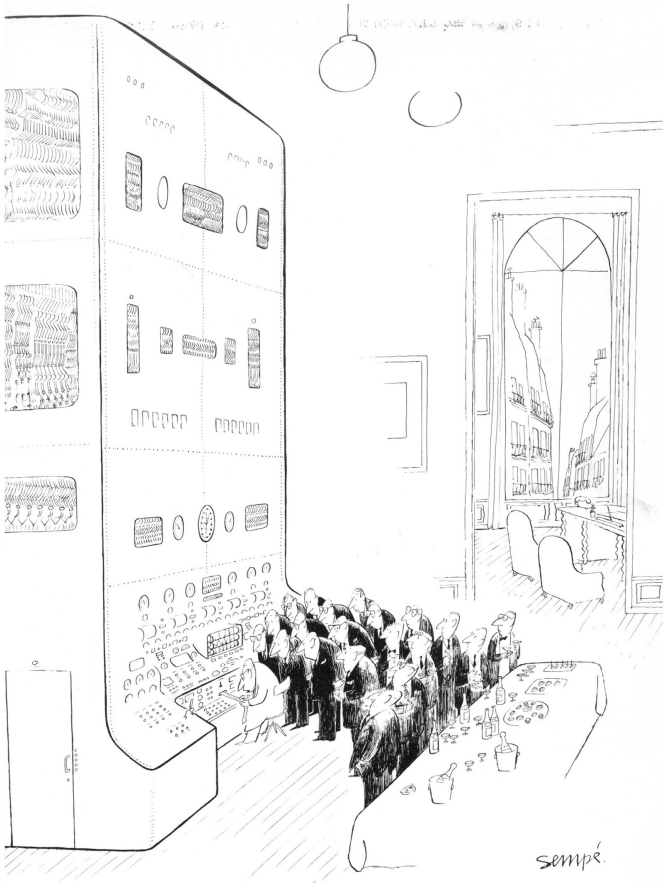
TP Info : Suites récurrentes, Listes



"Je suis une calculatrice I.B.M. N° X 124. On a mis trois ans cinq mois quatorze jours à me construire, six mois vingt-sept jours dix-sept heures à me monter ici. Malheureusement, demain à six heures vingt-deux, la baraque va s'effondrer..."

1 Suites récurrentes

Le but de cette partie est de développer des outils qui permettent d'étudier expérimentalement les suites définies par une relation de récurrence. Dans toute la suite, on se donne donc une fonction f et un réel α. La suite (u_n) est définie par :

$$u_0 = \alpha \quad \text{et} \quad \forall n \in \mathbb{N} \quad u_{n+1} = f(u_n)$$

1.1 Introduction à Matplotlib

La bibliothèque *matplotlib* permet d'afficher des graphes. Pour l'utiliser, on utilise la commande suivante :

```
import matplotlib.pyplot as plt
```

Si $x = [x_0, x_1, \dots, x_{n-1}]$ et $y = [y_0, y_1, \dots, y_{n-1}]$ sont deux listes de nombres, la commande

```
plt.plot(x, y)
```

permet de tracer une ligne brisée joignant les points de coordonnées $(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})$. Cette commande accepte des options permettant de changer la couleur la ligne. Par exemple, la commande

```
plt.plot(x, y, color='blue')
```

permettra de tracer une ligne bleue. Remarquons que conformément aux règles PEP8 et contrairement à ce qui se passe avec les affectations, il n'y a pas d'espace autour du signe « = » pour spécifier les options. Si vous souhaitez tracer une ligne en pointillées, vous pouvez utiliser la commande :

```
plt.plot(x, y, '--')
```

Si vous souhaitez tracer plusieurs lignes sur le même graphe, il suffit de placer plusieurs commandes les unes à la suite des autres.

```
plt.plot(x0, y0, color='blue')
plt.plot(x1, y1, color='red')
```

