

VIII Méthodes numériques

6 octobre 2026

Ce chapitre regroupe les différentes méthodes numériques que vous pourrez être amenés à mettre en œuvre, tant à l'écrit qu'à l'oral. Elles ne feront pas l'objet d'un cours, mais nous les travaillerons entre les écrits et les oraux.

1. Numpy

La bibliothèque **numpy** est la bibliothèque de calcul vectoriel la plus utilisée sous Python. C'est un projet *open source*, débuté en 2005.

L'objet de base de **numpy** sont les tableaux multidimensionnels, de type **numpy.ndarray**. Ce sont des tableaux *non dynamiques* (i.e. une fois créés, il n'est pas possible de modifier leur taille, au contraire des objets de type **list** de Python) et homogènes (tous les éléments du tableau sont de même type, pour nous des nombres flottants en double précision, codés sur 64 bits). Ces particularités là (et le fait qu'une grande partie de **numpy** est codée en C et en C++) expliquent la grande rapidité des calculs menés avec **numpy**.

Dans ce chapitre, nous supposons classiquement que la bibliothèque **numpy** a été importée avec l'alias **np** par la commande suivante.

```
import numpy as np
```

Il existe plusieurs manières de créer un tableau **numpy**.

- ▷ En convertissant un tableau Python (écrit comme listes de listes). Exemple :
`M = np.array([[1.,2.,3.],[4.,5.,6.]])`
- ▷ En utilisant une des fonctions de création de tableaux prédisponibles :
 - `np.arange(a,b)` (semblable à `range(a,b)`) et `np.linspace(a,b,num)` (`num` valeurs équiréparties entre `a` et `b`) pour créer des vecteurs (1 dimension) ;
 - `np.eye(n)` pour créer une matrice identité $n \times n$, ou `eye(n,p)` pour une matrice $n \times p$ (2 dimensions) ;

- `np.zeros(tuple)` (tableau de zéros), `np.ones(tuple)` (tableau de uns) où pour créer un tableau ayant autant de dimensions que `tuple` n'a d'éléments, les éléments de `tuple` donnant la taille de chaque dimension (ex : `np.zeros((2,3))` pour créer une matrice ayant deux lignes et trois colonnes).

L'opération inverse de la conversion faite par la fonction **array** est possible par la méthode `.tolist()` : si `M` est une matrice (de type **np.ndarray**), alors `M.tolist()` est la liste de listes correspondante.

Pour un tableau `M`, l'attribut `M.shape` contient les dimensions du tableau (sous forme de tuple). Si `M` est une matrice (tableau de dimension 2) à n lignes et p colonnes, et si $0 \leq i < n$ et $0 \leq j < p$, alors on accède (en lecture et en écriture) au coefficient $M_{i,j}$ par la commande `M[i,j]`, à la i^{e} ligne de `M` par la tranche `M[i, :]` (on obtient un vecteur) et à la j^{e} colonne de `M` par la tranche `M[:,j]` (on obtient aussi un vecteur).

Il est possible de réaliser toutes les opérations (vectorielles, ou non : `+`, `*`, `**` etc.) sur des tableaux **numpy** de même dimension : les opérations s'entendent alors *coefficient par coefficient*.

De même, on pourra appliquer les fonctions **numpy** (ex : `np.exp`, `np.sin` etc.) sur un tableau **numpy** : ces fonctions sont encore une fois appliquées *coefficient par coefficient*.

Le produit de deux matrices `M` et `N` ne se calcule alors pas par l'écriture `M*N`, mais par l'écriture `M.dot(N)`.

Enfin, une opération à valeur booléenne sur deux matrices renvoie... une matrice booléenne (ex : `M == N`, `M != N`, `M > N`). On utilisera les méthodes `.any()` (réalise un « ou logique » sur les coefficients de la matrice) et `.all()` (réalise un « et logique » sur les coefficients de la matrice). Exemple : `(M == N).all()` teste l'égalité entre `M` et `N`, `(M != N).any()` teste la différence entre `M` et `N`, `(abs(M - N) <= eps).all()` teste si les coefficients de `M` sont tous distants au plus de `eps` de ceux de `N`.

2. Équations différentielles

2.1. Généralités

La bibliothèque la plus utilisée pour résoudre des équations différentielles est `scipy.integrate`. Cette bibliothèque contient des fonctions de calcul approché d'intégrales, et une fonction de résolution approchée d'équations différentielles : `odeint`.

Cette fonction permet de résoudre l'équation différentielle vectorielle

$$X'(t) = F(X(t), t),$$

de la variable vectorielle X . Ici, X est donc une fonction à valeurs dans $\mathbb{R}^n$, et F est une fonction à variables dans $\mathbb{R}^n \times \mathbb{R}$ et à valeurs dans $\mathbb{R}^n$.

Exemple 2.1.1.

