il existe une fonction `empile` : `P.empile(a)` rajoute l'élément `a` à la pile
- il existe une fonction `depile` : `P.depile()` retire le dernier élément ajouté à la pile et en retourne la valeur. Cette fonction lève une exception si la pile est vide.

- il existe une fonction `est_vide` : `P.est_vide()` retourne un booléen qui vaut `True` si la pile est vide, `False` sinon.

Dans toute la suite, seules ces quatre fonctions peuvent être utilisées. On remarquera que `empile` et `depile` fonctionnent par effet de bord.

I.1 Que contiennent `P` et `a` à la fin de ces lignes :

```
P=pile_vide()
P.empile(4)
P.empile(2)
P.empile(5)
P.depile()
a=P.depile()
```

I.2 Coder une fonction `top`, telle que `top(P)` renvoie le dernier élément ajouté à la pile sans modifier la pile.

I.3 Coder une fonction `taille`, qui renvoie le nombre d'éléments dans la pile sans, in fine, l'avoir modifié.

II Utiliser l'exercice précédent pour recoder un parcours en profondeur d'un graphe `G`. On dispose de la fonction `voisin(G:graphe,s:str)->list[str]` retournant la liste des voisins de `s`, définis comme les sommets accessibles en un seul arc/une seule arête depuis `s`.

Vendredi

I On cherche à résoudre par méthode d'Euler explicite le système d'équations différentielles suivant :

$$\begin{cases} \frac{d^2u}{dt^2} &= 4\pi\frac{dv}{dt} + 4\pi^2 Au \\ \frac{d^2v}{dt^2} &= -4\pi\frac{du}{dt} + 4\pi^2 Bv \\ \frac{d^2w}{dt^2} &= -4\pi^2 Cw \end{cases}$$

où A, B et C sont des constantes. On note h le pas, supposé constant, de la méthode.

I.1 Ecrire, en introduisant les fonctions $u_p = \frac{du}{dt}$, $v_p = \frac{dv}{dt}$ et $w_p = \frac{dw}{dt}$ un système d'équations différentielles du premier ordre à 6 équations.

I.2 On commence le programme de la façon suivante :

```
import numpy as np
pi=np.pi
def resol(init,tau_f,N):
    '''
    init : tableau contenant les valeurs unitiales de u,v,w,up,vp,wp
    tau_f : flottant ; on résout sur [0,tau_f]
    N : nombre de points de calcul
    '''
    global A,B,C #coefficients définis dans l'espace global
    h=... #pas temporel
    tau=np.arange(0,tau_f,h)
    u=np.zeros(N)
    v=np.zeros(N)
    w=np.zeros(N)
    up=np.zeros(N)
```

```
    vp=np.zeros(N)
    wp=np.zeros(N)
    u[0],v[0],w[0],up[0],vp[0],wp[0]=...

    ----
    return tau,u,v,w,up,vp,wp
```

Compléter les ... et les ----