Convergence de suites et approximation

■ Comment manipuler une suite?

SF1 Définir explicitement une suite

Le cas d'une suite définie explicitement est en tout point semblable à celui d'une fonction.

Exemple

Définissons la suite u donnée, pour tout $n \in \mathbb{N}^*$, par $u_n = \frac{1}{n^2+1}$.

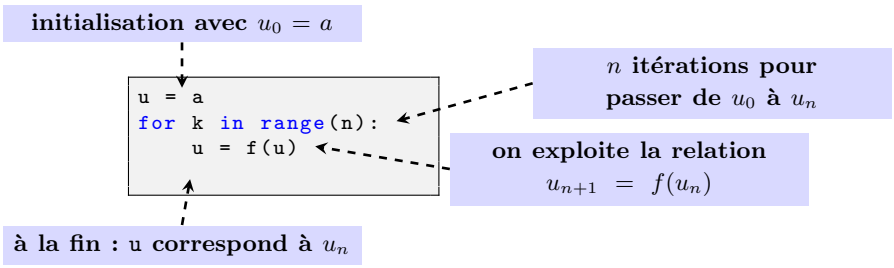
```
def u(n):
    return 1/(n**2+1)
```

En effet :

```
>>> u(2)
0.2
>>> u(5)
0.038461538461538464
```

SF2 Définir une suite par récurrence

Considérons le cas d'une suite définie par une valeur initiale $u_0 = a$ et une relation : $\forall n \in \mathbb{N},\ u_{n+1} = f(u_n)$. On définit une variable initialisée à a puis, au sein d'une boucle `for`, si l'on connaît à l'avance le nombre d'itérations (ou d'une boucle `while` si le nombre d'itérations est conditionné par un test), on remplace successivement les valeurs de la variable par les termes de la suite.



Exemples

1 ► Considérons la suite u définie par $u_0 = 2$ et : $\forall n \in \mathbb{N},\ u_{n+1} = \sqrt{1+u_n}$.

Écrivons une fonction d'en-tête `u(n)` qui renvoie la valeur de u_n.

```
def u(n):
    v = 2
    for k in range(n):
        v = sqrt(1 + v)
    return v
```

Ici, u est le nom de la fonction alors que v n'est qu'une variable locale à l'intérieur de la fonction.

```
>>> u(2)
1.6528916502810695
>>> v
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'v' is not defined
```

2 ► Considérons la suite de Fibonacci définie par $F_0 = 0$, $F_1 = 1$ et : $\forall n \in \mathbb{N}$, $F_{n+2} = F_{n+1} + F_n$.
On vérifie que cette suite diverge vers $+\infty$.
Déterminons le premier terme de cette suite qui soit supérieur à 1000.

```
F = [0, 1]  # il est plus efficace de travailler avec deux termes successifs
while F[1] < 1000:
    s = F[1]
    t = F[0] + F[1]
    F = [s, t]
```

On trouve :

```
>>> F[1]
1597
```

SF3 Donner les premiers termes d'une suite

◇ Dans le cas d'une suite $\left(f(n)\right)_{n\in\mathbb{N}}$ donnée explicitement à l'aide d'une fonction f, on peut directement calculer l'image du vecteur correspondant aux entiers de 0 à n.

Exemple

Définissons une liste contenant les 10 premiers termes de la suite $\left(\frac{1}{n+1}\right)_{n\in\mathbb{N}}$.

```
>>> [1/(n+1) for n in range(10)]
[1.0, 0.5, 0.3333333333333333, 0.25, 0.2, 0.16666666666666666,
    0.14285714285714285, 0.125, 0.1111111111111111, 0.1]
```

On peut également utiliser une boucle en modifiant une liste existante :

```
L = [0]*10
for n in range(10):
    L[n] = 1/(n+1)
```

◇ Dans le cas d'une suite donnée par une relation de récurrence, il suffit de construire pas à pas une liste en exploitant les derniers termes ajoutés ou bien de l'initialiser à la bonne taille puis de modifier ses termes.

