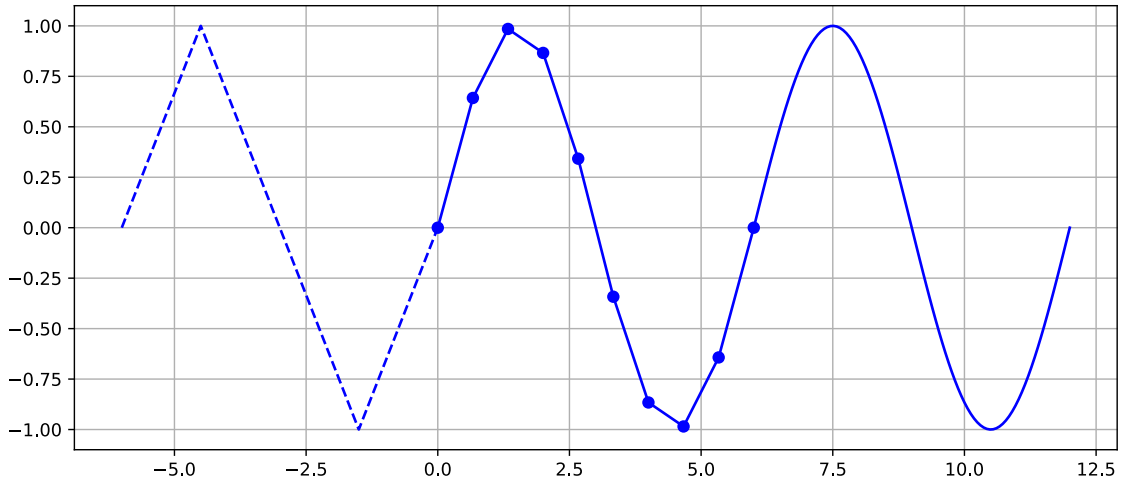
Python pour le supérieur

5) Approximations



Approximations

- Recherche de zéros
  - Dichotomie
  - Méthode de Newton-Raphson
- Calcul différentiel
  - Méthode d'Euler explicite
  - Méthode de Runge-Kutta 4
- Calcul intégral
  - Méthode des rectangles
  - Méthode de Simpson

Zéros : dichotomie

Théorème des valeurs intermédiaires.

Soient $[a, b], a < b$, un intervalle non vide de $\mathbb{R}$ et f une fonction continue de $[a, b]$ dans $\mathbb{R}$.
Alors $\forall u \in [f(a), f(b)], \exists\, c \in [a, b], f(c) = u$

Théorème de Bolzano.

Soit f une fonction continue de $[a, b]$ dans $\mathbb{R}$, telle que $f(a)f(b) \leq 0$. Alors il existe $c \in [a, b]$ tel que $f(c) = 0$

Zéros : dichotomie

```
Entrée : fonction f, entier N

Répéter N fois:
    m ← (a+b)/2

    Si f(a)f(m) < 0:
        b ← m

    Sinon :
        a ← m

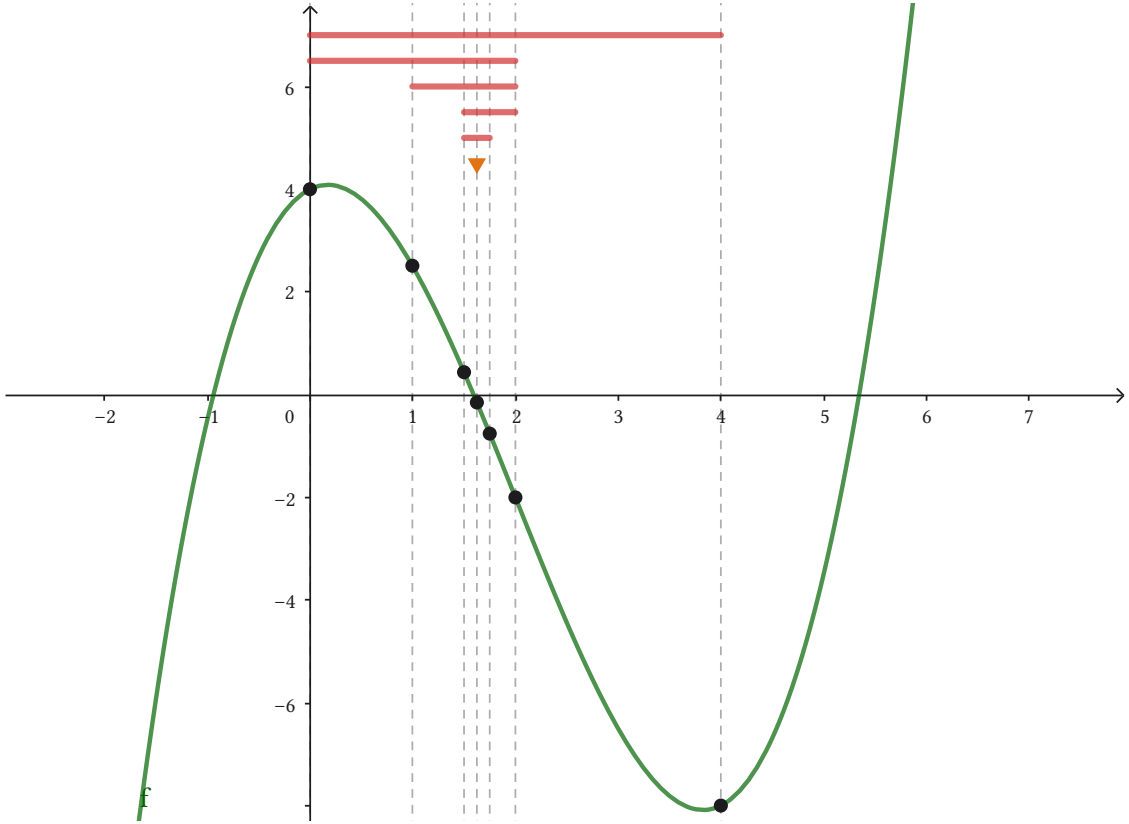
Retourner (a+b)/2
```

On a : $f(c) = 0$ avec $c \in [\alpha, \beta]$ et $\varepsilon = \beta - \alpha = \dfrac{b-a}{2^{N+1}}$

m est une valeur approchée à ε près de c

Zéros : dichotomie

Exemple 1 : $f(x) = \dfrac{x^3}{2} - 3x^2 + x + 4$. Zéro sur $[0\,;4]$?



