Approximation numérique

Dans de nombreux domaines, il n'est pas toujurs facile d'obtenir les valeurs exactes des quantités étudiées. Parfois, on peut espérer en obtenir une approximation numérique. La plupart du temps, cela revient donc à construire des suites qui convergent vers la valeur cherchée.
On pourra illustrer quelques ***vitesses de convergence*** *et essayer de comprendre la notion d'erreur dans ces approximations.*

1 Méthode numérique d'intégration et application

Le théorème fondamental de l'analyse nous permet de calculer la valeur de l'intégrale d'une fonction f continue sur un segment, mais il exige de connaitre au moins une primitive de f sur l'intervalle considéré. Dans de nombreux cas, et en dépit de toutes les transformations usuelles, on ne pourra donc pas faire appel à ce théorème.
Malgré tout, nous allons voir qu'il est possible de construire des suites de fonctions permettant d'approcher une intégrale : on parle alors de **méthodes numériques d'intégration**. Celles-ci reposent en particulier sur **la convergence des sommes de Riemann**.

Soit $n \in \mathbb{N}^*$. Considérons f une fonction continue sur $[a,b]$ à valeurs réelles, et notons (x_i) une subdivision du segment $[a,b]$ de sorte que $x_0 = a < x_1 < \ldots < x_n = b$. On rappelle que la **somme de Riemann associée à f et aux points** (x_i) désigne la suite (S_n) définie par :

$$S_n = \sum_{i=0}^{n-1} (x_{i+1} - x_i) f(c_i), \text{ où pour tout } i \in [\![0, n-1]\!],\ c_i \in [x_i, x_{i+1}]$$

Et dans le cas particulier où la subdivision est choisie à pas constant, la suite (S_n) est alors définie pour tout $n \in \mathbb{N}^*$ par :

$$S_n = \sum_{i=0}^{n-1} \frac{(b-a)}{n} f(c_i)$$

et on peut montrer que $S_n \underset{n\to+\infty}{\longrightarrow} \displaystyle\int_a^b f(t)\,dt$.

Exercice 1 (de la méthode des rectangles à la méthode d'Euler). [ ]
On considère ici la fonction $f : t \longmapsto \dfrac{1}{1+t^2}$ sur le segment $[0,1]$, et on note pour tout $n \in \mathbb{N}^*$, $x_0, \ldots, x_n$ les points de la subdivision à pas constant.

1. (a) On pose pour tout $i \in [\![0, n-1]\!]$, $c_i = x_i$. Donner l'expression de la somme de Riemann associée, puis en donner une interprétation géométrique :

   (b) On pose pour tout $i \in [\![0, n-1]\!]$, $c_i = x_{i+1}$. Donner l'expression de la somme de Riemann associée, puis en donner une interprétation géométrique :

2. Dans le deuxième cas, on a ainsi : $S_n = \sum_{i=1}^{n} \frac{1}{n} f(\frac{i}{n})$ **(méthode des rectangles à droite)**.
   Dans le langage Python, construire la fonction $somme(n : int) \to float$ qui pour tout entier n non nul, calcule les termes successifs de la suite puis renvoie la valeur de S_n, une approximation de l'intégrale de f sur $[0,1]$.

3. **Application** On considère un **problème de Cauchy** de la forme :

$$\begin{cases} y' = F(t,y)\ , t \in [a,b] \\ y(a) = y_0 \end{cases}$$

On note pour tout $n \in \mathbb{N}^*$, $a = t_0 < t_1 < \ldots < t_n = b$ la subdivision à pas constant définie par :

$$\forall\ k \in [\![0,n]\!],\ t_k = a + k.h\ \text{, avec } h = \frac{b-a}{n}$$

En particulier, si y désigne l'unique solution du problème de Cauchy, on en déduit par intégration :

$$\int_{t_0}^{t_1} y'(t)\ dt = \int_{t_0}^{t_1} F(t,y(t))\ dt \Leftrightarrow y(t_1) - y(t_0) = \int_{t_0}^{t_1} F(t,y(t))\ dt \Leftrightarrow y(t_1) = y_0 + \int_{t_0}^{t_1} F(t,y(t))\ dt$$

On obtient alors une approximation de $y(t_1)$ en approchant l'intégrale par la méthode des rectangles. On répète alors l'opération pour obtenir $y_1, \ldots, y_n$ des valeurs approchées de la solution aux points de la subdivision.

(a) En utilisant l'approximation donnée, justifier la relation obtenue entre y_1 et y_0. En itérant le procédé, retrouver alors le **schéma d'Euler explicite** permettant de construire la suite de points y_k.