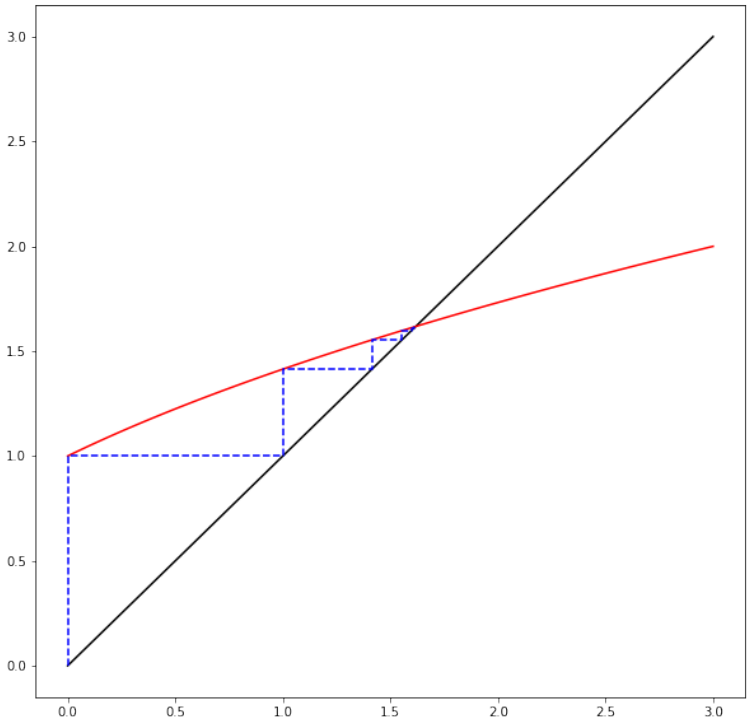
Afin de changer la taille d'affichage du graphe, on utilisera la commande

```
plt.rcParams['figure.figsize'] = [10, 10]
```

1.2 Graphe de l'escalier

1. Écrire une fonction `u(f, alpha, n)` qui renvoie le terme u_n.
2. Écrire une fonction `listes_graphe(f, x_min, x_max, m)` qui renvoie le tuple formé des listes
$$x = [x_0, \dots, x_m] \quad \text{et} \quad y = [f(x_0), \dots, f(x_m)]$$
où $x_k = x_{\min} + (k/m)(x_{\max} - x_{\min})$ pour tout $k \in [\![0, m]\!]$.
3. Après avoir importé la bibliothèque *Numpy*, tracer le graphe de la fonction sinus sur l'intervalle $[0, \pi/2]$.
4. Écrire une fonction `listes_escalier(f, alpha, n)` qui renvoie le tuple formé des listes de $2n+1$ éléments
$$x = [u_0, u_0, u_1, u_1, u_2, \dots, u_{n-1}, u_n] \quad \text{et} \quad y = [0, u_1, u_1, u_2, u_2, \dots, u_n, u_n]$$
Cette fonction doit avoir une complexité linéaire en n.

5. Écrire une fonction `escalier(f, alpha, n, x_min, x_max, m)` qui affiche le graphe de la fonction f sur $[x_{\min}, x_{\max}]$ avec une discretisation utilisant $m+1$ points, le graphe de la première bissectrice ainsi que les n premières marches de l'escalier de la suite (u_n). On devra obtenir un graphe comme celui-ci.