Zéros : dichotomie

```
1  def f(x):
2      return x**3/2 - 3*x**2 + x + 4
3
4  (a, b, N) = (0, 4, 10)
5
6  for _ in range(N):
7      m = (a+b)/2
8
9      if(f(a)*f(m)<0):
10         b = m
11     else:
12         a = m
13
14 print((a+b)/2, "est une valeur approchée d'un zéro à",
   (b-a)/2**(N+1), "près.")
```

Exécution du code

```
1.591796875 est une valeur approchée d'un zéro à
1.9073486328125e-06 près.
```

Zéros : méthode de Newton-Raphson

Théorème. Soit $(a, b) \in \mathbb{R}^2$, $a < b$. Soit f une fonction $\mathcal{C}^2$ définie sur $[a, b]$ telle que $f(a)f(b) < 0$ et telle que les dérivées première et seconde ne s'annulent pas sur $[a, b]$.

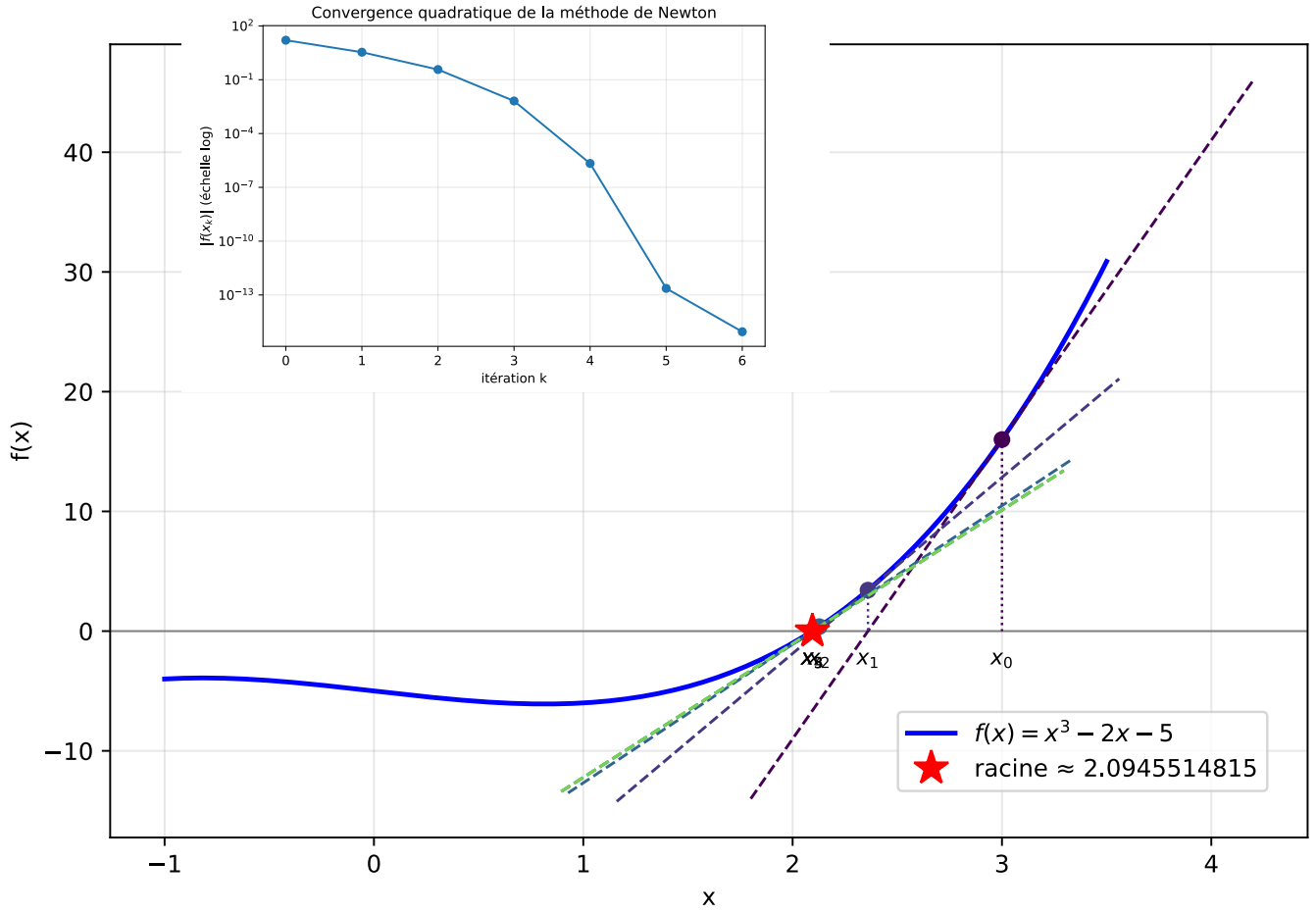
Alors $f(x) = 0$ admet une unique solution α dans $[a, b]$.

De plus, α est la limite de la suite $(x_k)_{k\in\mathbb{N}}$ définie par :

$$\begin{cases} x_0 = a & \text{si } f'f'' < 0 \\ x_0 = b & \text{si } f'f'' > 0 \\ x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)} & \end{cases}$$

et la convergence de cette suite est **quadratique**.