Exemple

Définissons une fonction d'en-tête `F(n)` renvoyant la liste des termes d'indice 0 à n de la suite de Fibonacci.

```
def F(n):
    L = [0, 1]
    for k in range(2, n):
        L.append(L[-1] + L[-2])
    return L

def Fbis(n):
    L = [0]*n
    L[1] = 1
    for k in range(2,n):
        L[k] = L[k-1] + L[k-2]
    return L
```

Ce qui donne :

```
>>> F(10)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
>>> Fbis(10)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

Exercice 1

On considère la suite donnée par :

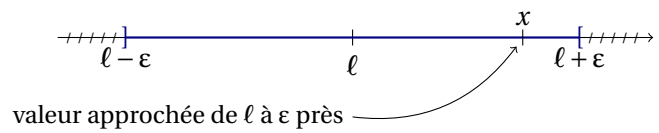
$$\begin{cases} u_0 = 1 \\ u_{n+1} = \dfrac{1}{2}\left(u_n + \dfrac{2}{u_n}\right) \quad \text{pour } n \geqslant 0 \end{cases}.$$

Écrire une fonction d'argument n et renvoyant le vecteur constitué par $u_0, u_1, \ldots, u_n$.

■ Comment obtenir une valeur approchée d'une limite de suite ?

SF4 Comprendre le principe d'une approximation

Le principe de toute approximation réside sur le fait que, pour tout réel $\varepsilon > 0$, tout réel x dans l'intervalle $]\ell - \varepsilon, \ell + \varepsilon[$ vérifie $|\ell - x| < \varepsilon$ donc fournit une approximation de ℓ à ε près (c'est-à-dire avec une erreur d'au plus ε).



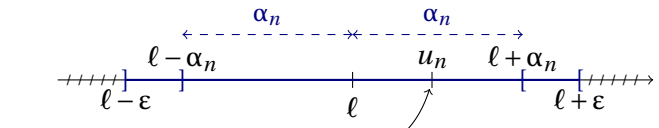
valeur approchée de ℓ à ε près

SF5 Exploiter une majoration de l'erreur

Considérons une suite $(u_n)_{n\in\mathbb{N}}$ et un réel ℓ tels qu'il existe une suite $(\alpha_n)_{n\in\mathbb{N}}$ vérifiant :

$$\forall n \in \mathbb{N},\ |u_n - \ell| \leqslant \alpha_n \quad \text{et} \quad \alpha_n \xrightarrow[n\to+\infty]{} 0.$$

Il résulte tout d'abord du théorème d'existence de limite par encadrement que la suite $(u_n)_{n\in\mathbb{N}}$ est convergente de limite ℓ.
Par ailleurs, pour tout réel $\varepsilon > 0$, **il suffit** que $\alpha_n < \varepsilon$ pour que u_n fournisse une valeur approchée de ℓ à ε près.



valeur approchée de ℓ à ε près

Exemple

Considérons la suite $(x_n)_{n\in\mathbb{N}}$ définie par $x_0 = 7$ et : $\forall n \in \mathbb{N},\ x_{n+1} = \dfrac{x_n - 1}{\ln(x_n) - 1}$.
On montre que la suite $(x_n)_{n\in\mathbb{N}}$ est convergente et, en notant ℓ sa limite et à l'aide de l'inégalité des accroissements finis, on obtient :

$$\forall n \in \mathbb{N},\ |x_n - \ell| \leqslant (0{,}17)^n.$$

Déduisons-en une fonction d'en-tête `approx(eps)` donnant une valeur approchée de ℓ à `eps` près.

