Anti-sèche

Importations

- `import matplotlib.pyplot as plt`
- `import numpy as np`
- `import random as rd`

Raccourcis

- `f` : Une fonction.
- `u0,u1` : 1ers termes de (u_n)
- `n` : entier

I Suites numériques

A) Suites récurrentes

$u_{n+1} = f(u_n)$, u_0 : premier terme

Calcul du terme de rang n.

```
  u=u0
for i in range(n):
    u=f(u)
print(u)
```

Liste des termes $(u_0, \dots, u_n)$

```
L=[u0]
for i in range(n):
    L.append(f(L[-1]))
```

B) SRL$_2$

$\forall n \geq 0 \quad u_{n+2} + au_{n+1} + bu_n = 0$

```
u,v = u0,u1
for i in range(n-1):
   w = -a*v-b*u
   u,v=v,w
print(v)
```

C) Somme

$$S_n = \sum_{k=1}^{n} \frac{1}{k}$$

```
S = 0
for k in range(1,n+1):
    S += 1/k
```

II Fonctions réelles

A) Définition

$f : x \mapsto x^2 - 2$

- Défintion par une lambda-expression :

```
f = lambda x :x**2-2
```

Revient au même que :

```
def f(x):
    return x**2-2
```

B) Méthode de dichotomie

Voir fiche Révision 10 IV) B).

C) Méthode de Newton

Voir fiche Révision 10 IV) C).

D) Méthode d'Euler

Voir fiche Révision 14 IV) B).

E) Méthode des rectangles - Sommes de Riemann

Voir fiche Révision 13 V) B).

III Listes

A) `range, arange, linspace`

a. `range(a,b)` contient `b-a` éléments (donc fera `b-a` passages de boucle)

b. `range(n)` siginifie `range(0,n)`.

Liste de **n points** régulièrement répartis sur l'intervalle `[a,b]` (il y a donc **n − 1 intervalles**)

```
h = (b-a)/(n-1) #
X=[a+k*h for k in range(n)]
# ou
X = np.linspace(a,b,n)
# ou
X = np.arange(a,b+h,h)
```

```
 h = 1E-4 #  signifie 10**(-4)
```

B) Ajout - Suppression

```
L.append(5) # ajoute au bout de L
L.pop()     # supprime le dernier
            # terme de L
L.pop(4)    # supprime le terme
            # d'indice 4
L1+L2       # concaténation de
            # deux listes
```

C) Insertion

- Insertion en position n° 5 d'un item `truc` dans la liste `L` :

```
L = L[:5]+[truc]+L[5:]
```

D) Slicing

```
len(L) # nb de termes de L
L[-1] # dernier terme
L[-2] # avant-dernier etc.
L[:3] # Tous jq'au no 3 **exclu**
L[4:] # À partir du no4 **inclus**
L[a:b:r] # termes d'indices
         # dans [a,b[ de
         # r en r (a<b, r négatif
         # possible)
L[::-1] # renversement de la liste
```

E) Réplication

- Construction de `L=[1,1,1,0,0]`

```
L = 3*[1]+2*[0]
```

F) Listes en compréhension

- Liste des x^2 pour $x \in \{0, \dots 5\}$:

```
  L = [ x**2  for x in range(6)]
```

- Liste des termes de L qui sont < 20 :

```
  M = [ a for a in L if a<20]
```

- Liste de listes (liste de 4 listes de 3 termes) :

```
  [[i+j for j in range(3)] for i
      in range(4)]
# On obtient :
[[0, 1, 2], [1, 2, 3], [2, 3, 4],
    [3, 4, 5]]
```

G) Copie de listes

- ☠ Attention : `M = L` ne crée pas une copie de la liste L. Il faut pour utiliser `M = L[:]` ou `M = L.copy()`.
- Si la liste contient elle-même des listes, il faut faire une copie en profondeur avec la fonction `deepcopy()` du module `copy`

Lycée Chateaubriand
Classe de B_2^B
MY Patel
2018-2019

IV Chaînes de caractères

☠ Les chaînes de caractères ne sont pas modifiables.

Les items d'une chaîne de caractères sont indicés comme ceux des listes.