L'équation

$$y' + 3ty = e^t$$

correspond à la fonction

$$F : (x, t) \mapsto e^t - 3tx.$$

L'équation

$$\begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} = \begin{pmatrix} 2x(t) + 3ty(t) \\ x(t) + e^t \end{pmatrix}$$

correspond à la fonction

$$F : \left(\begin{pmatrix} x \\ y \end{pmatrix}, t \right) \mapsto \begin{pmatrix} 2x + 3ty \\ x + e^t \end{pmatrix}.$$

Les vecteurs seront implémentés par les objets de type `array` de `numpy`.

Dans son utilisation la plus simple, la fonction `odeint` prend trois arguments :

- ▷ une fonction F , qui prend elle-même en argument un vecteur X et un flottant t , et renvoie $F(X, t)$;
- ▷ un vecteur y_0 (consition initiale) ;
- ▷ une liste de temps t .

Cette fonction renvoie une liste de vecteurs de même longueur que t . Si

$$t = [t_0, \dots, t_{n-1}],$$

alors un appel de `odeint(F, t0, t)` renvoie une liste

$$X = [X_0, \dots, X_{n-1}]$$

où $X_0, \dots, X_{n-1}$ sont des approximations de la solution du problème de Cauchy

$$X'(t) = F(X(t), t), \quad X(t_0) = y_0.$$

2.2. Exemple : équation d'ordre 1

On cherche à résoudre l'équation

$$y'(t) = t \sin(y(t)),$$

avec la condition initiale $y(0) = 1$, sur l'intervalle de temps $[0, 4]$.

Ici, on a

$$F : (x, t) \mapsto t \sin(x).$$

On peut donc utiliser tout simplement la fonction `odeint`.

```
1 import numpy as np
2 from scipy.integrate import odeint
3 import matplotlib.pyplot as plt
4
5 def F(x, t):
6     return t*np.sin(x)
7
8 y0=1.
9 t_vec = np.linspace(0,4)
10 X_vec = odeint(F,y0,t_vec)
11
12 plt.clf()
13 plt.plot(t_vec,X_vec)
14 plt.grid()
15 plt.axis("equal")
16 plt.show()
```

On obtient alors le tracé suivant (voir figure 1).

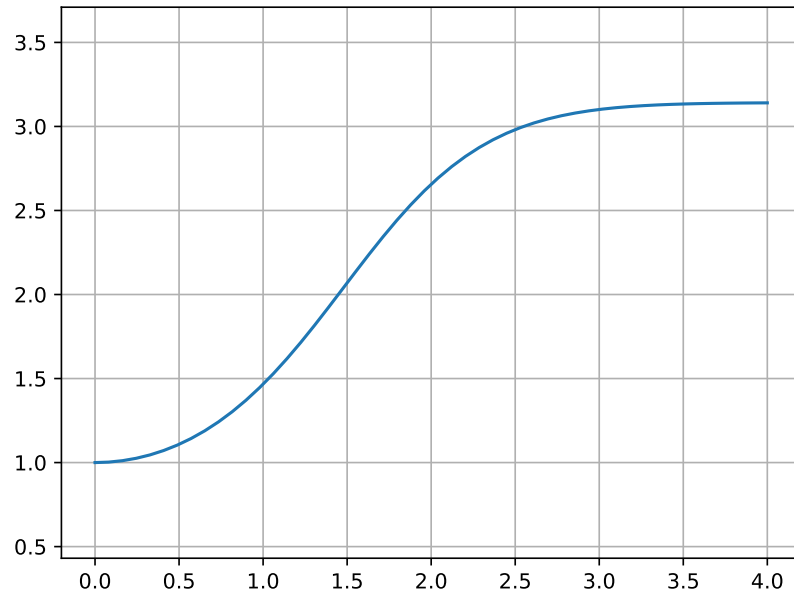
2.3. Exemple : équation d'ordre 2

On considère l'équation régissant les oscillations du pendule simple (en reprenant les notations physiciennes) :

$$\ddot{\theta} + \omega_0^2 \sin(\theta) = 0,$$

où ω_0 est la pulsation propre du système (on a $\omega_0 = \sqrt{\frac{g}{\ell}}$, où g est l'accélération de la pesanteur et où ℓ est la longueur du pendule).

On prendra $\ell = 1m$, $g = 9,81ms^{-2}$ et les conditions initiales $\theta(0) = \frac{\pi}{3}$ et $\dot{\theta}(0) = 0s^{-1}$.

FIGURE 1 – Résolution approchée de $y'(t) = t \sin(y(t))$ par `odeint`

Ici, on vectorialise le problème en

$$X(t) = \begin{pmatrix} \theta(t) \\ \dot{\theta}(t) \end{pmatrix},$$

soit

$$X'(t) = \begin{pmatrix} \dot{\theta}(t) \\ \ddot{\theta}(t) \end{pmatrix} = \begin{pmatrix} \dot{\theta}(t) \\ -\omega_0^2 \sin(\theta(t)) \end{pmatrix}$$