```
def f(x):
    return (x-1)/(log(x)-1)

def approx(eps):
    x = 7
    n = 0
    while (0.17)**n > eps:
        x = f(x)
        n = n+1
    return x
```

```
def approxbis(eps): # alternative
    x = 7
    maj = 1
    n = 0
    while maj > eps:
        x = f(x)
        n = n+1
        maj = maj * 0.17
    return x
```

On trouve par exemple :

```
>>> approx(1e-5)
6.305395279271691
```

En fonction de l'expression de α_n, il arrive que l'on puisse explicitement déterminer un entier n tel que $\alpha_n < \varepsilon$ auquel cas on peut ensuite utiliser une boucle `for`.

Exemple

Reprenons l'exemple précédent pour illustrer cette idée.
On a :

$$\begin{aligned}(0,17)^n \leqslant \varepsilon &\iff n\ln(0,17) \leqslant \ln(\varepsilon)\\ &\iff n \geqslant \frac{\ln(\varepsilon)}{\ln(0,17)}.\end{aligned}$$

On en déduit le programme suivant :

```
def approxter(eps):
    n = ceil(log(eps) / log(0.17))
    x = 7
    for k in range(n):
        x = f(x)
    return x
```

SF6 Utiliser des suites adjacentes

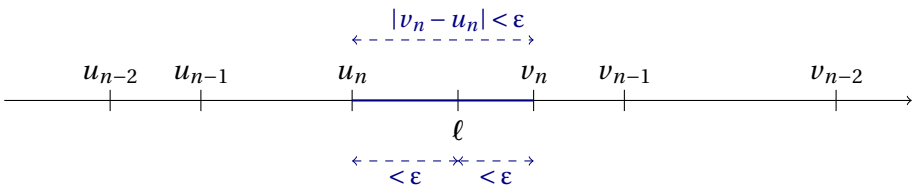
On rappelle que deux suites sont dites *adjacentes* lorsque l'une est croissante, l'autre décroissante et l'écart entre les deux converge vers 0.
Pour fixer les idées, considérons le cas d'une suite $(u_n)_{n\in\mathbb{N}}$ croissante et d'une suite $(v_n)_{n\in\mathbb{N}}$ décroissante telles que :

$$|v_n - u_n| \xrightarrow[n\to+\infty]{} 0.$$

Dans ces conditions, les deux suites convergent et leur limite est la même. De plus, en notant ℓ cette limite, on a :

$$\forall n \in \mathbb{N},\ u_n \leqslant \ell \leqslant v_n.$$

Il s'ensuit que, pour tout réel $\varepsilon > 0$, il suffit que $|v_n - u_n| < \varepsilon$ pour que tout réel entre u_n et v_n fournisse une valeur approchée de ℓ à ε près.



Exemple

On définit les deux suites u et v par $u_0 = 1$, $v_0 = \sqrt{2}$ et :

$$\forall n \in \mathbb{N},\ u_{n+1} = \frac{u_n + v_n}{2} \quad \text{et} \quad v_{n+1} = \sqrt{u_n v_n}.$$

On admet que ces deux suites sont adjacentes.
Déduisons-en une fonction d'en-tête `approx(eps)` renvoyant une valeur approchée de la limite commune de ces deux suites à `eps` près.

```
def approx(eps):
    u = 1
    v = sqrt(2)
    while abs(v-u) > eps:
        u, v = (u+v)/2, sqrt(u*v)
    return (u+v)/2
```

On obtient par exemple :