(b) On se place dans un cas particulier :

$$\begin{cases} y' = \tan(t)y \\ y(0) = 1 \end{cases}$$

Dans le langage Python, construire la fonction $y(n : int) \to list$ qui, pour tout entier n non nul donné, renvoie la liste L contenant les valeurs $[y_0, \ldots, y_n]$ livrant ainsi les approximations de f aux points $t_0, \ldots, t_n$.

(c) Résoudre rigoureusement le problème de Cauchy, puis construire le programme $euler(n : int) \to None$ qui, pour tout entier n non nul donné, affiche sur un même graphe la solution approchée, ainsi que l'unique solution du problème de Cauchy. On n'oubliera pas d'afficher une légende permettant de reconnaître les deux solutions.

Remarques

1. Cette méthode d'Euler peut aussi s'appliquer à une équation différentielle qui ne serait pas linéaire, à condition de pouvoir la présenter sous forme résolue : $y' = F(t, y)$ et avec une condition initiale.

2. On peut étendre la méthode à la résolution des équations différentielles d'ordre 2, mais il faudra d'abord se ramener à un système différentiel de la forme $X' = F(t, X)$. Par exemple, si y désigne l'unique solution d'un tel problème de Cauchy d'ordre 2 avec des conditions initiales données, on a à l'aide des opérations matricielles :

$$y'' + a(t)y' + b(t)y = c(t) \Leftrightarrow \begin{pmatrix} y' \\ y'' \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ b(t) & a(t) \end{pmatrix} \begin{pmatrix} y \\ y' \end{pmatrix} = \begin{pmatrix} 0 \\ c(t) \end{pmatrix}$$

Et donc, en posant $X = \begin{pmatrix} y \\ y' \end{pmatrix}$, $B(t) = \begin{pmatrix} 0 \\ c(t) \end{pmatrix}$ et $A(t) = \begin{pmatrix} 0 & -1 \\ b(t) & a(t) \end{pmatrix}$, on est ramené à résoudre le système différentiel suivant :

$$X' + A(t)X = B(t) \Leftrightarrow X' = \underbrace{-A(t)X + B(t)}_{=F(t,X)}$$

Il suffit encore d'appliquer la **méthode d'Euler explicite à ce système différentiel** en considérant le schéma suivant :

$$\begin{cases} X_0 = \begin{pmatrix} \lambda \\ \mu \end{pmatrix} \\ \forall\, k \in [\![0, n-1]\!],\ X_{k+1} = X_k + h.F(t_k, X_k) \end{cases}$$

2 Autres exemples d'approximation

Exercice 2 (comparaison des méthodes numériques d'intégration)**.** [ ]

On considère encore la fonction $f : t \longmapsto \dfrac{1}{1+t^2}$ sur le segment $[0, 1]$, et on note pour tout $n \in \mathbb{N}^*$, $x_0, \ldots, x_n$ les points de la subdivision à pas constant. On décide alors d'affiner l'approximation sur les intervalles rencontrés :

- par exemple, on peut approcher f par une fonction affine par morceaux décrivant un trapèze sur chaque intervalle $[x_i, x_i + 1]$ en utilisant la fonction définie par :

$$y = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i) + f(x_i)$$

de sorte qu'on pose :

$$\boxed{S_n = \sum_{i=0}^{n-1} \frac{(b-a)}{n}\left(\frac{f(x_i) + f(x_{i+1})}{2}\right) \quad \textbf{(méthode des trapèzes)}}$$

- de la même façon, on peut approcher f par une fonction définie par morceaux décrivant un polynôme d'interpolation passant par trois points, et donc sur chaque intervalle $[x_i, x_i + 1]$, on a un polynôme satisfaisant l'équation :

$$y = f(x_i)\frac{(x - m_i)(x - x_{i+1})}{(x_i - m_i)(x_i - x_{i+1})} + f(m_i)\frac{(x - x_i)(x - x_{i+1})}{(m_i - x_i)(m_i - x_{i+1})} + f(x_{i+1})\frac{(x - x_i)(x - m_i)}{(x_{i+1} - x_i)(x_{i+1} - m_i)}$$

avec $m_i = \dfrac{x_i + x_{i+1}}{2}$ et de sorte qu'on pose :

$$\boxed{S_n = \sum_{i=0}^{n-1} \frac{(b-a)}{n}\left(\frac{f(x_i) + 4f(m_i) + f(x_{i+1})}{6}\right) \quad \textbf{(méthode de Simpson)}}$$

1. Dans le langage Python, construire les fonctions $trapezes(n : int) \to float$ et $simpson(n : int) \to float$, qui pour tout entier n non nul, renvoie la valeur de S_n.