On a donc

$$F : \left(\begin{pmatrix} x \\ y \end{pmatrix}, t \right) \mapsto \begin{pmatrix} y \\ -\omega_0^2 \sin(x) \end{pmatrix}.$$

```

17 import numpy as np
18 from scipy.integrate import odeint
19 import matplotlib.pyplot as plt
20
21 l = 1 #m
22 g = 9.81 #ms**-2
23 omega0 = np.sqrt(g/l)
24
```

```

25 X0 = np.array([np.pi/3, 0.])
26
27 def F(X, t):
28     x, y = X
29     return np.array([y, -omega0**2*np.sin(x)])
30
31 t_vec = np.linspace(0, 10, num=100)
32 X_vec = odeint(F, X0, t_vec)

```

Remarquons qu'alors `X_vec` est un vecteur (que l'on peut voir comme une liste), dont les composantes sont des vecteurs de la forme $\begin{pmatrix} \theta_i \\ \dot{\theta}_i \end{pmatrix}$. Afin de tracer la trajectoire, ou bien un portrait de phase, il convient de récupérer au préalable les premières coordonnées de ces vecteurs (les angles) et les secondes coordonnées (les vitesses angulaires).

```

33 Theta_liste = [X[0] for X in X_vec]
34 Thetap_liste = [X[1] for X in X_vec]

```

On peut alors tracer l'évolution de l'angle θ en fonction du temps (voir figure 2).

```

35 plt.clf()
36 plt.plot(t_vec, Theta_liste)
37 plt.grid()
38 plt.axis("equal")
39 plt.show()

```

On peut aussi le portrait de phase ($\dot{\theta}$ en fonction de θ) (voir figure 3).

```

40 plt.clf()
41 plt.plot(Theta_liste, Thetap_liste)
42 plt.grid()
43 plt.axis("equal")
44 plt.show()

```

2.4. Méthode d'Euler

La méthode d'Euler est la méthode numérique la plus élémentaire permettant de déterminer des solutions approchées d'une équation différentielle d'ordre 1, que l'on écrit

$$X'(t) = F(X(t), t), \quad (\mathcal{E})$$

avec la condition $X(t_0) = X_0$. Ici, X est une fonction vectorielle, mais nous la représenterons comme une fonction réelle.

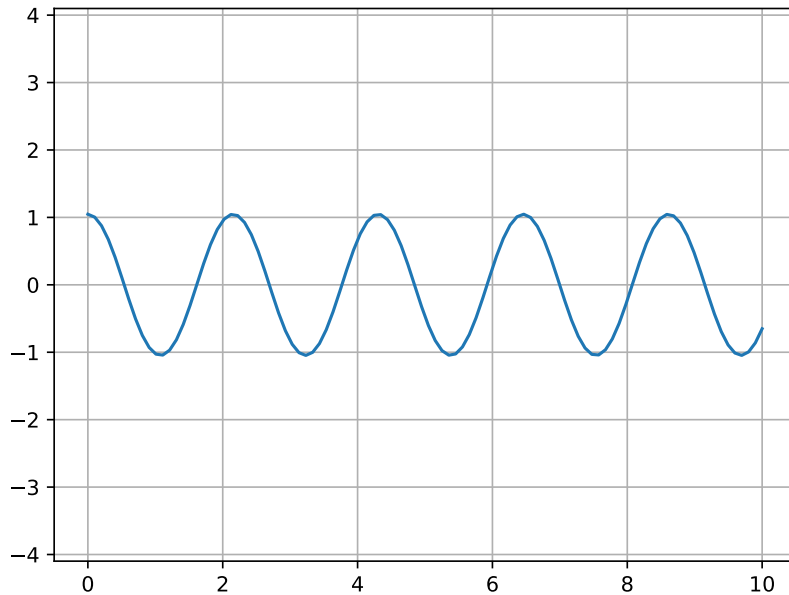


FIGURE 2 – Pendule simple : trajectoire angulaire

Pour $t = t_0$, on connaît la valeur de la solution : c'est X_0 . On essaie maintenant de trouver une valeur approchée de $X(t_1)$, où $t_1 > t_0$. On pourra noter $h = t_1 - t_0$, *i.e.* $t_1 = t_0 + h$. Cette quantité h est le pas temporel.

L'idée de la méthode d'Euler est d'approcher la courbe de la solution X , inconnue, par sa tangente (voir figure 4).

On sait en effet que sa tangente passe par le point de coordonnées (t_0, X_0) et est de pente

$$X'(t_0) = F(X_0, t_0).$$

Le point de cette tangente d'abscisse $t_1 = t_0 + h$ a pour ordonnée

$$X_1 = X_0 + hF(X_0, t_0).$$

On itère alors ce processus. Si la liste des temps $t = [t_0, \dots, t_n]$ est donnée, comme pour `odeint`, on pose pour tout $0 \leq i < n$:

$$h_i = t_{i+1} - t_i,$$

et l'on a alors

$$X_{i+1} = X_i + h_i F(X_i, t_i).$$

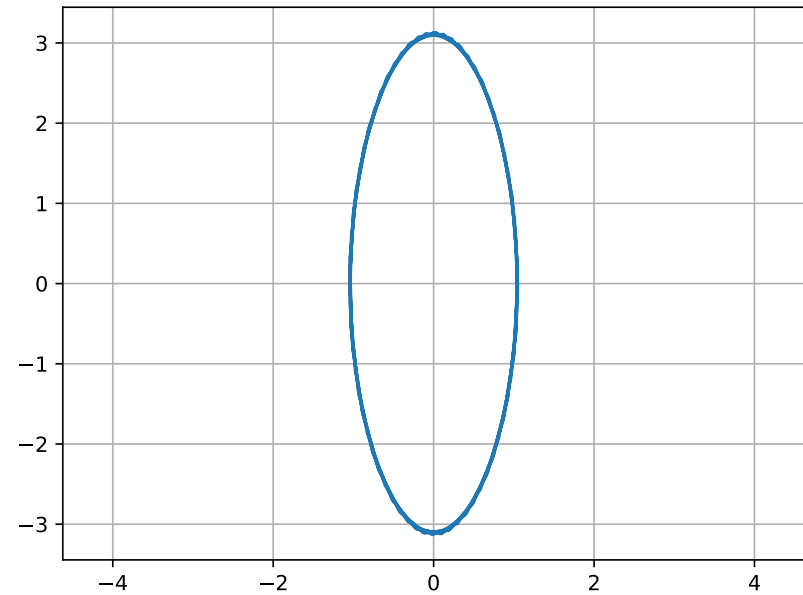


FIGURE 3 – Pendule simple : portrait de phase

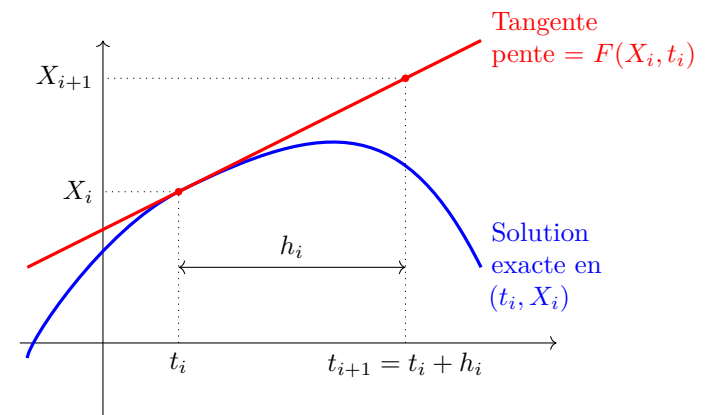


FIGURE 4 – Représentation de la méthode d'Euler

Si le pas temporel h est fixé, on a alors pour tout i :

$$t_i = t_0 + ih,$$

et alors

$$X_{i+1} = X_i + hF(X_i, t_i).$$

Remarque 2.4.1 (schéma numérique explicite d'Euler).

On prend souvent le point de vue suivant pour expliquer cette méthode d'Euler : on remplace dans l'équation différentielle le terme X' par le taux d'accroissement $\frac{X_{i+1} - X_i}{h_i}$, ce qui revient à résoudre l'équation

$$\frac{X_{i+1} - X_i}{h_i} = F(X_i, t_i).$$

Remarque 2.4.2 (schéma numérique implicite d'Euler).

Il existe d'autres manières de mettre en œuvre la méthode d'Euler. Par exemple, la méthode *implicite* revient à résoudre l'équation

$$\frac{X_{i+1} - X_i}{h_i} = F(X_{i+1}, t_{i+1}).$$

La résolution de cette équation est souvent plus compliquée (il y a une équation à résoudre), mais la solution fournie est souvent de meilleure qualité que la solution explicite.

Si l'on subdivise l'intervalle de résolution $[a, b]$ en n segments, cela revient à prendre un pas constant

$$h = \frac{b - a}{n}.$$

Dans tous les cas, on peut simplement écrire une fonction qui effectue ce calcul, et renvoie en sortie la liste des positions $[X_0, \dots, X_n]$.

a. Code : liste de temps fixée

Voici un premier exemple ayant les mêmes spécifications que la fonction `odeint`, *i.e.* qui prend en argument la fonction F , la condition initiale X_0 (que l'on suppose vectorielle) et la liste des temps t .

```
45 def euler(F, X0, t):
46     X = X0
47     L = [X]
48     n = len(t)
49     for i in range(1, n):
50         h = t[i] - t[i-1]
51         X = X + h * F(X, t[i-1])
52         L.append(X)
53     return L
```

b. Code : nombre de segments fixé

On peut aussi écrire cela en fonction du nombre n de points que l'on utilise pour subdiviser régulièrement le segment $[a, b]$. On a alors le pas de temps $h = \frac{b - a}{n}$ et l'on va calculer cela en les temps

$$t_i = a + ih.$$

```
54 def euler(F, X0, a, b, n):
55     h = (b - a) / n
56     X = X0
57     L = [X]
58     for i in range(n - 1):
59         t = a + i * h
60         X = X + h * F(X, t)
61         L.append(X)
62     return L
```

c. Code : pas fixé

On peut aussi écrire cela en fonction du pas h . On a toujours

$$t_i = a + ih,$$

mais il convient de s'arrêter juste avant d'avoir $t_i > b$.

```
63 def euler(F, X0, a, b, h):
64     t = a
65     X = X0
66     L = [X]
67     while t + h <= b:
68         t = t + h
69         X = X + h * F(X, t)
70         L.append(X)
71     return L
```

On remarquera que la méthode d'Euler est bien moins précise qu'`odeint` (voir figure 5).