```
>>> approx(1e-5)
1.1981402347355923
```

Exercice 2

Soit u la suite donnée par $u_0 = -1$ et :

$$\forall n \in \mathbb{N},\ u_{n+1} = \frac{e^{u_n} - 3}{2}.$$

L'étude mathématique de cette suite montre qu'elle converge vers un réel ℓ entre -2 et -1 et que l'on a :

$$\forall n \in \mathbb{N},\ |u_n - \ell| \leqslant \frac{1}{2^n}.$$

Écrire une fonction d'argument p et qui donne une valeur approchée de ℓ à 10^{-p} près.

Exercice 3

On définit les trois suites a, b et p par $a_0 = \frac{1}{3\sqrt{3}}$, $b_0 = 2a_0$ et, pour tout $n \in \mathbb{N}$, par les relations :

$$a_{n+1} = \frac{a_n + b_n}{2}\ ,\ b_{n+1} = \sqrt{a_{n+1} b_n}\ \text{ et }\ p_n = \frac{1}{b_n}.$$

Une étude mathématique, montre que :

$$\forall n \in \mathbb{N},\ 0 \leqslant \pi - p_n \leqslant \frac{3\sqrt{3}}{4^n}.$$

À l'aide de cette majoration, écrire une fonction d'argument p et qui donne une valeur approchée de π à 10^{-p} près.

Exercice 4

On considère les suites u et v donnée par $u_0 = 2$, $v_0 = 1$ et pour tout $n \in \mathbb{N}$:

$$u_{n+1} = \frac{u_n + \lambda v_n}{1+\lambda} \quad \text{et} \quad v_{n+1} = \frac{u_n + \mu v_n}{1+\mu},$$

où $\lambda = 0,05$ et $\mu = 4$.

1. Écrire un programme permettant d'obtenir les valeurs de u_n et v_n pour $n \in [\![0, 10]\!]$.

2. En admettant que ces deux suites sont adjacentes, déterminer le plus petit entier n tel que :

$$|u_n - v_n| \leqslant 10^{-4}.$$

3. Définir le vecteur `U` dont les composantes sont $u_0, u_1, \ldots, u_{50}$.

4. Représenter graphiquement $u_0, u_1, \ldots, u_{50}$.

5. Représenter sur le même dessin $u_0, u_1, \ldots, u_{50}$ et $v_0, v_1, \ldots, v_{50}$.

6. Mêmes questions avec $\lambda = 0,1$ et $\mu = 6$.

Exercice 5

Soit u la suite donnée par $u_0 = \frac{1}{5}$ et, pour tout $n \in \mathbb{N}$, $u_{n+1} = f(u_n)$, où f est la fonction définie sur $\mathbb{R}$ par :

$$f(x) = \frac{x^2 + 2x + 1}{x^2 + 1}.$$

1. Programmer la fonction f.

2. On peut montrer, à l'aide de l'inégalité des accroissements finis, que la suite u converge vers une limite ℓ et que l'on a pour tout $n \in \mathbb{N}^*$:

$$|u_n - \ell| \leqslant \left(\frac{1}{4}\right)^{n-1}.$$

Déterminer un entier n tel que u_n fournisse une valeur approchée de ℓ à 10^{-10} près. Afficher n et le terme u_n correspondant.

■ Comment approcher les solutions d'une équation de la forme $f(x) = 0$?

En général, les méthodes utilisent des suites définies par récurrence et exploitent la continuité de la fonction f. Cependant, le seul algorithme à maîtriser de façon autonome est le suivant.

SF7 Programmer une recherche dichotomique

Considérons une fonction f *continue* sur un intervalle $[a, b]$ et telle que $f(a)$ et $f(b)$ soient de signes opposés.

D'après le théorème des valeurs intermédiaires, il existe au moins un réel en lequel f s'annule.

Pour déterminer une valeur approchée d'un tel réel (mais c'est également l'idée d'une démonstration possible du théorème des valeurs intermédiaires), on peut utiliser l'***algorithme de dichotomie*** qui consiste à définir deux suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ de la façon suivante :

– on pose $a_0 = a$ et $b_0 = b$;

– une fois $a_0, \dots, a_k$ et $b_0, \dots, b_k$ construits, on considère $f\left(\frac{a_k+b_k}{2}\right)$:

- si $f\left(\frac{a_k+b_k}{2}\right)$ est du même signe que $f(b_k)$ alors on pose $b_{k+1} = \frac{a_k+b_k}{2}$ et $a_{k+1} = a_k$;
- sinon, on pose $a_{k+1} = \frac{a_k+b_k}{2}$ et $b_{k+1} = b_k$.

Dans ces conditions, $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ sont deux suites adjacentes.

La continuité de f sur $[a, b]$ permet d'en déduire que ces suites convergent vers un réel x_0 vérifiant $f(x_0) = 0$.