Zéros : méthode de Newton-Raphson



Méthode de Newton : tangentes successives et convergence

Zéros : méthode de Newton-Raphson

Exemple 2 : Calcul approché de la racine cubique d'un réel positif c

On pose $f(x) = x^3 - c$ sur l'intervalle $[0, c+1]$

L'unique zéro de la fonction est $\sqrt[3]{c}$

La fonction dérivée est $f'(x) = 3x^2$

On définit la suite $(x_k)_{k\in\mathbb{N}}$ par :

$$\begin{cases} x_0 = \lfloor \sqrt[3]{c} + 1 \rfloor \\ x_{k+1} = x_k - \dfrac{x_k^3 - c}{3\,x_k^2} \end{cases}$$

Zéros : méthode de Newton-Raphson

```python
c = 3250

x = 0

while(x**3 < c):
    x += 1

for k in range(5):
    x = x - (x**3-c) / (3*x**2)

print(f"La racine cubique de {c} est environ {x}.")
```

Exécution du code

```
La racine cubique de 3250 est environ 14.812480342036853
```

Remarque. `x^3 = 3250.0000000000005`

Zéros : exercices

1. Adapter le code précédent pour calculer la racine n-ième d'un nombre c avec n et c donnés en entrée.

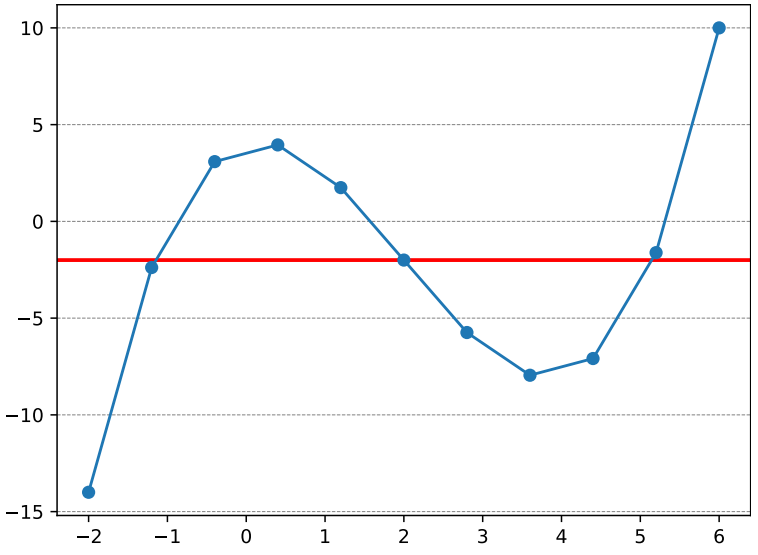
2. Créer un programme qui recherche un zéro par la **méthode de la sécante**.

   Celle-ci consiste à remplacer $f'(x_k)$ par $\dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$

   Choix des premiers termes

   `pagebreak`

3. Représenter sur $[-2\,;6]$ une interpolation linéaire de $f(x) = \dfrac{x^3}{2} - 3x^2 + x + 4$



Le problème de Cauchy

Soit $[a, b]$ un intervalle de $\mathbb{R}$, $a < b$.

On cherche $y : [a, b] \to \mathbb{R}$ tel que

$$\begin{cases} y'(t) = f(t, y(t)) & \text{pour tout } t \in [a, b] \\ y(a) = y_0 & \text{(condition initiale)} \end{cases}$$

avec $f : [a, b] \times \mathbb{R} \to \mathbb{R}$ **continue** et **lipschitzienne (*) en** y.

Le **théorème de Cauchy-Lipschitz** nous assure l'existence et l'unicité de la solution.

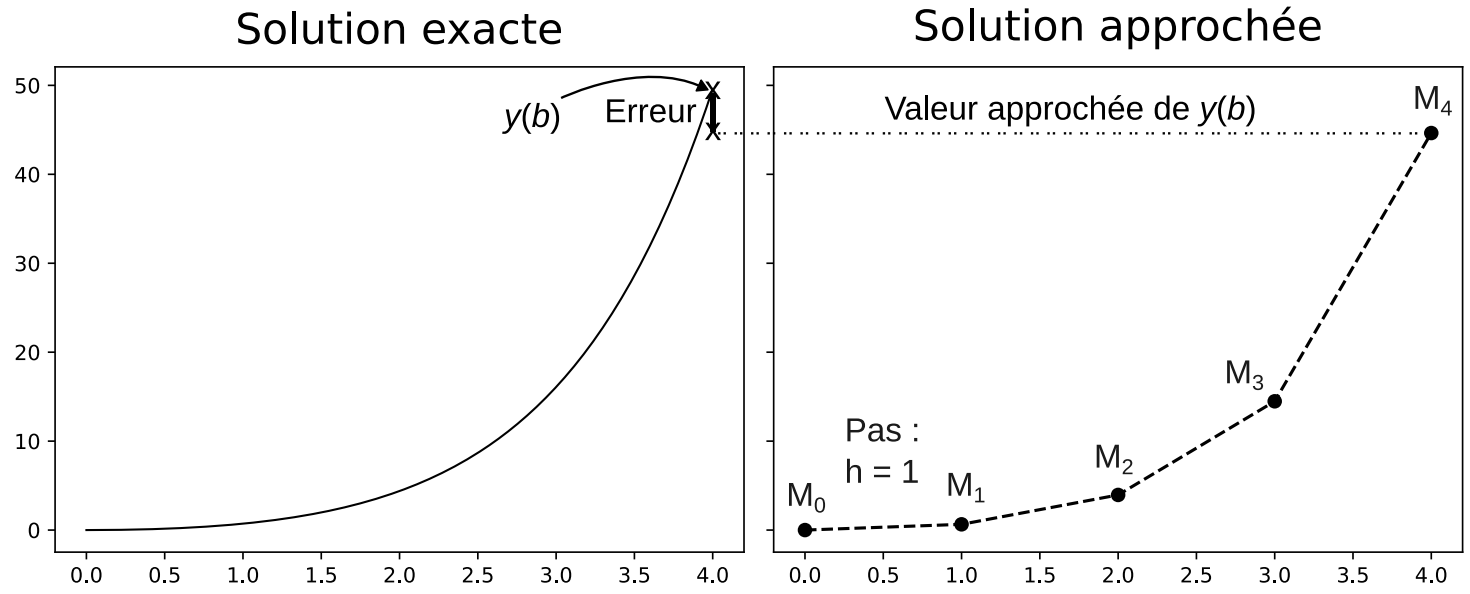
(*) $\exists k > 0, \forall t \in [a, b], \forall (x, y) \in \mathbb{R}^2, |f(t, x) - f(t, y)| \leq k|x - y|$

