Exercice 1.

D'après Taylor-Young, si f est C^2 :

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + o(h^2)$$

Donc si $f''(x) \neq 0$, on a $\frac{f(x+h)-f(x)}{h} - f'(x) \sim \frac{f''(x)}{2}h$.

Si f est C^3 :

$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2}h^2 + \frac{f^{(3)}(x)}{6}h^3 + o(h^3)$

$f(x-h) = f(x) - f'(x)h + \frac{f''(x)}{2}h^2 - \frac{f^{(3)}(x)}{6}h^3 + o(h^3)$

Donc, $f(x+h) - f(x-h) = 2hf'(x) + \frac{f^{(3)}(x)}{3}h^3 + o(h^3)$

$\frac{f(x+h)-f(x-h)}{2h} - f'(x) = \frac{f^{(3)}(x)}{6}h^2 + o(h^2) \sim \frac{f^{(3)}(x)}{6}h^2$ si $f^{(3)}(x) \neq 0$.

Si $f^{(3)}(x) = 0$, l'approximation est en $o(h^3)$.

Ainsi, on préfère approcher $f'(x)$ par $\frac{f(x+h)-f(x-h)}{2h}$.

Exercice 2.

```
def rectangleg(f,a,b,n):
    somme = 0
    h = (b-a)/n
    x = a
    for k in range(0,n):
        somme += f(x)
        x += h
    somme *= h
    return somme
```

Méthode des rectangles à droite : on permute les deux lignes de la boucle.
Méthode du point médian : on initialise x avec x = a + h/2
Méthode des trapèzes : on initialise somme avec somme = (f(a)+f(b))/2, on permute les deux lignes de la boucle et on exécute la boucle une fois de moins (range(1,n) par exemple).

Exercice 3.

lambda x: x**(1/3)-100 définit bien une fonction continue et monotone sur $[10^5, 10^7]$, changeant de signe en 10^6.
Le problème est que Python utilise 16 chiffres significatifs, donc en manipulant des nombres de l'ordre de 10^6, Python ne retiendra que 10 décimales, et l'écart voulu de 10^{-12} ne sera pas possible.

On peut rajouter dans la fonction dicho l'instruction print(a,b,c) pour comprendre le problème → c est confondu avec a, donc l'écart entre b et a ne diminue plus.

Exercice 4.

```
def recherche_dicho(liste, x):
    a, b = 0, len(liste) - 1
    while b >= a:
        c = (a + b)//2
        if liste[c] == x:
            return c
        elif liste[c] < x:
            a = c + 1
        else:
            b = c - 1
    return - 1
```

Exercice 5.

a. Soit $c = (2a + b)/3$ et $d = (a + 2b)/3$.
Si le minimum est atteint sur $[a, c]$, alors f est croissante sur $[c, b]$ et donc $f(c) \leq f(d)$.

Par contraposée, si $f(c) > f(d)$, le minimum n'est pas atteint sur $[a, c]$ et est donc atteint sur $[c, b]$. On poursuit donc avec l'affectation $a = c$.
De même, si $f(c) < f(d)$, le minimum n'est pas atteint sur $[d, b]$ et est donc atteint sur $[c, b]$. On poursuit alors avec $b = d$.
Si $f(c) = f(d)$, alors le minimum est atteint sur $[c, d]$. Ce cas très rare ne mérite peut-être pas d'être traité à part et peut être inclus dans un des deux cas précédents.

```
def recherche_min(f,a,b,epsilon):
    while b-a > epsilon:
        c,d = (2*a + b)/3,(a + 2*b)/3
        if f(c)> f(d):
            a = c
        else:
            b = d
    return (a+b)/2
```

b. On n'obtient pas la précision de 10^{-12} demandée, mais seulement une précision de l'ordre de 10^{-8}. Cela s'explique par le fait que x^2 est de l'ordre de 10^{-16} quand x est de l'ordre de 10^{-8}, donc les calculs deviennent faux et le programme ne fonctionne plus.
On peut rajouter print(c,d,f(c),f(d)) dans la boucle pour s'en rendre compte.

