Code de partage avec Capytale : 0440-2108983

Corrigé

Nous verrons aujourd'hui quelques manipulations habituelles avec Python pour l'étude des suites.

1 Calculer des termes d'une suite

1.1 Avec une formule explicite

Calculer les 100 premiers termes de la suites $(u_n)_{n\in\mathbb{N}}$ définie par : $u_n = \dfrac{1}{1+n}$

Comme la formule est explicite, on peut simplement lancer la boucle suivante :

```
for i in range (0,100):
  print(1/(1+i))
```

Variante en définissant une fonction (mais ce n'est pas nécessaire).

```
def u(n):
  return 1/(1+n)

for i in range (0,100):
  print(u(i))
```

1.2 Avec une formule récursive

Calculer les 100 premiers termes de la suites $(u_n)_{n\in\mathbb{N}}$ définie par :

$u_0 = 1000$ puis $\forall n \in \mathbb{N}, u_{n+1} = 1 + \dfrac{1}{1+u_n}$

Emettre une conjecture sur sa limite

Ici la formule n'est pas explicite, donc on doit calculer les termes de manière itérative : u_1 à l'aide de u_0, puis u_2 à l'aide de u_1... Une boucle permet de faire ce travail, par exemple à l'aide d'une variable `u` qui contient la valeur courante (celle du terme en court, et on « écrase » la valeur précédente qui n'est donc pas gardée en mémoire) :

```
u=1000
for n in range(1,100):
  u=1+1/(1+u)
  print(u)
```

On remarque que les valeurs se rapprochent de $\sqrt{2}$ $(\simeq 1,4142\ldots)$, on pressent donc que la suite converge vers $\sqrt{2}$, ce qui est le cas.

2 Un nouvel outil, la boucle *while*

La boucle while

Comme la boucle *for*, la boucle *while* ou « tant que », permet d'exécuter plusieurs fois une instruction, mais dans ce cas le nombre d'itération dépend d'une condition et n'est pas prédéfini.
L'algorithme de cette boucle s'écrit donc :
Tant que "condition"
→ *réaliser "instruction"*

Ce type de boucle peut être utilisé pour étudier l'atteinte ou le dépassement d'un seuil, ce qui correspond bien à des situations modélisées par des suites réelles. Voyons-le sur un exemple.

Avec la suite précédente, trouver le premier terme dont la valeur a un écart avec la limite inférieur à 10^{-5}

Le programme est très proche du précédent puisqu'on calcule de manière itérative les termes de la suite. Mais au lieu de le faire un nombre de fois prédéfini, on va le faire jusqu'à atteindre notre objectif (ou plutot « tant que » notre objectif n'est pas atteint). Ici l'objectif porte sur l'écart entre u_n et $\sqrt{2}$ qui vaut $|u_n - \sqrt{2}|$ et on continue les calculs de termes « tant que » cet écart est trop grand, i.e. supérieur à 10^{-5}

```
u=1000
while np.abs(u-np.sqrt(2))>10**(-5):
  u=1+1/(1+u)
print(u)
```

On peut compléter ce programme en ajoutant un compteur, afin de savoir pour quel rang de la suite on atteint la précision escomptée (ici on trouve $n = 8$ finalement).

```
u=1000
n=0 # on crée un compteur
while np.abs(u-np.sqrt(2))>10**(-5):
  u=1+1/(1+u)
  n=n+1 #on augmente la valeur du compteur à chaque passage dans la boucle
print(u)
```

3 Représentation graphique

Comme nous l'avons vu en cours de maths, une suite peut-être vue comme une fonction, on peut donc adapter nos outils de représentation graphique de fonction avec Python au cas d'une suite.

Représenter graphiquement les 100 premiers termes de la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par :
$u_n = \left(1 + \frac{1}{n}\right)^n$
Emettre une conjecture sur sa limite

De manière analogue à la **partie 1.1**, comme la suite est définie par une formule explicite, on pourrait d'emblée calculer n'importe quel terme comme u_{1000}).
Pour mieux visualiser l'évolution de la suite, on calcule un grand nombre de termes, par exemple les 500 premiers (attention u_0 n'est pas défini ici) :

```
for i in range (1,501):
  print ((1+1/i)**i)
```

Variante en définissant une fonction (mais ce n'est pas nécessaire).

```
def u(n):
  return (1+1/n)**n

for i in range (1,501):
  print(u(i))
```

Et comme demandé, on représente un nombre important de termes (ici 100) pour visualiser l'évolution. C'est ici analogue à la représentation de fonctions car la suite est définie par une formule explicite. Il s'agit simplement de définir une liste d'abscisses qui correspond aux nombres entiers (ici de 1 à 99).

```
import numpy as np
n=np.linspace(1,100,100)
u=(1+1/n)**n
plt.plot(n,u,'+')
plt.show()
```

Quelle que soit la méthode, on a l'impression que la suite converge vers e, ce qui est le cas.

Nota bene : nous avons représenté ici une suite explicite, pour laquelle l'analogie avec la fonction est simple. Pour une suite récursive, il y a une petit difficulté que nous verrons prochainement.