```
def dichotomie(f, a, b, epsilon):
    g, d = a, b  # bornes de gauche et de droite, initialement a et b
    while abs(g-d) > epsilon:
        m = (g+d)/2
        if f(g) * f(m) < 0:  # si f(g) et f(m) sont de signes opposés
            d = m
        else:
            g = m
    return (g+d)/2  # toute valeur entre g et d convient
```

Exemple

Déterminons des valeurs approchées de $\sqrt{10}$.
Il suffit de définir une fonction qui s'annule en $\sqrt{10}$ (sans utiliser cette valeur).

```
def rac10(x):
    return x*x - 10
```

On obtient par exemple :

```
>>> dichotomie(rac10, 3, 4, 1e-8)
3.1622776575386524
>>> sqrt(10)  # pour comparer
3.1622776601683795
```

Exercice 6

1. Étudier la fonction f définie sur $\mathbb{R}$ par : $f(x) = x^3 - 3x - 1$.

2. Justifier que f s'annule en trois réels x_0, x_1 et x_2 vérifiant :

$$-2 < x_0 < -1 < x_1 < 0 < x_2 < 2.$$

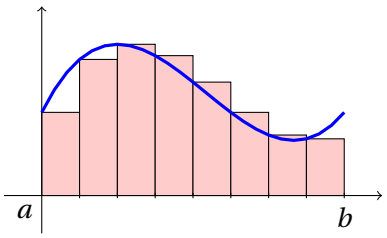
3. Déterminer, par dichotomie, des valeurs approchées à 10^{-5} près de x_0, x_1 et x_2.

■ Comment déterminer une valeur approchée d'une intégrale ?

SF8 Connaître et programmer la méthode des rectangles

Soit $f : [a, b] \to \mathbb{R}$ continue par morceaux et $n \geqslant 1$ un entier.
Pour tout $k \in [\![0, n]\!]$, on pose $a_k = a + k\frac{b-a}{n}$.

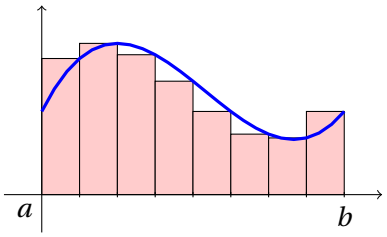
La *méthode des rectangles à gauche* consiste à remplacer f sur $[a_k, a_{k+1}]$ par la fonction constante qui coïncide avec f au point a_k :



Cela revient à remplacer $\displaystyle\int_a^b f(t)\mathrm{d}t$ par :

$$\mathrm{R}_n(f) = \frac{b-a}{n}\sum_{k=0}^{n-1} f(a_k).$$

La *méthode des rectangles à droite* consiste à remplacer f sur $[a_k, a_{k+1}]$ par la fonction constante qui coïncide avec f au point a_{k+1} :



Cela revient à remplacer $\displaystyle\int_a^b f(t)\mathrm{d}t$ par :

$$\mathrm{R}'_n(f) = \frac{b-a}{n}\sum_{k=1}^{n} f(a_k).$$

Voici une façon de programmer la méthode des rectangles à gauche :

```
def gauche(f, a, b, n):
    R = 0
    for k in range(n):
        R = R + f(a + k*(b-a)/n)
    return (b-a)/n * R
```

Exemple

Appliquons cette méthode au cas de $\ln(2) = \displaystyle\int_1^2 \frac{1}{t}\mathrm{d}t$.