Méthode d'Euler explicite

Idée. Sur un petit pas h, on remplace la courbe par sa tangente :

$$y(t+h) \approx y(t) + h\,y'(t) = y(t) + h\,f(t, y(t))$$

Convergence. Grâce à la continuité et au caractère lipschitzien de f, on a : $y_n \to y(t_n)$ quand $h \to 0$.



Méthode d'Euler explicite

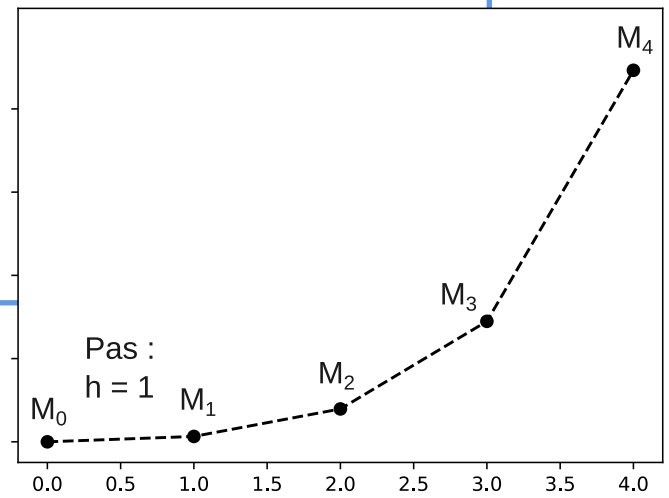
Méthode d'Euler explicite. On pose $h = \dfrac{b-a}{N}$

La suite des abscisses $(t_k)_{0\leq k\leq N}$ des points M_k est définie par $t_k = a + kh$.

Pour trouver leurs ordonnées, on utilise l'approximation de la dérivée :

$$f(t_k, y(t_k)) := y'(t_k) \approx \frac{y(t_{k+1}) - y(t_k)}{t_{k+1} - t_k}$$

$$\text{soit} \quad y(t_{k+1}) \approx y(t_k) + hf(t_k, y(t_k))$$



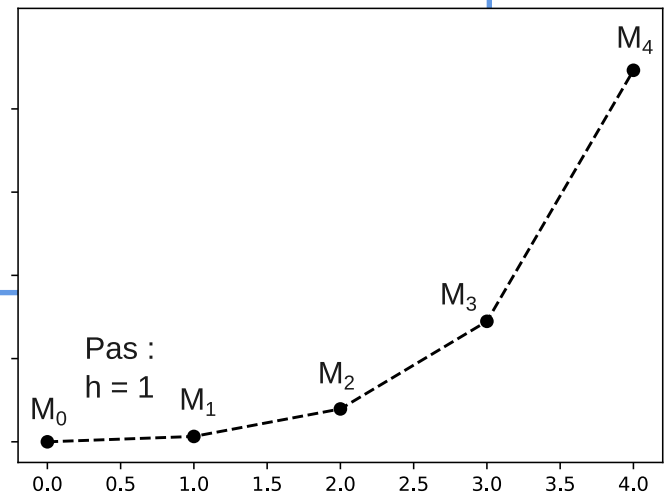
Méthode d'Euler explicite

On définit la suite $(u_k)_{0 \leq k \leq N}$ avec :

$$\begin{cases} u_0 = y(a) \\ u_{k+1} = u_k + hf(t_k, u_k) \end{cases}$$

On a alors $u_k \approx y(t_k)$ pour tout $k \in \{0, ..., N\}$.

La ligne brisée $M_0(t_0, u_0)...M_{N(t_N, u_N)}$ est la courbe d'une fonction continue et affine par morceaux qui approche la courbe de y.