2.5. Exemple d'EDP : l'équation de la chaleur

On peut reprendre les idées précédentes pour résoudre une équation aux dérivées partielles. Il suffit de remplacer les dérivées partielles successives par leurs équivalents

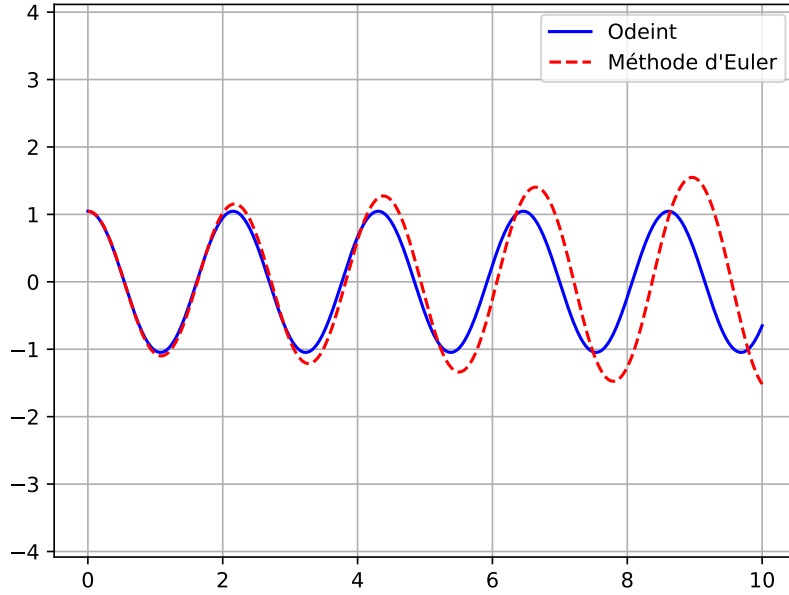


FIGURE 5 – Pendule simple : comparaison des solutions données par la méthode d'Euler et par odeint.

discrets, comme ci-dessus. Par exemple, une dérivée du type $\frac{\partial y}{\partial t}$ sera remplacée par un terme du type $\frac{y_{i+1} - y_i}{\Delta t_i}$, et une dérivée partielle du type $\frac{\partial^2 y}{\partial t^2}$ sera remplacée par un terme du type $\frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta t_i^2}$. Ce « remplacement » n'est pas toujours unique : le choix d'un type de remplacement particulier est un *schéma numérique*. Il reste alors souvent à résoudre les équations fournies par ce schéma. Nous allons détailler un exemple simple : l'équation de la chaleur sur une barre métallique.

On modélise l'échauffement d'une barre d'aluminium. On note $T(x, t)$ la température de la barre au point d'abscisse x et au temps t . On suppose que l'extrémité droite de la barre est de température fixée $T_1 = T(1, t) = 273,15$ K et que l'extrémité gauche est initialement à une température $T_0(0) = T(0, 0) = 273,15$ K, et qu'ensuite la barre subit un échauffement continu selon le modèle suivant : pour tout temps $t \geq 0$, avec $\omega = 1 \text{ Ks}^{-1}$ et $T_{fin} = T(0, 100) = 373,15$ K,

$$T_0(t) = T(0, t) = \begin{cases} T_0(0) + \omega t & \text{si } t \leq 100 \text{ s,} \\ T_{fin} & \text{si } t > 100 \text{ s.} \end{cases}$$

Au temps $t = 0$ s, toute la barre est à température $T(x, 0) = T_1$. Avec $\kappa = 98,8 \cdot 10^{-6} \text{ m}^2\text{s}^{-1}$, la température respecte l'équation de la chaleur :

$$\frac{\partial T}{\partial t}(x, t) = \kappa \frac{\partial^2 T}{\partial x^2}(x, t).$$

Pour résoudre de manière approchée cette équation différentielle, on discrétise spatialement la barre en la divisant en N_x segments. Le pas spatial de discrétisation sera donc $h_x = \frac{1}{N_x} \text{ m}$. On se donne aussi un pas temporel h_t (exprimé en s). Ainsi, on va calculer la température :

$$T_{i,j} = T(ih_x, jh_t).$$

Les conditions initiales se traduisent de la manière suivante : pour tout i, j ,

$$T_{i,0} = T_{N_x,j} = T_1 = 273,15 \text{ K,}$$

et, exprimée en K,

$$T_{0,j} = \begin{cases} 273,15 + h_t j & \text{si } j \leq \frac{100}{h_t}, \\ 373,15 & \text{si } j > \frac{100}{h_t}. \end{cases}$$