2. On souhaite comparer la **vitesse de convergence** de ces méthodes. Pour cela, construire la fonction $comparaison(\epsilon : float) \to tuple$ qui, pour toute précision ϵ, calcule S_n tant que $|S_n - \pi/4| > \epsilon$, puis renvoie le plus petit indice n_0 pour lequel :

$$|S_{n_0} - \pi/4| \leq \epsilon$$

et ceci pour chacune des méthodes numériques d'intégration : méthode des rectangles à droite, trapèzes et de Simpson.

Remarque On illustre ici la vitesse de convergence de ces méthodes, mais pour les comparer rigoureusement, il faut chercher à majorer l'**erreur d'approximation**. Ainsi, en notant :

$$\epsilon_n = |S_n - \int_a^b f(t)\, dt|$$

on peut montrer que pour une fonction suffisamment régulière, on a des vitesses de convergence polynomiales :

erreur	rectangles	trapèzes	Simpson
ϵ_n	$O(\frac{1}{n})$	$O(\frac{1}{n^2})$	$O(\frac{1}{n^4})$

Mais si on veut gagner du temps, on peut aussi faire appel à la commande `quad` du module **scipy.integrate**, et on veillera à placer une fonction en argument, ainsi que les valeurs du segment sur lequel on travaille.

Par exemple, si on cherche à obtenir une valeur approchée de $\displaystyle\int_0^1 \frac{1}{1+t}\,dt$:

```python
def f(t):
    return 1/(1+t)
```

et ainsi, on obtient rapidement l'approximation cherchée :

```
In : from scipy.integrate import *
quad(f,0,1)
                    (0.6931471805599454, 7.695479593116622e-15)
```

Cette fonction peut également prendre la valeur `inf` dans les bornes, et on retiendra qu'elle renvoie un second résultat : il s'agit simplement d'une majoration de l'erreur d'approximation.

Exercice 3 (approximation d'un zéro d'une fonction par dichotomie)**.** [ ]

On considère l'équation $x^2 - \frac{5}{2}x + 1 = 0$, $x \in [1,4]$. On peut alors rechercher l'unique solution de cette équation à l'aide du **principe de dichotomie** : il s'agit en fait de définir par récurrence deux suites (a_n) et (b_n) encadrant la solution donnée.

On pose $f : x \in \mathbb{R} \longmapsto x^2 - \frac{5}{2}x + 1$ et on définit les suites $(a_n), (b_n)$ par $a_0 = 1, b_0 = 4$, et pour tout $n \in \mathbb{N}$,

$$a_{n+1} = \begin{cases} a_n, \text{ si } f(a_n)f(\frac{a_n+b_n}{2}) < 0 \\ \frac{a_n+b_n}{2}, \text{ sinon} \end{cases} \quad \text{et } b_{n+1} = \begin{cases} \frac{a_n+b_n}{2}, \text{ si } f(a_n)f(\frac{a_n+b_n}{2}) < 0 \\ b_n, \text{ sinon} \end{cases}$$

1. Dans le langage Python, construire la fonction $bornes(n : int) \longrightarrow tuple$ qui pour tout entier n donné, renvoie les valeurs a_n et b_n encadrant la solution à l'équation donnée.
2. Construire alors le programme $approx(p : int) \longrightarrow int$ qui pour tout entier p donné, renvoie le plus petit indice n_p pour lequel $|b_{n_p} - a_{n_p}| \leq 10^{-p}$.

En particulier, a_{n_p} et b_{n_p} désigneront des valeurs approchées par défaut et par excès de la solution à 10^{-p} près.

Pour finir, on s'intéresse à la vitesse de convergence de ces deux suites.

3. Construire le programme $vitesse(N : int) \longrightarrow None$ qui pour tout entier N donné, renvoie un graphe représentant l'évolution du nombre d'itérations (n_p) nécessaires pour approcher la solution à 10^{-p} près en fonction de $p \in [\![1, N]\!]$.
4. Justifier rapidement que les deux suites (a_n) et (b_n) définissent bien des suites adjacentes dont la limite n'est rien d'autre que $\ell \in [1,4]$ vérifiant $f(\ell) = 0$. Montrer alors que l'erreur d'approximation vérifie :

$$\epsilon_n = |a_n - \ell| = O(\frac{1}{2^n})$$

*Cette dernière égalité nous permet en fait d'affirmer que la vitesse de convergence du principe de dichotomie est **géométrique**.*

Remarque Cet algorithme est assez efficace, mais il convient d'abord de localiser l'unique solution de l'équation $f(x) = 0$. D'ailleurs, ce sera aussi le cas pour les autres méthodes de recherche des zéros d'une fonction donnée : méthode du balayage, méthode de Newton...