Exemple 3 : y'=-y + cos(t), y(0)=0

Méthode d'Euler explicite

```python
from math import *
import matplotlib.pyplot as plt
import numpy as np

(a, b, N) = (0, 2*pi, 10)

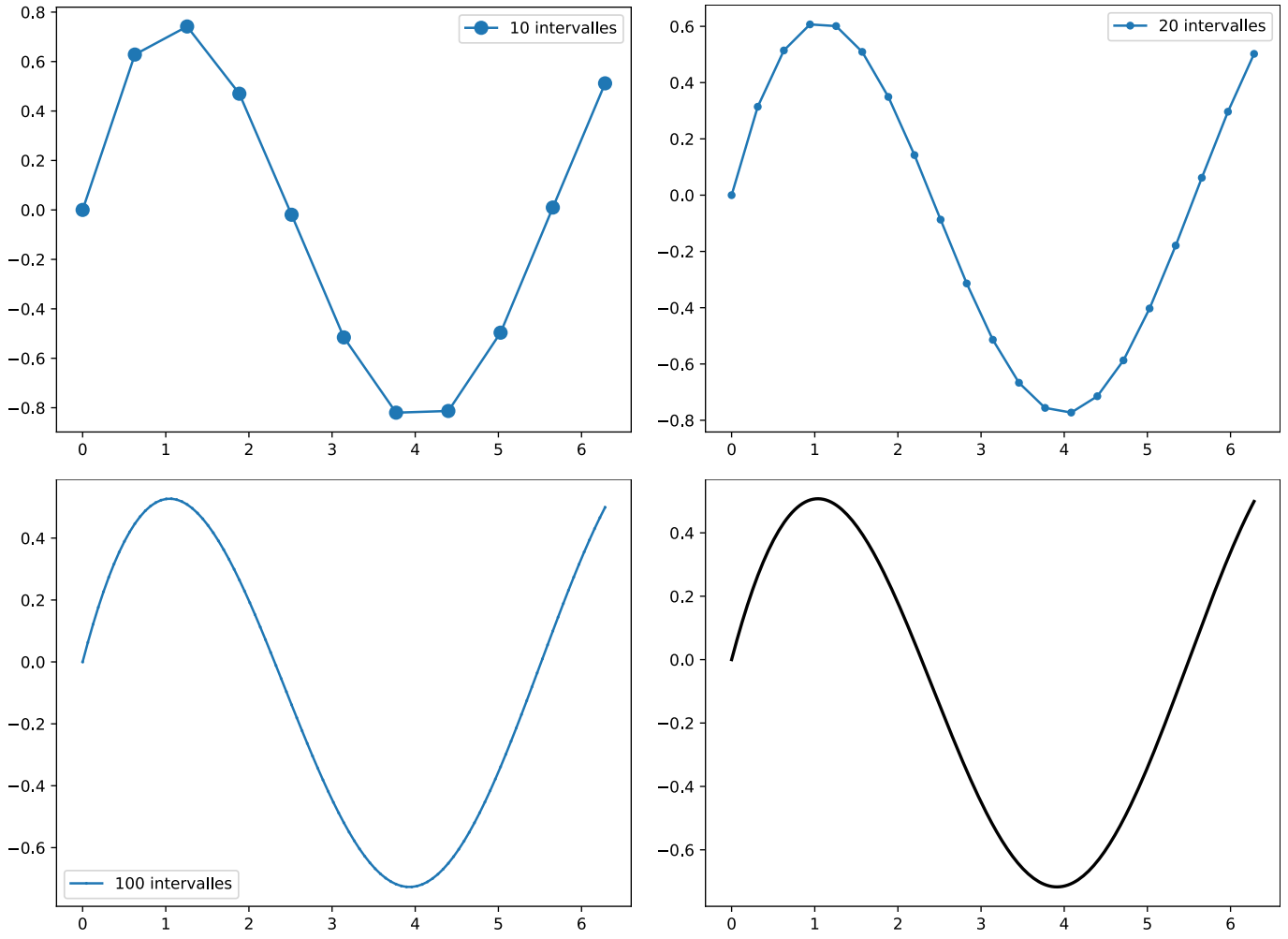
def f(t, y):              # y' = f(t, y)
    return -y + cos(t)

y = 0                     # condition initiale y0 en a
Y = [y]
T = np.linspace(a, b, N+1)
h = (b-a) / N

for k in range(0,N):      # Schéma d'Euler explicite
    y = y + h * f(T[k], y)
    Y.append(y)

plt.plot(T, Y, 'o-', ms=80/N, label=f"{N} intervalles")
plt.legend()
plt.show()
```

Méthode d'Euler explicite



Méthode de Runge-Kutta 4

La méthode d'Euler explicite consistait à poser la relation de récurrence $u_{k+1} = u_k + hf(t_k, u_k)$.

Pour la méthode de Runge-Kutta 4, le schéma est le suivant :

$$u_{k+1} = u_k + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right)$$

avec :

$$k_1 = f(t_k,\ u_k) \qquad k_3 = f\left(t_k + \frac{h}{2},\ u_k + \frac{h}{2}k_2\right)$$

$$k_2 = f\left(t_k + \frac{h}{2},\ u_k + \frac{h}{2}k_1\right) \qquad k_4 = f(t_k + h,\ u_k + hk_3)$$

Comparaison Euler / Runge-Kutta 4

```python
from math import *
import numpy as np

(a, b, N) = (0, 2, 100)
u = 1                          # condition initiale y0
T = np.linspace(a, b, N+1)
h = (b-a)/N

def f(t, y):                   # y' = f(t, y)
    return y

def RK(t, u):                  # Schéma de Runge-Kutta 4
    k1 = f(t, u)
    k2 = f(t+h/2, u+h*k1/2)
    k3 = f(t+h/2, u+h*k2/2)
    k4 = f(t+h, u+h*k3)
    return h/6*(k1 + 2*k2 + 2*k3 + k4)

for k in range(0, N):
    u = u + RK(T[k], u)
...
```

Comparaison Euler / Runge-Kutta 4

```
21 print("Valeur connue :  ", e**2)
22 print("Valeur calculée :", u, "avec", N, "intervalles")
23 print("Erreur :", e**2-u)
```

Exécution du code

```
Valeur connue :   7.3890560989306495
Valeur calculée : 7.389056079552106 avec 100 intervalles
Erreur : 1.9378543214543242e-08
```

On remplace la ligne 20 par `u = u + h*f(T[k], u)`

Exécution du code

```
Valeur connue :   7.3890560989306495
Valeur calculée : 7.244646118252337 avec 100 intervalles
Erreur : 0.14440998067831234
```