1.3 Étude de quelques cas

1. Tracer les escaliers pour les fonction suivantes :
   (a) $\alpha = 0$ et $f(x) = \sqrt{1+x}$
   (b) $\alpha = 0$ et $f(x) = \cos(x)$
   (c) $\alpha = 0$ et $f(x) = a(1+a^2)/(1+x^2)$ pour $a = 1/2$, $a = 1$ et $a = 2$
   (d) $\alpha = 5/7$ et $f(x) = ax(1-x)$ pour $a = 1$, $a = 2$, $a = 3$ et $a = 4$
2. Afin de mieux comprendre ce qui se passe pour les fonctions paramétrées $f_a(x) = a(1+a^2)/(1+x^2)$ et $f_a(x) = ax(1-x)$, étant donné un intervalle I et un réel α, on définit la suite (u_n) de fonctions de I dans $\mathbb{R}$ par :

$$\forall a \in I \quad u_0(a) = \alpha$$

$$\forall n \in \mathbb{N} \quad \forall a \in I \quad u_{n+1}(a) = f_a(u_n(a))$$

   (a) Écrire une fonction `bifurcation(f, alpha, a_min, a_max, m, n_min, n_max)` qui prend en entrée une fonction f de signature `f(a, x)`, un réel α, des réels $a_{\min}$ et $a_{\max}$ définissant l'intervalle $I = [a_{\min}, a_{\max}]$ un entier m définissant un niveau de discretisation pour tracer le graphe de f et des entiers $n_{\min}$ et $n_{\max}$. Cette fonction affichera les graphes des fonctions $a \to u_k(a)$ pour tout $k \in [\![n_{\min}, n_{\max}]\!]$.
   (b) Utiliser cette fonction avec $f_a(x) = a(1+a^2)/(1+x^2)$ et $I = [1/2, 2]$. Tester pour différentes valeurs de α.
   (c) Utiliser cette fonction pour $f_a(x) = ax(1-x)$ et $I = [1, 4]$. Tester pour différentes valeurs de α. On essaiera notamment la valeur $\alpha = 1/\pi$.

2 Bloc de somme maximale

On s'intéresse à des listes de nombres $t = [t_0, \dots, t_{n-1}]$, on note

$$s_{i,j} = \sum_{k=i}^{j-1} t_k$$

et l'on souhaite déterminer la valeur de $m = \max_{0 \leqslant i \leqslant j \leqslant n} s_{i,j}$.

Quelques exemples :
- Pour $t = [5, 3, -3, 2]$, on a $m = 5 + 3 = 8$;
- pour $t = [-2, -3]$, on a $m = 0$;
- pour $t = [1, -2, 5, -1, 7, -1]$, on a $m = 5 - 1 + 7 = 11$;
- pour $t = [4, -1, 2, 3, -1, 2]$, on a $m = 4 - 1 + 2 + 3 - 1 + 2 = 9$.

1. Écrire une fonction `somme_max(t)` qui réponde au problème posé de la manière la plus simple possible (en testant toutes les possibilités). Quelle est la complexité temporelle de cette fonction ?
2. On considère un tableau auxiliaire v de taille $n+1$ tel que $v_j = \max_{0 \leqslant i \leqslant j} s_{i,j}$.
   (a) Justifier que pour $1 \leqslant j \leqslant n$, on a $v_j = \max(0, v_{j-1} + t_{j-1})$.
   (b) En déduire une fonction `somme_max_lin(t)` calculant m et de complexité linéaire.
3. Modifier la fonction `somme_max_lin` pour qu'elle n'utilise plus de tableau auxiliaire (sans bien sûr effectuer plus d'opérations que la fonction précédente).

3 Deuxième plus grand élément

Dans cet exercice, je vous demande d'être particulièrement attentifs à la correction de vos fonctions : il est assez facile d'écrire des fonctions qui marchent « le plus souvent », beaucoup moins d'écrire des fonctions qui marchent *tout le temps*.

1. Écrire une fonction `deuxieme(t)` qui renvoie la deuxième plus grande valeur de t. Si le maximum de t apparaît au moins deux fois, cette valeur sera égale au maximum ; si t n'a qu'un élément, cette valeur n'est pas définie (et la fonction peut donc avoir n'importe quel comportement).
2. Écrire une fonction `deuxieme_distinct(t)` qui renvoie la deuxième plus grande valeur *distincte* de t. Si tous les éléments de t sont égaux, le comportement de la fonction n'est pas défini.