A) Slicing

```
c = "Je suis en biospé"
c[2:5] # ' su'
c[-1]  # dernier caract.
c[::-1] # chaine renversee
```

B) Scinder une chaîne

```
In [1]: 'je suis python'.split()
Out[1]: ['je', 'suis', 'python]
```

```
In [2]: 'porte-manteau'.split('-')
Out[2]: ['porte', 'manteau']
```

C) Liste de chaînes

```
In [3]: '*'.join(['a','b','c'])
Out[3]: 'a*b*c'
```

V Conversion de types

Chaine de caractères en liste :

```
In [4]: list("abc")
Out[4]: ['a','b','c']
```

Entier en chaine de caractères :

```
In [5]: str(123)
Out[5]: '123'
```

Chaine de caractères en entier :

```
In [6]: int('101')
Out[6]: 101
```

Booléen en entier

```
In [7]: True+0
Out[7]: 1
```

VI Matrices

A) Construction basique

Une matrice est une liste de lignes. On entre la liste précédée du mot-clé `array` :

a. $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$

```
A = np.array([[1,2],[3,4]])
```

b. Matrice ligne $L = (1, 2, 1)$

```
L = np.array([[1,2,1]])
```

c. Matrice colonne $C = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

```
C = np.array([[0],[0]])
```

B) Fonctions et opérations

```
A.shape() # tuple de la taille :
          # 2,2 ici.
A.reshape(4,1) # Redimensionne

L = np.arange(1,5,0.1)

B = A.copy() # Copie de A

A.T # Transposition
np.dot(A,A) # Produit matriciel
```

C) Extraction - Affectation

Dans les tableaux numpy, les indices commencent à `0`

- Modifier un coeff dans une matrice :

```
A[3,4] = 12
```

- Récupérer la sous-matrice de A pour $i = 1, 2, 3$ et $j = 3, 4$

```
M =  A[1:4,3:5]
```

- Mettre des `0` en position `(0,1),(1,1),(3,2)` :

```
I = (0,1,3) # les i's
J = (1,1,2) # les j's

A[I,J] = 0
```

- Incruster une matrice `M` de taille `(n,p)` dans `A` depuis le coin supérieur gauche `(i0,j0)` :

```
A[i0:i0+n,j0:j0+p] = M
```

D) Autres opérations

```
A*A # Produit élem par élem.

A.any() # vrai ssi un coeff
        # au moins est non nul

A.all() # vrai ssi tous les coeffs
        # sont non nuls

np.where(A>0) # Où se trouvent les
              # Coeffs > 0 de A
```

E) Matrices remarquables

```
np.zeros((2,3)) # Matrice nulle
np.ones((4,4))  # Matrice Atilla
np.diag([1,2,3]) # Matrice Diago
np.eye(3) # Matrice I3
np.zeros_like(A) # nulle de meme
                 # taille que A
np.ones_like(A)
```

VII Probabilités

A) Variables aléatoires

Voir les fiches consacrées.

B) Fonctions de base

Entier au hasard dans `[a,b]` (a, b inclus !)

```
k = rd.randint(a,b)
```

Entier au hasard dans `range(a,b)` (donc b exclu !)

```
k = rd.randrange(a,b)
```

Flottant au hasrad dans `]0,1[`

```
r = rd.rand()
```

Booléen valant `True` avec probabilité p :

```
p= 0.37
rd.rand()<p
```

VIII Tri

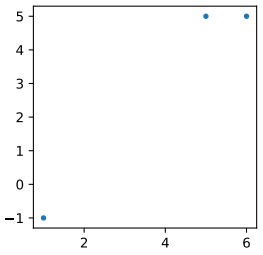
Voir fiche consacrée

IX Graphiques

A) Graphique simple

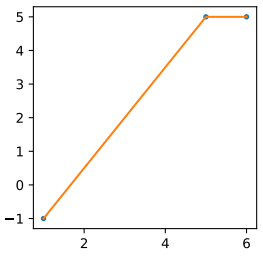
- Tracé d'une liste de points non reliés :

```
ABS = [1,5,6] #liste de 3 abscisses
ORD = [-1,5,5]
plt.plot(ABS,ORD,'.')
plt.show()
```



- Tracé de la courbe de f sur 3 points :

```
ABS = [1,5,6]
ORD = [f(x) for x in ABS]
plt.plot(ABS,ORD) # points reliés
plt.savefig('image.jpeg') # sauvegarde
```



- Tracé de la courbe de f sur un intervalle $[a, b]$

a. Si la fonction f a été construite à partir de fonctions du module numpy, le tracé sur un intervalle $[a, b]$ peut être obtenu de la manière suivante :

```
X = np.linspace(a,b) # cf. IIIA)
Y = f(X)
plt.plot(X,Y)
```

b. Sinon, on peut programmer les listes des abscisses et des ordonnées en compréhension. Ici, liste des abscisses de la forme $[0, 0.01, 0.02, \dots, 0.99]$ (100 termes !)

```
X = [ 0.01*k  for k in range (100)]
Y = [f(x) for x in X]
plt.plot(X,Y)
```