```
def inverse(x):
    return 1/x
```

Ce qui donne :

```
>>> gauche(inverse, 1, 2, 1000)
0.6933972430599373
```

Il faut également savoir exploiter une majoration de l'erreur. Dans le cas de la méthode des rectangles à gauche ou à droite, si f est de classe $\mathscr{C}^1$ sur $[a,b]$ alors :

$$\left|\int_a^b f(t)\mathrm{d}t - \mathrm{R}_n(f)\right| \leqslant \frac{(b-a)^2}{2n}\sup_{[a,b]}|f'|.$$

Il s'ensuit que la méthode des rectangles à gauche ou à droite converge «à la vitesse» $\mathrm{O}\left(\frac{1}{n}\right)$.

SF9 **Adapter les idées précédentes à d'autres méthodes d'approximation**

D'autres méthodes sont plus rapides que la méthode des rectangles à droite ou à gauche. Par exemple, la *méthode du point milieu* consiste à remplacer f sur $[a_k, a_{k+1}]$ par la fonction constante qui coïncide avec f au point milieu $\frac{a_k+a_{k+1}}{2}$.

Cela revient à remplacer $\displaystyle\int_a^b f(t)\mathrm{d}t$ par :

$$\mathrm{M}_n(f) = \frac{b-a}{n}\sum_{k=0}^{n-1} f\left(\frac{a_k+a_{k+1}}{2}\right).$$

Si f est de classe $\mathscr{C}^2$ alors :

$$\left|\int_a^b f(t)\mathrm{d}t - \mathrm{M}_n(f)\right| \leqslant \frac{(b-a)^3}{24n^2}\sup_{[a,b]}|f''|.$$

```
def milieu(f, a, b, n):
    M = 0
    for k in range(n):
        M = M + f( ((a + k*(b-a)/n) + (a + (k+1)*(b-a)/n)) / 2)
    return (b-a)/n * M
```

Exemple

Considérons à nouveau l'exemple de $\ln(2) = \displaystyle\int_1^2 \frac{1}{t}\mathrm{d}t$.

```
>>> milieu(inverse, 1, 2, 1000)
0.6931471493099519
```

Exercice 7

Pour tout réel x, on pose :

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} \mathrm{e}^{-\frac{1}{2}t^2}\,\mathrm{d}t.$$

On admet que $\Phi(0) = \frac{1}{2}$ et, pour tout réel x : $\Phi(-x) = 1-\Phi(x)$.
Les intégrales seront calculées à l'aide de la méthode des rectangles.

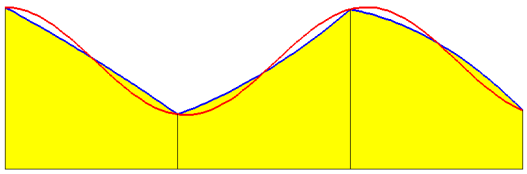
1. Écrire une fonction `f` ayant pour argument un réel *positif* x et un entier $\mathrm{N} \geqslant 1$ et renvoyant une valeur approchée, calculée avec N rectangles, de :

$$\frac{1}{\sqrt{2\pi}}\int_0^{x} \mathrm{e}^{-\frac{1}{2}t^2}\,\mathrm{d}t.$$

2. Écrire une fonction `Phi` ayant pour argument un réel x et un entier $\mathrm{N} \geqslant 1$ et donnant comme réponse une valeur approchée de $\Phi(x)$ telle que l'intégrale entre 0 et x soit calculée avec N rectangles.

Exercice 8

La *méthode de Simpson* consiste à remplacer f sur $[a_k, a_{k+1}]$ par la fonction polynomiale de degré 2 qui coïncide avec f aux points a_k, a_{k+1} et $\frac{a_k+a_{k+1}}{2}$:



On peut montrer que cela revient à remplacer $\int_a^b f(t)\mathrm{d}t$ par :

$$\mathrm{S}_n(f) = \frac{b-a}{6n}\sum_{k=0}^{n-1}\left[f(a_k) + 4f\left(\frac{a_k+a_{k+1}}{2}\right) + f(a_{k+1})\right].$$

Écrire une fonction `simpson(f, a, b, n)` qui calcule la somme exprimée ci-dessus et comparer cette méthode aux diverses méthodes des rectangles.