On approche alors la quantité $\frac{\partial T}{\partial t}(ih_x, jh_t)$ par

$$\frac{T_{i,j+1} - T_{i,j}}{h_t}.$$

On approche la quantité $\frac{\partial T}{\partial x}(ih_x, jh_t)$ par

$$\frac{T_{i+1,j} - T_{i,j}}{h_x}$$

et donc la quantité $\frac{\partial^2 T}{\partial x^2}(ih_x, jh_t)$ par

$$\frac{\frac{T_{i+1,j} - T_{i,j}}{h_x} - \frac{T_{i,j} - T_{i-1,j}}{h_x}}{h_x} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{h_x^2}.$$

On résout donc

$$\frac{T_{i,j+1} - T_{i,j}}{h_t} = \kappa \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{h_x^2},$$

ce qui donne pour tout $i \in \llbracket 1, N_x - 1 \rrbracket$ et pour tout $j > 0$:

$$T_{i,j+1} = T_{i,j} + \kappa h_t \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{h_x^2}.$$

On peut donc calculer cela de la manière suivante, étant donné h_x (ou N_x) et h_t :

- ▷ on crée une matrice T ayant $N_x + 1$ lignes et autant de colonnes que d'instant de calculs ;
- ▷ on initialise la matrice T en remplissant la première colonne ($t = 0$ s), la première ligne ($x = 0$ m) et la dernière ligne ($x = 1$ m) ;
- ▷ on calcule tous les coefficients de la matrice T en les calculant colonne par colonne, en utilisant la formule précédente.

On obtient par exemple les courbes suivantes (voir figures 6 et 7).

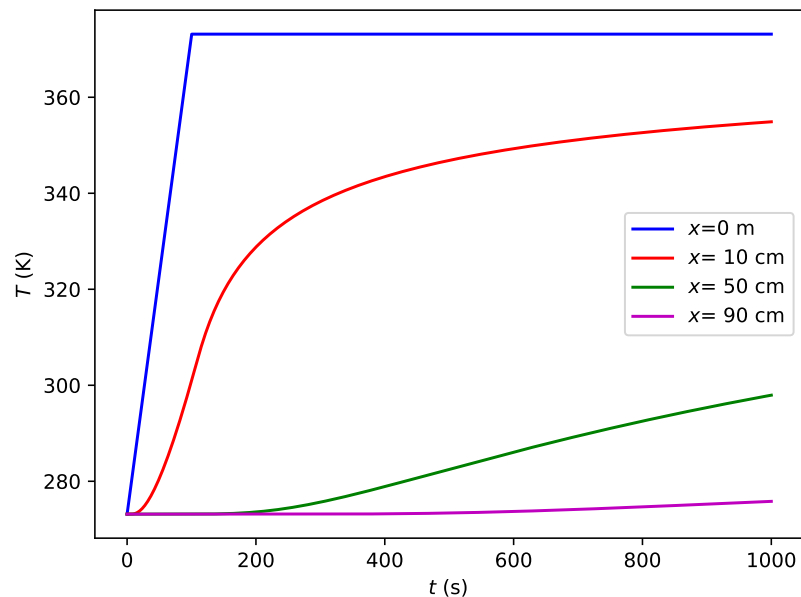


FIGURE 6 – Équation de la chaleur : solution en fonction de t .

- Exercice 2.5.1.** 1. Mettre en œuvre la méthode précédente, en calculant la matrice T . On prendra $N_x = 10^2$, $h_x = 10^{-2}$ m, $h_t = 1$ s et l'on calculera jusqu'au temps $t = 10^3$ s, ce qui reviendra à prendre $N_t = 10^3$ et à construire une matrice T à $N_x + 1$ lignes et $N_t + 1$ colonnes.
2. Produire des graphiques semblables aux courbes des figures 6 et 7.

Remarque 2.5.2.

On utilisé un schéma numérique qui remplace le terme $\frac{\partial T^2}{\partial x^2}$ par des termes faisant

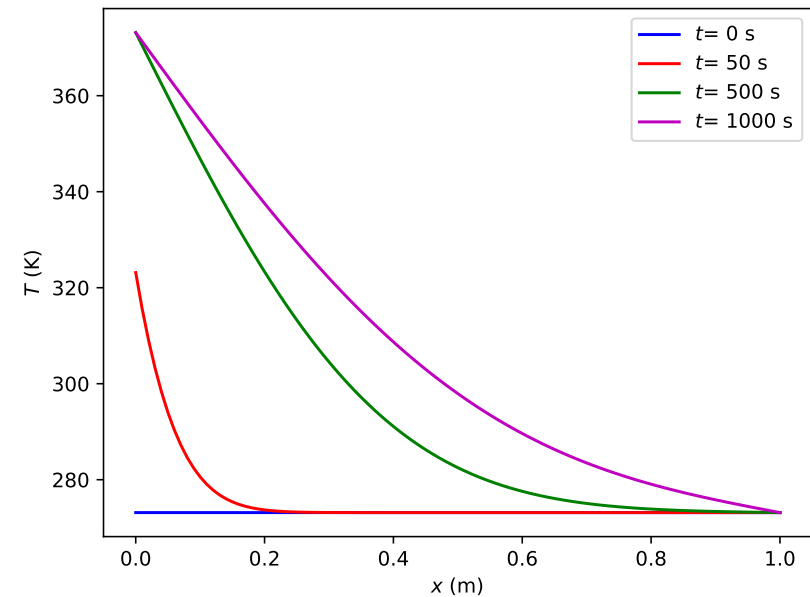


FIGURE 7 – Équation de la chaleur : solution en fonction de x .