B) Graphique multi-fenêtre (mosaïque)

a. On peut utiliser range dans un plot (list(range) est inutile).

b. Exemple de graphique avec sous-graphiques :

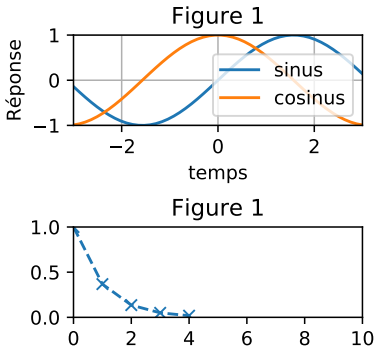
- Premier sous-graphique : tracé des fonctions sin et cos sur $I = [-3, 3]$.
- Second sous-graphique : tracé des termes $u_0, \dots, u_4$ où $u_n = e^{-n}$

```
# Points à tracer  fonctions
I  = np.linspace(-3,3)
Y1 = np.cos(I)
Y2 = np.sin(I)
# Points à tracer suites
N = np.arange(5)
U = np.exp(-N)
# Fenêtre graphique :
plt.figure(figsize=(3,3))  #Taille
# 1er sous-graphique
ax1 = plt.subplot(211) # cf. Rem
ax1.set_title('Figure 1') #Titre
ax1.set_xlim(-3,3) # axe des x
ax1.set_ylim(-1,1) # axe  des y
ax1.set_xlabel('temps')# noms des
ax1.set_ylabel('Réponse') #  axes

ax1.plot(I,Y1,label='sinus')
ax1.plot(I,Y2,label='cosinus')
ax1.grid('on') # Grille
ax1.legend(loc='best') # cf. Rem

# 2d sous-graphique
ax2 = plt.subplot(212)
ax2.set_title('Figure 2')
ax2.set_xlim(0,10) # Bornes
ax2.set_ylim(0,1)

ax2.plot(N,U),'--',marker='x')
plt.tight_layout() # cf. Rem
plt.show()
plt.savefig('image.pdf')
```



Rem.1

a. **Ligne 11 : `subplot(xyz)`** : crée une grille de graphiques contenant x×y sous-fenêtres et le tracé courant se fait dans la numéro z (numérotées de haut en bas puis de gauche à droite).

b. **Ligne 21 : `loc='best'`** : place la légende au mieux sur le graphique.

c. **Ligne 31 : `tight_layout`** : espace au mieux les graphiques pour éviter le chevauchement des légendes.

X Fichiers

```
with open("nom-fichier",mode) as leFichier:
    for ligne in leFichier:
        print(ligne)
```

mode prend une des trois valeurs suivantes que vous devez fixer :

a. `'w'` (write, écriture) : le fichier est créé si il n'existe pas, sinon effacé et réécrit par dessus.

b. `'a'` (append, ajout) : le fichier existe déjà, vous pouvez écrire à la suite de ce qui y existe.

c. `'r'` (read, lecture) : vous ne pouvez que lire le contenu du fichier, sans le modifier.

XI Remarques diverses

a. **`return`** : sans `return`, ce qui est calculé par une fonction ne peut pas être récupéré par l'utilisateur. Mettre un `print` ne fait qu'un affichage, donc ne sert pas à utiliser ce qui est calculé par la fonction.

b. **Boucles :** Si on doit répéter une action, son usage est obligatoire. Quelle boucle choisir ?

1) Si le nombre de répétitions est connu : **for**. Ex : calculer les 30 premiers termes d'une suite.

2) Si le nombre **maximal** de répétitions est connu : **for avec interruption** de boucle. Exemple : trouver le premier 1 dans une liste.

3) Si le nombre de répétitions est inconnu : **while**. Ex : calculer le rang du premier terme d'une suite dépassant `1000`.

c. Si la condition `C` de boucle contient des variables, ces dernières doivent être **recalculées avant la fin de la boucle**, sinon `C` garde la même valeur. Dans ce cas, soit on n'entre jamais dans la boucle (si `C` vaut `False`) soit on n'en sort jamais (si `C` vaut `True`).

d. **`while`** :

— Il est plus facile de commencer par écrire la **condition `S` de sortie de boucle**, puis d'écrire la condition `C` du `while` en prenant la **négation de `S`**.