c.
```
def recherche_max(f,a,b,epsilon) :
    return recherche_min(-f,a,b,epsilon)
```

Exercice 6.

a. Pour $q \geq 1$, on a $V_q = \frac{1}{dt}\left(X_q - X_{q-1}\right)$

b. Pour $q \geq 2$, on a

$$A_q = \frac{1}{dt}\left(V_q - V_{q-1}\right) = \frac{1}{dt}\left(\frac{1}{dt}\left(X_q - X_{q-1}\right) - \frac{1}{dt}\left(X_{q-1} - X_{q-2}\right)\right)$$

$$\text{donc } A_q = \frac{1}{(dt)^2}\left(X_q - 2X_{q-1} + X_{q-2}\right)$$

c. $\ddot{X}(t) + A(t)\dot{X}(t) + B(t)X(t) = C(t)$ devient à l'instant t_q :

$A_q + A(t_q)V_q + B(t_q)X_q = C(t_q)$

$\frac{1}{(dt)^2}\left(X_q - 2X_{q-1} + X_{q-2}\right) + A(t_q)\frac{1}{dt}\left(X_q - X_{q-1}\right) + B(t_q)X_q = C(t_q)$

$X_q - 2X_{q-1} + X_{q-2} + dt.A\left(t_q\right)\left(X_q - X_{q-1}\right) + (dt)^2 B\left(t_q\right)X_q = (dt)^2 C(t_q)$

$\left((dt)^2 B\left(t_q\right) + dt.A\left(t_q\right) + I_n\right)X_q = \left(2I_n + dt.A\left(t_q\right)\right)X_{q-1} - X_{q-2} + (dt)^2 C(t_q)$

D'où $M_q X_q = N_q$ avec $M_q = (dt)^2 B\left(t_q\right) + dt.A\left(t_q\right) + I_n$

et $N_q = \left(2I_n + dt.A\left(t_q\right)\right)X_{q-1} - X_{q-2} + (dt)^2 C(t_q)$

d. On a besoin de X_0 (connu) et X_1 (à déterminer).

$X_1 - X_0 = dt\, V_1$ et $V_1 = dt\, A_1$
Alors, $A_1 + A(t_1)V_1 + B(t_1)X_1 = C(t_1)$ donne
$X_1 - X_0 + dt\, A(t_1)(X_1 - X_0) + (dt)^2\, B(t_1)X_1 = (dt)^2\, C(t_1)$
Soit $M_1X_1 = X_0 + dt\, A(t_1)X_0 + (dt)^2\, C(t_1) = N_1$
Si on définit N_1 comme N_q en posant $X_{-1} = X_0$, on peut intégrer le calcul de X_1 dans la boucle en posant $X_{-1} = X_0$.

```
def calcul(n, A, B, C, T, npts):
    tq = 0
    dt = T/(npts – 1)
    Xqm1 = np.linspace(0,10,n)  # X_(0)
    Xqm2 = np.linspace(0,10,n)  # X_(-1)
    X = [Xqm1]
    for q in range(npts-1):       # ou range(1, npts) pour les vrais valeurs de q
        tq += dt
        Mq = dt*dt*B(tq) + dt*A(tq) + np.diag([1]*n)
        Nq = np.dot(np.diag([2]*n) + dt*A(tq), Xqm1) - Xqm2 + dt*dt*C(tq)
        Xqm2, Xqm1=Xqm1, np.linalg.solve(Mq,Nq)
        X.append(Xqm1)
    return np.array(X)
```

Remarques

- L’addition d’arrays effectue des additions terme à terme (comme une somme de matrices), tandis que l’addition de listes effectue une concaténation.
- De même, la multiplication d’un array par un nombre multiplie chaque coefficient de l’array.
- La multiplication matricielle de deux arrays s’effectue à l’aide de la fonction dot du module numpy. Cette fonction accepte une matrice ligne à la place d’une matrice colonne.
- En utilisant des matrix au lieu d’array, on peut effectuer les multiplications matricielles avec l’opérateur * mais dans ce cas, il faut utiliser des matrices colonnes à la place des lignes lors des calculs, ce qui alourdit le programme (M.T renvoie la transposée de M).