intervenir des $T_{\dots,j}$. On pouvait aussi le faire en $j + 1$, ce qui donne l'équation

$$\frac{T_{i,j+1} - T_{i,j}}{h_t} = \kappa \frac{T_{i+1,j+1} - 2T_{i,j+1} + T_{i-1,j+1}}{h_x^2},$$

On voit alors que le vecteur $(T_{0,j+1}, \dots, T_{N_x,j+1})$ est solution d'un système linéaire, que l'on peut résoudre numériquement. Cela dépasse toutefois le cadre de ce cours.

3. Intégrales

3.1. Fonction quad

La bibliothèque `scipy.integrate` fournit la fonction `quad`, qui prends trois arguments :

- ▷ une fonction `f` ;
- ▷ un flottant `a` (ou éventuellement la valeur `-inf` de `numpy`) ;
- ▷ un flottant `b` (ou éventuellement la valeur `inf` de `numpy`).

Un appel de `quad(f,a,b)` renvoie alors un couple constitué

▷ une valeur approchée de $\int_a^b f(t) dt$;

▷ un majorant de l'erreur réalisée lors de cette approximation.

Prenons comme exemple la fonction $f : t \mapsto e^{-t^2}$. Nous allons calculer des valeurs approchées des intégrales

$$I = \int_0^1 e^{-t^2} dt, \quad J = \int_0^{+\infty} e^{-t^2} dt.$$

On utilise pour cela le code suivant.

```
72 from scipy.integrate import quad
73 import numpy as np
74
75 def f(x) :
76     return np.exp(-x**2)
77
78 I, eI = quad(f,0,1)
79 J, eJ = quad(f,0,np.inf)
```

On sait que

$$J = \frac{\sqrt{\pi}}{2}.$$

On peut donc vérifier la qualité de l'approximation donnée.

```
80 test_J = abs(J - .5*np.pi**.5) < eJ
```

3.2. Méthode des rectangles

Le fait d'approcher une intégrale par la somme des aires de rectangles dont les bases forment une subdivision régulière de $[a, b]$ s'appelle la *méthode des rectangles* (voir la figure 8 pour un exemple). Il y a une infinité de manières de choisir la hauteur de ces rectangles. La figure 8 correspond à la situation où la hauteur de chaque rectangle est fixée sur son côté gauche : on parle de la méthode des rectangles à gauche.

On approche donc $\int_a^b f(t) dt$ par l'une des sommes

$$S_n = \frac{b-a}{n} \sum_{k=0}^{n-1} f\left(a + k \frac{b-a}{n}\right) \quad \text{ou} \quad S'_n = \frac{b-a}{n} \sum_{k=1}^n f\left(a + k \frac{b-a}{n}\right),$$

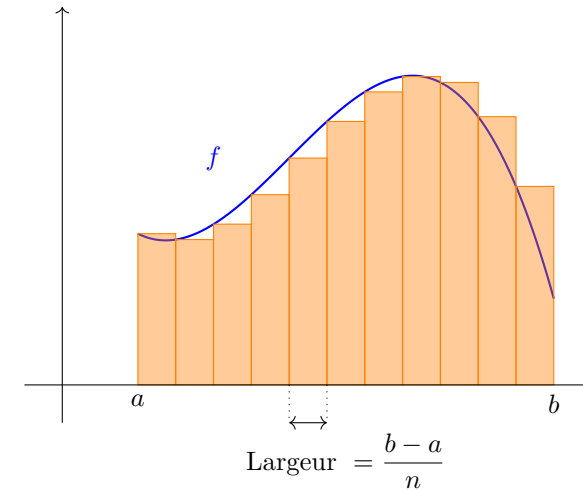


FIGURE 8 – Exemple de méthode des rectangles ($n = 11$, à gauche).

en fonction que l'on utilise une méthode des rectangles à gauche (S_n) ou à droite (S'_n).

Le théorème de convergence des sommes de Riemann montre que l'approximation d'une intégrale réalisée par la méthode des rectangles converge vers la valeur de l'intégrale.