Exercices

1. Résoudre l'équation différentielle

$$\begin{aligned} y' &= -y + e^{-t} \\ y(0) &= 0 \end{aligned}$$

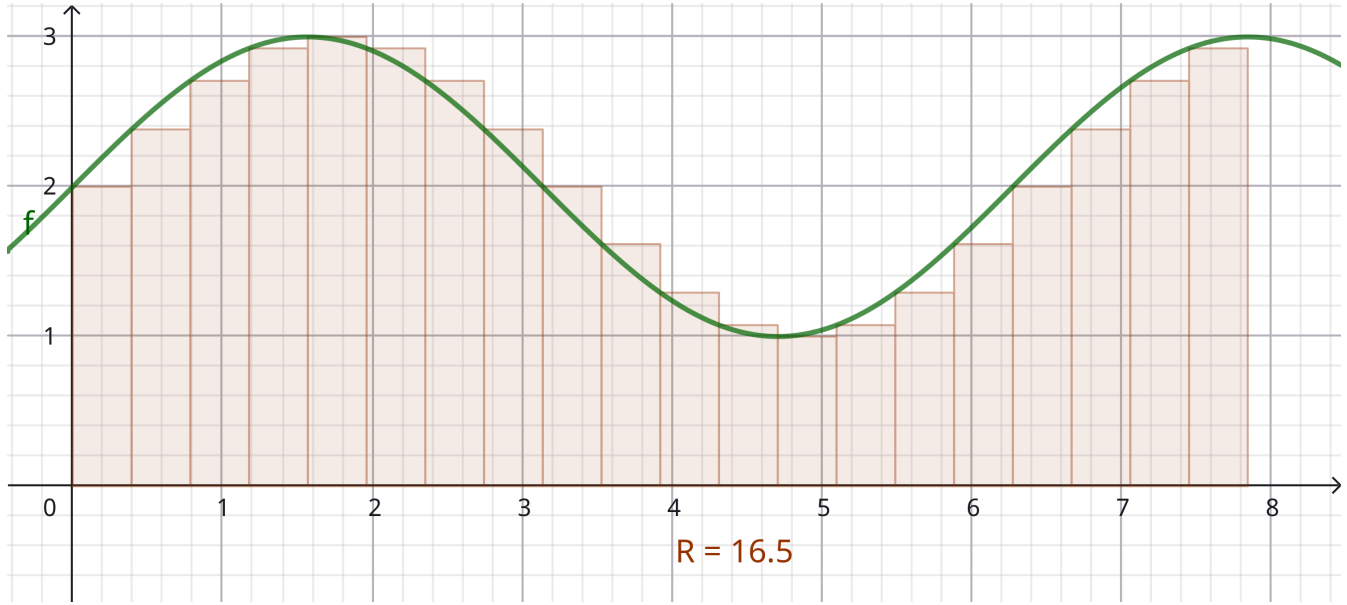
Représenter, sur un même schéma, les courbes sur $[0, 2\pi]$ de la solution exacte et de la solution approchée par Euler explicite avec 40 intervalles.

2. Comparer les valeurs obtenues pour $\sqrt{e}$ avec Euler explicite et Runge-Kutta 4 pour 100 intervalles, et comparer l'erreur en utilisant la valeur pré-enregistrée de Python.

Méthode des rectangles : présentation

On souhaite calculer une **valeur approchée** de l'intégrale d'une fonction f définie sur un intervalle $[a, b]$ grâce à la somme des aires de rectangles.

Généralement, les N rectangles ont une largeur fixe h égale à $\frac{b-a}{N}$ (le « pas ») et une hauteur variable.

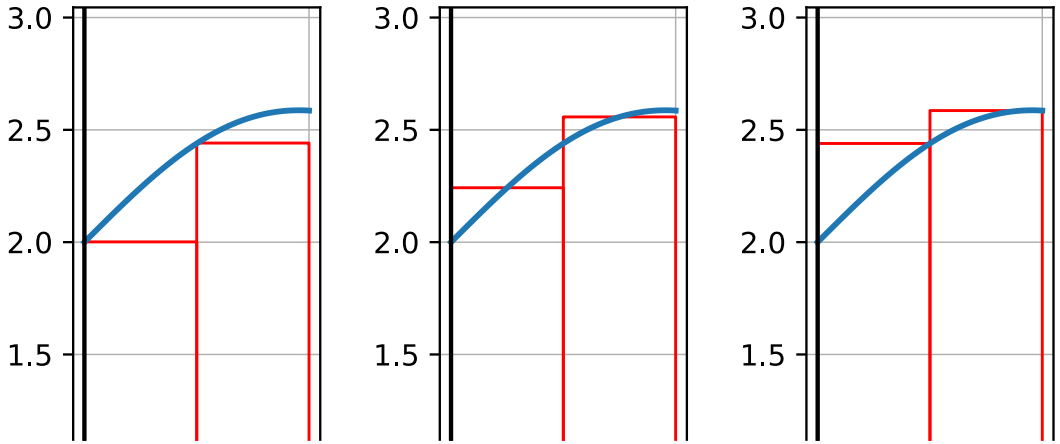


Méthode des rectangles : valeurs

L'abscisse « à gauche » du k-ième rectangle a pour valeur $a + kh$.

L'abscisse « à droite » a pour valeur $a + (k + 1)h$.

La hauteur du k-ième rectangle a pour valeur $f(a + (k + d)h)$ avec $d \in [0, 1]$, décalage relatif (ou « paramètre de position »).



Rectangles gauche d = 0

Rectangles milieu d = 1/2

Rectangles droite d = 1

Méthode des rectangles : exemple

La valeur approchée de $\displaystyle\int_a^b f(x)\ \mathrm{d}x$ donnée par la méthode des rectangles de paramètre d, avec N rectangles est :

$$h\sum_{k=0}^{N-1} f(a+(k+d)h) \quad \text{avec } h=\frac{b-a}{N}$$

Exemple 4 : Calcul de la valeur approchée de

$\displaystyle\int_0^3 \left(3+x-\sqrt{1+x^3}\right)\ \mathrm{d}x$ avec 100 rectangles au point milieu.

Méthode des rectangles : calcul

```
 1  from math import *
 2
 3  def f(x):
 4      return 3 + x - sqrt(1+x**3)
 5
 6  a, b = 0, 3          # intervalle
 7  N, d = 100, 1/2     # nb de rectangles, décalage
 8
 9  h = (b-a)/N          # pas
10  S = 0                # initialisation de la somme
11
12  for k in range(N):
13      S += f(a+(k+d)*h)
14
15  print(h*S)
```

Exécution du code

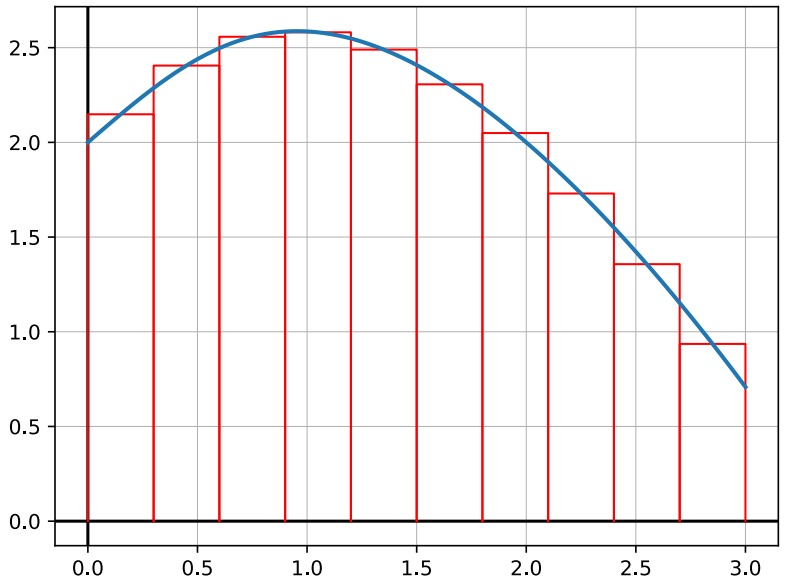
```
6.158688316906336
```

Méthode des rectangles : visualisation

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from math import *
4
5  def f(x):
6      return 3 - sqrt(1+x**3) + x
7
8  a, b = 0, 3
9  N, d = 10, 1/2
10
11 h = (b-a)/N
12
13 # Configuration du graphique
14 plt.grid(lw=0.5)
15 plt.axhline(y=0, c='black')
16 plt.axvline(x=0, c='black')
17
18 # Tracé des rectangles
19 for k in range(N):
20     RX = [[a+k*h], [a+k*h], [a+(k+1)*h], [a+(k+1)*h]]
21     RY = [0, f(a+(k+d)*h), f(a+(k+d)*h), 0]
22     plt.plot(RX, RY, 'r', lw='1')
```

Méthode des rectangles : visualisation

```
24  # Tracé de la courbe
25  X = np.linspace(a, b, 101)
26  Y = [f(x) for x in X]
27  plt.plot(X, Y, lw='2')
28
29  plt.show()
```



Méthode de Simpson : présentation

La **méthode de Simpson** améliore la précision de la méthode des rectangles (majoration de l'ordre de $\frac{(b-a)^5}{N^4}$ contre $\frac{(b-a)^3}{N^2}$).

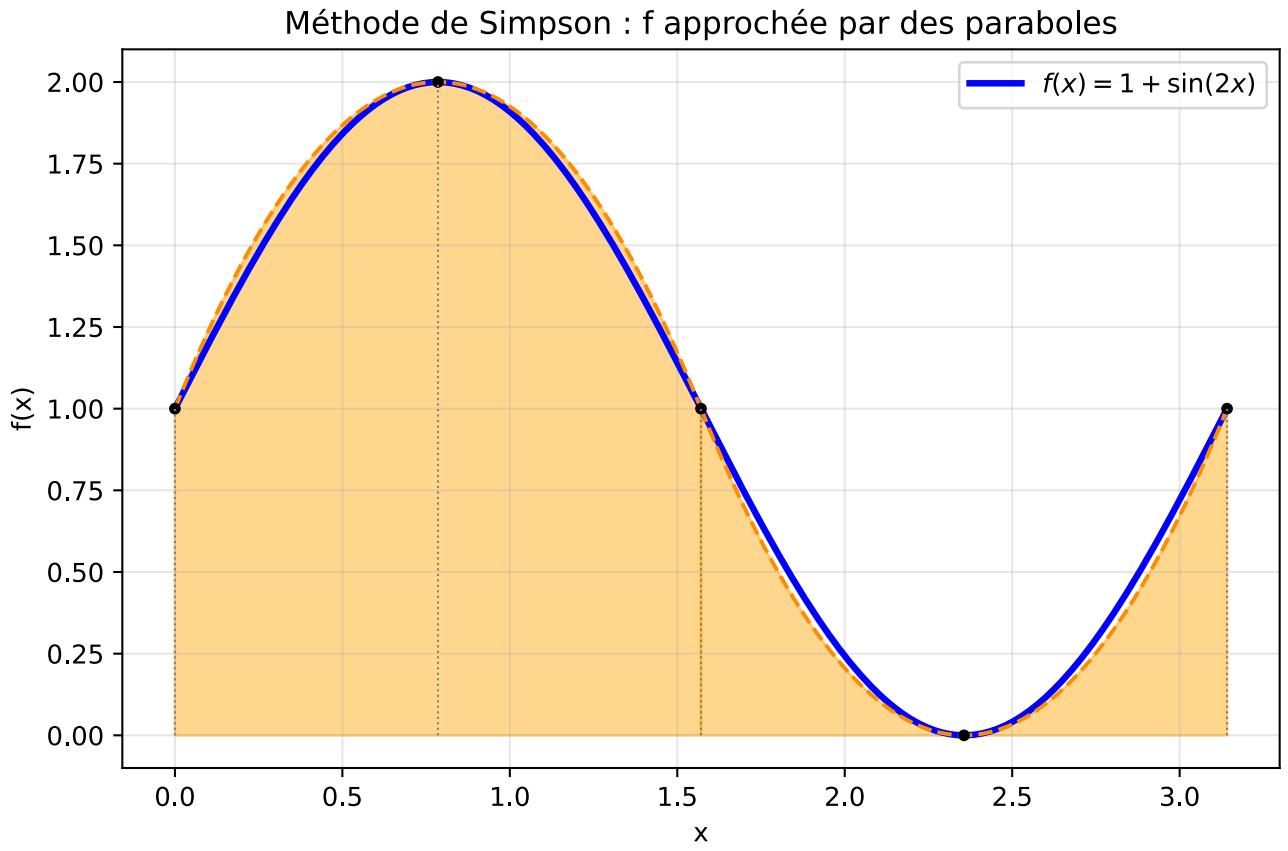
Elle consiste à évaluer f en trois points pour chaque intervalle :

- l'extrémité gauche : $a + kh$
- le milieu : $a + \left(k + \frac{1}{2}\right)h$
- l'extrémité droite : $a + (k+1)h$

en affectant des « poids » : 1 pour les extrémités et 4 pour le milieu.

On peut montrer que cette méthode donne une valeur exacte avec un seul intervalle si la fonction est parabolique (*ie.* de type $x \mapsto px^2 + qx + r$).

Méthode de Simpson : présentation



Méthode de Simpson : présentation

La valeur approchée de $\int_a^b f(x)\,\mathrm{d}x$ donnée par la méthode de Simpson, avec N sous-intervalles, est :

$$\frac{h}{6}\sum_{k=0}^{N-1}\left(f(a+kh)+4f\left(a+\left(k+\frac{1}{2}\right)h\right)+f(a+(k+1)h)\right)$$

soit, sous formule simplifiée :

$$\frac{h}{6}\left(f(a)+f(b)+2\sum_{k=1}^{N-1}f(a+kh)+4\sum_{k=0}^{N-1}f\left(a+\left(k+\frac{1}{2}\right)h\right)\right)$$

Méthode de Simpson : calcul