On la calcule aisément (ici pour la méthode des rectangles à gauche, pour S_n).

```
81 def rectangle(f, a, b, n) :
82     S = 0
83     pas = (b-a)/n
84     for k in range(n) :
85         S = S + f(a+k*pas)
86     return S*pas
```

On peut alors l'appliquer à la fonction f codée précédemment.

```
87 I2 = rectangle(f,0,1,1000)
```

Comme pour la méthode d'Euler, on peut adapter aisément le code précédent à une situation où le pas n'est pas constant. Par exemple, si on donne une liste de temps $t = [t_0, \dots, t_{n-1}]$, alors la méthode des rectangles (à gauche) approchera

l'intégrale de f par la valeur

$$\sum_{i=0}^{n-2} (t_{i+1} - t_i) f(t_i).$$

On pourra alors écrire le code suivant.

```

88 def rectangle(f, t) :
89     S = 0
90     n = len(t)
91     for i in range(n-1) :
92         dt = t[i+1] - t[i]
93         S = S + f(t[i])*dt
94     return S

```

3.3. Méthode des trapèzes

Le fait d'approcher une intégrale par la somme des aires des trapèzes dont les bases forment une subdivision régulière de $[a, b]$ s'appelle la *méthode des trapèzes* (voir la figure 9 pour un exemple).

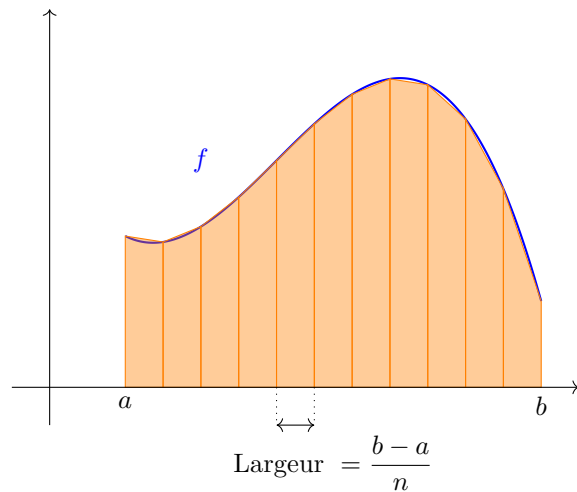


FIGURE 9 – Exemple méthode des trapèzes ($n = 11$).

à droite :

$$T_n = \frac{1}{2}(S_n + S'_n) = \frac{b-a}{2n}(f(a) + f(b)) + \frac{b-a}{n} \sum_{k=1}^{n-1} f\left(a + k \frac{b-a}{n}\right).$$

Pour les mêmes raisons que précédemment, T_n converge vers $\int_a^b f(t) dt$, mais de manière plus rapide qu'avec la méthode des rectangles (sous quelques hypothèses, très souvent vérifiées).

On la calcule aisément.

```

95 def trapezes(f, a, b, n) :
96     pas = (b-a)/n
97     S = (f(a)+f(b))/2
98     for k in range(1,n) :
99         S = S + f(a+k*pas)
100    return S*pas

```

On peut alors l'appliquer à la fonction f codée précédemment.

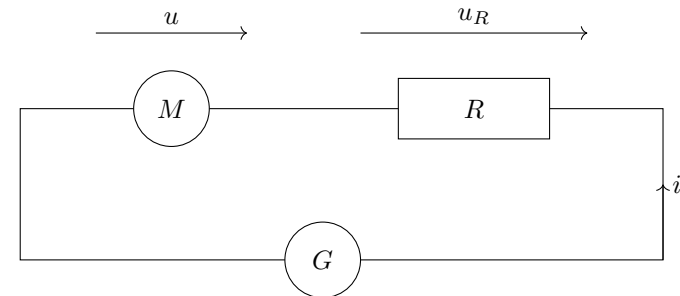
```

101 I3 = trapezes(f,0,1,1000)

```

3.4. Exercice

On considère le montage électrique suivant : un générateur de tension est monté en série avec un moteur électrique et une résistance ($R = 100 \Omega$), comme montré ci-dessous.



On mesure en fonction du temps la tension $u(t)$ aux bornes du moteur et la tension $u_R(t)$ aux bornes de la résistance, ce qui permet d'obtenir une mesure de

l'intensité $i(t)$ parcourant le circuit, jusqu'à un temps t_0 . On peut donc calculer l'énergie électrique reçue par le moteur par :

$$E = \int_0^{t_0} u(t)i(t) dt.$$

Une carte d'acquisition a fait des relevés de ces mesures, et a stocké dans le fichier `mesuresTension.txt`, pour chaque mesure :

- ▷ l'instant t de la mesure (exprimé en ms) ;
- ▷ la tension $u(t)$ aux bornes du moteur (exprimée en V) ;
- ▷ la tension $u_R(t)$ aux bornes de la résistance (exprimée en V).

La première ligne du fichier correspond aux titres des colonnes, chacune des lignes suivantes correspondant à une mesure. Voici par exemple les quatre premières lignes de ce fichier.

```
# t (ms), u (V), u_R (V)
0,0.0,0.0
8,12.94679,1.454499
17,12.19301,3.124836
```

1. Écrire un script permettant de lire les données contenues dans le fichier `mesuresTension.txt`. On affectera les mesures de temps à une liste `t`, les mesures de la tension u à une liste `u` et les mesures de tension u_R à une liste `uR`.
2. Calculer l'énergie reçue par le moteur sur toute la durée de la mesure.
3. Tracer sur un même graphique et en fonction du temps t l'évolution des tensions $u(t)$ et $u_R(t)$.