```python
from math import *

def f(x):
    return 3 - sqrt(1+x**3) + x

a, b, N = 0, 3, 100

h = (b-a)/N
S = f(a) + f(b)

for k in range(1, N):
    S += 2*f(a + k*h)

for k in range(N):
    S += 4*f(a + (k+1/2)*h)

print(S*h/6)
```

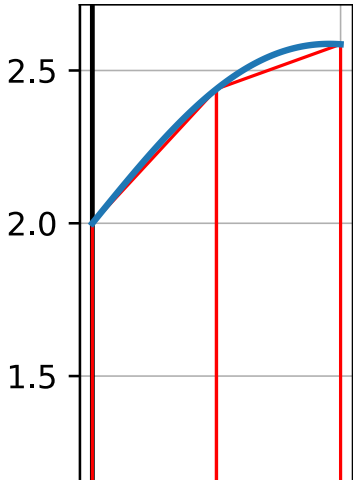
Exécution du code

```
6.158592642459585
```

Exercices

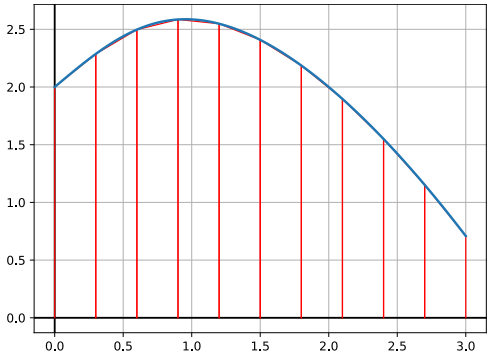
La **méthode des trapèzes** consiste à approcher l'intégrale par une somme d'aires de trapèzes.

1. Créer un programme pour calculer la valeur approchée de $\int_0^3 \left(3 + x - \sqrt{1 + x^3}\right) \, \mathrm{d}x$ avec 100 trapèzes.



2. Adapter le code de visualisation pour afficher des trapèzes au lieu de rectangles.

3. Calculer la valeur exacte de $\int_{-1}^{1} \frac{\pi}{4} \cos\left(\frac{\pi x}{2}\right) \, \mathrm{d}x$ puis comparer cette valeur avec celle calculée par les 3 méthodes pour 100 subdivisions.



Solutions

```python
from math import *
import matplotlib.pyplot as plt
import numpy as np

(a, b) = (0, 2*pi)     # intervalle

def f(t, y):         # y' = f(t, y)
    return -y + cos(t)

def s(t):        # fonction solution
    return 1/2*(cos(t) + sin(t) - e**(-t))

for N in [10, 50, 100, 250]:
    y = s(a)      # condition initiale y0 en a
    Y = [y]
    T = np.linspace(a, b, N+1)
    h = (b-a) / N   # pas

```

Solutions

```
19      for k in range(0,N):    # Schéma d'Euler expl
20          y = y + h * f(T[k], y)
21          Y.append(y)
22
23      print(N,"intervalles : erreur =", round(100*(s(b)-
    Y[-1])/s(b),2), "%")
24      plt.plot(T, Y, ls="--", label=f"{N} intervalles")
25
26  # Tracé de la fonction solution
27  L = np.linspace(a, b, 200)
28  plt.plot(L, [s(t) for t in L], c="black", lw="2")
29
30  plt.legend()
31  plt.show()
```

Fin